GÖMÜLÜ SİSTEMLER İÇİN FPGA TABANLI GRAFİK SİSTEMİ TASARIMI

Kahraman Serdar AY
Yüksek Lisans Tezi

Elektrik-Elektronik Mühendisliği Anabilim Dalı
Kasım – 2010

JÜRİ VE ENSTİTÜ ONAYI

Kahraman Serdar AY'ın **"Gömülü Sistemler İçin FPGA Tabanlı Grafik Sistemi Tasarımı"** başlıklı **Elektrik-Elektronik Mühendisliği** Anabilim Dalındaki, Yüksek Lisans Tezi 26.11.2010 tarihinde, aşağıdaki jüri tarafından Anadolu Üniversitesi Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliğinin ilgili maddeleri uyarınca değerlendirilerek kabul edilmiştir.

	Adı-Soyadı	**İmza**
Üye (Tez Danışmanı)	**: Doç. Dr. ATAKAN DOĞAN**	**.....**
Üye	**: Yard. Doç. Dr. EMİN GERMEN**	**.....**
Üye	**: Yard. Doç. Dr. CÜNEYT AKINLAR**	**.....**

Anadolu Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun ..... tarih ve ..... sayılı kararıyla onaylanmıştır.

Enstitü Müdürü

ÖZET

Yüksek Lisans Tezi

GÖMÜLÜ SİSTEMLER İÇİN FPGA TABANLI GRAFİK SİSTEMİ TASARIMI

Kahraman Serdar AY

Anadolu Üniversitesi
Fen Bilimleri Enstitüsü
Elektrik-Elektronik Mühendisliği Anabilim Dalı

Danışman: Doç. Dr. Atakan DOĞAN
2010, 138 sayfa

Bu tezde iki boyutlu bir grafik işleme ardışık düzeninin FPGA yongasında gerçeklenmesi ele alınmıştır. Tezin hazırlanmasında, gömülü sistemlerdeki değişken grafik ihtiyaçları; piyasada var olan hazır grafik kartlarının gömülü sistem tasarımı kıstaslarına uygun olmaması ve maliyetleri; FPGA ile tasarımın doğasından gelen esnek yapılandırılabilirliğin grafik sistemlerinde sağlayacağı avantajlar motive edici unsurlar olmuştur.

Tezde yapılan prototip çalışmada BitBLT, Alfa Karıştırma, Bresenham Metodu ile Doğru Çizme, Poligon Çizme, Poligon Doldurma gibi pikselleştirme IP Çekirdekleri (IP Cores) VHDL dilinde gerçeklenmiştir. Pikselleştirme öncesi renderleme safhaları olan Model Dönüşümleri ve Görüntü Dönüşümleri, FPGA yongasında gömülü CPU IP Çekirdeğinde çalışmak üzere C dilinde gerçeklenmiştir. Aynı zamanda grafik IP Çekirdeklerinin, C fonksiyonları ile CPU tarafından sürülmesi sağlanmıştır. Donanım olarak Xilinx University Program Virtex-II Pro Geliştirme Kiti, geliştirme ortamı olarak Xilinx EDK ve ISE 7.1 kullanılmıştır.

Anahtar Kelimeler: FPGA, BitBLT, Alfa Karıştırma, Bresenham, Pikselleştirme

ABSTRACT

Master of Science Thesis

FPGA BASED GRAPHICAL SYSTEM DESIGN FOR EMBEDDED SYSTEMS

Kahraman Serdar AY

Anadolu University
Graduate School of Sciences
Electrical and Electronics Engineering Program

Supervisor: Assoc. Prof. Dr. Atakan DOĞAN
2010, 138 pages

In this thesis, realization of a two dimensional sequential graphic-processing system on FPGA is considered. The motivating factors in the preperation of the thesis include varying graphic needs of the embedded systems, inconveniency of commercial graphic cards for the embedded systems and their high costs, advantages that will be available to graphic systems due to the flexible configurability provided by FPGA.

In the prototype work, the rasterization IP Cores, such as line drawing with BitBLT, Alpha Blending, Bresenham Method, polygon drawing and polygon filling, are implemented in VHDL. The rendering phases of model transformations and image transformations before rasterization are implemented in C language to work on embedded CPU IP Core. At the same time, graphic IP Cores are driven by CPU through C functions. As hardware Xilinx University Program Virtex-II Pro Development Kit and as development environment Xilinx EDK, ISE 7.1 are used.

Keywords: FPGA, BitBLT, Alpha Blending, Bresenham, Rasterization

TEŞEKKÜR

Çalışmalarımda beni yönlendiren ve benden yardımlarını esirgemeyen sayın hocam Doç. Dr. Atakan DOĞAN'a, beni destekleyen aileme ve Tuğba ÖZBİLGİN'e teşekkür ederim.

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

1. GİRİŞ

Savunma, otomotiv ve GSM sektörleri başta olmak üzere birçok alanda gömülü sistemlerin grafiksel ihtiyaçları hızla artmaktadır. İşlem yükü açısından maliyetli olan grafik algoritmaları genellikle ana işlemcide değil, harici grafik işleme birimlerinde gerçeklenmektedir. Bu nedenle özel amaçlı ya da basit grafik algoritmalarını destekleyen genel amaçlı grafik işleme yongaları tasarlanmaktadır; fakat, değişken grafik ihtiyaçları ve sürekli gelişen dijital ekran teknolojisi bu yongaları demode hale getirmektedir.

Gömülü sistemlerde kullanılmak istenen grafik işleme donanımları için çeşitli kriterler söz konusudur. Standart olmayan çözünürlüklerin desteklenmesi, düşük güç tüketimi, PCI haricinde daha basit veri yolları üzerinden haberleşebilme, daha ilkel sürücü ve kütüphanelerle birlikte çalışabilme ve daha az parasal maliyet bu kriterlerden bazılarıdır. Bilgisayarlar için geliştirilmiş ekran kartları en basitinden en karmaşığına birçok grafik algoritmasını desteklemekte, hatta programlanabilmektedir. Ancak, ekran kartlarının parasal maliyetleri, güç tüketimleri ve entegrasyon maliyetleri nedeniyle gömülü sistem kriterlerine uygun olmadıkları görülmektedir.

Bunların ışığında paralel işlem yapabilme kabiliyeti ve tasarımda yenilenebilirliği sağlaması açısından FPGA ile grafik sistemi tasarımı eğilim gösterilmesi gereken bir konudur. Bu tezde yapılan çalışmada FPGA içerisinde gömülü olan bir işlemci ve bu işlemciye veri yolu ile bağlı grafik işleme çevresel birimlerinden oluşan iki boyutlu bir grafik sisteminin gerçeklenmesi ele alınmıştır.

