GRAFİK SİSTEMLERİ İÇİN FPGA CİHAZLARINDA ÇALIŞMAK ÜZERE TASARLANMIŞ MATRİS ÇARPIM MOTORU

İsmail KOYUNCU

DÜZCE ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

ELEKTRİK EĞİTİMİ ANABİLİM DALINDA

YÜKSEK LİSANS

DERECESİ İÇİN GEREKLİ ÇALIŞMALAR YERİNE GETİRİLEREK

ONAYA SUNULAN TEZ

OCAK 2008

Fen Bilimleri Enstitüsü'nün Onayı

Prof. Dr. Demet KAYA

Enstitü Müdürü

Bu tezin Yüksek Lisans Derecesinde bir tez olarak gerekli çalışmaları yerine getirdiğini onaylarım.

Yrd. Doç. Dr. Recep DEMİRCİ

Elektrik Eğitimi Anabilim Dalı Başkanı

Bu tezin Yüksek Lisans Derecesinde bir tez olarak onaylanması, düşüncemize göre, amaç ve kalite olarak tamamen uygundur.

Yrd. Doç. Dr. İbrahim ŞAHİN

Tez Danışmanı

Jüri Üyeleri

1. Yrd. Doç. Dr. İbrahim ŞAHİN ……

2. Yrd. Doç. Dr. Pakize ERDOĞMUŞ ……

3. Yrd. Doç. Dr. Recep DEMİRCİ ……

ABSTRACT

A MATRIX MULTIPLICATION ENGINE FOR GRAPHIC SYSTEMS DESIGNED TO RUN ON FPGA DEVICES

KOYUNCU, İSMAİL

Master of Science, Department of Electrical Education

Advisor: Assist. Prof. Dr. İbrahim ŞAHİN

JANUARY 2008, 76 pages

In Computer Graphics, as the number of objects in an animation scene and the number of vertices used to define each object increase, calculations of the three dimensional (3D) transformations require huge amount of CPU time. As a result, it becomes impossible to calculate transformations in real time. In this study, three different hardware modules have been designed to speed up three dimensional transformations using FPGA (Field Programmable Gate Array) chips. The modules were tested and the correctness of the modules' results were verified by comparing the modules' results with PCs' results for the same set of input data. Modules' data processing speeds were compared to various general purpose computers. The results showed that using the modules, 3D graphic transformations can be speeded up by factor of up to 12.

Keywords: FPGA, Computer Graphics, 3D Transformation, Hardware Module.

ÖZET

GRAFİK SİSTEMLERİ İÇİN FPGA CİHAZLARINDA ÇALIŞMAK ÜZERE TASARLANMIŞ MATRİS ÇARPIM MOTORU

KOYUNCU, İSMAİL

Yüksek Lisans, Elektrik Eğitimi Anabilim Dalı

Tez Danışmanı : Yrd. Doç. Dr. İbrahim ŞAHİN

OCAK 2008, 76 Sayfa

Bilgisayar grafiklerinde, üç boyutlu (3B) dönüşümlerde, animasyon sahnesindeki dönüşüme uğrayan nesne sayısı ve bu nesneleri tanımlamada kullanılan nokta sayısı arttıkça dönüşümü hesaplamak için çok fazla CPU zamanı gerekmektedir. Sonuçta, özellikle gerçek zamanlı grafik uygulamalarının hesaplanması imkânsız hale gelmektedir. Bu çalışmada, FPGA (Alan Programlanabilir Kapı Dizileri) çiplerini kullanarak üç boyutlu dönüşümleri hızlandırmak için donanım modülleri tasarlanmıştır. Tasarlanan modüller gerçek veri üzerinde işlemler yapılarak test edilmiş ve modüllerin ürettiği sonuçların doğrulanması yapılmıştır. Modüllerin veri işleme hızı değişik bilgisayarlarla karşılaştırılmıştır. Karşılaştırma sonuçları göstermiştir ki, tasarlanan modüller kullanılarak üç boyutlu grafik dönüşümleri 12 kata kadar daha hızlı gerçekleştirilebilmektedir.

Anahtar Kelimeler: FPGA, Bilgisayar Grafikleri, 3B Dönüşüm, Donanım Modülü.

Kardeşim Ayşe Yesir ve Eşi Ulaş Ercan Yesir'e ithaf ediyorum.

TEŞEKKÜR

Tez danışmanlığımı üstlenerek araştırma konusunun seçimi, yürütülmesi ve sunuma hazırlanması sırasında, değerli bilimsel görüş ve önerilerinden yararlandığım Yrd. Doç. Dr. İbrahim ŞAHİN'e teşekkür eder ve en içten minnet duygularımı sunarım.

Değerli görüşleriyle bana yol gösteren Yrd. Doç. Dr. Pakize ERDOĞMUŞ'a, yapıcı eleştirileriyle yardımlarını esirgemeyen Yrd. Doç. Dr. Recep DEMİRCİ'ye ve Yrd. Doç. Dr. A. Turan ÖZCERİT'e teşekkürü borç bilirim.

Tezin hazırlanmasında ilgileri, sabırları ve maddi manevi destekleri için aileme, çalışma arkadaşlarım olan Sadık KAŞİFOĞLU'na, Murat KARABACAK'a, Yunus BİÇEN'e ve tezin hazırlanması konusunda yardımlarını esirgemeyen Ferzan KATIRCIOĞLU'na teşekkürlerimi sunarım.

İÇİNDEKİLER

ABSTRACT…….. iii

ÖZET……...…….. iv

TEŞEKKÜR……...…… vi

İÇİNDEKİLER……...…….. vii

ŞEKİLLER DİZİNİ……..x

TABLOLAR DİZİNİ……...…… xiii

1. GİRİŞ……...……... 1

2. ÖN BİLGİ VE DİĞER ÇALIŞMALAR……..…….…… 4

2.1. FPGA Çipleri……...…..…...……...…….. 4

2.2. FPGA Tabanlı Özel Amaçlı Bilgisayarlar ...……...…… 6

2.3. Bilgisayar Grafikleri ve Üç Boyutlu Dönüşümler…….. 10

2.3.1. Öteleme (Translation)……...…….. 11

2.3.2. Döndürme (Rotation)……... 12

2.3.3. Ölçeklendirme (Scaling)…….. 14

2.3.4. Yansıma (Reflection)……... 14

2.4. Diğer Çalışmalar……...……. 17

3. ÜÇ BOYUTLU DÖNÜŞÜMLER İÇİN FPGA MODÜL TASARIMI…..... 21

3.1. Modüllerin Genel Yapısı……..……..……. 22

3.2. Tamsayı (Integer) Modülü……...…….. 25

3.2.1. Hafıza Haritası……...…….. 26

3.2.2. Tamsayı Modül Kontrolörü ve Çalışması…...……..... 28

3.3. Kesirli Sayı (Floating-Point) Modülleri…….……...……. 33

3.3.1. Kesirli Sayı Toplama ve Çarpma Üniteleri…...…….. 33

3.3.2. Tek Hafıza Üniteli Kesirli Sayı (Floating-Point) Modülü..... 36

3.3.2.1. Tek Hafıza Üniteli Kesirli Sayı Modül Kontrolörü

ve Çalışması……... 39

3.3.3. Çift Hafıza Üniteli Kesirli Sayı (Floating-Point) Modülü….. 44

3.3.3.1. Çift Hafıza Üniteli Kesirli Sayı Modül Kontrolörü

ve Modülün Çalışması……...…….. 47

4. DENEY DÜZENEKLERİ VE TEST SONUÇLARI…….. 52

4.1. Tamsayı (Integer) Modülü……...…….…….... 52

4.2. Tek Hafıza Üniteli Kesirli Sayı (Floating-Point) Modülü...……. 57

4.3. Çift Hafıza Üniteli Kesirli Sayı (Floating-Point) Modülü……..…. 60

5. SONUÇ VE GELECEKTE YAPILABİLECEK ÇALIŞMALAR.……..... 63

5.1. Gelecekte Yapılabilecek Çalışmalar..…….…... 64

KAYNAKÇA……...…...… 65

EKLER……...……..69

Ek-A : Test Verisi Üretimi İçin Yazılan C++ Program Kodu…...……. 69

Ek-B : Bilgisayar İşlem Süresi Hesaplanması İçin Hazırlanan Program Kodu.. 72

ŞEKİLLER DİZİNİ

Şekil 2.1. FPGA genel yapısı. 6

Şekil 2.2. Xilinx 4000 serisi FPGA çipinin CLB blok şeması. 7

Şekil 2.3. RC sistem genel yapısı. 7

Şekil 2.4. WildForce FPGA kartı... 8

Şekil 2.5. Nesnelerin noktalarla oluşturulması. 10

Şekil 3.1. Matris çarpım modülü blok diyagramı. 23

Şekil 3.2. Modülün ikinci seviye blok diyagramı. 23

Şekil 3.3. *Veri İşlem Ünitesi* blok diyagramı. 24

Şekil 3.4. *Çekirdek* blok diyagramı. 26

Şekil 3.5. Bütün modüller için ortak hafıza haritası. 26

Şekil 3.6. Tamsayı modül kontrolürünün *Durum* diyagramı. 30

Şekil 3.7. Tamsayı modül simülasyonu 1. 35

Şekil 3.8. Tamsayı modül simülasyonu 2. 35

Şekil 3.9. *Toplama Ünitesi* blok diyagramı. 35

Şekil 3.10. *Çarpma Ünitesi* blok diyagramı. 35

Şekil 3.11. Tek Hafıza Üniteli Kesirli Sayı modülü *Çekirdek* blok diyagramı..... 37

Şekil 3.12. *SabitYazac* blok diyagramı. 38

Şekil 3.13. *DegiskenYazacı* blok diyagramı. 39

Şekil 3.14. Tek Hafıza Üniteli Kesirli Sayı modül algoritması. 42

Şekil 3.15. Tek Hafıza Üniteli Kesirli Sayı modülün çalışmasını gösteren zaman diyagramı. 43

Şekil 3.16. Tek Hafıza Üniteli Kesirli Sayı modülün simülasyon sonuçları. 44

Şekil 3.17. Çift Hafıza Üniteli Kesirli Sayı modül blok şeması. 46

Şekil 3.18. Çift Hafıza Üniteli Kesirli Sayı modül *Çekirdek* blok diyagramı..... 46

Şekil 3.19. Çift Hafıza Üniteli Kesirli Sayı modül algoritması. 49

Şekil 3.20. Çift Hafıza Üniteli Kesirli Sayı modülün çalışmasını gösteren zaman diyagramı. 50

Şekil 3.16. Çift Hafıza Üniteli Kesirli Sayı modülün simülasyon sonuçları 1. 51

Şekil 3.16. Çift Hafıza Üniteli Kesirli Sayı modülün simülasyon sonuçları 2. 52

Şekil 4.1. *Virtex 2* FPGA çipine göre sentezlenmiş Tamsayı modülün PC'lerle karşılaştırılması. 56

Şekil 4.2. *Virtex 2P* FPGA çipine göre sentezlenmiş Tamsayı modülün PC'lerle karşılaştırılması. 56

Şekil 4.3. *Virtex 4* FPGA çipine göre sentezlenmiş Tamsayı modülün PC'lerle karşılaştırılması. 57

Şekil 4.4. *Virtex 5* FPGA çipine göre sentezlenmiş Tamsayı modülün PC'lerle karşılaştırılması. 57

Şekil 4.5. *Virtex 4* FPGA çipine göre sentezlenmiş Tek Hafıza Üniteli Kesirli Sayı modülün PC'lerle karşılaştırılması..... 60

Şekil 4.6. *Virtex 5* FPGA çipine göre sentezlenmiş Tek Hafıza Üniteli Kesirli Sayı modülün PC'lerle karşılaştırılması..... 60

Şekil 4.7. *Virtex 5* FPGA çipine göre sentezlenmiş İki Hafıza Üniteli Kesirli Sayı modülün PC'lerle karşılaştırılması..... 63

TABLOLAR DİZİNİ

Tablo 4.1. Tamsayı modülün FPGA çip istatistikleri. 53

Tablo 4.2. Deneylerde kullanılan PC'lerin özellikleri. 53

Tablo 4.3. PC'lerin tamsayı tabanlı nesne nokta verilerini işleme süreleri. 54

Tablo 4.4. Tek Hafıza Üniteli Kesirli Sayı modülün FPGA çip istatistikleri. 57

Tablo 4.5. PC'lerin kesirli sayı türünde oluşturulan nesne nokta verilerini işleme süreleri. 58

Tablo 4.6. Çift Hafıza Üniteli Kesirli Sayı modülün FPGA çip istatistikleri. 60

Tablo 4.7. PC'lerin kesirli sayı türünde oluşturulan nesne nokta verilerini işleme süreleri. 61

Tablo 4.8. *Virtex 5* Çipinin nesne nokta verilerini işleme süreleri..... 62

1. GİRİŞ

Üç boyutlu (3B) grafik dönüşümler, grafik paketlerinin ve grafik uygulama programlarının vazgeçilmez parçalarıdır. Bu dönüşümler binlerce noktadan oluşan karmaşık şekillere uygulandığında, grafik programları işlemci gücünü önemli miktarda kullanmaktadır. Özellikle animasyon işlemlerinde, animasyon sahnesindeki nesne sayısı ve bu nesneleri tanımlamada kullanılan nokta sayısı arttıkça, animasyonun gerçek zamanlı hesaplanması neredeyse imkansız hale gelmektedir. Örneğin tipik bir animasyon filminin bir sahnesinde onlarca grafik nesne bulunabilmekte ve nesneleri tanımlamak için yüz binin üzerinde nokta kullanılabilmektedir [1]. Böyle bir animasyon sahnesinin DVD standardına göre saniyede 24 defa hesaplanması gerekmektedir. Bunun anlamı, sahneyi tanımlamada kullanılan yüz binin üzerindeki noktanın yeni değerlerinin saniyede 24 defa hesaplanması gerektiğidir. Üç boyutlu animasyonda bir noktanın yeni değerinin hesaplanabilmesi için 4x4 ile 4x1'lik bir matris çarpımının gerçekleştirilmesi söz konusudur. Yüz bin noktalı bir sahnenin 1 saniyelik animasyonunda toplam 38.4 milyon çarpma ve 28.8 milyon toplama işleminin yapılması gerekmektedir. Bu ise tipik bilgisayarlarda gerçek zamanlı animasyonları imkansız hale getirmektedir.

Bu duruma çözüm olarak değişik yaklaşımlar geliştirilmiştir [2]. Bu yaklaşımlardan bazıları; gelişmiş grafik kartlarının, grafik işlemler için tasarlanmış

özel amaçlı bilgisayarların [3], daha hızlı işlemcilerin ya da paralel işlemcilerin kullanılması olarak özetlenebilir. Bu sunulan çözümlerin bazı dezavantajları bulunmaktadır. Örneğin süper bilgisayarlar oldukça maliyetlidirler. Normal bilgisayarlar, grafik işlemlerinin gerçekleştirilmesinde yeterli performans gösterememektedirler. Özel olarak tasarlanan kartlarda ise herhangi bir tasarım veya üretim hatası telafi edilememekle birlikte yeniden tasarım süreci oldukça uzun zaman kaybına sebep olmaktadır.

Çalışmanın Amacı

Bu çalışmanın asıl amacı; yukarıda sayılan yöntemlere bir alternatif olarak üç boyutlu grafik dönüşümleri hızlandırmak amacıyla, FPGA çipleri üzerinde çalışabilecek donanım modüllleri tasarlamaktır. Böylelikle 3B grafik dönüşümlerinde,

- **Daha ucuz :** (FPGA çipleri paralel bilgisayarlara, süper bilgisayarlara hatta özel tasarlanmış grafik kartlarına göre daha düşük maliyete sahiptirler)
- **Daha hızlı:** (FPGA çipleri normal bilgisayarlarla karşılaştırıldığında daha yüksek performans göstermektedirler)
- **Daha esnek:** (FPGA çipleri çok kısa sürelerde defalarca tekrar programlanabilmektedirler) **bir yapı ortaya koymaktır.**

Bu amaçla bir tamsayı tabanlı ve iki kesirli sayı tabanlı olmak üzere toplam üç değişik 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 gelen amaçlı bilgisayarlarla karşılaştırılmıştır.

Tezin İkinci Bölümünde FPGA cipleri ve RC sistemler, bilgisayar grafikleri ve üç boyutlu dönüşümler ve konu ile ilgili diğer çalışmalar kısaca özetlenmiştir. Üçüncü Bölümde, tasarlanan modüller detaylarıyla anlatılmıştır. Dördüncü Bölümde, yapılan test çalışmaları ve bu çalışmalardan elde edilen sonuçlar sunulmuştur. Beşinci Bölümde ise sonuçlar değerlendirilmiştir. Ayrıca bu bölümde daha hızlı bir modül tasarımı için gelecekte yapılabilecek çalışmalar hakkında tavsiyelerde bulunulmuştur.

2. ÖN BİLGİ VE DİĞER ÇALIŞMALAR

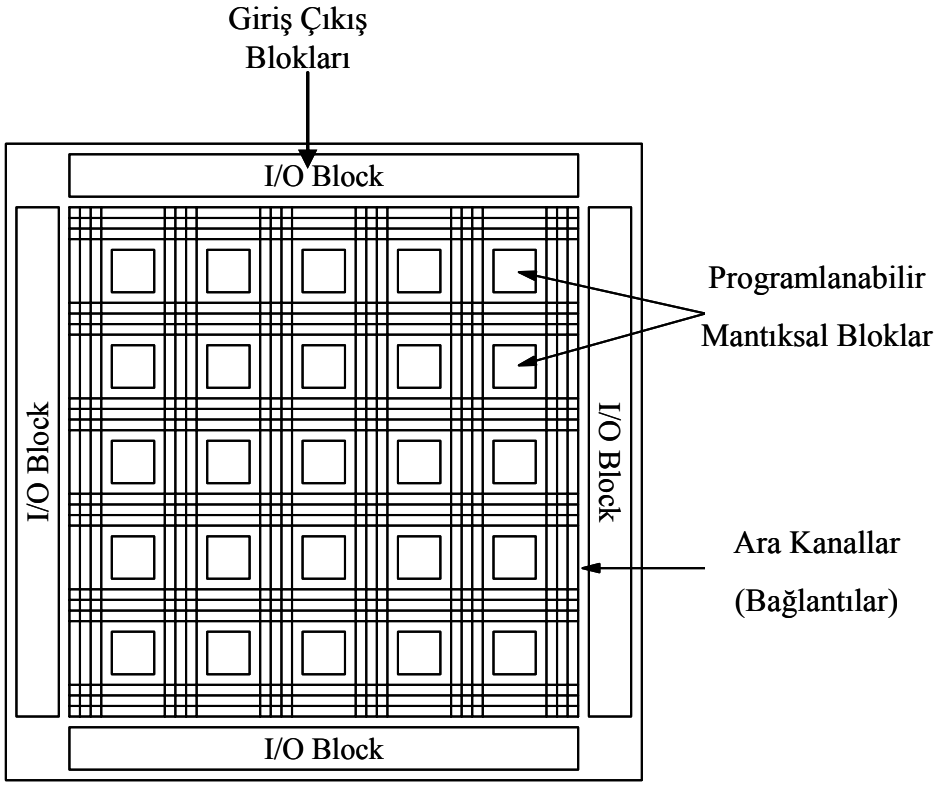
2.1. FPGA Çipleri

FPGA çipleri programlanabilir tümdevrelerdir. Tasarımcının ihtiyacı olan mantıksal fonksiyonları gerçekleştirebilmesi amacıyla, kullanıldığı yerde programlanabilir olarak üretilirler. Kullanıcının tasarladığı mantıksal devreye göre, mantıksal bloklar, aralarındaki bağlantılar ve giriş/çıkış blokları programlanır. Şekil 2.1'de genel yapısı verilen FPGA çipi, programlanabilir üç bileşenden oluşur:

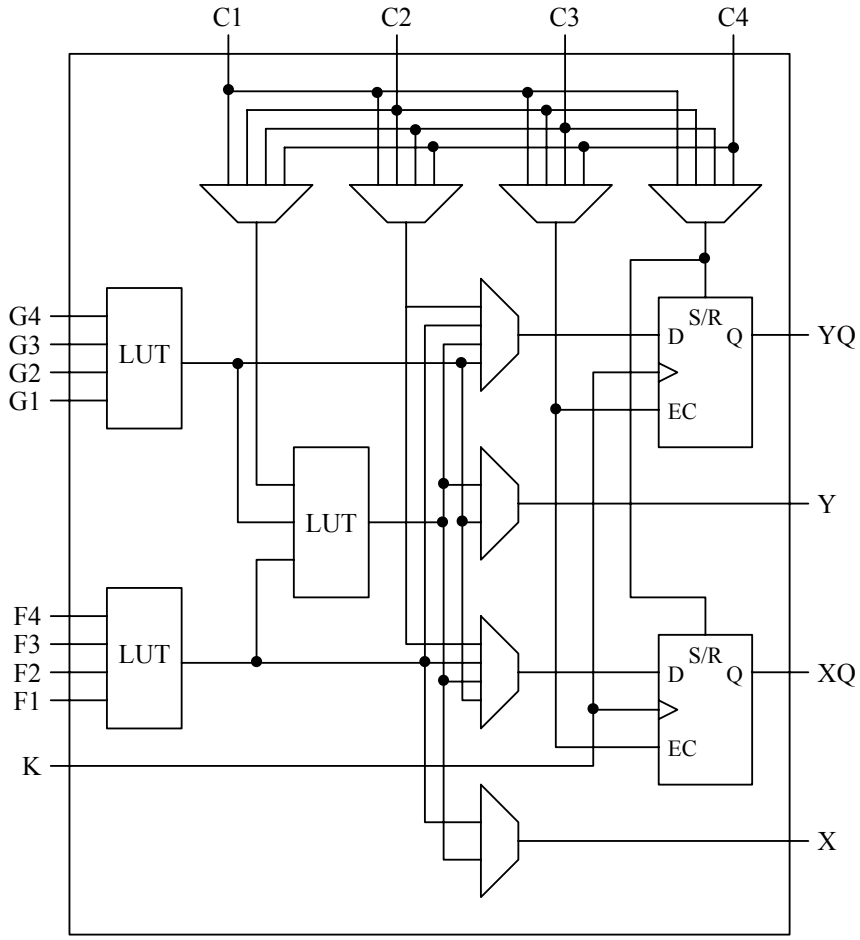
a. Yapılandırılabilir Mantıksal Bloklar (Configurable Logic Blocks (CLB)): Şekil 2.2'de blok şeması verilen CLB blokları, mantıksal fonksiyonların oluşturulabildiği Look-up table (LUT) ve Flip-Flop'lardan oluşmaktadır. CLB, kullanıcının oluşturmak istediği mantıksal devre için fonksiyonel elemanlar sağlar. CLB mimarisinin esnekliği ve simetrisi, uygulamaların kolaylıkla yerleştirilmesine ve yönlendirilmesine olanak tanır [4].

b. Giriş Çıkış Blokları (Input/Output Blocks (IOB)): IOB'ler çipin iç sinyal hatları ile çipin pinleri arasında programlanabilir bir arabirim sağlarlar. IOB'ler sayesinde FPGA pinleri giriş, çıkış ya da çift yönlü olarak programlanabilir. FPGA çipinin türüne göre bir çipteki IOB sayısı (dolayısıyla pin sayısı) 256 ile 1000 arasında veya daha fazla olabilmektedir.

c. Ara Bağlantılar (Interconnections): Bu birimler hem CLB'ler arasında hem de CLB'ler ile IOB'ler arasında bağlantıları yapılandırmada kullanılırlar. Programlanabilir olduklarından çok esnek bir yapıya sahiptirler.



Şekil 2.1. FPGA genel yapısı [5].

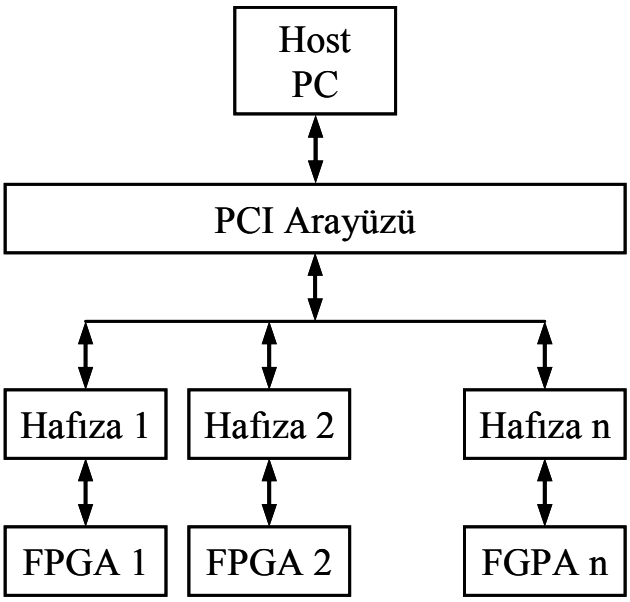


Şekil 2.2. Xilinx 4000 serisi FPGA çipinin CLB blok şeması [5].

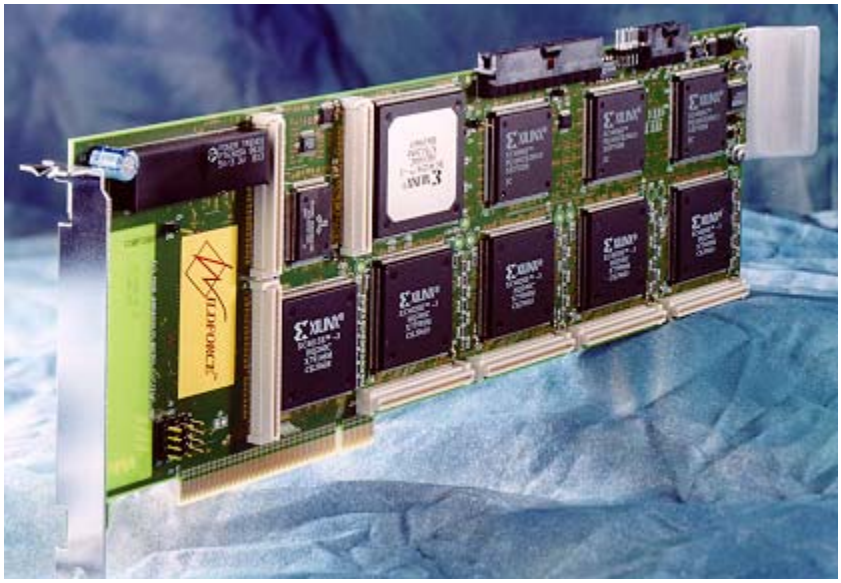
2.2. FPGA Tabanlı Özel Amaçlı Bilgisayarlar

FPGA tabanlı Özel Amaçlı Bilgisayarlar (FPGA-based Custom Computing Machines *F-CCMs*), donanım ve yazılımı bir araya getiren özel veri işleme platformlarıdır [6]. Genellikle bir *F-CCM* (*Reconfigurable Computing* (*RC*) sistem olarak da bilinir) bir adet genel amaçlı bilgisayar ve bu bilgisayara bağlı üzerinde bir ya da daha fazla FPGA ve hafıza çipleri bulunduran elektronik karttan oluşur [7, 8, 9]. Bu sistemler genel amaçlı işlemcilerin sunduğu programlanabilme özelliği ile FPGA çiplerinin sağladığı hız avantajının bir araya getirildiği sistemlerdir [10].

Uygulama programları *F-CCM*'lere uyarlanırken, programın yüksek işlemci gücü gerektiren bölümleri özel olarak tasarlanmış donanım modülü kullanılarak FPGA çipleri üzerinde çalıştırılır ve bu sayede programın daha hızlı çalışması sağlanır. İyi tasarlanmış bir *F-CCM* ve donanım modülü sayesinde, uygulama programlarının yoğun CPU gerektiren bölümlerini 10 ile 100 kat arasında hızlandırmak mümkündür [1, 5]. Hali hazırda çeşitli grafik işleme algoritmaları ve optimizasyon algoritmaları dahil olmak üzere bir çok algoritma *F-CCM*'lere uygulanmış ve genel amaçlı bilgisayarlarla karşılaştırıldığında, *F-CCM*'lerin şu ana kadar tespit edilen en düşük çalışma zamanını verdiği görülmüştür [11, 12]. Birçok firma RC sistemlerde kullanılmak üzere FPGA kartları tasarlamış ve piyasaya sunmuştur. Şekil 2.3'te RC sistemlerin genel yapısı görülmektedir. Bu yapıda *n* adet FGPA çipi ve bunlara ait yerel hafızalar bulunur. FPGA kartı bilgisayar ile veri iletişimini bir PCI ara yüzü üzerinden yapar. Şekil 2.4'te Annapolis Micro Systems firması tarafından genel amaçlı tasarlanmış bir FGPA kartı görülmektedir.



Şekil 2.3. RC sistem genel yapısı.



Şekil 2.4. WildForce FPGA kartı [13].

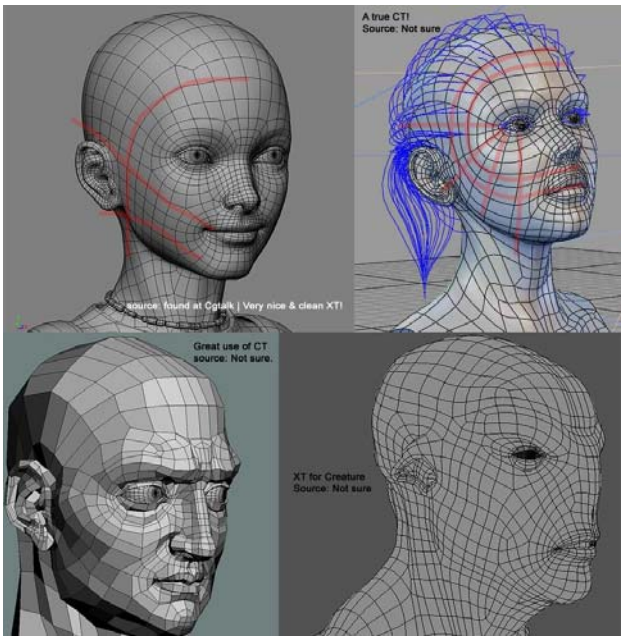
FPGA kartlarının kullanımı ise şu şekildedir: İlk olarak, daha önceden tasarlanmış donanım modülünün yapılandırma (*configuration*) bilgileri PCI yolu üzerinden FGPA çiplerine yüklenir. Ardından, yine PCI üzerinden, işlenecek veri kart üzerindeki yerel hafıza ünitelerine aktarılır. Bu aşamadan sonra FPGA çiplerine özel bir başlat sinyal gönderilerek, çiplerin hafızadaki veriyi işlemesi sağlanır. Çipler bu verileri işlerken, ana bilgisayar diğer işlerine devam eder. Çipler işlemi bitirince, bilgisayara bir kesme sinyali göndererek işlenmiş verinin hazır olduğunu bildirir. Son olarak bilgisayar yine PCI üzerinden FPGA kartı üzerindeki hafızalara erişerek işlenmiş veriyi alır [7]. FPGA yerel hafızası ile bilgisayar hafızası arasında hızlı veri iletişimini gerçekleştirmek için DMA (Direct Memory Access-Doğrudan Hafıza Erişimi) kullanır.

FPGA Çiplerin Avantajları

FPGA çipleri, ilk üretim aşamasında, üretimin piyasaya hızlı bir şekilde sunulması söz konusu olduğunda çok büyük avantaj sağlarlar. ASIC (Application Specific Integrated Circuit- Uygulamaya Özel Tümdevre) çipleri, uzun süre gerektiren çeşitli tasarım ve üretim aşamalarından geçtikten sonra ancak kullanıma hazır hale gelebilmektedir. Fakat FPGA çipleri kullanılarak bu süre sadece tasarlanan ürünün FPGA çiplerine yüklenmesi kadar kısa bir sürede gerçekleştirilebilmektedir. Ayrıca tasarımda yapılan bir hatadan dolayı ASIC çipleri, kullanılamaz hale gelmesine rağmen bu dezavantaj FPGA çiplerinde yeniden programlanabilme özelliğinden dolayı, çok kısa bir sürede ve ekonomik bir zarara uğranılmadan telafi edilebilmektedir. Çiplerin programlanma süresi, boyutlarına ve tasarıma bağlı olarak yaklaşık 200ms'den daha az sürmektedir. FPGA çipleri sınırsız olarak tekrar programlanabilir ve 550MHz saat hızlarına çıkabilirler [4, 14]. Çipleri yapılandırmak için tasarlanan donanım modülleri, sadece bir özel problemin çözümü için tasarlandıklarından yazılıma göre çok hızlı çalışmaktadırlar. Ayrıca birden fazla FPGA çipi birbiri ile paralel olarak çalışabildiği hatta tek bir çip içerisinde birden fazla modül yerleştirilebildiği için yazılıma göre çok yüksek hız kazançları elde edilebilmektedir. FPGA çipleri üretici firmaları arasında, Altera, Xilink, Flex, Lucent, Actel sayılabilir. Bu firmalar kendi üretmiş oldukları çiplere özel isimler vermektedirler. Örneğin Xilinx firması, üretmiş olduğu çiplere Spartan, Virtex [14] gibi isimler verirken Altera firması ürettiği çiplere Cyclone ve Stratix [15] gibi isimler vermektedir.

2.3. Bilgisayar Grafikleri ve Üç Boyutlu Dönüşümler

Grafik tasarımlar ve animasyonlar için çeşitli grafik paketleri (API-Application Programming Interface-Uygulama Programlama Arayüzü) geliştirilmiştir. Bu paketler genellikle grafik işlemleri yerine getirebilmek için birçok fonksiyon içerirler. Bu grafik paketlerinden en yaygın olarak bilinenleri OpenGL (Graphic Library) ve Direct X'tir. Paket ile gelen fonksiyonlara C/C++ başta olmak üzere birçok programlama dilinden erişim mümkündür. Bu paketlerle animasyonların oluşturulabilmesi için nesnelerin 2 veya 3 boyutlu ortamda matematiksel olarak noktalar, çizgiler ve yüzeyler olarak modellenmesi gerekmektedir. Basit bir nesneyi oluşturmak için yüzlerce nokta tanımlanması gerekmektedir. Şekil 2.5'te noktaların tanımlanarak birleştirilmesiyle oluşturulmuş çeşitli şekiller verilmiştir.



Şekil 2.5. Nesnelerin noktalarla oluşturulması [16].

Grafik paketlerinde nesneler tanımlanırken (modellenirken), nesne üzerinde bulunan noktaların kartezyen koordinat sistemindeki üç boyutlu koordinat değerleriyle (*x, y, z*) tanımlanırlar. Üç boyutlu (3B) grafik dönüşümler, grafik paketlerinin ve grafik uygulama programlarının vazgeçilmez parçalarıdır.

Nesneler üzerinde genel olarak *öteleme, döndürme* ve *ölçeklendirme* olmak üzere temel bir kaç değişik dönüşüm veya bunların kombinasyonu uygulanır. Bu dönüşümlerden üç boyutlu *döndürme* ve *ölçeklendirme* işlemleri 3x3 ile 3x1 matris çarpımı olarak gerçekleştirilir. Fakat *öteleme* işlemi diğer işlemler gibi doğrusal bir işlem olmadığından matris çarpımı şeklinde ifade edilemez. Bu işlemi de doğrusal hale getirebilmek için kartezyen koordinat sisteminde (*x, y, z*) olarak tanımlanan nokta değerleri homojen koordinat sistemine çevrilerek bütün grafik dönüşümleri standart 4x4 ile 4x1 matris çarpımı haline getirilir. Bu dönüşüm işlemi sırasında nokta *x, y, z* ile birlikte bir dördüncü değer olarak *w* ile (*x, y, z, w*) olarak ifade edilir ($w \neq 0$ olmak kaydıyla). *w* değeri olarak genellikle 1 alınır.

Aşağıdaki eşitliklerde dönüşüm denklemleri parametrik (2, 3, 4, 5, 11) ve matris çarpım notasyonları (1, 7, 10) verilmiştir. Bu denklemlerde, *P* dönüşüme uğrayan noktanın orijinal koordinatı, *P*' ise yeni koordinatını temsil etmektedir.

2.3.1. Öteleme (Translation)

Öteleme işleminde *tx*, *ty* ve *tz* üç boyutlu ortamda nesnenin her boyuttaki öteleme miktarını gösterir. $P = (t_x, t_y, t_z)$ koordinatında bulunan bir noktanın $T(t_x, t_y, t_z)$ öteleme vektörü ile ötelenmesinin ardından yeni koordinatı,

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (1)$$

matris çarpımıyla hesaplanır. Bu hesaplamanın sonucunda yeni nokta değerleri

$$\begin{aligned} x' &= x + t_x \\ y' &= y + t_y \\ z' &= z + t_z \end{aligned} \quad (2)$$

eşitlikleri ile hesaplanır. Buradan kısaca *öteleme* dönüşümü parametrik olarak

$$P' = T(t_x, t_y, t_z) \cdot P \quad (3)$$

şeklinde de belirtilebilir.

2.3.2. Döndürme (Rotation)

Üç boyutlu uzayda *döndürme* işlemi belirli bir eksen etrafında gerçekleştilirilir. Bundan dolayı, bir nesne üzerinde *döndürme* işlemi yapabilmek için hem döndürme açısını hem de döndürme ekseninin belirtilmesi gereklidir. *Döndürme* işleminde θ nesnenin dönme açısını göstermektedir. *P*(*x, y, z*) noktasını *z* ekseni etrafında θ açısı kadar döndürmek için,

$$\begin{aligned} x' &= x\cos\theta - y\sin\theta \\ y' &= x\sin\theta + y\cos\theta \\ z' &= z \end{aligned} \quad (4)$$

eşitlikleri kullanılarak,

$$P' = R_z(\theta) \cdot P \tag{5}$$

denklemi elde edilir.

z- ekseni etrafında *döndürme* dönüşümünü (6) homojen koordinat sisteminde matris çarpımı olarak belirtmek istersek,

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{6}$$

eşitliğine ulaşırız. Buna göre z- ekseni etrafında *döndürme* dönüşüm matrisi de,

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}$$

olacaktır. Benzer şekilde, *x* ve y eksenleri etrafında *döndürme* işlemi de (8, 9) aşağıda verilen dönüşüm matrisleri kullanılarak gerçekleştirilir.

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9}$$

2.3.3. Ölçeklendirme (Scaling)

Homojen koordinat sisteminde $P(x, y, z)$ noktasının orijin noktasına göre *ölçeklendirme* işleminde *sx*, *sy* ve *sz* nesnenin her bir boyuttaki ölçeklendirme katsayısını belirtir. Buna göre *ölçeklendirme* dönüşüm denklemi,

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{10}$$

şeklinde yazılır ya da kısaca,

$$P' = S(s_x, s_y, s_z) \cdot P \tag{11}$$

olarak yazılabilir.

2.3.4. Yansıma (Reflection)

Temel dönüşümler olan *öteleme*, *döndürme* ve *ölçeklendirme* dönüşümleri kadar yaygın olmasa da bazı özel amaçlar için kullanılan diğer bir dönüşüm işlemi ise *yansıma* dönüşümüdür.

Üç boyutlu *yansıma* dönüşümü orijin noktasına ya da koordinat eksenlerine göre gerçekleştirilir. *Yansıma* dönüşümü sadece koordinat değerlerinin işaretlerini değiştirme yoluyla elde edilebilir. Eğer bir noktayı oluşturan noktaların *x, y*, ve *z* koordinatlarının işaretleri değiştirilirse orjin noktasına göre yansıma dönüşümü (12) elde edilmiş olur. Bunun için kullanılacak *yansıma* dönüşüm matrisi de

$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (12)$$

olur. İki koordinatın işaretinin değişmesiyle koordinat eksenlerine (13), tek koordinatın işaretinin değiştirilmesiyle de koordinat düzlemlerine (14) göre yansımalar elde edilmiş olmaktadır.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (13)$$

x eksenine göre *y* eksenine göre *z* eksenine göre

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (14)$$

xy düzlemine göre *yz* düzlemine göre *xz* düzlemine göre

yansıma dönüşümleri elde edilmiş olur [17].

Yukarıdaki eşitliklerden de görüleceği üzere dönüşüm işlemleri 4x4 ile 4x1 matris çarpımı halinde gerçekleştirilmektedir. Bunların dışında grafik paketlerinde, *eğme* (shearing), *gölgelendirme* (shadow) gibi daha birçok dönüşüm yine 4x4 ile 4x1 matris çarpımı şeklinde gerçekleştirmektedir.

Hatta bu dönüşümlerin kombinasyonu da yine aynı formatta olmaktadır.

Bu dönüşümler tek bir noktanın dönüşümü için geçerlidir. Eğer nesneye uygulanmak istenirse nesnenin bütün noktalarının birer birer dönüşüm matrisiyle çarpılması ve yeni nokta değerlerinin hesaplanması gereklidir.

Bu şekillerden bir sahnede yüzlerce olabilir. Grafik paketleri bu matematiksel modeller üzerinde yukarıda açıklanan grafik dönüşümleri uygulayarak animasyonları oluşturur. Yani basit bir kamera hareketiyle ilgili bir animasyonda, bütün nesnelerin sahnedeki yeni görünümleri (nesneleri tanımlayan noktaların yeni değerleri) bu dönüşümlerde verilen matris çarpımları yoluyla hesaplanır. Örneğin tipik bir animasyon filminin bir sahnesinde onlarca grafik nesne bulunabilmekte ve bu nesneleri tanımlamak için yüz binin üzerinde nokta kullanılmaktadır [1]. Böyle bir animasyon sahnesinin DVD standardına göre saniyede 24 defa hesaplanması gerekmektedir. Bunun anlamı, sahneyi tanımlamada kullanılan 100 binin üzerindeki noktanın yeni değerlerinin saniyede 24 defa yeniden hesaplanmasıdır. Üç boyutlu animasyonda bir noktanın yeni değerinin hesaplanabilmesi için 4x4 ile 4x1'lik bir matris çarpımının gerçekleştirilmesi söz konusudur. 100 bin noktalı bir sahnenin 1 saniyelik animasyonunda toplam 38.4 milyon çarpma ve 28.8 milyon toplama işleminin yapılması gerekmektedir.

Bu dönüşümler binlerce noktadan oluşan karmaşık şekillere uygulandığında, grafik programları çok fazla CPU zamanı gerektirmektedir. Özellikle animasyon işlemlerinde, animasyon sahnesindeki nesne sayısı ve bu nesneleri tanımlamada kullanılan nokta sayısı arttıkça, animasyonun gerçek zamanlı hesaplanması imkansız hale gelmektedir.

Bu çalışmada yukarıda belirtilen matris çarpımını *n* tane nokta üzerinde gerçekleştirebilecek bir modül tasarlanmıştır. Modülün detayları üçüncü bölümde verilmiştir.

2.4. Diğer Çalışmalar

FPGA tabanlı özel tasarlanmış modüller bilimsel alanda yüksek performans, esneklik ve düşük maliyet gibi önemli parametreleri elinde bulundurmakla büyük ilgi çekmiştir.

Bilgisayar teknolojisi ve bilgisayar grafikleri [18, 19], bulanık mantık [20], görüntü ve sinyal işleme [21, 22], güç elektroniği [23], mikroişlemcilerin geliştirilmesi [24, 25] ve özel amaçlı tasarımlar [26, 27, 28, 29] hakkında pek çok çalışmalar yapılmıştır.

Çalışmamızda temel etken bilgisayar grafiklerini hızlandırmak amacıyla çeşitli üç boyutlu matris çarpım modülü tasarımı olduğu için yapılan literatür taraması da bu yönde olmuştur. Bu yönde yapılan çalışmalardan bir tanesi F. Bensaali ve arkadaşlarının yaptığı çalışmadır [2]. Çalışmada yüksek hızlı donanım tanımlama dillerinden biri olan Handel-C dili kullanılarak OpenGL gibi grafik paketleri ve sinyal işleme uygulamaları için düşük maliyetli ve yüksek hızlı, üç boyutlu matris çarpım modülü gerçekleştirilmiştir. Tasarımı yapılan modül, tamsayı (integer) tabanlı olarak çalışmaktadır. Çalışmada hedeflenen tasarım Celoxica RC-1000-FPGA kartında, Xilinx XCV200E-6 Virtex-E FPGA çipi kullanılarak gerçekleştirilmiştir.