Gerçeklenen grafik sisteminde temel amaç, grafik oluşturma sürecinde işlem yükü fazla olan pikselleştirme ve pikselleştirme sonrası operasyonlarda iş yükünü CPU'dan alacak çevresel birimler oluşturmak ve IP Çekirdekleri (IP Core) olarak tasarlanan bu çevresel birimler sayesinde FPGA yongasında yenilenebilir, geliştirilebilir bir grafik sisteminin oluşturulabileceğini göstermektir. Grafik sistemi CPU ve veri yolu (data bus) dahil olmak üzere tamamen IP Çekirdeklerinden oluşmaktadır. Hedeflenen tasarımda grafik işlemleri C dilinde yazılan grafik kütüphaneleri ve VHDL dilinde gerçeklenen grafik IP Çekirdekleri ile ortaklaşa gerçekleştirilir. CPU'da çalışacak olan grafik

fonksiyonlarının görevi grafik oluşum sürecinde modelleme aşamasını ve renderleme aşamasındaki ilklendirmeleri yapmak, pikselleştirme ve pikselleştirme sonrası operasyonlar için de ilgili grafik işleme birimlerini sürerek grafik oluşum sürecini yönetmektir.

Grafik oluşum sürecinde görüntüleme aşaması CPU'dan tamamen bağımsızdır. Bu çalışmada kullanılan Xilinx VGA IP Çekirdeği, çerçeve arabelleğine (frame buffer) yazılan piksel verilerini ekrana göndermekle görevlidir. Bu IP Çekirdeği sayesinde görüntüleme safhasında VGA monitörler kullanılabilmektedir. Başka türde ekranlar kullanılmak istenirse yapılması gereken tek şey VGA IP Çekirdeği yerine, kullanılmak istenen ekranı sürebilen ve sistemde kullanılan veri yolu protokolüne uyumlu bir IP Çekirdeği tasarlayıp sisteme eklemek olacaktır. Ayrıca çerçeve arabelleğinin ve modellemede kullanılan verilerin yer aldığı DDR belleğin sürülebilmesi için de XUPV2P geliştirme kiti ile birlikte sunulan DDR IP Çekirdeği kullanılmıştır. Bu çalışmada iki tip IP Çekirdeği tasarlanmıştır. Bunlar:

- Bresenham,
- BitBLT ve Alfa Karıştırıcı.

Bu IP Çekirdekleri CPU'nun ve DDR sürücüsünün de üzerinde yer aldığı veri yolunda kendi adres uzaylarına sahip çevresel birimlerdir. Bu çevresel birimleri sürmek için oluşturulan ve CPU'da çalışan fonksiyonlar adreslenmiş yazmaçları sayesinde bu IP Çekirdeklerine erişirler. Kesikli, düz doğru renderlemek için Bresenham IP Çekirdeği kullanılırken, poligon doldurma, BitBLT ve alfa karıştırma operasyonları için BitBLT IP Çekirdeği kullanılır.

Modelleme ve koordinat dönüşümleri tamamen CPU IP Çekirdeği üzerinde yapılmaktadır. Bu çalışmada örnek olarak poligonlarla karakter modelleme yapan ve IP Çekirdeklerini kullanarak istenilen büyüklükte karakter renderleyen, taşıyan ve arka plan üzerinde gösterebilen fonksiyonlar yazılmıştır.

1.1 Bugüne Kadar Yapılan Çalışmalara Genel Bakış

Singh, Bellec [1] grafik sistemlerinde FPGA kullanımının getireceği avantajları araştırmışlar, çember çizme algoritmasını baz alarak birtakım deneyler

yapmışlardır. Vardıkları sonuçlara göre FPGA tabanlı grafik sistemlerinin performansının genel amaçlı grafik yongaları ile özel amaçlı grafik yongalarının arasında olduğunu gözlemişlerdir. FPGA'nın grafik sistemlerine yeniden yapılandırılabilme özelliğini sağladığını ve bunun getirdiği avantajları vurgulamışlardır.

Chin [2] FPGA ile paralel işlem yapabilme kabiliyetinin grafik algoritmalarının gerçeklenmesinde CPU'lara göre avantaj sağladığı göstermiş, FPGA ile renk dönüşüm modülü tasarımı yapmıştır.

Holten [3] paralel işlem kabiliyetine sahip çevre birimleri ve CPU tabanlı hibrit 3D grafik sistemi tasarım prensiplerini anlatmış ve FPGA'lar sayesinde geliştirilebilir sistem tasarımı üzerinde durmuştur.

Warner [5] üç boyutlu cisimlerin iki boyutlu düzlemde ifade edilmesinde kullanılan matris çarpımlarının VHDL'de gerçeklenen matris çarpım motoru ile gerçekleştirilmesini başarmıştır.

Knutsson [6] açık kaynak IP Çekirdeklerinden faydalanarak üç boyutlu grafik sistemi tasarımı yapmaya çalışmıştır. Farklı veri iletim yolu standartlarına göre tasarlanmış IP Çekirdekleri ile çalışmanın getirdiği sıkıntıları ve veri yolu dönüştürücülerin performans açısından sorun yarattığını gözlemlemiştir.

Gray, Woodson, Chau ve Retzlaff [7] askeri sistemlerdeki değişken grafik ihtiyaçlarını karşılamak üzere FPGA tabanlı grafik motoru tasarımı üzerinde durmuşlar, sadece pikselleştirme sonrası operasyonları hedefleyen BitBLT modülü tasarımı yapmışlardır.

Altera [8] FPGA yongasında gömülü işlemci ve VGA IP Çekirdeği aracılığı ile basit grafik sistemi tasarımı yapmıştır.

logiBITBLT IP Core [10] açık kaynak olmayan paralı bir IP Çekirdeğidir ve Bit BLT operasyonları için tasarlanmıştır. Sadece pikselleştirme sonrası operasyonları destekleyen bir IP Çekirdeği için maliyetinin fazla oluşu bu tezde yapılan çalışmanın önemini ortaya koymaktadır.

POWERVR MBX IP Core [11] ve DAVE_2D [14] maliyeti çok yüksek iki boyutlu ardışık düzen IP Çekirdeği olup açık kaynak değillerdir ve yeni veri yolu standartlarını destekler hale getirilemezler. Bu dezavantajlı durumları ve

FPGA'nın grafik sistemlerinde etkin kullanılabilirliğini gösteriyor oluşları tezin yapılmasında motive edici unsur olmuştur.

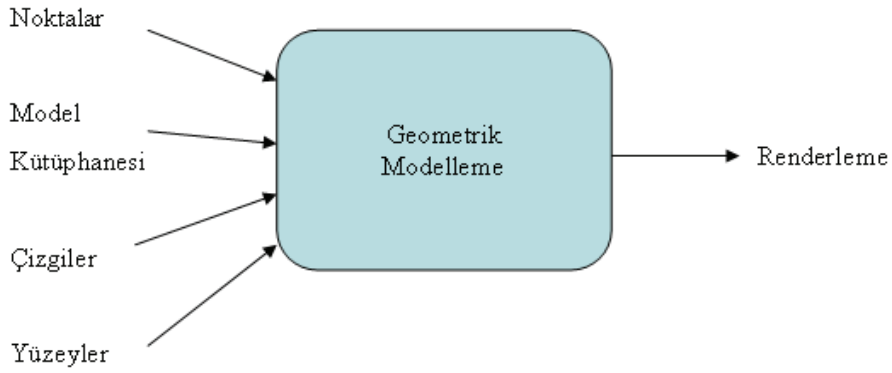
Automotive Graphics System [12] otomotiv sektöründe artan grafik ihtiyacı ve FPGA ile grafik tasarımına yönelik eğilimler üzerine yapılmış bir çalışmadır.