Çalışmanın sonucunda, donanım tabanlı hızlandırma yapılarının bilgisayar grafikleri ve görüntü işleme için ümit verici yaklaşımlardan biri olduğu ve donanımın her zaman yazılımdan hızlı olduğu temel kuralına dayanarak tüm grafik kartı algoritmalarında var olan pahalı kartlar yerine, FPGA çipleri kullanılarak, paralel matris çarpım yaklaşımı ile üç boyutlu matris dönüşümleri için hızlı ve düşük maliyetli çözüm üretilebileceği vurgulanmıştır.

Bu alanda yapılmış çalışmalardan bir diğeri Nicolas Boullis ve Arnaud Tisserand'ın sabit matris ile donanım tabanlı çarpıcının optimizasyonu alanında yaptığı çalışmadır [30]. Bu çalışmada sabit matrisler ile donanım çarpım modülünün optimizasyonuna dair bazı gelişmeler sunulmuştur. Çalışmanın amacı, geliştirilen modül kullanılarak sabit matris ile vektör çarpımının otomatik olarak yapılmasıdır. Sabit matris çarpımı problemlerinin giderilmesi için yeni algoritmalar geliştirilmiştir. Geliştirilen modüllerin sentezlenmesinde Xilink ISE araçları ve FPGA çipi olarakta Virtex XCV 200-5 kullanılmıştır. Elde edilen sonuçlarda diğer çözüm önerilerine göre % 40'a varan hız kazancı sağlandığı belirtilmiştir.

Yukarıda da değinildiği gibi yapılan çalışmalar sadece bilgisayar teknolojileri alanı ile sınırlı kalmamış birçok alanda kullanılabilir hale gelmiştir. Bu çalışmalardan birisi, E. Cesur'un FPGA çiplerini kullanarak gerçekleştirdiği iki boyutlu Gabor Filtresi çalışmasıdır [31]. Çalışma 1988 yılında L. O. Chua ve L. Yang tarafından ortaya atılıp, analog yapıda bulunan ve çok hızlı işlem yapabilen Hücresel Sinir Ağları (H.S.A.) alanında gerçekleştirilmiştir. H.S.A. ile görüntü işlemede Gabor süzgeci ön işlem olarak kullanılmıştır. Gabor süzgecinin sayısal devrelerde gerçekleştirilmesinde birçok problemle karşılaşıldığından bu sorunların

ortadan kaldırılması için B. Shi H.S.A. Gabor tip süzgeçler geliştirilmiştir. Sunulan çözümlerin gerçeklemesi FPGA üzerinde yapılmıştır.

Yapılan çalışmalardan bir diğeri A. Uçar'ın FPGA Türkçe fonemlerin sınıflandırılmasında kullanılan sinir ağının FPGA çiplerine uygulanması ile ilgili çalışmasıdır [32]. Çalışmada Türkçe fonemlerin sınıflandırılmasında kullanılan sinir ağı, FPGA çipi kullanılarak gerçekleştirilmiştir. Gerçek zamanlı hızlara ulaşmanın, sinir ağlarının donanımsal olarak gerçekleştirilmesiyle mümkün olabileceği belirtilmiş ve bu amaçla radyal tabanlı fonksiyon ağı tasarlanarak oluşturulan donanım mimarisi FPGA çipi için sentezlenmiştir. Sentez sonuçları FPGA ile gerçeklenen sistemin, gerçek zamanlı fonem sınıflandırması yapabilen bir sistem olduğunu göstermiştir.

Bir diğer çalışma Murat Çakıroğlu ve arkadaşlarının 80C51 mikrodenetleyicilerinde timer-counter yapılarının FPGA mimarileri kullanılarak geliştirilmesi ile ilgili çalışmasıdır [33].

Çalışmadaki temel amaç, 80C51 mikrodenetleyicilerinin endüstriyel sahadaki kullanılabilirliğini arttırabilmek için timer-counter, seri haberleşme birimlerinin sayısı ve hafıza birimlerinin boyutu gibi etkenleri yeniden düzenleyerek daha modüler bir hale getirmek ve endüstriyel alandaki bazı eksikliklerini gidermektir. Yapılan tasarım çalışmaları yüksek seviyeli donanım tanımlama dili olan VHDL dilinde yapılmış ve tasarlanan sistemin uygulaması ise büyük kapasite ve esneklik gibi birçok avantajından dolayı FPGA çiplerinde yapılmıştır.

Sistemin sentezlenmesi ve geliştirilmesi aşaması Xilinx ISE 5.1.i ve Mentor Graphics programları ile yapılmış, gerçekleme aşamasında ise Xilinx firmasının

ürettiği çip ailelerinden biri olan Virtex ailesinin XCV300 FPGA çipi kullanılmıştır. Çalışma sonuçlarına göre tasarlanan mikrodenetleyicinin çalışma frekansı 40Mhz'dir. Sistem kanallı olarak tasarlanmış ve mikrodenetleyiciden daha kısa bir komut işleme süresi elde edildiği belirtilmiştir.

3. ÜÇ BOYUTLU DÖNÜŞÜMLER İÇİN FPGA MODÜL TASARIMI

Bu çalışmada üç boyutlu homojen dönüşümler için FPGA tabanlı ortamlarda kullanılabilecek modüller tasarlanmıştır. Grafik paketleri hem tamsayı tabanlı dönüşümleri hemde kesirli sayı tabanlı dönüşümleri desteklemektedirler. Bu yüzden bu çalışmada üç değişik donanım modülü tasarlanmıştır. Bütün modüller sabit bir 4x4 matris ile bir dizi 4x1 matris serisini çarparak yeni bir dizi 4x1 matris serisi oluşturacak şekilde düzenlenmiştir. Modüllerden ilki tamsayı (integer) türünde veri işleyebilen Tamsayı modüldür. İkinci olarak tasarlanan modül, kesirli sayılar (floating-point) üzerinde çalışan Tek Hafıza Üniteli Kesirli Sayı modülüdür. Üçüncü aşamada tasarlanan modül ise çift hafıza ünitesine sahip, optimize edilerek daha hızlı hale getirilen ve ikinci aşamada tasarlanan kesirli sayı modülüne göre yaklaşık 2,5 kat daha hızlı çalışabilen, Çift Hafıza Üniteli Kesirli Sayı modülüdür. Modüller matris çarpım işlemi sırasında, istenilen sayıda noktayı (yüz, bin, 100 bin v.b.) çarpabilecek yapıda tasarlanmıştır. Modüller çarpma işlemini gerçekleştirmeye başlamadan önce kaç tane nokta için hesaplama yapılacağını veri yolu aracılığı ile hafızadan okumaktadır.

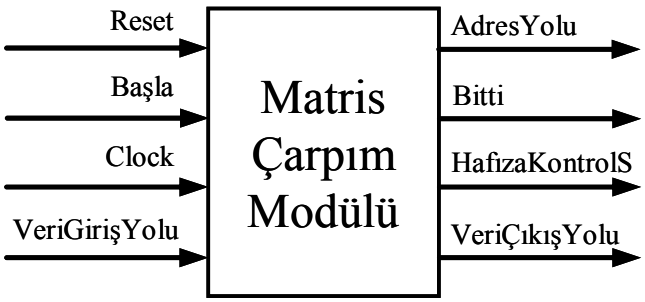
Çalışma da oluşturulan bütün modüller bir donanım tanımlama dili olan VHDL'de tasarlanmış ve sentez aracı olarak Xilinx ISE 9.2 kullanılmıştır.

Tasarlanan modüllerden ilki olan Tamsayı modülünde 32-bit işaretli tamsayı formatı kullanılmıştır. Kesirli Sayı modüllerde kullanılan kesirli sayılar ise 32-bit ve IEEE 754-1985 standardına uygun biçimdedir.

3.1. Modüllerin Genel Yapısı

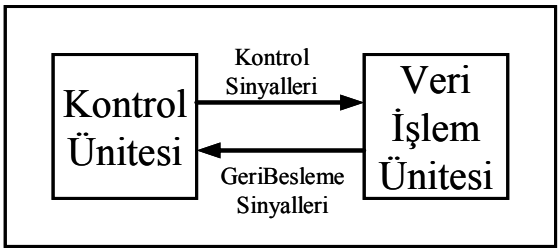
Bu çalışmada tasarlanan üç modülün en üst seviye, ikinci seviye ve veri işleme üniteleri aynı yapıya sahiptir. Modül türüne göre veri işleme ünitesinde yer alan çekirdek ve modül kontrolörü değişik yapıda tasarlanmıştır. Bu kısımda modüllerin bu ortak yapısı anlatılacak, ardından her bir modülün yapısı ayrıntılarıyla anlatılacaktır.

Şekil 3.1'de tasarlanan modüllerin en üst seviye blok diyagramı görülmektedir. Modüller üzerinde 32-bitlik veri giriş ve çıkış yolları ayrı ayrı verilmiştir. Bunun sebebi modüllerin herhangi bir özel FPGA kartına göre tasarlanmamış olmasıdır. Modül kullanıcıları, kullanılacağı karta uygun bir hafıza ara yüzü aracılığıyla veri giriş ve çıkış yollarını birleştirerek çift yönlü tek bir veri yoluyla hafıza ünitesine bağlantı yapmalıdırlar. Modüller 32-bitlik bir adres yoluna sahiptir. Bu sayede 4 GB'lık bir hafıza alanını adresleme imkanı sunulmuştur. `Reset`, `Basla` ve `Bitti` sinyalleri modüllerin zamanlaması ve modül ile modülün bağlı olduğu bilgisayar arasında senkronizasyonu (hand-shaking) sağlamak için kullanılır. Hafıza ile senkronizasyonu sağlamak ve veri okuma yazma işlemlerini yapabilmek için ayrıca hafıza kontrol sinyalleri bulunmaktadır.



Şekil 3.1. Matris Çarpım modülü blok diyagramı.

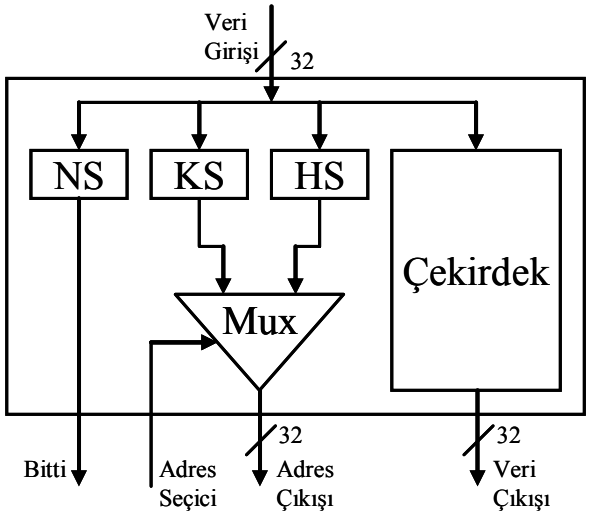
Şekil 3.2'de matris çarpım modüllerinin ikinci seviye blok diyagramı görülmektedir. *Kontrol Ünitesi* ve *Veri İşlem Ünitesi* olmak üzere, modüller iki bölümde tasarlanmıştır. *Kontrol Ünitesi* gerekli zamanlarda ihtiyaç duyulan kontrol sinyallerini üreterek, hafızadan verilerin alınmasını, verilerin *Veri İşlem Ünitesi* üzerinde işlenerek sonuçların hafızaya geri yazılmasını sağlar. *Veri İşlem Ünitesi* ise kontrolörden aldığı sinyallerle 4x4 ile 4x1 matris çarpımını yapan donanımı içerir. Çalışma kapsamında tasarlanan üç değişik modül için üçer değişik *Kontrol Ünitesi* ve *Veri İşlem Ünitesi* tasarlanmıştır.



Şekil 3.2. Modülün ikinci seviye blok diyagramı.

Şekil 3.3'te *Veri İşlem Ünitesi'*nin blok diyagramı görülmektedir. *Veri İşlem Ünitesi*, *Çekirdek* ve *Veri Erişim Sayaçları* olmak üzere iki bölüm olarak tasarlanmıştır. Veri erişim kısmında üç adet sayaç kullanılmıştır. *NoktaSayacı* (*NS*) dönüşüme uğrayacak nokta sayısının kontrolü içindir. Modül dönüşüm işlemine

başlamadan önce, dönüşüme uğrayacak nokta adedi bu yazaca yüklenir. Herbir nokta işlendikten sonra sayacın değeri bir azaltılır. Sayaca eklenen özel bir tasarım sayesinde sayaç, değeri sıfıra ulaştığında modülün kontrol ünitesine `Bitti` sinyali göndererek dönüşümün sonlanmasını sağlar. *KaynakSayacı* (*KS*) dönüşüme uğrayacak noktaların hafıza adresini, *HedefSayacı* (*HS*) ise dönüşüm sonrasında oluşan yeni nokta değerlerinin hafızada yazılacağı adresi takip etmek için kullanılmıştır. Bu sayaçların değeri dönüşüm işlemine başlamadan önce hafızadan okunur. Her bir nokta için *x, y, z* ve *w* olmak üzere hafızada dört değer bulunduğundan, her bir noktanın işlenmesi sırasında *KS* ve *HS*'nin değerleri dörder defa arttırılır. Bu bölümde yer alan multiplexer, adres seçicidir. Kontrolörden gelen `AdresSecici` sinyaline göre *KS* ya da *HS*'nin adres yoluna yazılmasını sağlar. *Veri İşleme Ünitesi*'nde yer alan *Çekirdek* her bir modül türüne göre ayrı ayrı tasarlanmıştır. Bundan sonraki bölümlerde modüller anlatılırken *Çekirdek* yapısı üzerinde yoğunlaşılacaktır.

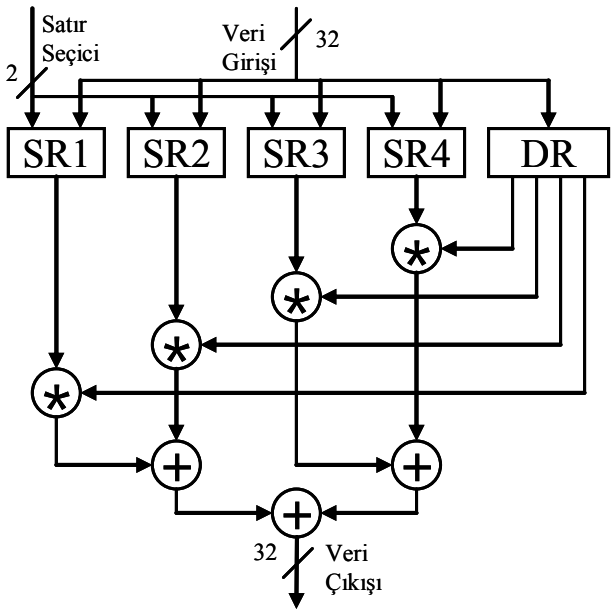


Şekil 3.3. *Veri İşlem Ünitesi* blok diyagramı.

3.2. Tamsayı (Integer) Modülü

Şekil 3.4'te tamsayı modülüne ait *Çekirdek* ünitesinin blok diyagramı görülmektedir. *Çekirdek*, dört adet 4x32-bitlik *SabitYazac*'lar (*SR1...SR4*), bir adet 4x32-bitlik *DonusumYazacı* (*DR*), dört adet 32-bitlik tamsayı çarpıcı ve üç adet 32-bitlik tamsayı toplayıcından oluşmaktadır. 4x32-bitlik *SabitYazac*'lar dönüşüm matrisinin değerlerini saklamak için kullanılır. Her bir *SabitYazac*'ın içerisinde dört adet 32-bitlik yazaç yerleştirilmiştir. Dönüşüm matrisinin 16 elemanını saklamak için 4 adet *SabitYazac* içinde toplam 16 adet 32-bitlik veri depolama kapasitesi vardır. Her bir *SabitYazac*'ta dönüşüm matrisinin bir kolonundaki değerler tutulmaktadır. Dönüşüm işlemi başlamadan önce dönüşüm matrisinin değerleri bu yazaçlara sırası ile hafızadan okunur. Okunan bu değerler dönüşüm işlemi bitene kadar bu yazaçlar içerisinde tutulurlar. Dönüşüme uğrayacak noktanın değerleri (*x, y, z, w*) ise *DonusumYazacı*'nda tutulmaktadır. Bir noktanın dönüşümü sırasında, önce noktanın (*x, y, z, w*) değerleri hafızadan *DR* yazacına okunur. Ardından *SR* yazacındaki değerler ile çarpılıp toplanarak noktanın yeni değerleri (*x′, y′, z′, w′*) olarak hesaplanır. Hesaplanan bu değer doğrudan hafızaya yazılır. *SabitYazac*'ın ve *DonusumYazacı*'nın detayları 3.3.2. nolu Tek Hafıza Üniteli Kesirli Sayı (Floating-Point) Modülü bölümünde anlatılmıştır.

Çekirdek içerisinde kullanılan toplama ve çarpma üniteleri 32-bit işaretli tamsayı *Çarpma* ve *Toplama Üniteleri*'dir. Bu üniteler için özel bir tasarım yapılmamış, VHDL sentez aracının bu üniteleri sentezlemesi sağlanmıştır.



Şekil 3.4. *Çekirdek* blok diyagramı.

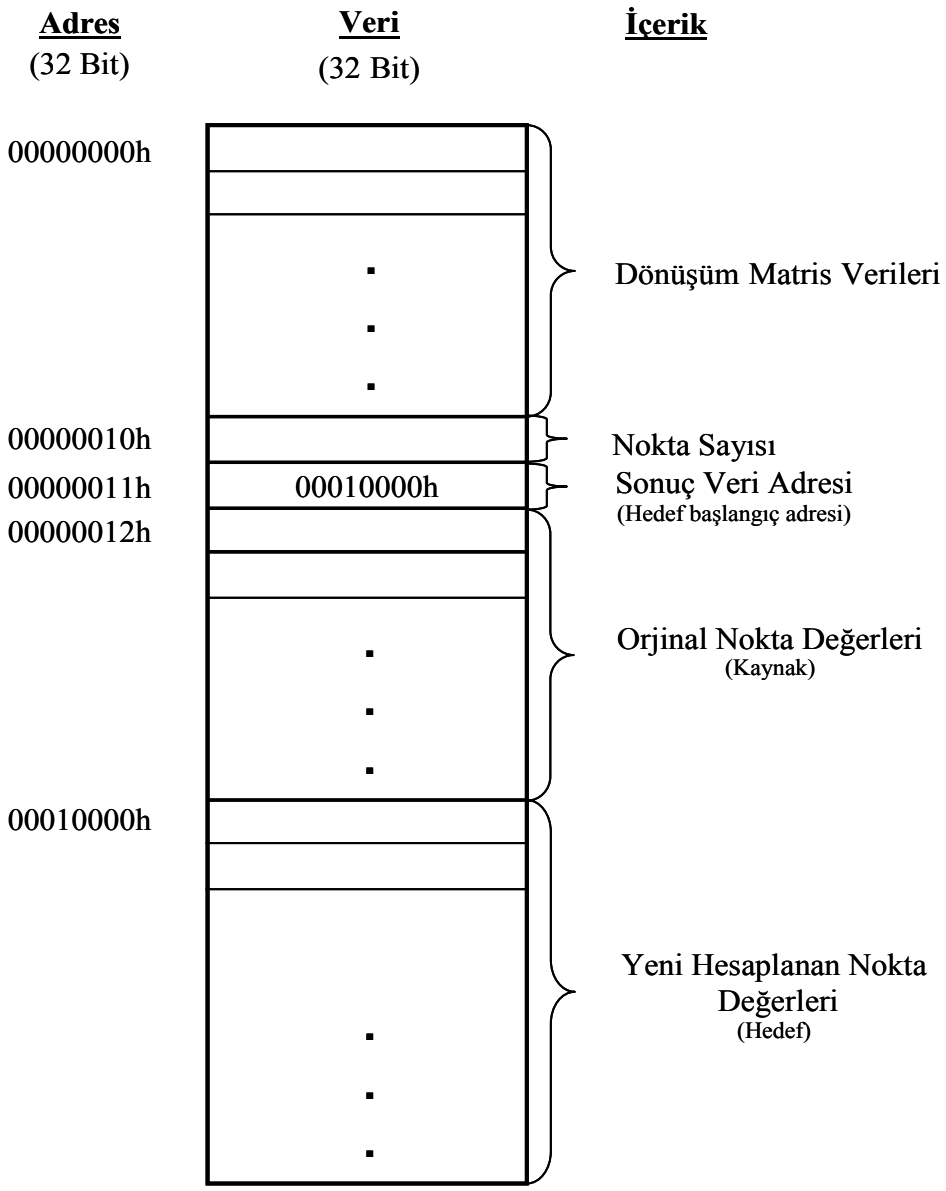
3.2.1. Hafıza Haritası

Şekil 3.5'te tasarlanan her üç modülün kullandığı ortak hafıza haritası görülmektedir. Hafızada adreslenebilir herbir veri alanının 32-bit olduğu varsayılmıştır. Hafıza haritası beş bölümden oluşmaktadır. Bu bölümler; Dönüşüm Matrisi veri alanı, Nokta Sayacı ve Sonuç Veri Adresi alanları, dönüşüme uğrayacak noktaların saklandığı kaynak alanın ve yeni değerlerin yazılacağı hedef alanlarıdır.