2. İKİ BOYUTLU GRAFİK OLUŞUM SÜRECİ

Grafik oluşturma süreci 3 temel aşamadan oluşmaktadır. Bu aşamalar şunlardır [15]:

- Modelleme
- Renderleme
- Görüntüleme

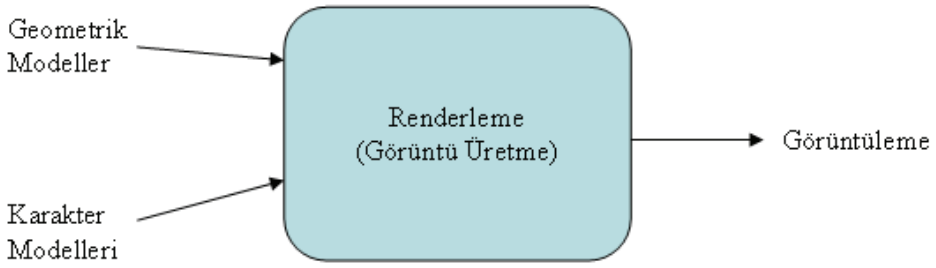
Modelleme, geometrik modelleme ve doku (texture) modellemeden oluşmaktadır. Geometrik modelleme, objenin bir takım matematiksel ve geometrik ilkeller *(primitive)* ile modelinin kurulmasıdır (Şekil 2.1). Başka bir deyişle modelin iskeletinin oluşturulmasıdır. En basit geometrik ilkeller; noktalar, çizgiler, çokgenler ve yüzeylerdir. Geometrik modellemede gerçek objenin sentetik (yapay) modeli bu ilkeller yardımıyla kurulur ve renderleme aşamasına gönderilir. Renderlenmiş model gerçek nesnenin bir prototipi olur [24].



Şekil 2.1. Geometrik modelleme safhası [24]

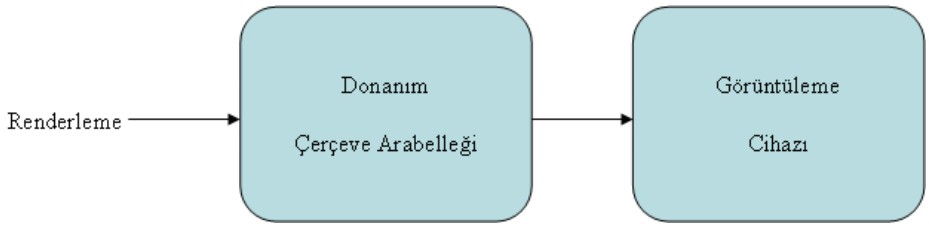
Doku modelleme ise oluşturulan iskeletin giydirilmesidir. Doku kaplama veya örtü anlamlarına gelmektedir. Doku kaplama, 2 boyutlu dokuları 3 boyutlu geometrik modellere uygulayarak 3 boyutlu katı cisimler oluşturulmasını sağlar. Doku kaplama konusu 3 boyutlu grafik modellemenin konusudur. İki boyutlu grafik ilkellerinin alanlarının, motif (pattern) bit haritaları ile doldurulması ile karıştırılmamalıdır.

İkinci aşama olan renderleme, sentetik modelin görüntüleme cihazlarının anlayabileceği şekle sokulması ile ilgilidir. Renderleme, görüntü parçalarına (geometrik ve doku modelleri) gerekli dönüşümleri yapıp onları bir bütün haline getirme işlemidir (Şekil 2.2).



Şekil 2.2. Renderleme safhası [24]

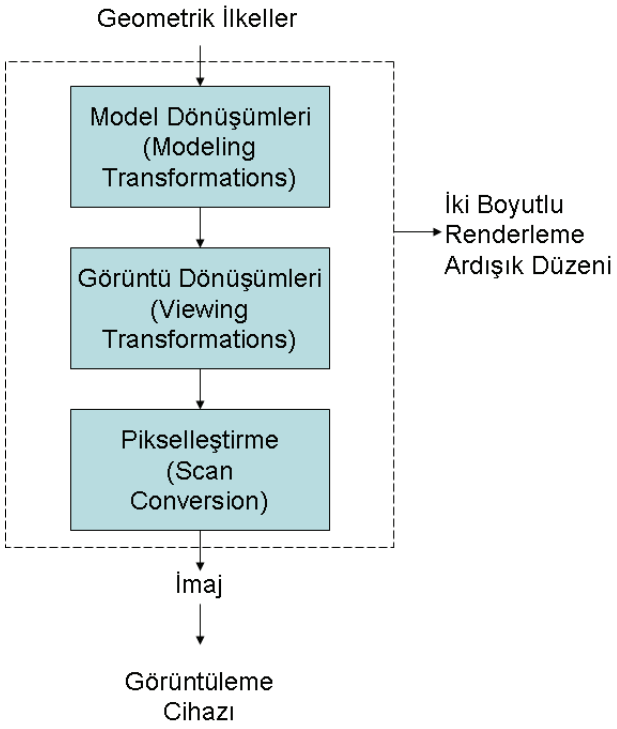
Son olarak görüntüleme aşaması ise görüntünün görüntüleme cihazlarına iletilmesi ve görüntülenmesidir. Son hale getirilmiş görüntü verisi (renderlenmiş görüntü) çerçeve arabelleği denen ve sadece görüntü verisinin saklanması için tahsis edilmiş olan bellek alanında tutulur. Çerçeve arabelleği grafik sisteminin bir parçasıdır. Grafik sisteminde görüntülemeden görevli birim buradaki grafik verilerini görüntüleme cihazına gönderir (Şekil 2.3).



Şekil 2.3. Görüntüleme safhası

2.1 Renderleme

Renderleme geometrik ilkeller ile başlar. Geometrik ilkeller ile modellenen nesneler görüntülenme aşamasına kadar model dönüşümleri, görüntü dönüşümleri ve pikselleştirme gibi renderleme aşamalarından geçer. Şekil 2.4'te renderleme ardışık düzeni gösterilmektedir [24].



Şekil 2.4. İki boyutlu renderleme ardışık düzeni

2.1.1 Model dönüşümleri

Geometrik ilkeller renderleme ardışık düzenine girmeden önce son görsel imajın tanımlı olduğu koordinat sistemi üzerinde tanımlı değillerdir. Bir geometrik ilkelin grafik sistemindeki ifadesi onun matematiksel tanımı ve bazı özniteliklerinden ibarettir. Örneğin, bir çemberin matematiksel tanımı için çap bilgisi ve öznitelikleri olarak da rengi, çizgi kalınlığı vs. yeterli olabilir. Bu çember iki boyutlu renderleme ardışık düzenine girdikten sonra gerçek koordinat sisteminin bir parçası olacak ve konumu burada belirlenecektir. Bir çokgeni ise

köşe noktaları ile tanımlayabiliriz. Bu köşe noktalarının koordinat değerleri çokgenin kendi tanımlı olduğu koordinat sistemine göre belirlenir. Model dönüşümleri sonucunda geometrik ilkeller grafik sistemindeki bütün objelerin yer aldığı gerçek koordinat sistemi üzerinde tanımlı hale gelirler.