İlk bölüm hafıza haritasının `0…0h` adresinden başlayarak `0…Ah` adresine kadar ki ilk 16 adresden oluşur ve her biri 32-bit olan 16 adet dönüşüm matris verilerinin saklanması için kullanılır. `0…10h` adresinde tasarlanan modülün kaç tane veriyi işleyeceği bilgisi bulunmaktadır. Yapılan hesaplamalar sonucunda elde edilen yeni değerlerin hafızada hangi adresten itibaren yazılmaya başlanacağını gösteren adres bilgisi Sonuç Veri Adresi adı altında `0…11h` adresinde tutlur. Hafızada `0…12h`

adresinden itibaren noktanın orijinal değerleri ve ardından hesaplanan yeni değerlerin yazılacağı alan yer almaktadır. `0…10h` ile `0…11h` adresleri sırasıyla dönüşüme uğrayacak nokta adedini ve hesaplanan yeni değerlerin hafızada yazılmaya başlanacağı ilk adresi saklamak için ayrılmıştır. Bu alanlara yazılacak değerler PC tarafından hesaplanarak yazılmalıdır.

Dönüşme uğrayacak orjinal nokta değerleri `0…12h` adresinden başlayarak `Kaynak`'a yerleştirilmelidir. Bu alanın boyutu dönüşüme uğrayacak nokta sayısına göre değişiklik göstermektedir. `Kaynak` alanın bitiminde sonraki herhangi bir adresten itibaren yeni hesaplanan değerlerin yazıldığı hedef alanı başlayabilir. `Hedef` alanın boyutu da işlenecek nokta sayısına göre değişiklik göstermektedir. İstenirse `Kaynak` ve `Hedef` alanları üst üste çakıştırılabilir. Bu durumda modül bir adresten okuduğu bilgileri dönüşüme uğrattıktan sonra yine aynı adrese yazar. Bu yaklaşım özellikle aynı animasyon modelinin yeni animasyon değerlerinin hesaplanacağı durumlarda PC ile FPGA arasında veri alışverişini azaltmak için çok önemlidir. Bu yaklaşımda önce orjinal veriler ve ilk dönüşüm matrisi hafızaya yazılır. Ardından her seferinde modül hafızasına sadece yeni dönüşüm matrisi yüklenerek modülün hafızada var olan bir önceki nokta değerlerinden bir sonraki nokta değerlerini hesaplaması sağlanır. Yeni değerler hafızadan alınır ve dönüşüm matrisi tazelenerek işlem yeniden başlatılır. Bu yaklaşım PC ile FPGA arasında veri alışverişini %50'ye yakın bir oranda azaltır.

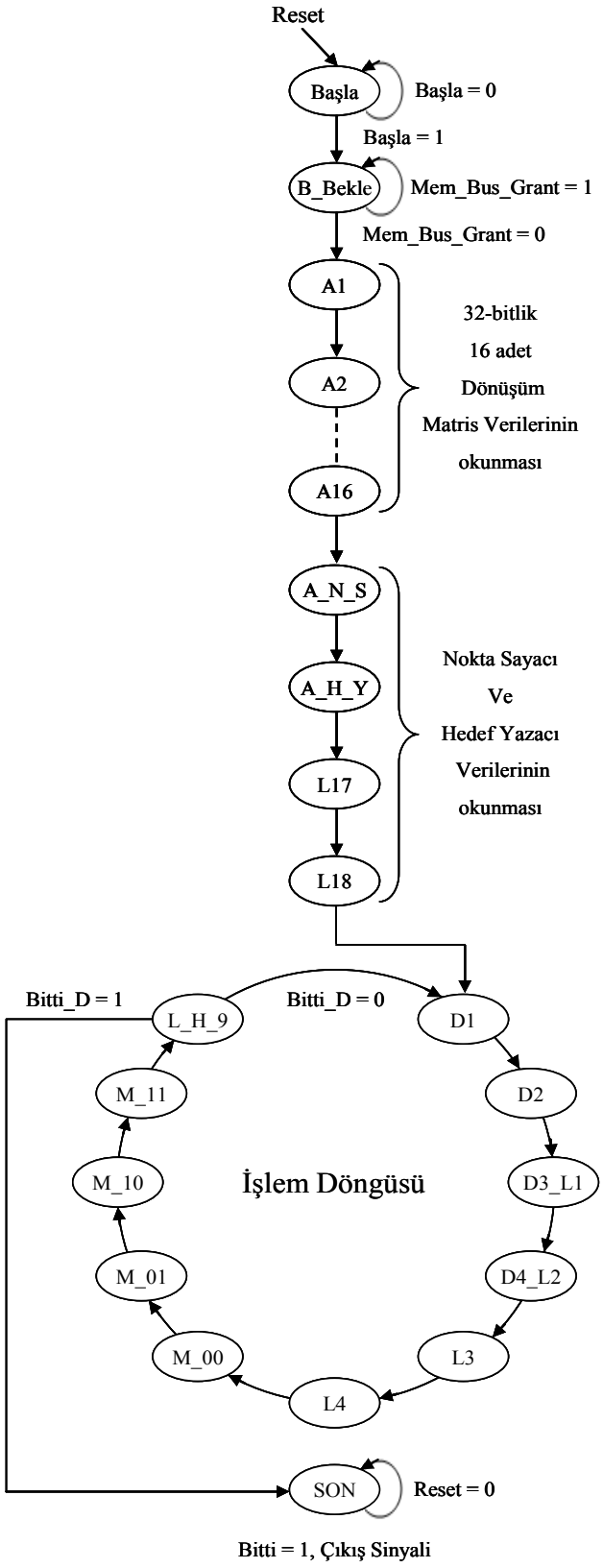


Şekil 3.5. Bütün modüller için ortak hafıza haritası.

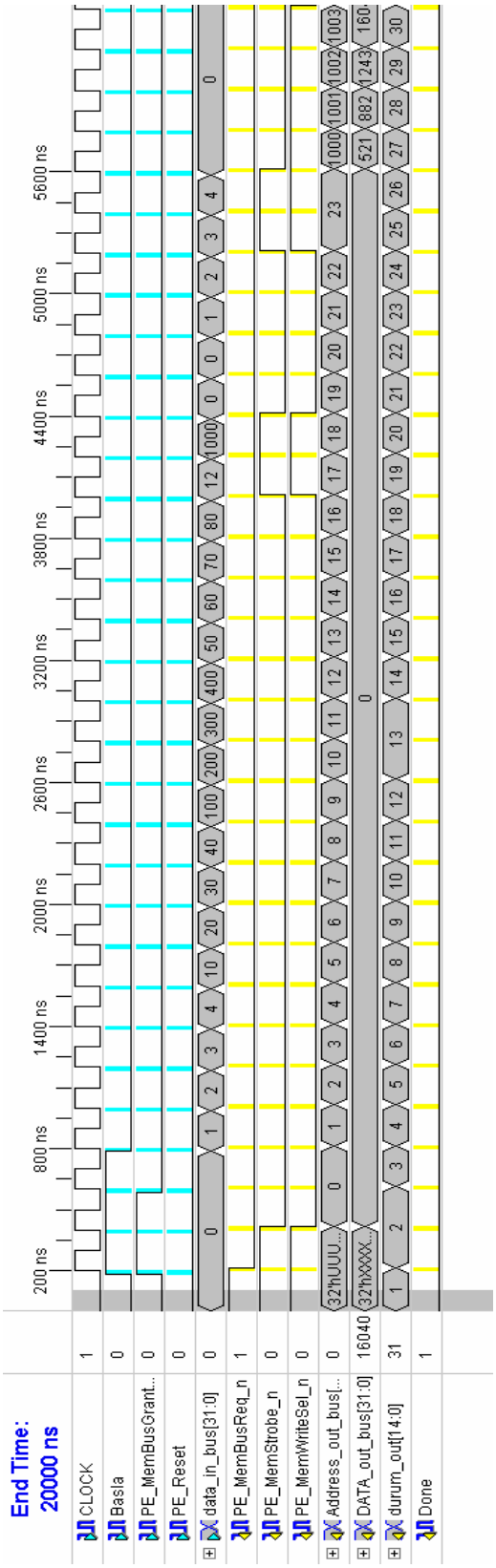
3.2.2. Tamsayı Modül Kontrolörü ve Çalışması

Modülün kontrolörü toplam 30 *durum*'dan (state) oluşan bir Sonlu Durum Makinesi (Finite State Machine-FSM) olarak tasarlanmıştır. Şekil 3.6'deki *Durum* diyagramında görüldüğü gibi modül ilk çalışmaya başladığında *Reset durum*'unda

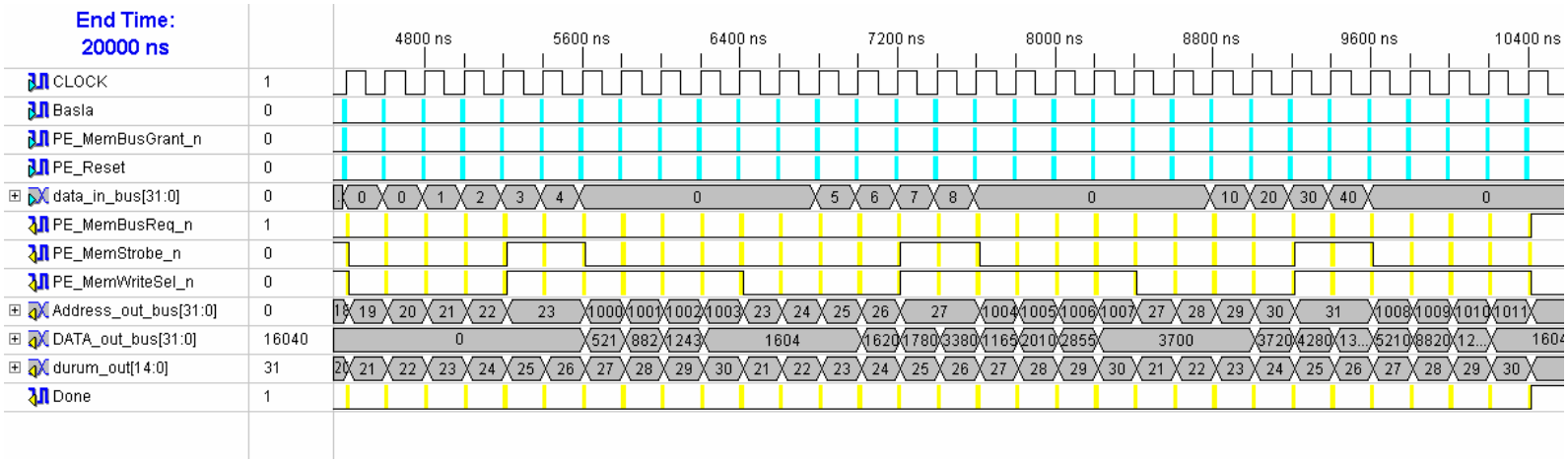
bekler. Bağlı olduğu bilgisayardan `Basla` sinyalini aldığında (Basla = 1 olduğunda), `Reset` *durum*'undan çıkarak, hafızaya erişmek için sistem yoluna istek sinyali gönderir ve yolun kendine tahsis edilmesini bekler. Yol tahsis edildikten sonra, kontrolör hafızada `0…0h` adresine giderek ilk önce dönüşüm matrisinin değerlerini hafızadan `Çekirdek` içinde bulunan *SabitYazac*'larda uygun yerlere yükler. Bu işlemler, kontrolörde 18 *durum* boyunca yapılır. Okunan bu değerler bütün noktaların dönüşümü tamamlanıncaya kadar bu yazaçlar içinde tutulur. Modül, dönüşüm matrisini okuduktan sonra hafızadan kaç tane noktanın dönüşüme uğrayacağı bilgisini ve hesaplanan yeni değerlerin hafızada kaydedileceği adres bilgisini `0…10h` ve `0…11h` adreslerinden sırasıyla `NS` ve `HS` veri erişim sayaçlarına kaydedilir. Bu işlemler iki *durum* boyunca yapılır. Bu aşamadan sonra, modül on *durum*'luk bir döngüye girer ve döngünün her tekrarlanışında bir noktanın değerlerini (*x, y, z, w*) hafızadan *DegiskenYazacı*'na okur ve *SabitYazac'*ı ve *DegiskenYazacı*'ndaki değerleri uygun zamanlama ile çarpıp toplayarak yeni değerleri hesaplar. Ardından yine döngü içerisinde hesaplanan yeni değerleri, direkt olarak hafızada *HedefSayacı*'nın gösterdiği yere yazar. Döngünün her tekrarlanışından sonra *NoktaSayacı* bir azaltılır, *HedefSayacı* ve *KaynakSayacı* dört arttırılır. *NoktaSayacı* sıfıra ulaştığında kontrolöre `Bitti` sinyalini göndererek bu on *durum*'luk döngüden çıkılmasını sağlar. Döngüden çıkıldığında, işlemin bittiğini belirtmek için kontrolör bağlı olduğu bilgisayara bir kesme sinyali gönderir ve bekleme konumuna geçer. Modül bekleme konumunda tekrar `Reset` sinyali gelinceye kadar kalır. Şekil 3.7 ve 3.8'de Tamsayı modül simülasyonunun dalga formları verilmiştir.



Şekil 3.6. Tamsayı modül kontrolürünün *Durum* diyagramı.



Şekil 3.7. Tamsayı modül simülasyonu 1.



Şekil 3.8. Tamsayı modül simülasyonu 2.

3.3. Kesirli Sayı (Floating-Point) Modülleri

Tek hafıza üniteli ve çift hafıza üniteli olmak üzere iki değişik kesirli sayı modülü tasarlanmıştır. Tek hafıza üniteli modül, giriş ve çıkış matris verilerini hafızadan alarak işledikten sonra uygun zamanlama ile sonuçları yine aynı Veri İşlem Ünitesi üzerinden aynı hafıza ünitesine göndermektedir. Çift hafıza üniteli kesirli sayı modülü ise işlenecek verileri birinci hafızadan alarak gerekli hesaplamaları yaptıktan sonra sonuçları ikinci hafızaya göndermektedir.

Her iki kesirli sayı modülünün en üst seviye, ikinci seviye ve veri işleme ünitesi tasarımları Şekil 3.1, 3.2 ve 3.3'te görüldüğü gibidir. Her bir modülde Veri işleme ünitesinde yer alan çekirdek farklı olarak tasarlamıştır.

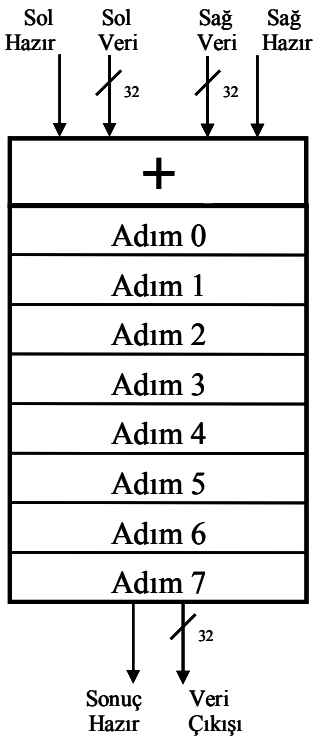
Kesirli sayı modüllerinde, kesirli sayıların toplanması ve çarpılması için daha önceden tasarlanmış kesirli sayı çarpma ve toplama üniteleri kullanılmıştır [5, 8, 9]. Aşağıda öncelikle çarpma ve toplama ünitelerinin yapısından ve çalışma prensiplerinden bahsedilmiştir. Ardından modül çekirdeklerinin detayları verilmiştir.

3.3.1. Kesirli Sayı Toplama ve Çarpma Üniteleri

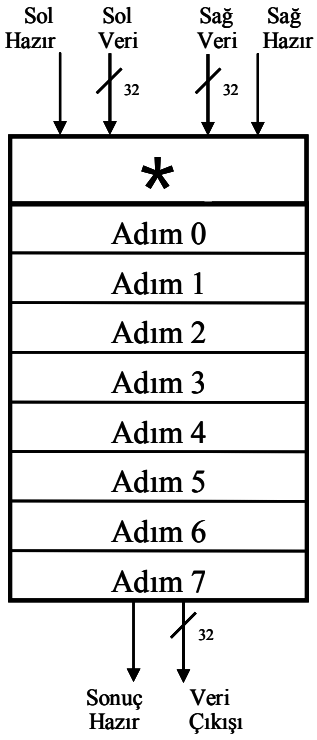
Şekil 3.9 ve 3.10'daki blok diyagramlarda toplama ve çarpma ünitelerinin prensip blok şeması verilmiştir. Üniteler kanallı (pipelined) olarak tasarlanmışlardır ve işlemleri 8 saat darbesi boyunca gerçekleştirmektedirler. Kanallı olmayan *Toplama* ve *Çarpma Üniteleri*'nde yeni bir işlemin başlayabilmesi için o anda yürütülen işlemin bitirilmesi gerekmektedir. Çalışmamızda kullanılan ünitelerde 8

aşamalı kanallı olarak tasarlanmıştır. Ünite ilk işlemin sonucunu 8 periyot sonra vermektedir. Bununla beraber üniteye ilk veri çifti verildikten sonra işleme tabi tutulacak diğer veriler için ünitenin sonucu üretmesini beklemeye gerek yoktur. Her saat darbesinde ünitelere bir veri çifti işlenmek üzere girilebilir. Üniteler aynı anda 8 veri çiftini değişik aşamalarda içinde barındırırlar. Bu durumda, bir veriyi işleme süresi 8 periyot, 10 veriyi işleme süresi 18 periyot ve 100 veriyi işleme süresi ise 108 periyottur. Yukarıdaki verilerden de anlaşılacağı üzere işlem sayısı arttıkça işlem başına harcanan periyot adedi azalmaktadır. Örneğin 1000 adet sayı çiftini çarpmak ya da toplamak için toplam 1008 periyot, sayı çifti başına 1008/1000 periyot harcanır.

Üniteler girişlerinde geçerli sayının olup olmadığını anlamak için `SagHazir` ve `SolHazir` sinyallerini izlerler. Bu iki sinyal birden '1' olduğunda sağ ve sol veri girişinden gelen değerler işlenmek üzere ünite içine alınırlar. Ünite çıkışında herhangi bir sonuç hazır olduğunda çıkışa sonuçla birlikte `SonucHazir` sinyali de gönderilir. Bir üniteden çıkan `SonucHazir` sinyali başka bir ünitenin `SagHazir` veya `SolHazir` girişine bağlanarak bir ünitenin kendinden sonra gelen başka bir üniteyi tetiklemesi sağlanır. Bu sayede üniteler birbirine zincirleme bir şekilde bağlanarak hesap ağaçları oluşturulabilmektedir. Oluşturulan hesap ağaçlarında, bu sinyaller ağaç içerisinde veri akışının senkronizasyonu için kullanılır. Böylelikle ağaç ne kadar büyük olursa olsun ekstra kontrol sinyaline ihtiyaç duyulmadan inşa edilebilir. Burada dikkat edilmesi gereken bir husus ağacın dengeli bir şekilde oluşturulmasıdır [5].



Şekil 3.9. *Toplama Ünitesi* blok diyagramı.

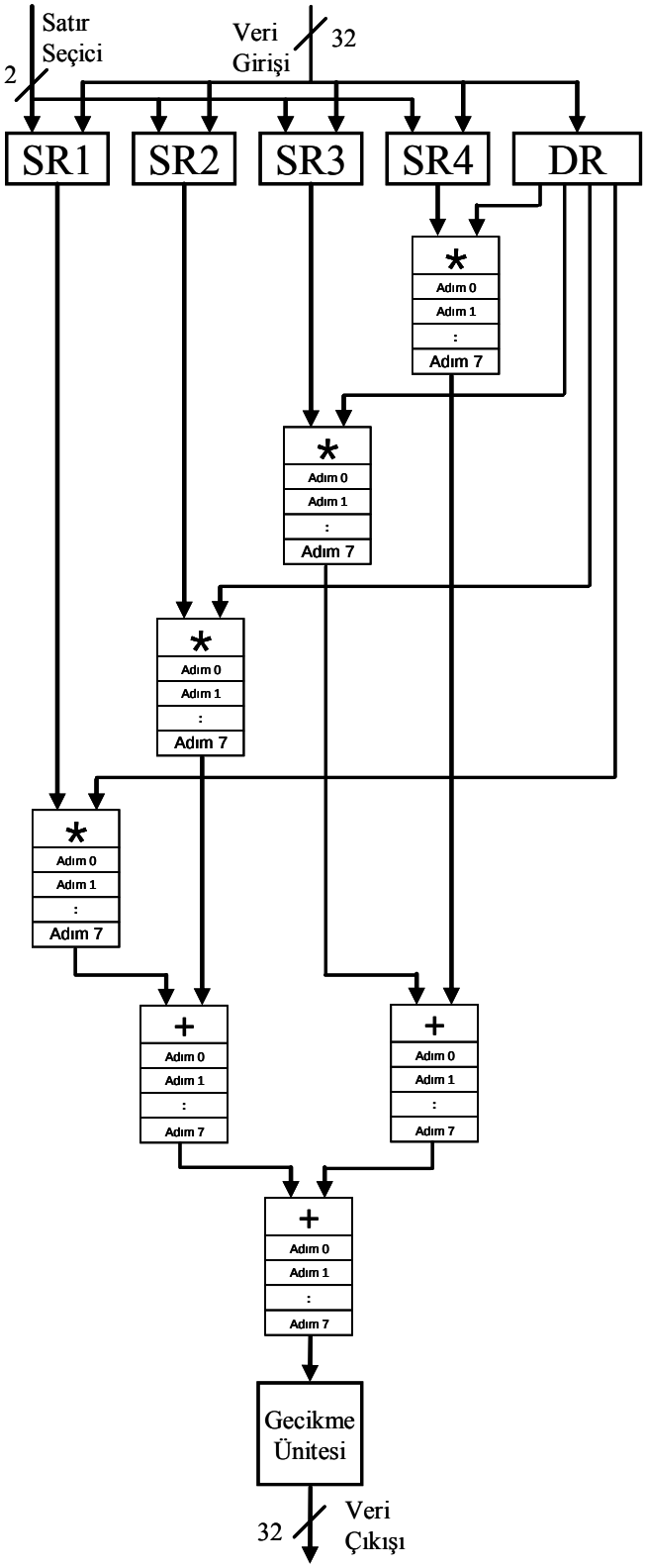


Şekil 3.10. *Çarpma Ünitesi* blok diyagramı.