Model dönüşümleri aşağıda verilmiştir. Burada, (x, y) nesnenin ilk konumu ve , y') nesnenin dönüşüm sonrası konumunu göstermektedir.

- **Nakil** (Translation)**:** Nesnenin kendi koordinat sisteminin başka bir koordinat sistemine çevrilmesidir. 't' taşıma miktarı ise,

$$= x + tx$$

$$= y + ty$$

- **Ölçeklendirme** (Scaling): Nesnenin boyutunun değiştirilmesidir. 's' ölçeklendirme oranı ise,

$$= x * sx$$

$$= y * sy$$

- **Döndürme** (Rotation): Nesneyi tanımlayan bütün köşe noktalarının (verteks) belirli bir açıyla döndürülmesidir. '∅' döndürme açısı ise,

$$x = x * - y *$$

$$y = x * + y *$$

- **Kaydırma** (Shearing): Parça kaydırarak nesnenin şeklinin deforme edilmesidir. 'h' deformasyon oranı ise,

$$= x + hx * y$$

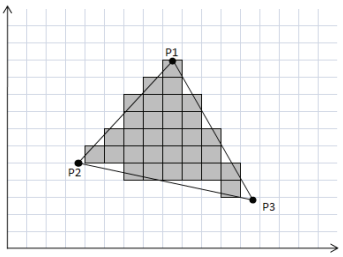
$$= y + hy * x$$

2.1.2 Görüntü dönüşümleri

Ekran üzerinde gösterilecek nesnelerin köşe noktalarının koordinatlarının gerçek koordinat sisteminden ekran koordinat sistemine dönüştürülmesidir. Bilgisayar grafiklerinde genellikle, ekranın sol üst köşesi başlangıç (0,0) noktası olarak kabul edilir. Buna göre +X ekseni sağa doğru, +Y ekseni ise aşağıya doğru artacaktır.

2.1.3 Pikselleştirme

Görsel nesneler ekran koordinat sisteminde tanımlandıktan sonra yapılması gereken iş, nesneleri ekran üzerindeki hangi piksellerin temsil edeceğini bulmaktır. Bu işleme pikselleştirme (scan conversion) denir. Örneğin, Şekil 2.5'teekran koordinat sisteminde tanımlanmış bir üçgenin piksellerle ifade edilişi gösterilmektedir.



Şekil 2.5. Ekran koordinat sistemine dönüştürülmüş bir üçgenin pikselleştirilmesi

Piksellerden oluşan bir görüntü hiçbir zaman gerçek dünyadaki görüntüyü tam olarak veremez. Örneğin, piksellerle hiçbir zaman sürekli bir çizgi elde edemeyiz. Ancak gerçek ve sürekli bir çizgiye yakınsayan bir görüntü elde edebiliriz. İşte bilgisayar grafiklerinde kullanılan pikselleştirme algoritmaları bu yakınsamayı maksimuma çıkarmayı hedefler. Örneğin, doğru çizme algoritmalarından Bresenham'ın Çizgi Algoritması bilgisayar grafiği dünyasında en çok kullanılan algoritmadır. Nesnelerin içlerinin doldurulması da çeşitli algoritmalar tarafından gerçeklenir. Bu algoritmalar hem görsel performans açısından hem de grafik sisteminin çalışacağı donanımın performans kriterleri açısından değerlendirilir ve en uygun olanları seçilir. Bu tezdeki prototip çalışmada pikselleştirme algoritmalarından Bresenham'ın Çizgi Algoritması, Orta Nokta Çember Algoritması ve Satır Tarama Yöntemi İle Poligon Doldurma Algoritması gerçeklenmiştir.

2.1.4 Pikselleştirme sonrası operasyonlar

Nesneler pikselleştirildikten sonra, nesnelerin dikdörtgensel bloklar halinde bit haritalarını elde etmiş oluruz. Bu bit haritaları üzerinde de işlemler

yapmamız gerekebilir. Bu işlemler genelde işlem yükü açısından iki boyutlu grafik sisteminin en maliyetli işlemleridir. Bunun nedeni daha önceki operasyonlar köşe noktaları üzerinde yapılırken, pikselleştirme sonrası operasyonların nesnenin her pikseli üzerinde yapılıyor olmasıdır (üç boyutlu grafik sistemlerinde köşe noktaları üzerindeki işlemler de maliyetli olabilmektedir). Pikselleştirme sonrası operasyonlar genelde bit bloklarının taşınması, kaynak ve hedef bit blokları üzerinde Boolean işlemlerin gerçekleştirilmesi ve kompozit görüntü elde etme işlemleridir [8]. Bu tezdeki prototip çalışmada, bit blokları üzerinde ızgara operasyonlarının yapılmasını sağlayan BitBLT bloğu ve nesne saydamlaştırma ve kompozit görüntü elde etmemizi sağlayan Alpha Blender bloğu FPGA yongasında gerçeklenmiştir.

3. GRAFİK SİSTEMİ ALGORİTMALARI

Bu bölümde, iki boyutlu grafik oluşum aşamalarının FPGA yongasında gerçeklenebilirliğinin gösterilebilmesi için belirlenen grafik algoritmaların detaylarına yer verilmiştir. Bu algoritmalar iki boyutlu grafik sistemlerinde sıkça kullanılmakta olup, FPGA yongasında verimli bir şekilde gerçeklenebilecek özelliktedirler. Sonraki bölümde ise, pikselleştirme öncesi ve pikselleştirme sonrası aşamaların gerçeklenme yöntemleri ve algoritmaların ardışık düzendeki işlevleri anlatılmıştır.

3.1 Bresenham'ın Çizgi Algoritması

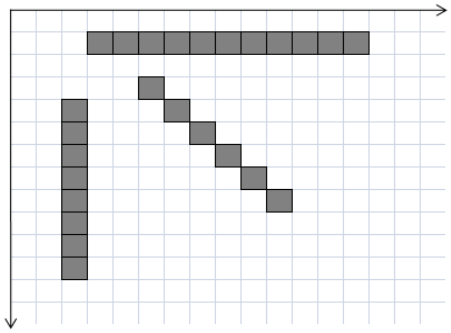
Bir bilgisayar grafiğinde çizilecek doğrunun aşağıdaki kriterlere uyması gerekir:

- Eğimi ve uzunluğundan bağımsız olarak doğru boyunca sabit bir kalınlığa sahip olmalı,
- Doğru koordinatlarda başlamalı ve bitmeli,
- İdeal doğruya yakın bir yol izlemeli.

Donanım ve yazılım açısından değerlendirildiğinde performans kriterleri de söz konusudur:

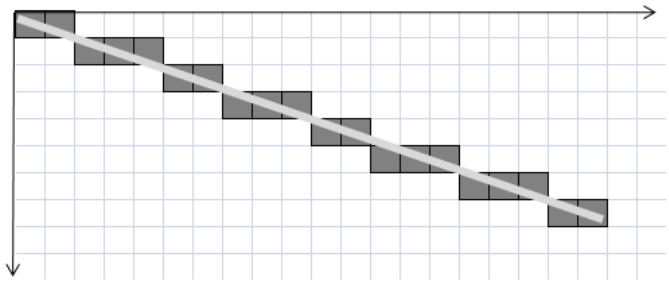
- En az piksel hesabı ile doğruyu oluşturabilmeli,
- Sadece tamsayılar kullanılarak koordinatlar hesaplanabilmeli,
- Sadece toplama, çıkarma ve bit kaydırma gibi Assembly seviyesinde maliyeti az operasyonlarla gerçeklenebilmeli [15].

Eğer sadece yatay, dikey ya da 45° eğimli doğrulara ihtiyacımız olsaydı, bu doğruları Şekil 3.1'de gösterildiği gibi piksellerle kolaylıkla ifade etmek mümkün olabilirdi.



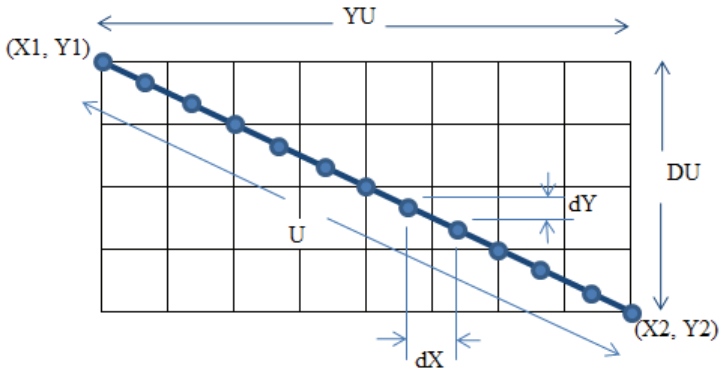
Şekil 3.1. Ekran koordinat sistemi üzerinde yatay, dikey ve 45° eğimli çizgiler

Ancak eğimi 45°'den az olan bir doğruyu piksellerle gerçeğe yakın bir şekilde oluşturabilmek için bir piksel satırında ya da sütununda birden fazla ve farklı sayılarda piksel doldurmak gerekebilir. Şekil 3.2'de eğimi 45°'den küçük olan bir doğrunun piksellerle ifade edilişi görülmektedir.



Şekil 3.2. Eğimi 45°'den küçük olan doğrunun pikselleştirilmesi

Doğru çizme algoritmalarının hepsi gerçek doğruya en yakın pikselleri bulmayı hedefler. Bresenham Algoritması, Digital Differential Analyzer (DDA) Algoritması'nın geliştirilmiş halidir. DDA en basit doğru çizme algoritmasıdır. DDA'nın temeli doğrunun matematiksel ifadesine dayanmaktadır. DDA doğruyu Şekil 3.3'teki gibi eşit aralıklı parçalara ayırır [15].



Şekil 3.3. DDA algoritması

Burada, yatay uzunluk (YU) ve düşey uzunluk (DU) şu şekilde bulunabilir:

$$YU = X2 - X1 \tag{3.1}$$

$$DU = Y2 - Y1 \tag{3.2}$$

Doğrunun gerçek uzunluğu $U =$ ile hesaplanabilir. Doğrunun uzunluğunun yaklaşık değeri ise,

$$U = + |DU| \tag{3.3}$$

şeklinde hesaplanabilir.

DDA algoritmasında doğru, Şekil 3.3'teki gibi U sayıda eşit parçaya bölündükten sonra, her parçanın yatay bileşeninin uzunluğu $dX =$, düşey bileşeninin uzunluğu ise $dY =$ olmaktadır. Eğer (X1, Y1) noktasından başlanılırsa, doğru boyunca gerekli noktalar (dX, dY) artırımları kullanılarak bulunabilir. Böylece bu koordinata en yakın pikseli boyayarak doğruyu çizebiliriz.

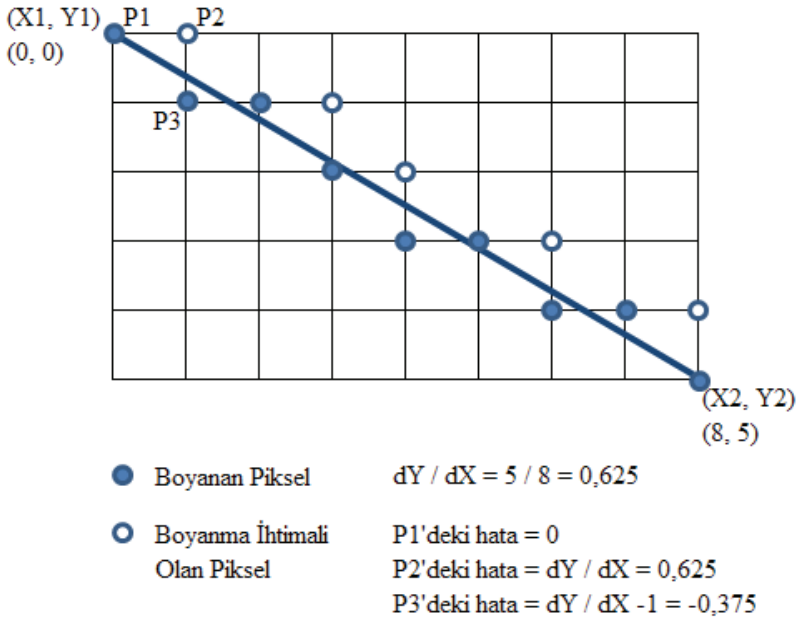
Görüldüğü gibi DDA algoritmasında gerçel sayılarla işlemler söz konusudur. Bu algoritmanın gerçeklenmesi hem bu açıdan hem de gerekenden fazla nokta hesaplama gerektirdiğinden maliyetli olacaktır.

Bresenham'ın Çizgi Algoritması yukarıda değinilen tüm kriterleri sağlayan bir algoritmadır [15]. Algoritmanın yalın hali ondalıklı katsayılara ihtiyaç duymaktadır. Ancak algoritma optimize edilerek sadece toplama, çıkarma ve bit kaydırma operasyonları kullanılarak gerçeklenmektedir.

Bresenham Algoritması ile gerçeklenen birim kalınlıklı bir doğrunun temel özelliği, doğrunun bulunduğu her satırda ya da her sütunda sadece 1 piksel ile temsil edilmesidir. Şekil 3.2'de görülen doğru her sütunda yalnız 1 piksel içerir. Fakat satırlarda 2 ya da 3 piksel ile gösterilmektedir. Dikkat edilirse bu doğru üzerinde ilerlerken yatay eksenin daha hızlı değiştiğini görürüz. Bresenham Algoritması ile gerçeklenecek bir doğrunun hangi eksendeki bileşeni daha hızlı değişiyorsa o eksen *majör eksen* olarak kabul edilir. Majör eksene dik çizilen doğrular yalnızca 1 pikselden geçebilir.