3.3.2. Tek Hafıza Üniteli Kesirli Sayı (Floating-Point) Modülü

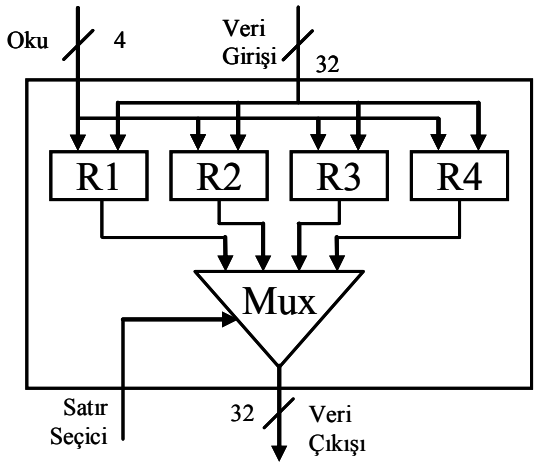
Tek Hafıza Üniteli Kesirli Sayı (Floating-Point) modülün en üst seviye, ikinci seviye ve veri işleme ünitesi blok diyagramları tamsayı modülü ile aynı özelliklere sahiptir (Şekil 3.1, 3.2 ve 3.3). Şekil 3.11'de kesirli sayı için tasarlanmış çekirdeğin blok diyagramı görülmektedir.

Çekirdek, dört adet 4x32-bitlik *SabitYazac* (*SR1...SR4*), bir adet 4x32-bitlik *DonusumYazac'ı* (*DR*), dört adet 32-bitlik kesirli sayı çarpıcı ve üç adet 32-bitlik kesirli sayı toplayıcıdan oluşmaktadır. 4x32-bitlik *SabitYazac*'lardan her birinin içinde 4 adet 32-bitlik yazaç bulunur ve bu yazaçlardan herbiri dönüşüm matrisinin bir sütununu tutmak için kullanılırlar. Dönüşüme uğrayacak noktanın değerleri (*x, y, z, w*) ise *DonusumYazacı*'nda tutulur.



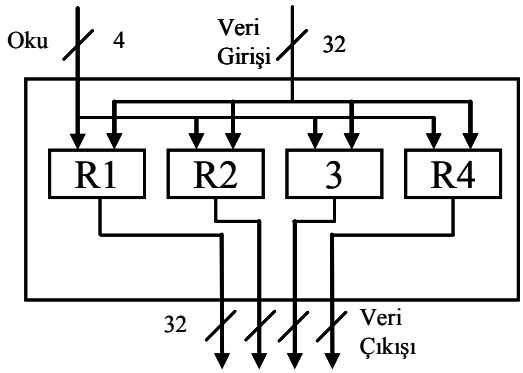
Şekil 3.11. Tek Hafıza Üniteli Kesirli Sayı modülü çekirdek blok diyagramı.

Şekil 3.12'de blok diyagramı verilen *SabitYazac*'ın çalışmasını her bir yazaç için ayrı ayrı tasarlanmış 4-bitlik `Oku` sinyali ve `SatirSecici` sağlamaktadır. Gelen verinin hangi yazaca kaydedileceğini `Oku` sinyali belirler. `SatirSecici` kullanılarak da hangi yazacın içindeki değerin çıkışa aktarılacağı belirlenir. Bu değerler *Çarpma Ünitesi*'nde işlenerek sekiz adım sonra çıkar. Verinin hazır olduğunu gösteren veri hazır sinyalleri de veri ile birlikte çıkarak *Toplama Ünitesi*'nin girişinde bulunan `SagHazir` ve `SolHazir` girişlerine girerek işlemi başlatırlar.



Şekil 3.12. *SabitYazac* blok diyagramı.

Şekil 3.13'te *DegiskenYazacı*'n blok diyagramı görülmektedir. Veri yolundan gelen veriler 4-bitlik `Oku` sinyali sayesinde sıra ile *R1...R4* yazaçlarına aktarılırlar. *DegiskenYazacı* 4x32 = 128-bitlik veri çıkışına sahiptir. Bu sayede yazaçlarda tutulan 4 değer aynı anda çıkışa aktarılabilmektedir.

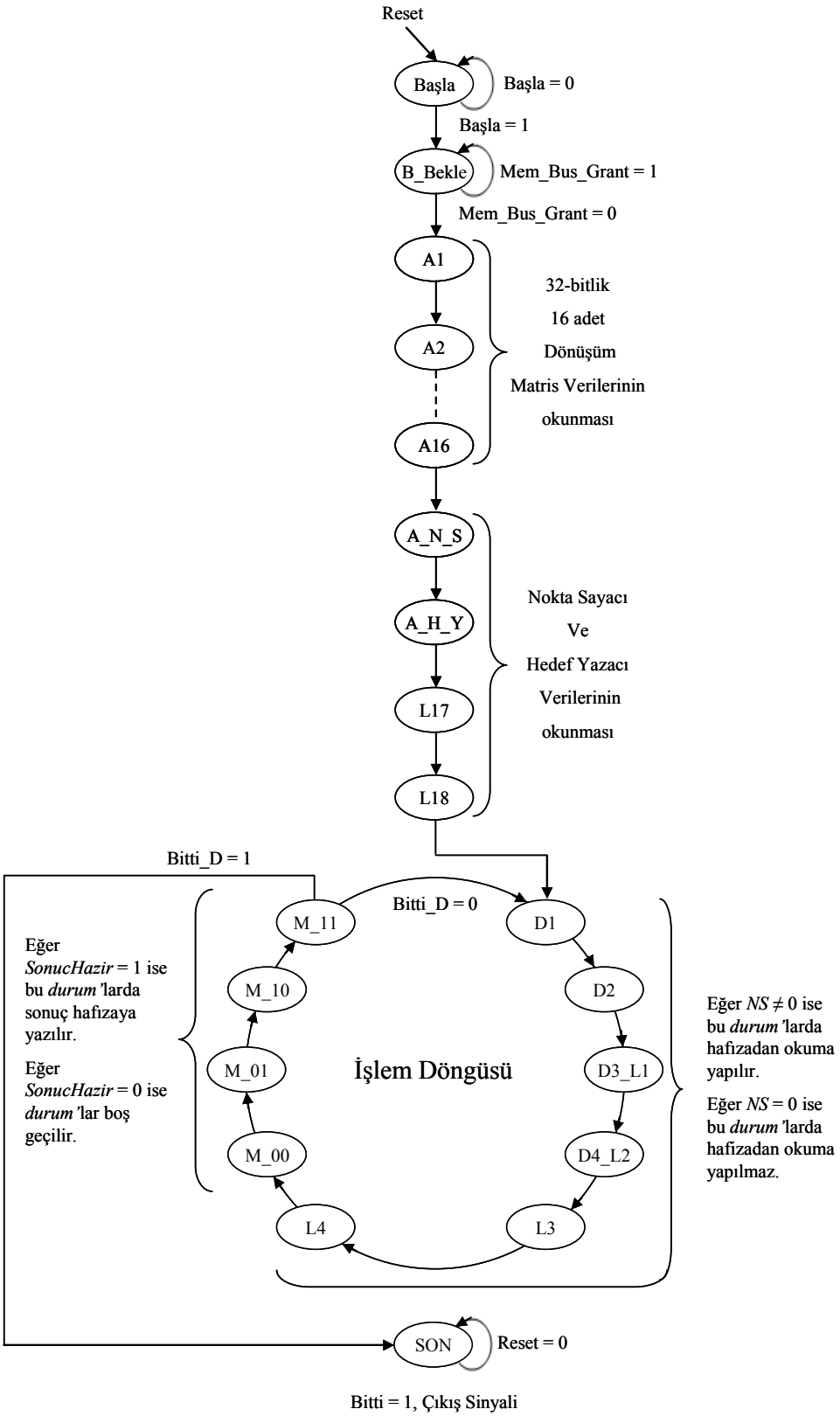


Şekil 3.13. *DegiskenYazacı* blok diyagramı.

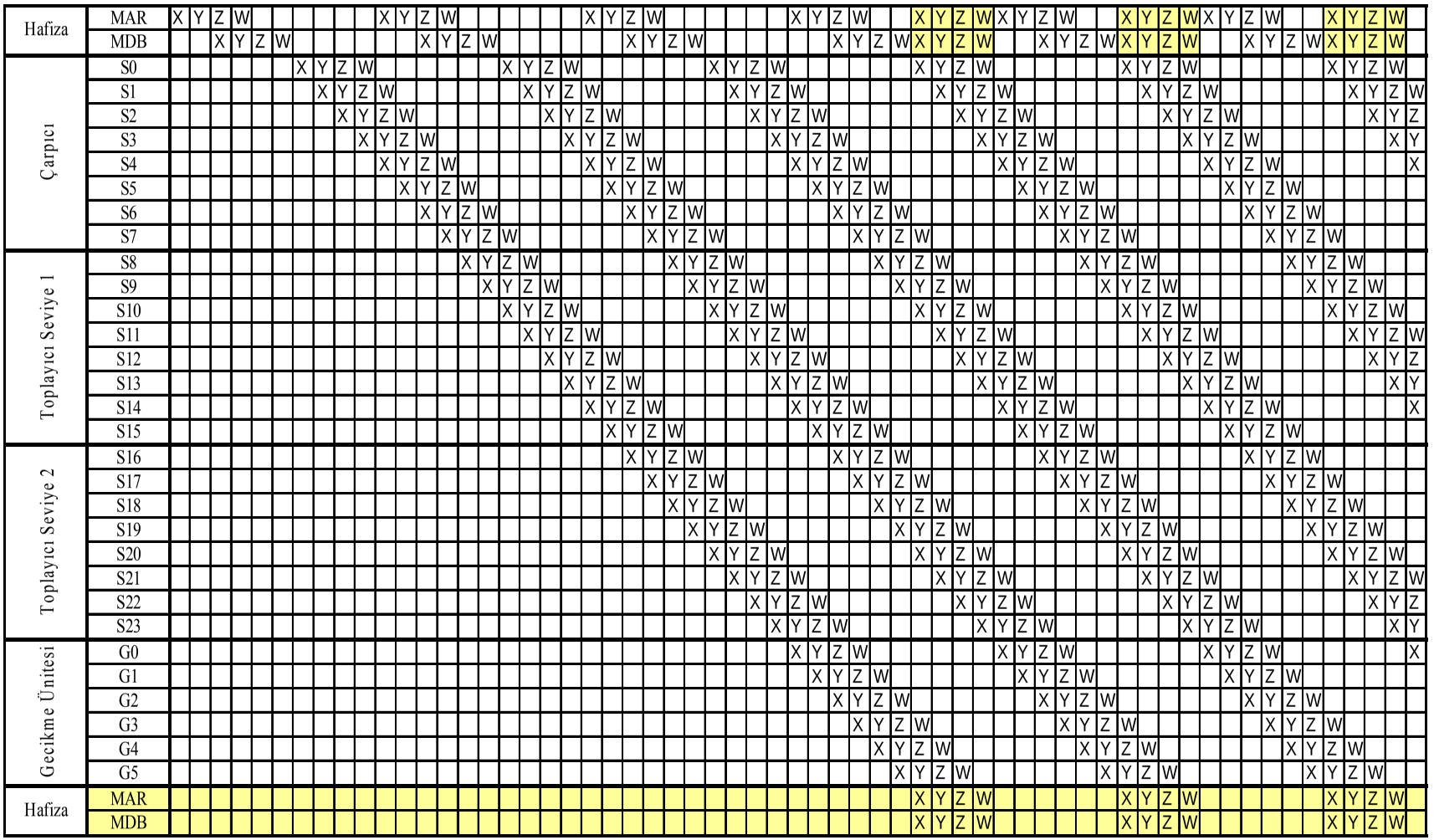
3.3.2.1. Tek Hafıza Üniteli Kesirli Sayı Modül Kontrolörü ve Çalışması

Şekil 3.14'te modül kontrolürünün çalışmasını gösteren *durum* diyagramı verilmiştir. Kontrolör Tamsayı modül kontrolörüne benzer şekilde toplam 28 *durum'dan* oluşan bir sonlu durum makinesi olarak tasarlanmıştır. Modül ilk çalışmaya başladığında `Reset` *durum*'unda bekler. Bilgisayardan `Basla` sinyalini aldığında, `Reset` *durum*'undan çıkarak, hafızaya erişmek için sistem yoluna istek sinyali gönderir ve yolun kendine tahsis edilmesini bekler. Yol tahsis edildikten sonra, kontrolör hafızada `0…0h` adresine giderek ilk önce dönüşüm matrisinin değerlerini hafızadan *SabitYazac*'ta uygun yerlere okur. Bu işlem, kontrolörde toplam 18 *durum* boyunca yapılır. Okunan bu değerler bütün noktaların dönüşümü tamamlanıncaya kadar bu yazaçlar içinde tutulur. Modül, dönüşüm matrisini okuduktan sonra, hafızadan kaç tane noktanın okunacağı bilgisini ve hesaplanan yeni değerlerin hafızada kaydedileceği adres bilgisini okur ve bunları, *NS* ve *HS* veri erişim sayaçlarına kaydeder. Bu işlemler toplam iki *durum* boyunca yapılır.

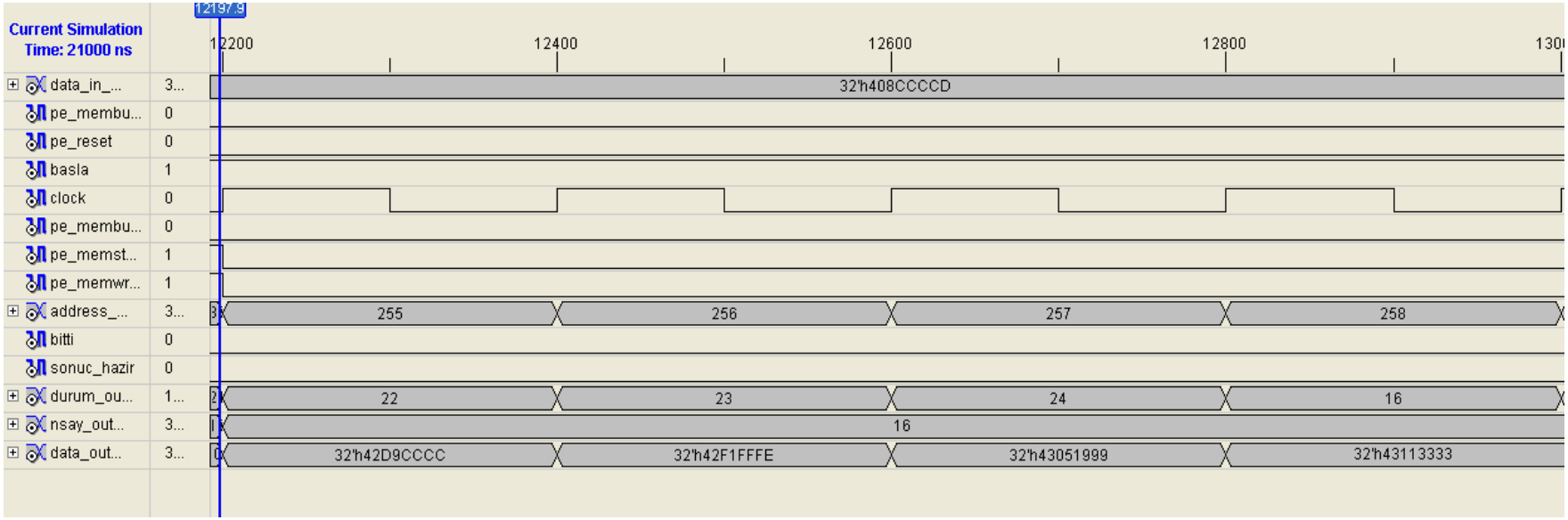
Bu aşamadan sonra, modül bir döngüye girer ve döngünün her tekrarlanışında bir noktanın değerlerini hafızadan *DegiskenYazac*ı'na okur, *SabitYazac* ve *DegiskenYazacı*'ndaki uygun elemanları uygun zamanda çarpıp toplayarak yeni değerleri hesaplar. Hesaplanan yeni değerler, hafızada *HedefSayacı*'nın gösterdiği yere yazılır. Döngünün her tekrarlanışından sonra *NoktaSayacı* bir azaltılır. *NoktaSayacı* sıfıra ulaştığında ise modül döngüden çıkar. Kontrolör döngünün herbir tekrarlanışını on `durum` boyunca gerçekleştirir. Döngüden çıkıldığında, kontrolör bilgisayara işlemin bittiğini belirtmek için bir kesme sinyali gönderir ve bekleme konumuna geçer. Modül bekleme konumunda tekrar `Reset` sinyali gelinceye kadar kalır. Şekil 3.15 ve 3.16'da tek hafıza üniteli kesirli sayı modülün zaman diyagramı ve simülasyon sonuçları verilmiştir. Bu kontrölerin tamsayı modül kontrolöründen farkı işlem döngüsünden kaynaklanmaktadır. Tamsayı modülünde işlem sonuçları 1 saat darbesi süresinde hesaplanabilmekte ve hemen hafızaya yazılmakta idi. Kesirli Sayı modülünde ise toplama ve çarpma üniteleri 8 periyot kanallı olduğundan hafızadan okunan ilk değerin sonucu toplam 24 saat darbesi sonra çıkışta görülmektedir. Veri okuma ve yazma işlemini senkronize etmek için kullanılan 6 saat darbesi gecikme ünitesinde dikkate alındığında, ilk sonuç toplam 30 periyot sonra çıkışta görülür. Kontrolör bu 30 periyot boyunca çıkışa sonuç yazmasa da her 10 periyotda bir veri setini hafızadan okur ve kanallı işlenmek üzere gönderir. Kanal içinde aynı anda 3 noktanın değerleri sıra ile işlem görür. Bu durum Şekil 3.15'te görülmektedir. Benzer bir durum son noktanın değerleri hafızadan okunduktan sonra gerçekleşir. Son nokta değerlerinden sonra işlem döngüsü veri okumayı bırakır ama kanal içindeki en son verinin sonucu hafızaya yazılana kadar (toplam 3 defa) tekrarlanmaya devam eder.



Şekil 3.14. Tek Hafıza Üniteli Kesirli Sayı modül algoritması



Şekil 3.15. Tek Hafiza Üniteli Kesirli Sayı modülün çalışmasını gösteren zaman diyagramı.

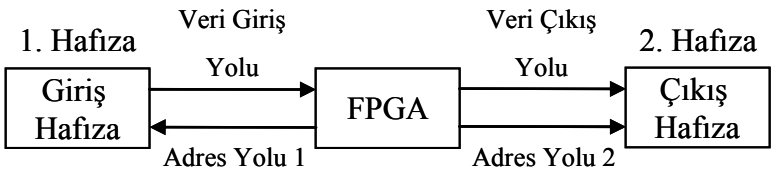


Şekil 3.16. Tek Hafıza Üniteli Kesirli Sayı modülün simülasyon sonuçları.

3.3.3. Çift Hafıza Üniteli Kesirli Sayı (Floating-Point) Modülü

Şekil 3.15'te görüldüğü gibi Tek Hafıza Üniteli Kesirli sayı modülünde azami bus kullanımı (*bus utilization*) %80'e ulaşmaktadır. Hafızadan okuma işlemi sırasında meydana gelen 2 saat darbelik gecikme (*latency*) sonucu adres ve veriyolu %20 oranında boş durmaktadır. Ayrıca veri okuma ve veri yazma işlemleri aynı veri yolu üzerinden yapıldığından, veriyolu okuma ve yazma işlemleri için paylaşımlı olarak kullanılmaktadır. Hem okuma işleminden kaynaklanan gecikmeyi önlemek, hem de modülü bir vektör işlemcisi olarak kullanabilmek için çift hafızalı sürümü tasarlanmıştır.

Tasarlanan modülün birinci seviye blok diyagramı Şekil 3.17'de görülmektedir. Burada birinci hafıza girişi hafızası, ikinci hafıza ise çıkış hafızası olarak kullanılmaktadır. Modül hiç kesintiye uğramadan aynı anda giriş hafızasından verileri okumakta, okunan verileri işlemekte ve işlem sonuçlarını çıkış hafızasına yazmaktadır.