Matematiksel olarak ifade edersek, majör eksen X olarak kabul edildiğinde Bresenham Algoritması, eğiminin mutlak değeri 1'den küçük ($|m|<1$) olan doğrular için geçerlidir. Ancak eğimi farklı diğer doğrular da gerekli eksen dönüşümleri yapılarak aynı algoritmayla gerçeklenebilir. Algoritma yatay, düşey ve 45°'lik doğrular için geçersizdir; zaten bu doğrular da hiçbir hesaplama gerektirmeden kolaylıkla çizilebilirler.

Bresenham Algoritması'nda majör eksene göre döngüsel olarak ilerlenir ve bulunulan pikselden sonraki pikselin aynı hizada mı yoksa çaprazda mı olacağına karar verilir. Eğimi sıfırdan küçük olan doğrular için alt çapraza geçilir, yani y değeri 1 azaltılır. Tersi durumda ise y değeri 1 arttırılarak üst çapraza geçilir. Bunun için hata kontrolü yapılır ve aynı hizada olan ya da çaprazında olan iki pikselden doğrunun gerçek yoluna en yakın olanı seçilir. Gerçek doğrunun eğimi olan dy/dx, bir piksel ilerlerken boyayacağımız pikselin doğrunun neresinde kaldığının bulunmasına yardımcı olur. Gerçek doğrunun olması gereken yer ile eldeki pikselin farkı karşılaştırılarak hata bulunur. Y koordinatını değiştirmeden aynı satırda ilerlerken hata her pikselde dy/dx kadar artacaktır. Bir noktada hata 0,5'ten daha büyük olacaktır, o zaman bir aşağıdaki piksele geçilir, hata 1'den çıkarılarak 0,5'ten daha küçük bir hata elde edilir. Bu işlem Şekil 3.4'te gösterilmiştir [24].



Şekil 3.4. Bresenham Algoritmasında hata analizi

```
function DogruCiz(X1, Y1, X2, Y2)
        real Hata := 0
        real dY := (Y2 - Y1)
        real dX := (X2 - X1)
        real dYBYdX := dY/dX
        int y := Y1

        for x from X1 to X2
                PikselBoya(x,y)
                Hata := Hata + dY / dX;
                if Hata >= 0.5 then
                        y := y + 1;
                        Hata := Hata - 1
```

Şekil 3.5. Bresenham Algoritması'nın pseudo kodu

Şekil 3.5'te pseudo kodu görülmekte olan Bresenham Algoritması'nda, karşılaştırma amacıyla kullanılan hata katsayısı için gerçel sayılar kullanılmaktadır. Bu karşılaştırma operasyonunu bazı matematiksel dönüşümler yaparak daha basit matematiksel işlemlerle yapılabilecek hale getirmek mümkündür. Şekil 3.5'teki algoritmada yapılan karşılaştırmayı şöyle açıklayabiliriz:

$$Hata > 0{,}5 \tag{3.4}$$

(3.4)'teki ifadenin her iki tarafında da gerçel sayılar vardır. Bu ifadede iki tarafı da 2 ile çarparsak sağ taraf tam sayı olacaktır.

$$2 * Hata > 1 \tag{3.5}$$

Hata katsayısı her iterasyonda dy/dx ile toplanmaktadır. Eşitsizliğin her iki tarafı da dx ile çarpıldığı takdirde iki taraf tamsayı haline getirilmiş olur:

$$2 * dx * Hata > dx \tag{3.6}$$

(3.6)'daki eşitsizliğin her iki tarafından dx çıkarılacak olursa sadece 0 ile karşılaştırma yaparak algoritma gerçeklenebilir.

$$2 * dx * Hata - dx > 0 \tag{3.7}$$

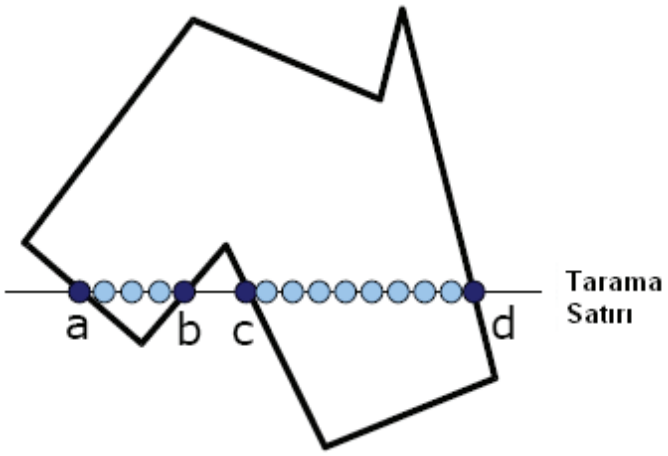
Bu şekilde tüm değerler önceden hesaplatılırsa Şekil 3.6'da verilen pseudo kod elde edilmiş olur. Bu kodda majör eksenin X ve eğimin 0 ile 1 arasında olduğu kabul edilmiştir. Görüldüğü gibi Geliştirilmiş Bresenham Algoritması'nda gerçel sayılarla işlem bulunmamaktadır. Bütün değişkenler tamsayı olup, sadece toplama, çıkarma ve bit kaydırma işlemleri kullanılmaktadır. Bu algoritma FPGA yongasında az sayıda kaynak harcanarak efektif bir şekilde gerçeklenebilir.

```
function DogruCiz(X1, Y1, X2, Y2)
        int Hata := 0
        int 2dY := (Y2 - Y1) << 1
        int 2dX := (X2 - X1) << 1
        int y := Y1

        for x from X1 to X2
                PikselBoya(x,y)
                Hata := Hata + 2dY;
                if Hata > 0 then
                        y := y + 1;
                        Hata := Hata – 2dX
```

Şekil 3.6. Geliştirilmiş Bresenham Algoritması'nın pseudo kodu

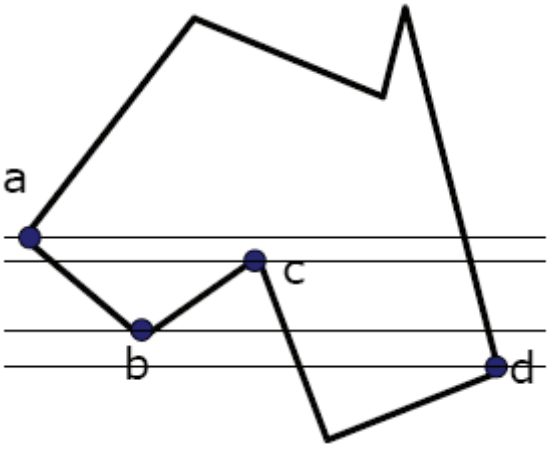
3.2 Satır Tarama Yöntemi ile Poligon Doldurma Algoritması