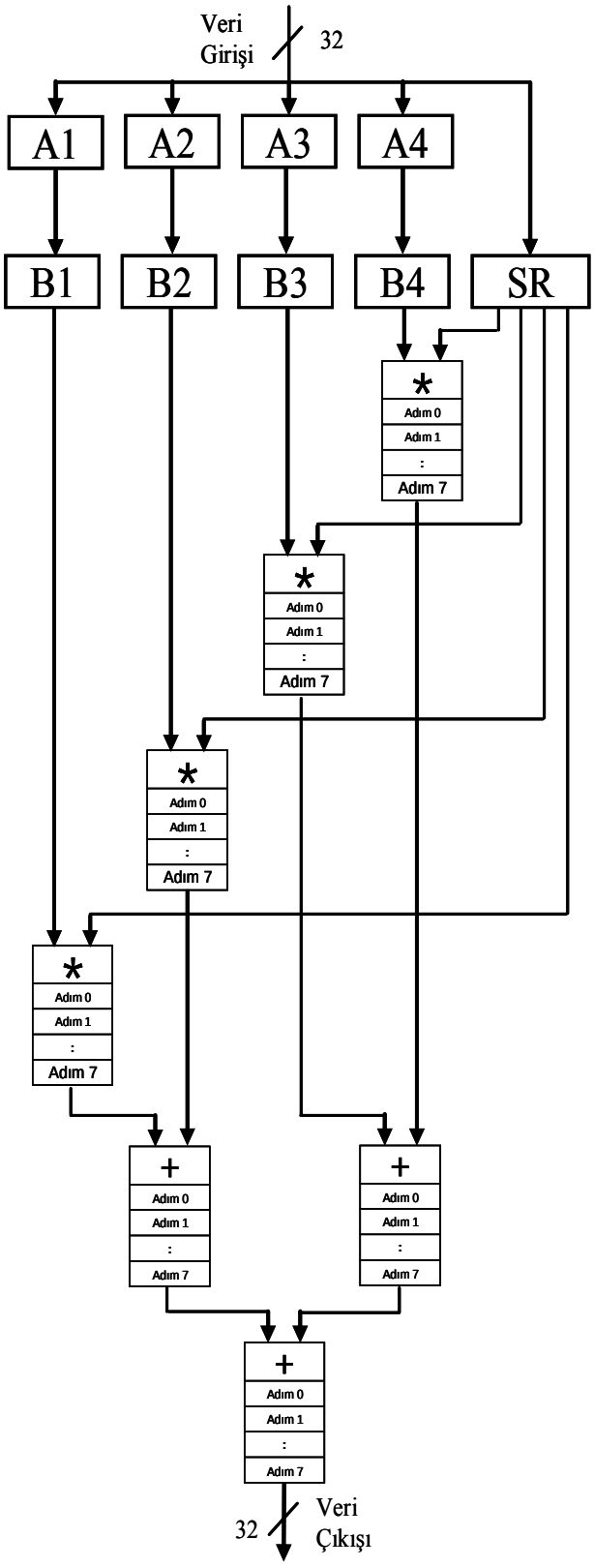


Şekil 3.17. Çift Hafıza Üniteli Kesirli Sayı modül blok şeması

Çift Hafıza Üniteli modülün ikinci seviye blok diyagramı Tek Hafıza Üniteli Kesirli Sayı (Floating-Point) Modülü ile aynı özelliklere sahip olduğundan burada tekrar gösterilmemiştir. Şekil 3.18'de Çift Hafıza üniteli kesirli sayı modülünün *Çekirdek* blok diyagramı görülmektedir. Diğer modüllerde olduğu gibi bu modülde

de gelen ilk veriler *SabitYazac*'a kaydedilir, daha sonra *NS* ve *HS* değerleri sıra ile okunur. Diğer modüllerden farklı olarak işlenecek nokta değerlerini saklamak için A ve B serisi olmak üzere 2 seride toplam 8 yazaç kullanılmıştır. Bunun sebebi, okuma işlemini kesintiye uğratmamaktır. Okunan nokta değerlerinin 4 saat darbesi boyunca değişmeden saklanması gerekmektedir.

Diğer yandan, giriş hafızasından gelen verilerinde kaybolmaması gerekmektedir. Burada A serisi tampon olarak kullanılmaktadır. Gelen nokta değerleri önce A serisine yazılır. Bu işlem 4 periyot boyunca yapılır. Bu sırada B serisindeki veriler işlenir. A serisi dolduğunda B serisindeki verinin işlemi bitmiş olur ve A, B'ye aktarılırken A1'e yeni noktanın ilk değeri okunur. Bu modülde birbirinden bağımsız çift hafıza kullanıldığından ve giriş ve çıkışın senkronize edilmesine gerek duyulmadığından, Çekirdek çıkışında herhangi bir gecikme ünitesi kullanılmamıştır.

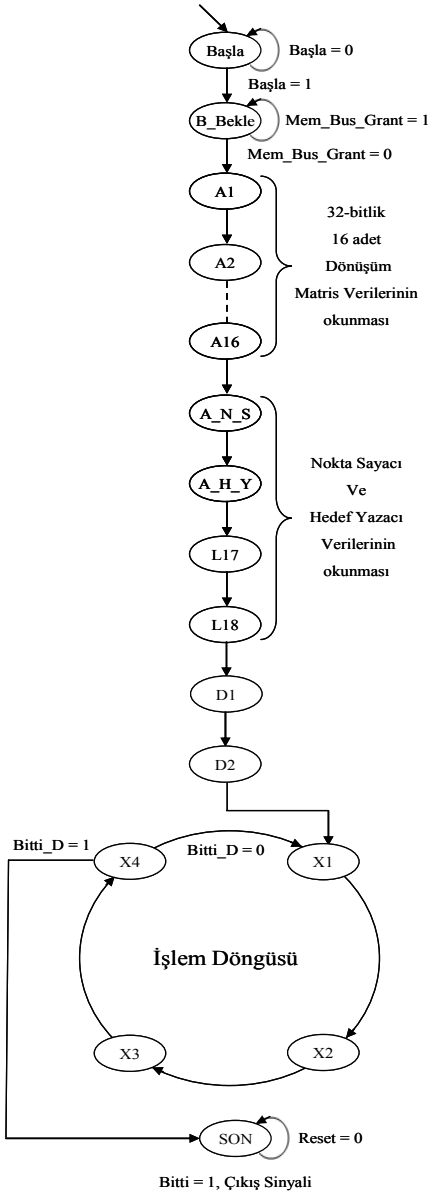


Şekil 3.18. Çift Hafıza Üniteli Kesirli Sayı modül *Çekirdek* blok diyagramı.

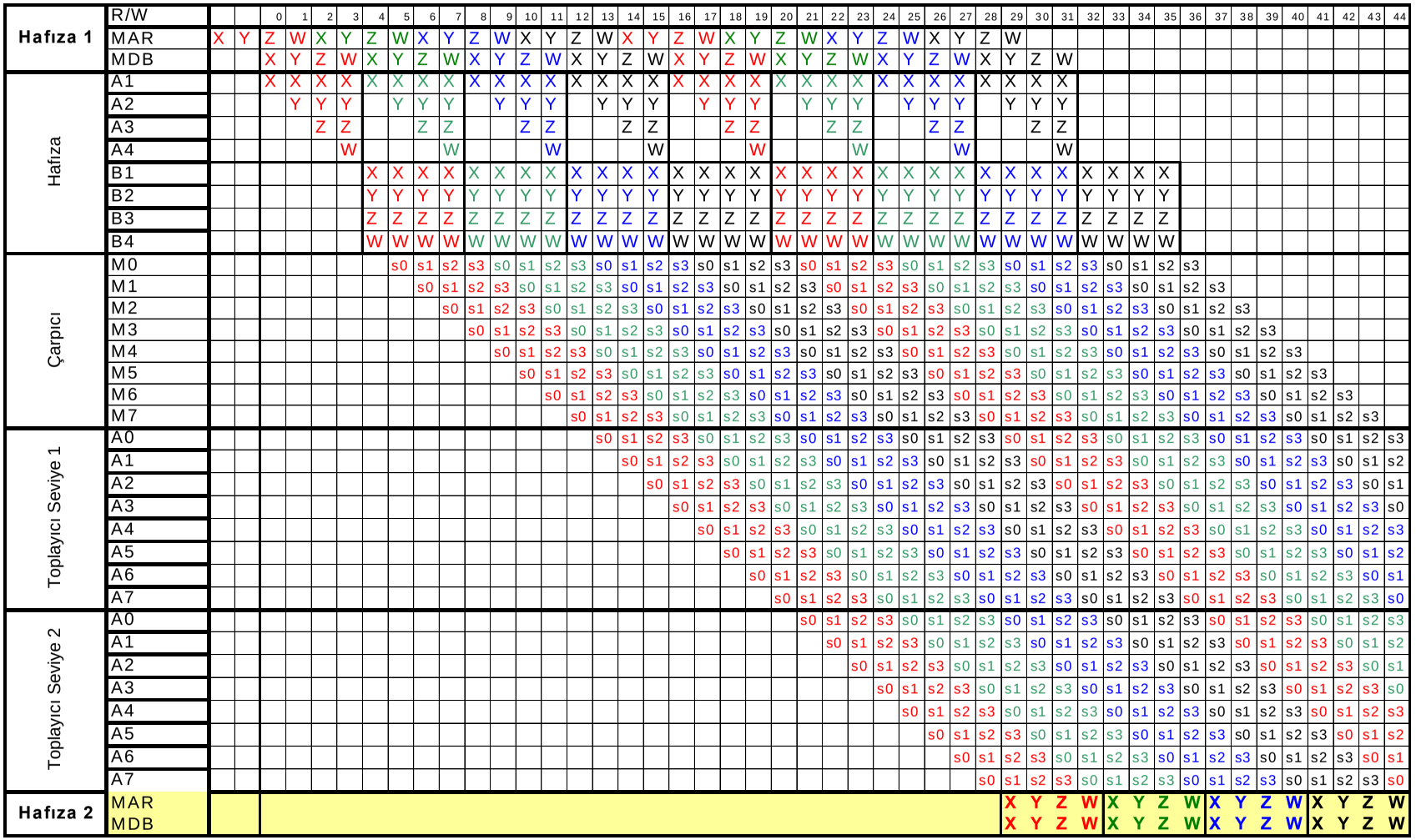
3.3.3.1. Çift Hafıza Üniteli Kesirli Sayı Modül Kontrolörü ve Modülün Çalışması

Şekil 3.19'da görüldüğü gibi modül kontrolörü toplam 26 *durum*'dan oluşan bir sonlu durum makinesi olarak tasarlanmıştır. Kontrolör ilk çalışmaya başladığında diğer modüller gibi *Reset durum*'unda bekler. `Basla` sinyalini aldığında ise `DegiskenYazacı` yükleninceye kadar yine aynı prosedürü takip eder. Bu aşamadan sonra, modül 4 *durum*'dan oluşan bir döngüye girer. Diğer modüllerden farklı olarak bu modülde giriş ve çıkış veri yolları ayrı ayrı kullanılacağı düşünülerek (birleştirilmeyeceği) düşünülerek tasarım yapılmıştır. Bu sayede modül bir yandan `Veri Giriş Yolu`'ndan bilgi okurken aynı anda veri çıkış yoluna veri yazılmaktadır. Bu durum modülün daha hızlı çalışması noktasından önemli bir hız kazancı sağlamaktadır. 4 *Durum'*dan oluşan döngünün her tekrarlanışında kontrolör bir noktanın değerlerini hafızadan *DegiskenYazacı*'na okur, *SabitYazac* ve *DegiskenYazacı*'ndaki uygun elemanları uygun zamanda çarpıp toplayarak yeni değerleri hesaplar ve hesaplanan yeni değerler, hafızada *HedefSayacı*'nın gösterdiği yere yazılır. Tek hafızalı modülde olduğu gibi bu modülde de çekirdek içindeki toplama üniteleri kanallıdır. Dolayısıyla döngü boyunca kanal dolana kadar yani çıkışta sonuç oluşuncaya kadar çıkışa değer yazılmaz, sadece okuma yapılır. Aynı şekilde bütün nokta değerlerinin okunması bittikten sonra kanal boşalana kadar okuma işlemi yapılmaz, sadece yazma yapılır. Burada birbirinden bağımsız çift hafıza olduğundan okuma ve yazma işlemlerinin senkronizasyonuna dolayısıyla gecikme ünitesine gerek yoktur. Tasarlanan diğer modüllerde olduğu gibi döngünün

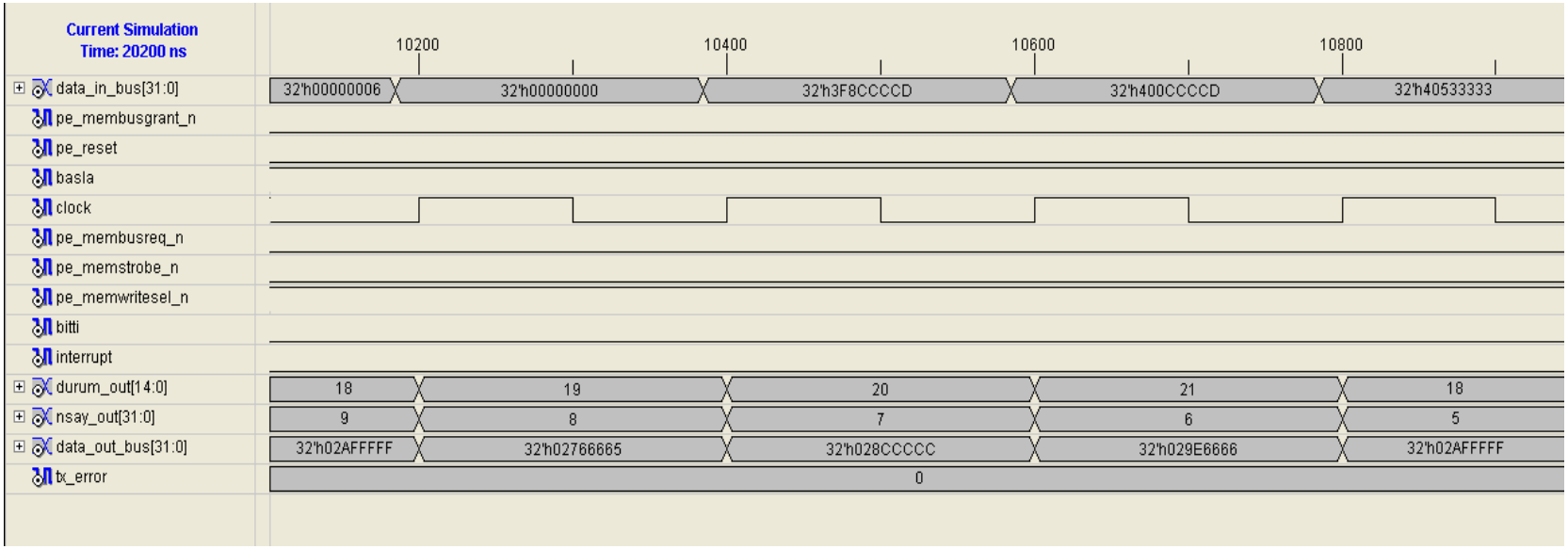
her tekrarlanışından sonra *NoktaSayacı* bir azaltılır. *NoktaSayacı* sıfıra ulaştığında ise modül döngüden çıkar. Döngüden çıkıldığında, kontrolör bilgisayara işlemin bittiğini belirtmek için bir `Bitti` sinyali gönderir ve bekleme konumuna geçer. Modül bekleme konumunda tekrar `Reset` sinyali verilinceye kadar kalmaktadır. Şekil 3.20'de 3.21 ve 3.22'de çift hafızalı modülün zaman diyagramı ve simülasyon sonuçları verilmiştir.



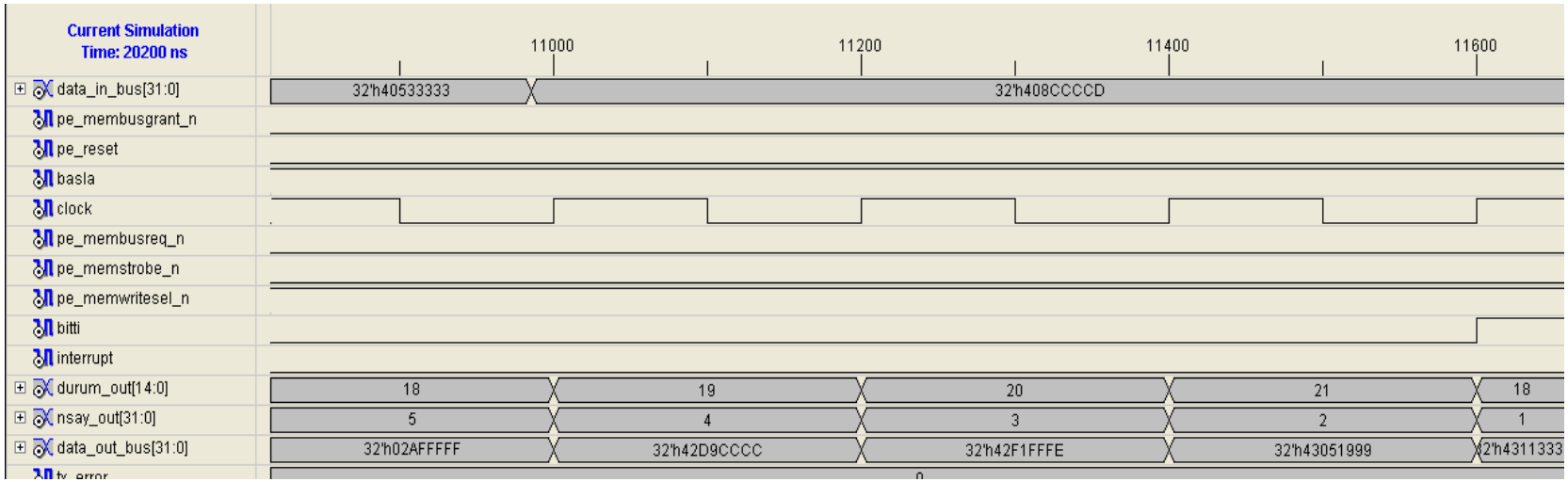
Şekil 3.19. Çift Hafızalı Üniteli Kesirli Sayı modül algoritması.



Şekil 3.20. Çift Hafıza Üniteli Kesirli Sayı modülün çalışmasını gösteren zaman diyagramı.



Şekil 3.21 Çift Hafiza Üniteli Kesirli Sayı modülün simülasyon sonuçları1.



Şekil 3.22. Çift Hafıza Üniteli Kesirli Sayı modülün simülasyon sonuçları 2.

4. DENEY DÜZENEKLERİ VE TEST SONUÇLARI

4.1. Tam Sayı (Integer) Modülü

Bu çalışmada tasarlanan üç boyutlu tamsayı dönüşüm modülü, değişik FPGA çipleri için sentezlenerek FPGA çip istatistikleri ve modülün azami saat frekansları incelenmiştir. Modülün belirlenen veriyi işleme süresi, ISE simülasyon programı [14] kullanılarak bulunmuş ve bu süre farklı PC'lerin aynı veriyi işleme süreleri ile karşılaştırılmıştır.

Modül *Xilinx* firmasının *Virtex 2*, *Virtex 2P*, *Virtex 4* ve *Virtex 5* çipleri [14] için sentezlenmiştir. Tablo 4.1'de sentezleme sonuçları görülmektedir. Kullanılan *slice register, LUTs* sayılarına bakıldığında, seçilen FPGA çiplerine teorik olarak modülün sırasıyla 8, 14, 20 ve 50 adet kopyasının aynı anda sığması mümkündür. Fakat kullanılan `IOB` (giriş/çıkış pinleri) yönünden düşünüldüğünde, ilk üç çipe iki adet, *Virtex 5* çipine ise 4 adet modülün rahat bir şekilde yerleştirilebileceği görülmektedir.

Bu çalışmada her bir çipte tek bir modülün performansı incelenmiştir. Özel bir tasarımla, bir FPGA çipine birden fazla modülün aynı anda yerleştirilmesi ve bunların paralel çalıştırılması ile, modülden daha yüksek performans elde edilebileceği görülmektedir.

Tablo 4.1. Tam Sayı modülün FPGA çip istatistikleri.

FPGA Çip Türü	Slice Reg. Sayısı / %	LUTs Sayısı / %	Slice FFs Sayısı / %	Bounded IOBs Sayısı / %
Virtex 2	748 / 12	671 / 10	673 / 03	119 / 45
Virtex 2P	749 / 07	670 / 06	781 / 07	119 / 30
Virtex 4	673 / 03	1094 / 05	673 / 03	119 / 37
Virtex 5	616 / 02	402 / 01	737 / 16	119 / 24

Modülün çalışma performansını karşılaştırmak için yapılandırmaları Tablo 4.2'de görülen üç adet bilgisayar seçilmiştir. Seçilen bu bilgisayarlardan PC-1 ve PC-3 tek çekirdekli olup PC-2 çift çekirdekli AMD işlemciye sahiptir. Karşılaştırma için seçilen bu bilgisayarlar diğer modüllerin karşılaştırılması için de kullanılmıştır.

Tablo 4.2. Deneylerde kullanılan PC'lerin özellikleri.

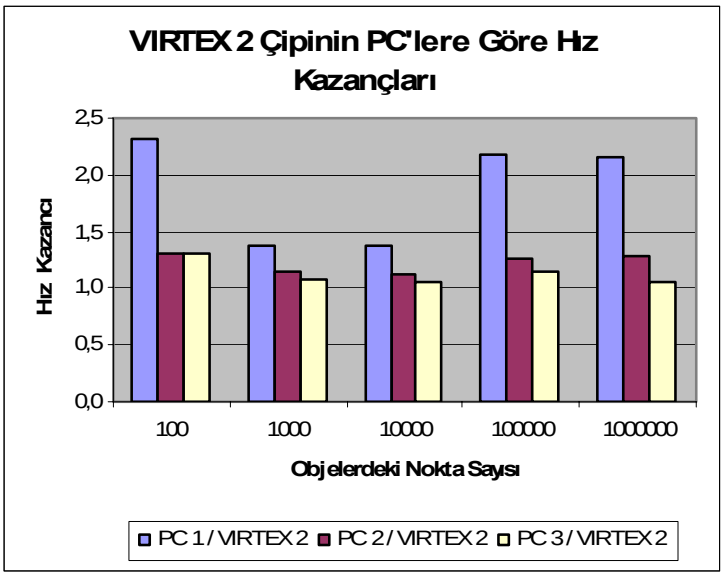
PC Adı	CPU Hızı (GHz)	CPU Cache Bellek (KB)	RAM Hafıza Boyutu (MB)	Hafıza Tipi	CPU Türü	FSB BUS Hızı (MHz)	BUS Boyutu (Bit)
PC-1	2.60	128	256	DDR 1	Intel Cel.	400	32
PC-2	2.00	2048	512	DDR 2	AMD Athl.	667	64
PC-3	1.73	2048	512	DDR 2	Intel Pent.	533	32

Deneylerde kullanılmak üzere 100, 1000, 10000, 100000 ve 1000000 noktası (`vertex`) olan nesne verileri Ek A'da verilen program kullanılarak oluşturulmuştur. Bu veriler, tamasayı formatında olup her bir nokta için x, y, z ve w olmak üzere dört adet rasgele tamsayı belirlenmiştir. Tablo 4.3'te PC'lerin deney verilerini işleme süreleri mikrosaniye (μs) türünden verilmiştir. Bu süreler verilerin sabit diskten okunması için gerekli süreyi içermemekte, sadece nokta dönüşüm süresini (matris çarpım işlemlerini) içermektedir.

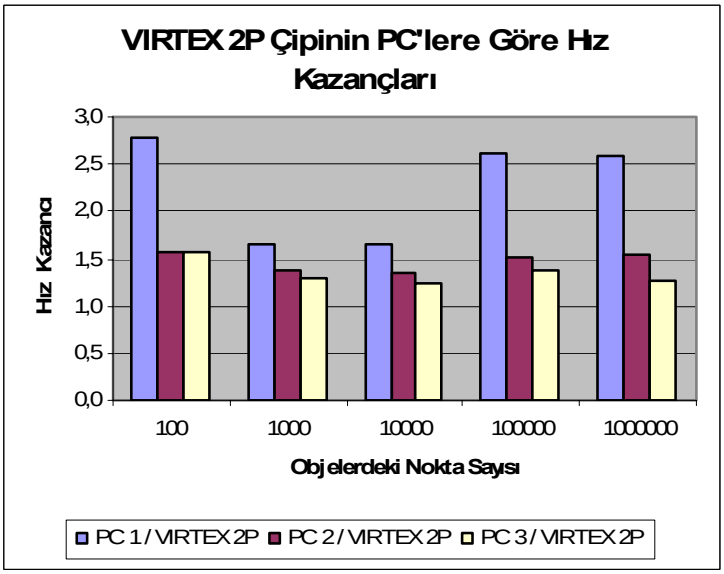
Şekil 4.1, 4.2 ve 4.3'de *Virtex 2, Virtex 2P* ve *Virtex 4*'e göre sentezlenmiş tamsayı modülün çalışma sürelerinin, seçilen PC'lerle karşılaştırılması görülmektedir. Her bir grafikte bir FPGA çipine göre sentezlenmiş modül yapılandırmasının, 100, 1000, 10000, 100000, 1000000 noktayı işleme süresi için PC'lere göre hız kazancı görülmektedir. *Virtex 2*'de modül, PC-1' e göre yaklaşık 1.4 ile 2.3 kat arasında, PC-2 ve PC-3 göre ise yaklaşık 1.1 ile 1.3 kat arasında daha hızlıdır. *Virtex 2P*'de PC-1'e göre yaklaşık 1.7 ile 2.8 kat arasında, PC-2 ve PC-3 göre 1.3 ile 1.6 kat arasında hız kazancı sağlanmıştır. *Virtex 4* üzerinde ise, modül, PC-1'e göre 2 ile 3.2 kat arasında daha hızlı, PC-2 ve PC-3'e göre ise 1.5 ile 1.8 kat arasında daha hızlıdır. Seçilen FPGA çipleri içinde en hızlısı olan *Virtex 5* üzerinde ise modül, PC-1'e göre 2 ile 5 kat, PC-2 ve PC-3'e göre ise 2 kat daha hızlıdır.

Tablo 4.3. PC'lerin tamsayı tabanlı nesne nokta verilerini işleme süreleri.

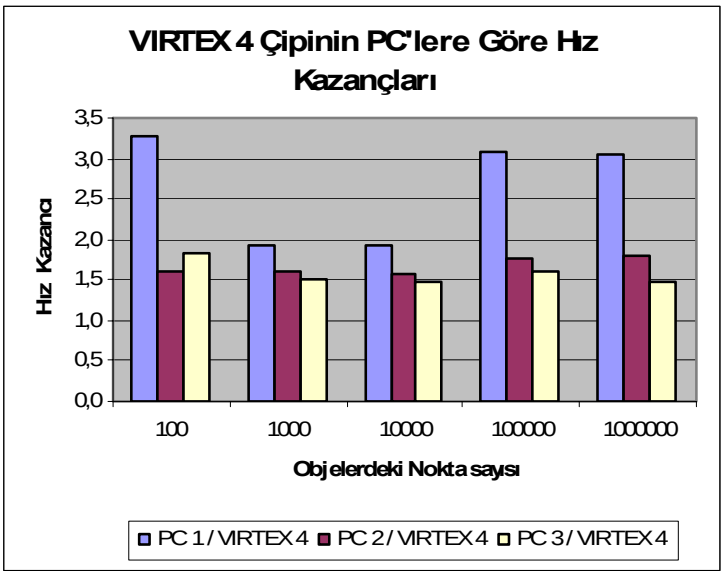
Nokta Sayısı	PC-1 (µs)	PC-2 (µs)	PC-3 (µs)
100	11.85	6.95	6.65
1000	69.18	57.18	53.72
10000	689.61	563.19	525.02
100000	10990.08	6338.23	5721.85
1000000	108713.01	64439.13	53080.19



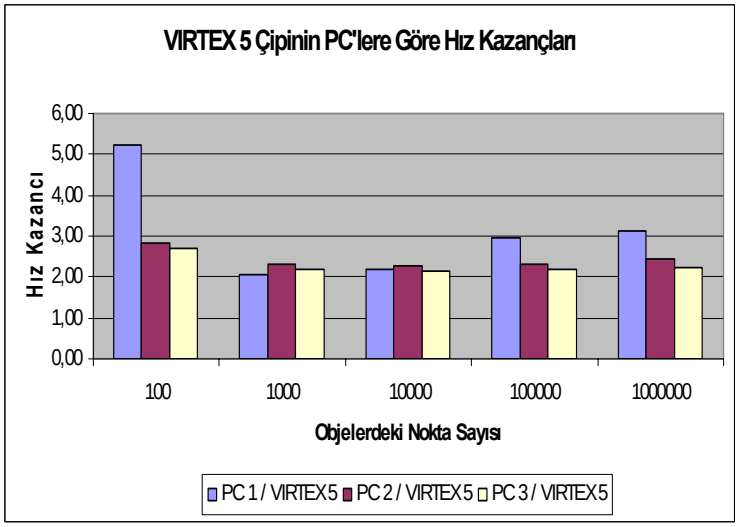
Şekil 4.1. *Virtex 2* FPGA çipine göre sentezlenmiş Tamsayı modülün PC'lerle karşılaştırılması.



Şekil 4.2. *Virtex 2P* FPGA çipine göre sentezlenmiş Tamsayı modülün PC'lerle karşılaştırılması.



Şekil 4.3. *Virtex 4* FPGA çipine göre sentezlenmiş Tamsayı modülün PC'lerle karşılaştırılması.



Şekil 4.4. *Virtex 5* FPGA çipine göre sentezlenmiş Tamsayı modülün PC'lerle karşılaştırılması.

Yukarıda verilen hız kazançları tek bir çip üzerinde, tek bir modül yapılandırılması çalıştırıldığında elde edilen sonuçlardır. Birden fazla çip üzerinde birden fazla modül yapılandırması paralel olarak çalıştırıldığında bu hız kazançlarının katlanarak artacağı gayet açık olarak görülmektedir. İlk çalışmada

geliştirilen modül, tamsayı türünde verileri işlemek üzere tasarlanmıştır. PC'ler, tamsayılara göre kesirli sayılarda daha fazla zaman harcamaktadırlar. Bu nedenle kesirli sayı versiyonu tasarlandığında hız kazancının çok daha yüksek olacağı düşünüldüğünden bir sonraki aşamada kesirli sayı versiyonu tasarlanılmıştır.

4.2. Tek Hafıza Üniteli Kesirli Sayı (Floating-Point) Modülü

İkinci aşamada tasarlanan üç boyutlu dönüşüm modülü, *Virtex 4* ve *Virtex 5* FPGA çipleri için sentezlenerek FPGA istatistiklerine ve modülün en azami saat frekansları incelenmiştir. Modülün belirlenen veriyi işleme süresi, simülasyon ile bulunarak bu süre değişik PC'lerin aynı veriyi işleme süreleri ile karşılaştırılmıştır.

Modül, *Xilinx* firmasının *Virtex 4* ve *Virtex 5* çipleri için sentezlenmiştir. Tablo 4.4'te verilen sentezleme sonuçları göstermiştir ki, kullanılan *slice register, LUTs* sayılarına göre seçilen FPGA çiplerine teorik olarak modülün beş adet kopyasının aynı anda sığması mümkündür. Fakat modülde bulunan *IOB* (giriş/çıkış pinleri) yönünden düşünüldüğünde çipe iki adet modülün rahat bir şekilde yerleştirilebileceği görülmektedir. Bu aşamada tek bir modülün performansı incelenmiştir. Özel bir tasarımla bir FPGA çipine birden fazla modülün aynı anda yerleştirilmesi ve bunların paralel çalıştırılması ile modülden daha yüksek performans elde edilebileceği açık olarak görülmektedir.

Tablo 4.4. Tek Hafıza Üniteli Kesirli Sayı modülünün FPGA çip istatistikleri.

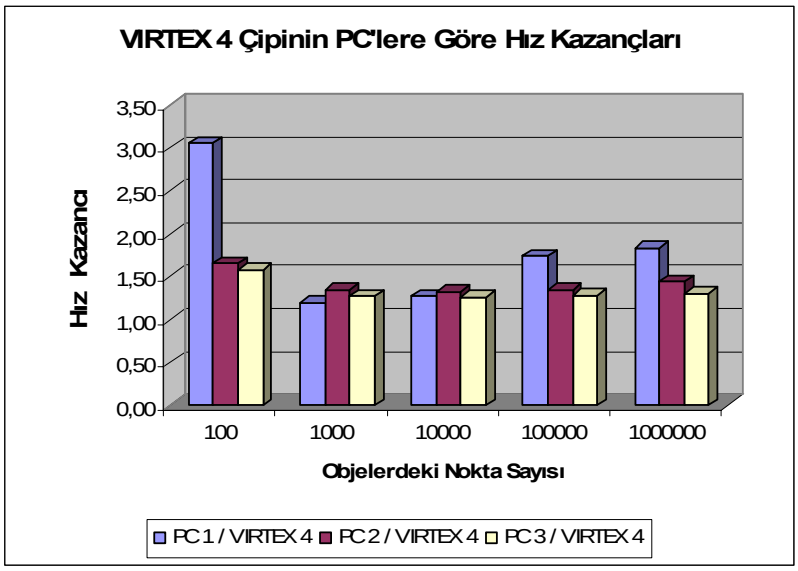
FPGA Çip Türü	Slice Reg. Sayısı / %	LUTs Sayısı / %	Slice FFs Sayısı / %	Bounded IOBs Sayısı / %
Virtex 4	4192 / 38	9085 / 83	4422 / 40	152 / 47
Virtex 5	4188 / 14	5005 / 17	6288 / 21	152 / 31

Modül veri işlemlerinin kontrolü için 100, 1000, 10000, 100000 ve 1000000 noktası (*vertex*) olan nesne verileri oluşturulmuştur. Bu veriler kesirli sayı formatında olup her bir nokta için *x, y, z* ve *w* olmak üzere dört adet rasgele kesirli sayı belirlenmiştir. Tablo 4.5'te PC'lerin deney verilerini işleme süreleri verilmiştir. Yine burada verilen süreler verilerin hard diskten okunması için gerekli süreyi değil, yalnızca nokta dönüşüm süresini (matris çarpım işlemlerini) içermektedir.

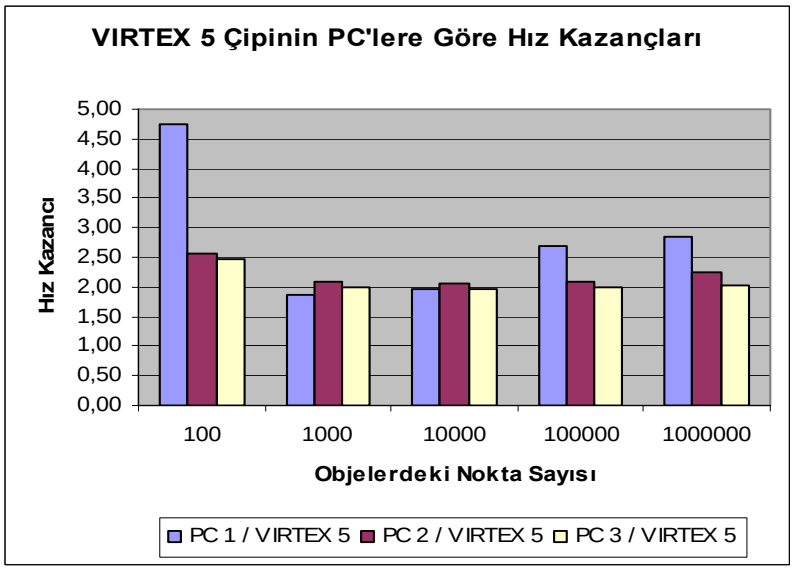
Şekil 4.5 ve 4.6'de *Virtex 4*'e ve *Virtex 5*'e göre sentezlenmiş modülün çalışma sürelerinin, seçilen PC'lerle karşılaştırılması verilmiştir. Sonuçlar yorumlanacak olursa *Virtex 4* çipinde modül, PC-1'e göre yaklaşık 1.2 ile 3 kat arasında, PC-2'ye göre 1.3 ile 1.5 ve PC-3'e göre ise yaklaşık 1.2 ile 1.5 kat arasında hız kazancı sağlandığı görülmektedir. *Virtex 5* çipinde ise, PC-1'e göre yaklaşık 1.8 ile 4.7 kat arasında, PC-2'ye göre 2.0 ile 2.5 ve PC-3 göre ise yaklaşık 1.9 ile 2.4 kat arasında hız kazancı sağlandığı görülmektedir.

Tablo 4.5. PC'lerin kesirli sayı türünde oluşturulan nesne nokta verilerini işleme süreleri.

Nokta Sayısı	**PC-1** (µs)	**PC-2** (µs)	**PC-3** (µs)
100	16.87	9.15	8.76
1000	64.99	72.87	68.96
10000	684.65	714.87	679.03
100000	9374.62	7271.61	6920.45
1000000	98939.94	77589.75	70163.72



Şekil 4.5. *Virtex 4* FPGA çipine göre sentezlenmiş Tek Hafıza Üniteli Kesirli Sayı modülün PC'lerle karşılaştırılması.



Şekil 4.6. *Virtex 5* FPGA çipine göre sentezlenmiş Tek Hafıza Üniteli Kesirli Sayı modülün PC'lerle karşılaştırılması.

Daha önce de belirtildiği gibi verilen hız kazançları tek bir çip üzerinde, tek bir modül yapılandırması çalıştırıldığında elde edilen sonuçlardır. Birden fazla çip

üzerinde birden fazla modül yapılandırması paralel olarak çalıştırıldığında hız kazançları katlanarak artacaktır.

4.3. Çift Hafıza Üniteli Kesirli Sayı (Floating-Point) Modülü

İkinci aşamada geliştirilen Tek Hafıza Üniteli Kesirli sayı modülü, hem okuma hem de yazma yapmak için tek hafızaya tek bir veri ve adres yolu üzerinden erişecek şekilde tasarlanmıştır. Veri ve adres yolu okuma ve yazma için paylaşımlı olarak kullanıldığından modül hızı sınırlı kalmaktadır. Bu duruma çözüm olarak Çift Hafıza Üniteli modül geliştirilmiştir. Bu modül sadece *Xilinx* firmasının *Virtex 5* çipi için sentezlenmiştir. Tablo 4.6'da görülen sentezleme sonuçları göstermiştir ki kullanılan *slice register*, *LUTs* ve *IOB* sayılarına göre seçilen FPGA çiplerine teorik olarak modülün dört adet kopyasının aynı anda sığması mümkündür. Fakat daha öncede belirtildiği gibi kullanılan *IOB* (giriş/çıkış pinleri) yönünden düşünüldüğünde çipe iki adet modülün rahat bir şekilde yerleştirilebileceği görülmektedir. Daha önce de bahsedildiği gibi burada verilen değerler sadece tek bir modül istatikleridir. Eğer kullanılan çipin özelliklerine göre daha fazla modül entegre edilecek olursa hız kazançları birkaç kat artacaktır.

Tablo 4.6. Çift Hafıza Üniteli Kesirli Sayı modülün FPGA çip istatistikleri.

FPGA Çip Türü	Slice Reg. Sayısı / %	LUTs Sayısı / %	Slice FFs Sayısı / %	Bounded IOBs Sayısı / %
Virtex 5	4281 / 14	4987 / 17	6288 / 21	184 / 38

Tasarımın kontrolün için standart olarak 100, 1000, 10000, 100000 ve 1000000 noktası olan nesne verileri oluşturulmuştur. Kullanılan veriler kesirli sayı

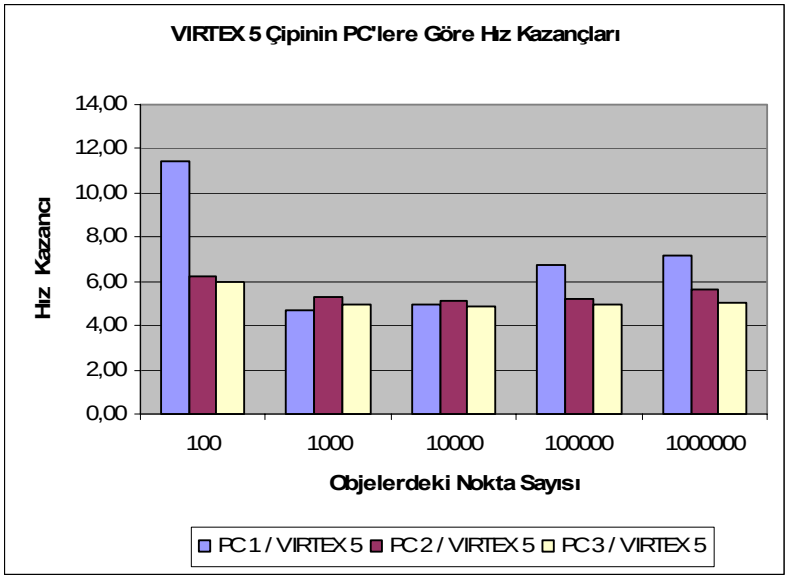
formatında olup her bir nokta için *x, y, z* ve *w* olmak üzere dört adet rasgele kesirli sayı belirlenmiştir. Tablo 4.7'de PC'lerin deney verilerini işleme süreleri verilmiştir. Bu süreler sadece nokta dönüşüm süresini (matris çarpım işlemlerini) içermektedir. Tablo 4.8'de *Virtex 5* çipinin deney verilerini işleme süreleri verilmiştir. Şekil 4.7'de *Virtex 5*'e göre sentezlenmiş modülün çalışma sürelerinin, seçilen PC'lerle karşılaştırılması görülmektedir. Daha öncede ifade edildiği gibi modüllerin veri işleme sürelerinin karşılaştırılması için 100, 1000, 10000, 100000, 1000000 nokta kullanılmıştır. *Virtex 5* çipinde modül, PC-1' e göre yaklaşık 11-12 kat arasında, PC-2'ye göre 4.0 ile 5.0 ve PC-3 göre ise yine 4.0-5.0 kat arasında hız kazancı sağlanmıştır. Çift hafıza üniteli kesirli sayı modülü, *Virtex 2*, *Virtex 2P* ve *Virtex 4* çiplerine sığmadığından sadece *Virtex 5* çip istatikleri verilmiştir.

Tablo 4.7. PC'lerin kesirli sayı türünde oluşturulan nesne nokta verilerini işleme süreleri.

Nokta Sayısı	**PC-1** (μs)	**PC-2** (μs)	**PC-3** (μs)
100	16.87	9.15	8.76
1000	64.99	72.87	68.96
10000	684.65	714.87	679.03
100000	9374.62	7271.61	6920.45
1000000	98939.94	77589.75	70163.72

Tablo 4.8. *Virtex 5* Çipinin nesne nokta verilerini işleme süreleri.

Nokta Sayısı	**Virtex 5** (μs)
100	1,470432
1000	13,872024
10000	138,803232
100000	1387,248108
1000000	13872,08323



Şekil 4.7. *Virtex 5* FPGA çipine göre sentezlenmiş Çift Hafıza Üniteli Kesirli Sayı modülün PC'lerle karşılaştırılması.

5. SONUÇ VE GELECEKTE YAPILABİLECEK ÇALIŞMALAR

Grafik uygulamaları ve animasyonları oluşturulurken yoğun olarak matris çarpımı işlemleri ve dönüşümleri yapılmaktadır. Bu işlemler ve dönüşümler binlerce noktadan oluşan karmaşık şekillere uygulandığında, grafik programları çok fazla CPU zamanı gerektirmektedir. Sahnedeki nesne sayısı ve bu nesneleri tanımlamada kullanılan nokta sayısı arttıkça, animasyonun gerçek zamanlı hesaplanması imkansız hale gelmektedir.

Bu problemin çözümü için gelişmiş grafik kartları, grafik işlemler için tasarlanmış özel amaçlı tasarlanmış bilgisayarlar veya paralel işlemcili bilgisayarlar gibi değişik yaklaşımlar geliştirilmiştir [2, 3].

Bu çalışma da yukarıda belirtilen yaklaşımlara bir alternatif olarak, üç boyutlu grafik dönüşümleri hızlandırmak amacıyla, FPGA çipleri üzerinde çalışabilecek donanım modüllleri tasarlanmıştır. Bu yaklaşımla 3B grafik dönüşümlerinde, paralel bilgisayarlara, süper bilgisayarlara hatta özel tasarlanmış grafik kartlarına göre daha düşük maliyete sahip, normal bilgisayarlarla karşılaştırıldığında daha yüksek performans gösteren ve çok kısa sürelerde defalarca tekrar programlanabilen daha esnek bir yapı ortaya konulmuştur.

Çalışma kapsamında bir tanesi tamsayılar üzerinde, ikiside kesirli sayılar üzerinde işlem yapabilecek şekilde olmak üzere toplam üç adet donanım modülü tasarlanmıştır. Tasarlanan modüller, gerçek veri üzerinde işlemler yapılarak test

edilmiş ve modüllerin ürettiği sonuçların doğrulanması yapılmıştır. Modüllerin veri işleme hızı değişik bilgisayarlarla karşılaştırılmıştır. Karşılaştırma sonuçlarına göre modüller kullanılarak tamsayı tabanlı üç boyutlu grafik dönüşümleri 2 ile 3 kat arasında, kesirli sayı tabanlı üç boyutlu grafik dönüşümleri ise 2 ile 12 kat arasında hızlandırmanın mümkün olacağı görülmüştür.

Hız kazançları tek bir FPGA üzerinde tek bir modül çalıştırılarak tespit edilmiştir. Bir FPGA çipine birden fazla modülün uygun şekilde yerleştirilmesiyle ya da birden fazla FPGA çipinde modülün birden fazla kopyasını çalıştırılmasıyla hız kazancının katlanarak artacağı düşünülmektedir.