Satır tarama algoritması poligonu kesen piksel satırlarının belirlenmesi, bu satırların poligonun hangi kenarlarını hangi noktalarda kestiğinin bulunması ve bu noktaların arasında kalan poligonun alanına ait kısımdaki piksellerin boyanmasından ibarettir [15]. Bir doğru bir poligonu en az iki olmak üzere her zaman çift sayıda noktadan keser. Bir satırın kestiği noktaları x değerlerine göre sıralarsak (1, 2, … ,n), sırası tek sayı olan nokta ile onun bir sağındaki nokta arasında kalan kısım poligonun içerisindedir. Çift sayı olanla onun bir sağındaki nokta arasındaki kısım ise poligonun dışındadır. Yani tarama satırı üzerinde ilerlerken ilk noktadan itibaren satır üzerindeki pikseller boyamaya başlanır ve bir sonraki noktada boyama bitirilir. İşlem bu şekilde devam eder. Ancak köşelerin üzerinden geçen satırlar için önemli bir istisna söz konusudur. Bu noktalar için ayrıca bir değerlendirme yapılması gerekir. Şekil 3.7'de herhangi bir köşeden geçmeyen bir satır üzerinde piksellerin nasıl boyandığını görebiliriz. Burada a ve c noktalarında piksellerin boyanmasına başlanmakta, b ve d noktalarında ise boyama bitirilmektedir.



Şekil 3.7. Satır tarama algoritmasında kenar kesim noktalarının bulunuşu

Köşeden geçen bir satır tek bir noktadan geçmesine rağmen iki kenar üzerinden de geçmiş olur. Bu durumda iki kenar için aynı noktayı iki kez listeye almak algoritmayı hataya uğratır. Aslında bu her köşe noktası için geçerli olmayabilir, yani bazı köşe noktalarının iki kez sayılması algoritmada hataya yol açmaz. Şekil 3.8'de farklı köşelerden geçen piksel satırları görülmektedir.

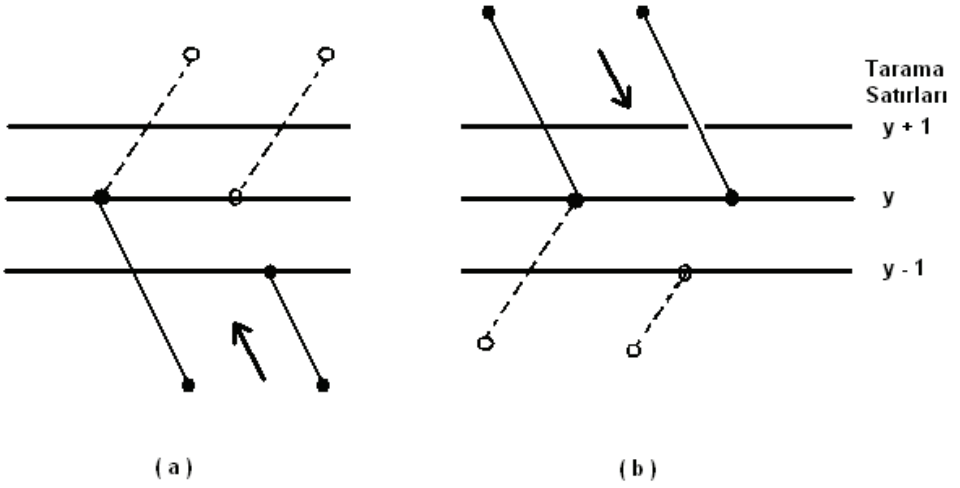


Şekil 3.8. Satırların ortak köşe noktalarından geçişi

Bu şekilde görülen b ve c köşelerinin iki kez sayılması problem yaratmaz. Bunun nedeni, bu noktaların üzerinden geçerken piksel boyama durumunun değişmiyor olmasıdır. Şekilde de görüleceği gibi, b noktasından önce poligonun dışında ilerlenirken, b noktasından sonra poligonun dışında devam edilmektedir.

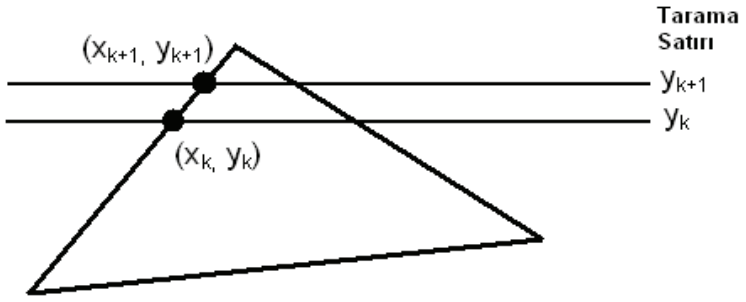
Aynı şekilde c noktasından önce poligonun içinde ilerlenirken, noktanın üstünden geçtikten sonra tekrar içinde olmaya devam edilmektedir. Fakat a ve d noktaları iki kez sayılamaz. Çünkü bu noktalarda poligonun dışındayken içine, içindeyken de dışına çıkılmaktadır. Yani bu noktalarda piksel boyama durumu değişmelidir. Eğer bu noktalar iki kez sayılırsa piksel boyama durumu aynı kalır ve hata oluşur. İki kez sayılması problem yaratan köşeyi bulmak için bir kural oluşturulabilir. Dikkat edilirse b ve c noktalarının birleştirdiği kenarların ortak olmayan noktaları, ortak noktayı kesen satırın aynı tarafında kalmaktadır. a ve d köşelerinde ise birleştirdikleri kenarların ortak olmayan noktalarından birisi satırın alt tarafında, diğeri üst tarafındadır.

Problemli olan köşe noktasının bulunmasında poligonun tanımlanış şekli de kolaylık sağlayabilir. Modelleme aşamasında poligonlar vektörel olarak tanımlarken köşe noktalarının saat yönünde ya da ters yönde sıralanması, satır tarama yöntemiyle poligon doldurma algoritmasının gerçeklenmesini kolaylaştıracaktır. Bu durumda yan yana sıralanan üç noktanın ortada olanının y değeri yanındakilerin y değerlerinin arasında ise bu nokta iki kez sayılamayacak olan noktadır. Bu tür köşe noktaları için bir düzeltme yapılması gerekir. Bu tür noktaların birleştirdiği kenarlardan birini 1 adım kısaltmak ve yeni bir nokta tanımlamak doğru çözüm olacaktır. Bu yöntem Şekil 3.9'da görülmektedir. Şekil 3.9 (a)'da saat yönünde sıralanmış nokta üçlüsünde y değeri artmaktadır. Bu durumda ilk kenarın diğerine ortak olan bitiş noktasının y değeri bir azaltılır ve yeni bir nokta oluşturulur. Şekil 3.9 (b)'de ise y değeri azalmaktadır. Bu durumda da ikinci kenarın ortak olan bitiş noktasının y değeri bir azaltılır. Böylece algoritma doğru çalışır.