Bu modüller FPGA çipleri için tasarlanmıştır. FPGA çipleririn şu anki saat hızları yaklaşık olarak 500MHz civarındadır. Modüller ASIC olarak gerçekleştirildiğinde çok daha yüksek saat hızlarına ulaşılacağı için PC'lere göre çok daha yüksek hız kazançlarına ulaşmak mümkün olacaktır.

5.1. Gelecekte Yapılabilecek Çalışmalar

Modüller tek bir FPGA çipi üzerinde tek modül olarak çalışacak biçimde tasarlanmıştır. Çip istatistiklerine bakıldığında bir çip içine iki modül kopyasının rahatça sığacağı görülmektedir. Hız kazancını arttırmak için modüllerin çiftli versiyonları tasarlanıp denenebilir. Ayrıca birden fazla FPGA çipinde birden fazla modül paralel olarak çalışmasını sağlamak için grafik programlarından gelen dönüşüm isteklerinin uygun şekilde sıralayıp çiplere dengeli bir şekilde dağıtan bir arayüz programı geliştirilebilir.

KAYNAKÇA

[1] Hearn, D. and Baker M.P. 2003. *"Computer Graphics with OpenGL 3E"*. Prentice Hall.

[2] Bensaali, F., Amira, A., Uzun, I.S. and Ahmedsaid, A. 2003. *"An FPGA Implementation of 3D Affine Transformations"*. IEEE International Conference on Electronics, Circuits and Systems: Proceedings. Sharjah, UAE.

[3] Anonim. 2007. http://www.sgi.com/.

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

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

[6] Şahin, I. ve Gloster, C.S. 2005. *"Evaluation of Ic Physical Design Optimization Algorithms For Acceleration Using Fpga-Based Custom Computing Machines"*. İleri Teknolojiler Sempozyumu, Konya.

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

[8] Gloster, C. and Sahin, I. 2001. *"Floating-Point Modules Targeted for Use with RC Compilation Tools",* Earth Science Technology Conference (ESTC), College Park, MD.

[9] Sahin, I., Gloster, C. and Doss, C. 2000. *"Feasibility of Floating-Point Arithmetic in Reconfigurable Computing Systems"*, Military and Aerospace Applications of Programmable Devices and Technology Conference, Washington, DC.

[10] Rincon, F. and Teres, L. 1998. *"Reconfigurable Hardware Systems"*. International Semiconductor Conference, vol. 1, pp. 45-54.

[11] Sridharan, K. and Priya, T.K. 2007. *"A Hardware Accelerator and FPGA Realization for Reduced Visibility Graph Construction Using Efficient Bit Representations"*. IEEE Transactions on Industrial Electronics, Vol. 54, No. 3.

[12] Bishop, B. and Kelliher, T.P. 2003. *"Specialized Hardware for Deformable Object Modeling"*. IEEE Transactions on circuits and Systems for Video Technology, Vol. 13, No. 11.

[13] Anapolis Micro Systems Inc. 1997. *"WildForce Reference Manual"*. Rev 3.4.

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

[15] Anonim 2007. http:// www.altera.com/products/devices/dev-index.jsp.

[16] Anonim 2007. http:// www.tr3d.com.

[17] Çetin, A. 2003. *"Bilgisayar Grafikleri"*. Seçkin Yayıncılık.

[18] Koyuncu, İ. and Şahin, İ. 2007. *"Generic Fpga Modules for Integer 2D and 3D Transformations"*. 12th. Conference for Computer Aided Engineering and System Modeling with Exhibition. Antalya, Turkey.

[19] Chodowiec, P., Gaj, K. 2003. *"Very compact FPGA Implementation of the AES"*, in: Proceedings of CHES 2003, Lecture Notes in Computer Science, vol. 2779, Springer, Berlin, pp. 319–333.

[20] McKenna, M. and M., Wilamowski. 2001. *"Implementing a Fuzzy System on a Field Programmable Gate Array"*. International Joint Colnference on Neural Networks, p.p. 189-194.

[21] Eadie, D., Shevlin, F. and Nisbet, A. 2002. *"Correction of Geometric Image Distortion Using FPGAs"*. Optical Mefrologv, Imaging, and Machine Vision Conference, Galway, IRL.

[22] Dillon, T. 2001. *"Two Virtex-II FPGAs Deliver Fastest, Cheapest, Best High-Performance Image Processing System"*. Xilinx Xcell Journal, 41.

[23] Hu, H., Jin, T., Zhang, X., Lu, Z. and Qian, Z. 2006. *"A Floating-point Coprocessor Configured by a FPGA in a Digital Platform Based on Fixed-point DSP for Power Electronics"*. IEEE IPEMC.

[24] Walke, R.L., Smith, R.W.M., Lightbody, G. 2000. *"20 GFLOPS QR Processor on a Xilinx Virtex-E FPGA"*. SPIE, San Diego, U.S.A.

[25] Borgatti, M., Lertora, F., Foret, B. and Cali, L. 2002. *"A Reconfigurable System Featuring Dynamically Extensible Embedded Microproccessor, FPGA ana Customizable I/O"*. IEEE Custom Integrated Circuits Conference 13-16.

[26] Cavuslu, M., Dikmese, A., Sahin, S., Kucuk, S. K. ve Kavak, A. 2006. *"Akıllı Anten Algoritmalarının IEEE 754 Kayan Sayı Formatı ile FPGA Tabanlı Gerçeklenmesi ve Performans Analizi"*. III.URSI-Türkiye'2006, pp.610-612.

[27] Beuchat, J.L. 2003. *"Modular Multiplication for FPGA Implementation of the IDEA Block Cipher"*, in: Proceedings of ASAP Application Specific Systems Architectures and Processors IEEE, Silverspring, MD.

[28] Rouvroy, G., Standaert, F., Quisquater, J. and Legat, J. 2004. *"Compact and Efficient Encryption/decryption Module for FPGA Implementation of the AES Rijndael Very Well Suited for Small Embedded Applications"*, in: Proceedings of ITCC 2004, Las Vegas, April 5–7,

[29] Visser, S.J., Dawood, A.S. and Williams, J.A. 2002. *"FPGA Based Real-time AdaptiveFfiltering for Space Applications"*, in: Proceedings of IEEE International Conference on Field-Programmable Technology, , pp. 322–326.

[30] Boullis, N. and Tisserand, A. 2005. *"Some Optimizations of Hardware Multiplication by Constant Matrices"*. *IEEE Tran.* On Computers, Vol. 54, No. 10 pp. 1271-1282.

[31] Cesur, E. 2006. *"2D CNN Gabor Fitle FPGA Implementation"* . Yüksek Lisans Tezi, Yıldız Teknik Üniversitesi, Fen Bilimleri Enstitüsü, Elektrik ve Elektronik Mühendisliği Anabilim Dalı, İstanbul.

[32] Uçar, A. 2007. *"Türkçe Fonemlerin Sınıflandırılmasında Kullanılan Sinir Ağının Fpga Uygulaması"*. Yüksek Lisans Tezi, Hacettepe Üniversitesi, Elektrik ve Elektronik Mühendisliği Anabilim Dalı, İstanbul.

[33] Çakıroğlu, M., Özcerit, A.T., Eskikurt, H.İ. ve Özdemir, Ç. 2003. *"80C51 Mikrodenetleyicilerinde Timer-Counter Yapılarının FPGA Mimarileri Kullanılarak Geliştirilmesi"*. 3.Uluslararası İleri Teknolojiler Sempozyumu, Gazi Üniversitesi, Ankara.

EKLER

Ek-A: Test Verisi Üretimi İçin Yazılan C++ Program Kodu

Bu program modüller için test verisi üretmek amacıyla için yazılmıştır. Program çalışırken komut satırında üç adet parametre girilmesini ister. Birinci parametre üretilecek verinin tipini belirler. Eğer '0' seçilirse 32-bitlik tamsayı, '1' seçilirse 32-bitlik kesirli sayı ile işlem yapılacağı belirtilmiş olur. İkinci parametre matris boyutunu belirtmektedir. Matris boyutu seçiminde '3' 3x3 matris boyutu, '4' seçimi yapılırsa 4x4 matris boyutu seçilmiş olur. Üçüncü parametre ise modülün işleyeceği nokta sayısını belirlemektedir. Son parametre de ise seçime bağlı olarak verilerinin kaydedileceği dosyanın adı verilir ve program çalıştırılır.

```
#include <iostream>
#include <fstream>
using namespace std;

void Usage()
{
	cout << "USAGE : ";
	cout << "C:\\..\\>DataGen <DataType>
				<MatrixSize>
				<VerCount>
				<FileName>\n\n";
	cout << " DataType : 1-> Integer (32-bit),
			     2-> Float   (32-bit)\n";
	cout << " MatrisSize    : 3->3x3,
				  4->4x4\n";
	cout << " VerticesCount : 1..2147438648\n";
	cout << " FileName      : FileName (Optional).
	If not given Program will make up a name.\n";
}


int CheckParams(int argc, char *argv[])
{
		int a,b,c;
		a=atoi(argv[1]);
		b=atoi(argv[2]);
		c=atoi(argv[3]);

	if (!((a==1)||(a==2)))
	{
	cout << "DataType must be either 1 or 2\n";
```

```
		return 0;
	}
	if (!((b==3)||(b==4)))
	{
	cout << "MatrixSize must be either 3 or 4\n";
		return 0;
	}
	if ((c<1)||(c>2147438648))
	{
	cout << "VerticesCount must be between 1 or 2147438648\n";
		return 0;
	}
	return 1;
}

void main (int argc, char *argv[])
{
		char FileName[50];
		ofstream Ofp;
		int i;
		int P1,P2,P3;
	//Argumanlari Kontrol Et
	if ((argc!=4) && (argc!=5))
	{
		Usage();
		exit(0);
	}
	else if (!CheckParams(argc, argv))
	       exit(1);
	//DosyaAdini Olustur
		P1=atoi(argv[1]);
		P2=atoi(argv[2]);
		P3=atoi(argv[3]);
	if (argc==5)
		strcpy(FileName,argv[4]);
	else
	{
	if (P1==1)
		strcpy(FileName,"Int_");
	else
		strcpy(FileName,"Flt_");
		strcat(FileName,argv[2]);
		strcat(FileName,"_");
		strcat(FileName,argv[3]);
		strcat(FileName,".dat");
	}
	Cout  << "FileName : "<< FileName<<endl;
	Ofp.open(FileName);
	       Ofp<<P1<< " "<<P2<< " "<< P3<< " "<< FileName<< endl;
	//3x3 matris carpimi
	if (P2==3)
	{
	       //3x3 matrisi yaz
	       for (i=0; i<9; i++)
	if (P1==1)
	       Ofp<< rand()/5000
	     << endl;
	else
	       Ofp<< (float)rand()/(float)5000
	       << endl;
	       //Nokta Sayisini ve Hedef Adresini yaz
	       Ofp<< argv[3]
	       << endl
	       <<11+P3*3<< endl;
```

```
	//Nolktalari Yaz;
		for (i=0; i<P3*3; i++)
			if (P1==1)
		Ofp<< rand()/66
		<< endl;
	else
		Ofp<< (float)rand()/(float)66
		<< endl;
	}
	else //4x4 matrix carpimi
	{
		//4x4 matrisi yaz
		for (i=0; i<16; i++)
			if (P1==1)
		Ofp<< rand()/5000
		<< endl;
	else
		Ofp<< (float)rand()/(float)5000
		<< endl;
		//Nokta Sayisini ve Hedef Adresini yaz
		Ofp << argv[3] << endl <<18+P3*4<< endl;
		//Nolktalari Yaz;
		for (i=0; i<P3*4; i++)
	if (P1==1)
		Ofp
		<< rand()/66
		<< endl;
	else
		Ofp<< (float)rand()/(float)66
		<< endl;
	}
	Ofp.close();
}
```

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

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

Adım 1 Başla.
Adım 2 CPU hızını ve veri dosyası adını gir.
Adım 3 Veriyi oku.
Adım 4 CPU saat değerini oku ve kaydet.
Adım 5 Verileri istenen şekilde işle.
Adım 6 CPU saat değerini yeniden oku ve kaydet.
Adım 7 CPUsaat verilerini kullanarak işlem süresini hesapla.
Adım 8 Sonucu ekrana yaz.

```
#include <fstream>
#include <iostream>
#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);
}

void main(int argc, char *argv[])
{
	const int Integer = 1;
```

```
	const int Float   = 2;
	unsigned     *T0,*T1;
	T0 = new unsigned[2];
	T1 = new unsigned[2];
	int CPU;
	int i,j;
	ifstream Fp;
	int DataType;
	int MatSize;
	int VCount;
	int DestAddr;
	char FName[20];
	int **TrMatris_i;
	int *Vertices_i;
	int *RVertices_i;
	float **TrMatris_d;
	float *Vertices_d;
	float *RVertices_d;

	if (argc != 3)
	{
		printf("Usage: mmul <FileName> <CPU Speed (Mhz)>\n");
		exit(1);
	}
	CPU = atoi(argv[2]);
	//Open the data file and read the first line (read the parameters)
	Fp.open(argv[1]);
	Fp >> DataType;
	Fp >> MatSize;
	Fp >> VCount;
	Fp >> FName;
	cout << "PARAMETERS ARE:\n";
	cout << "DataType =" << DataType << "\nMatSize = " << MatSize <<
"\nVCount = " << VCount << "\nFileName = " << FName << endl;
	//Form the translation matris
	if (DataType == Integer)
	{
	cout << "Creating Integer " << MatSize << "x" << MatSize << endl;
	TrMatris_i = (int **)(malloc(sizeof(int *) * MatSize));
		for (i=0; i<MatSize; i++)
		TrMatris_i[i] = (int *)(malloc(sizeof(int) * MatSize));
	}
	else
	{
	cout << "Creating Float" << MatSize << "x" << MatSize << endl;
	TrMatris_d = (float **)(malloc(sizeof(float *) * MatSize));
		for (i=0; i<MatSize; i++)
		TrMatris_d[i] = (float *)(malloc(sizeof(float) * MatSize));
	}
	//Read traslation matrix;
	if (DataType == Integer)
	{
		cout << "Reading Integer TR Matris\n";
		for (i=0; i<MatSize; i++)
			for (j=0; j<MatSize; j++)
				Fp >> TrMatris_i[i][j];
	}
	else
	{
		cout << "Reading Float TR Matris\n";
		for (i=0; i<MatSize; i++)
			for (j=0; j<MatSize; j++)
				Fp >> TrMatris_d[i][j];
	}
```

```
	//Print Translation Matrix
	if (DataType == Integer)
	{
		for (i=0; i<MatSize; i++)
		{
			for (j=0; j<MatSize; j++)
			cout << TrMatris_i[i][j] << " ";cout << endl;
		}
	}
	else
	{
		for (i=0; i<MatSize; i++)
		{
			for (j=0; j<MatSize; j++)
			cout << TrMatris_d[i][j] << " ";cout << endl;
		}
	}
	//Read Vertex Count and Destination address
	Fp >> VCount >> DestAddr;
	cout << "VCount =" << VCount << "   " << "DestAddr = " << DestAddr <<
endl;
	//Form Vertices and Result Vertices Arrays
	if (DataType == Integer)
	{
		Vertices_i = new int [VCount*MatSize];
		RVertices_i = new int [VCount*MatSize];
	}
	else
	{
		Vertices_d = new float [VCount*MatSize];
		RVertices_d = new float [VCount*MatSize];
	}

	//Read and Print Vertices
	if (DataType == Integer)
	{
		for(i=0; i<VCount*MatSize; i++)
			Fp >> Vertices_i[i];
		//for(i=0; i<VCount*MatSize; i++)
		//	cout <<  Vertices_i[i] << " ";
	}
	else
	{
		for(i=0; i<VCount*MatSize; i++)
		{
			Fp >> Vertices_d[i];
			//Vertices_d[i] = Vertices_d[i] /1000;
		}
		//for(i=0; i<VCount*MatSize; i++)
		//	cout <<  Vertices_d[i] << " ";
	}
	cout << "Baslamak icin hazir. Devam etmek icin enter tusuna
basiniz\n";
	cin.get();
	//cin.get();

	if (DataType == Integer)
	{
		if (MatSize == 3)   // 3x3 Matrix Multiplication
		{
			cout << "3x3 Integer Mutrix Multiplication \n";
			//Read the initial time info
			T0 = GetCycles();
```

```
                //Perform transformation
                for (i=0; i<VCount*3; i+=3)
                {
                RVertices_i[i] = TrMatris_i[0][0]*Vertices_i[i]
                        + TrMatris_i[0][1]*Vertices_i[i+1]
                        + TrMatris_i[0][2]*Vertices_i[i+2];
                RVertices_i[i+1]=TrMatris_i[1][0]*Vertices_i[i]
                        + TrMatris_i[1][1]*Vertices_i[i+1]
                        + TrMatris_i[1][2]*Vertices_i[i+2];
                RVertices_i[i+2]=TrMatris_i[2][0]*Vertices_i[i]
                        + TrMatris_i[2][1]*Vertices_i[i+1]
                        + TrMatris_i[2][2]*Vertices_i[i+2];
                }
                //Read the final time info
                T1 = GetCycles();
        }
        else
        {
                cout << "4x4 Integer Mutrix Multiplication \n";
                //Read the initial time info
                T0 = GetCycles();
                //Perform transformation
                for (i=0; i<VCount*4; i+=4)
                {
                RVertices_i[i] = TrMatris_i[0][0]*Vertices_i[i]
                        + TrMatris_i[0][1]*Vertices_i[i+1]
                        + TrMatris_i[0][2]*Vertices_i[i+2]
                        + TrMatris_i[0][3]*Vertices_i[i+3];
                RVertices_i[i+1]=TrMatris_i[1][0]*Vertices_i[i]
                        + TrMatris_i[1][1]*Vertices_i[i+1]
                        + TrMatris_i[1][2]*Vertices_i[i+2]
                        + TrMatris_i[1][3]*Vertices_i[i+3];
                RVertices_i[i+2]=TrMatris_i[2][0]*Vertices_i[i]
                        + TrMatris_i[2][1]*Vertices_i[i+1]
                        + TrMatris_i[2][2]*Vertices_i[i+2]
                        + TrMatris_i[2][3]*Vertices_i[i+3];
                RVertices_i[i+3]=TrMatris_i[3][0]*Vertices_i[i]
                        + TrMatris_i[3][1]*Vertices_i[i+1]
                        + TrMatris_i[3][2]*Vertices_i[i+2]
                        + TrMatris_i[3][3]*Vertices_i[i+3];
                }
                //Read the final time info
                T1 = GetCycles();
        }
}
else            //FLOAT MATRIX MULTIPLICATIONS
{
        if (MatSize == 3)
        {
                cout << "3x3 Float Mutrix Multiplication \n";
                //Read the initial time info
                T0 = GetCycles();
                //Perform transformation
                for (i=0; i<VCount*3; i+=3)
                {
                RVertices_d[i] = TrMatris_d[0][0]*Vertices_d[i]
                        + TrMatris_d[0][1]*Vertices_d[i+1]
                        + TrMatris_d[0][2]*Vertices_d[i+2];
                RVertices_d[i+1]=TrMatris_d[1][0]*Vertices_d[i]
                        + TrMatris_d[1][1]*Vertices_d[i+1]
                        + TrMatris_d[1][2]*Vertices_d[i+2];
                RVertices_d[i+2]=TrMatris_d[2][0]*Vertices_d[i]
                        + TrMatris_d[2][1]*Vertices_d[i+1]
                        + TrMatris_d[2][2]*Vertices_d[i+2];
```

```
                        }
                        //Read the final time info
                        T1 = GetCycles();
                }
                else
                {
                cout << "4x4 Float Mutrix Multiplication \n";
                        //Read the initial time info
                        T0 = GetCycles();
                        //Perform transformation
                        for (i=0; i<VCount*4; i+=4)
                        {
                        RVertices_d[i] = TrMatris_d[0][0]*Vertices_d[i]
                               + TrMatris_d[0][1]*Vertices_d[i+1]
                               + TrMatris_d[0][2]*Vertices_d[i+2]
                               + TrMatris_d[0][3]*Vertices_d[i+3];
                        RVertices_d[i+1]=TrMatris_d[1][0]*Vertices_d[i]
                               + TrMatris_d[1][1]*Vertices_d[i+1]
                               + TrMatris_d[1][2]*Vertices_d[i+2]
                               + TrMatris_d[1][3]*Vertices_d[i+3];
                        RVertices_d[i+2]=TrMatris_d[2][0]*Vertices_d[i]
                               + TrMatris_d[2][1]*Vertices_d[i+1]
                               + TrMatris_d[2][2]*Vertices_d[i+2]
                               + TrMatris_d[2][3]*Vertices_d[i+3];
                        RVertices_d[i+3]=TrMatris_d[3][0]*Vertices_d[i]
                               + TrMatris_d[3][1]*Vertices_d[i+1]
                               + TrMatris_d[3][2]*Vertices_d[i+2]
                               + TrMatris_d[3][3]*Vertices_d[i+3];

//cout << "OLD : " << Vertices_d[i] << " " << Vertices_d[i+1] << " " <<
Vertices_d[i+2] << " " << Vertices_d[i+3] << endl;
//cout << "NEW : " << RVertices_d[i] << " " << RVertices_d[i+1] << " " <<
RVertices_d[i+2] << " " << RVertices_d[i+3] << endl;
                        }
                        //Read the final time info
                        T1 = GetCycles();
                }
        }

/*
        if (DataType == 1)
        {
                cout << endl;
                for(i=0; i<VCount*3; i++)
                        cout << setw(3) << i << " = " << setw(5) <<
RVertices_i[i] << " ==>" << setw(5) << Vertices_i[i] << endl;
        }
        else
        {
                cout << endl;
                for(i=0; i<VCount*3; i++)
                        cout << setw(3) << i << " = " << setw(5) <<
RVertices_d[i] << " ==>" << setw(5) << Vertices_d[i] << endl;
        }
*/
        //Print the result
        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";
```