Şekil 3.9. Satır tarama algoritmasında problem yaratan köşe noktalarını birleştiren kenarlardan birinin kısaltılarak yeni nokta oluşturulması

Geometrik nesnelerin pikselleştirilmesinde diğer grafik algoritmalarında da faydalanıldığı gibi tutarlılık (coherence) avantajından faydalanılabilir. Burada tutarlılıktan kasıt nesneyi oluşturan birimlerin özelliklerinin her noktada aynı olmasıdır. Bu özellik işlem yükünün azaltılmasına yardımcı olur, çünkü bu sayede artımsal (incremental) hesaplar yapılabilir. Bu algoritmada ihtiyaç duyulan satır-kenar kesişim noktaları Bresenham'ın Çizgi Algoritması'nda olduğu gibi artımsal olarak hesaplanabilir. Bu algoritmayı matematiksel olarak şöyle ifade edebiliriz:



Şekil 3.10. Ardışık iki tarama satırının bir kenarı kesiş noktaları

Şekil 3.10'da poligonu kesen ardışık iki satırı ele alalım. Buradaki kenarın eğimi:

$$m = \qquad (3.8)$$

y eksenindeki değişim her zaman 1 dir:

$$- = 1 \tag{3.9}$$

Bu durumda,

$$= + \tag{3.10}$$

olacaktır. Yani kesim noktasının x değeri bir önceki kesim noktasının x değeri ve kesilen poligon kenarının eğimi ile bulunabilir. Burada eğimi ondalıklı bir sayı olarak ifade etmektense tamsayıların oranı olarak ifade etmek ondalıklı sayılarla işlem yapmaktan kurtarır.

$$m = \tag{3.11}$$

$$= + \tag{3.12}$$

Burada Δx ve Δy poligon kenarının iki bitim noktasının x ve y değerleri arasındaki farktır. Eğer istenirse ve mümkünse kesir sadeleştirilebilir.

Artımsal Hesap: Yukarıdaki formülden faydalanılarak kesim noktalarını hesaplamak için bir sayaç tutulur ve başlangıçta bu sayaç sıfıra eşitlenir. Her yeni tarama satırına geçildiğinde sayaç Δx kadar arttırılır. Eğer sayaç Δy' ye eşit ya da büyükse Δy'den küçük olana kadar sayaç Δy kadar azaltılır. x değeri de doğrunun eğim yönüne göre 1 arttırır ya da azaltılır. Bu algoritma sayesinde eğimin 1'den küçük ya da büyük olması ile ilgilenilmez. Tek bilmemiz gereken eğimin işaretidir.

3.3 BitBLT (Bit Block Transfer) İşlemi

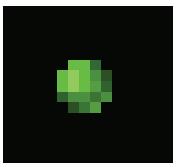
Grafik oluşturma sürecinde işlem yükü en yüksek olan aşama nesnelerin piksellerle ifade edilmesinden (rasterization, pikselleştirme) sonra başlar. Çünkü pikselleştirme öncesi modelleme ve koordinat dönüştürme gibi vektörel grafik operasyonlarda ilkel grafik nesnelerini ifade eden köşe noktaları üzerinde işlem yapılırken piksel operasyonlarında nesneyi oluşturan piksel verileri üzerinde işlem yapılır. Nesneler pikselleştirme işleminden sonra dikdörtgensel piksel veri blokları (Bitmap-Bit Haritası) şeklinde tanımlanır. Bu aşamadan sonra temel

ihtiyaç bit haritalarını bellekte bir yerden bir yere taşımak ya da pikselleri temsil eden dijital ifadelerin üzerinde temel Boolean işlemleri (one, zero, and, or, nand, nor, xor, nxor) gerçekleştirmektir. Literatürde bit haritaları üzerinde yapılan bu operasyonlar BitBLT ya da Bit Blit (Bit Block Image Transfer) terimi ile tanımlanır [5].

BitBLT operasyonu temel olarak *kaynak* ve *hedef* bit haritaları arasında ızgara operasyonlarının (raster operation) yapılmasıdır. Temel ızgara operasyonları yukarıda belirtilen Boolean işlemler ve genel olarak da kaynağın hedef üzerine yazılması, yani taşıma işlemidir. Izgara operasyonlarına *alfa karıştırma* operasyonu da eklenebilir. Bu işlemlerde *kaynak* ve *hedef,* ekran verisini bulunduran çerçeve arabelleği üzerinde ya da kullanıcı adres uzayındaki herhangi bir alanda olabilir [7].

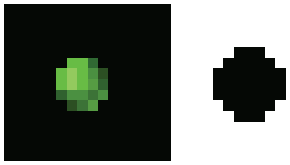
3.3.1 Maskeli BitBLT

BitBLT operasyonunun en temel kullanım yerlerinden birisi hareketli saydam nesnelerin (sprite) arka plan imajının üzerine yerleştirilmesidir. AND ve OR boolean operatörlerinin kullanıldığı iki BitBLT operasyonu ile gerçekleştirilen bu işleme *maskeli blit* adı verilmektedir. İki boyutlu grafik oluşturma sürecinde nesneler, pikselleştirme sonrasında dikdörtgensel bloklar halinde ifade edilirler. Bu blokların içerisindeki her nokta o nesnenin görüntülenmesi gereken kısmına genellikle ait olmaz. Örneğin Şekil 3.11'de görülen nesnenin ekran üzerinde görüntülenecek kısmının siyah olmayan bütün noktalar olduğunu kabul edelim.



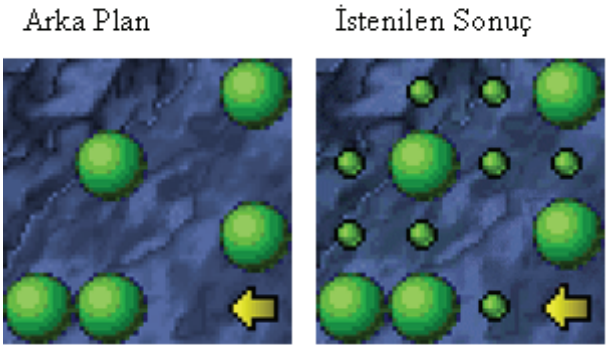
Şekil 3.11. Pikselleştirilmiş geometrik nesne

Bu nesneyi bir arka plan üzerine yapıştırırken bütün piksellerinin üzerinden geçip hangisinin çizilip çizilmeyeceğine karar vermek çok maliyetli bir çözüm olacaktır. Bu maliyetli çözüm yerine, maskeli blit yöntemi kullanılabilir. Bunun için ilgili nesnenin 1 bit (monokrom) maskesi oluşturulur.



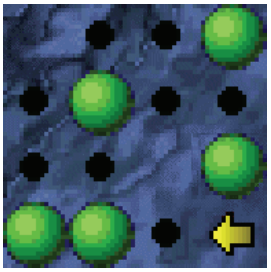
Şekil 3.12. Pikselleştirilmiş geometrik nesne ve maskesi

Maskede görüntülenecek bölge Şekil 3.12'de görüldüğü gibi siyah (RGB(0,0,0)), görüntülenmeyecek bölge ise beyaz (RGB(255,255,255)) olmalıdır. Çünkü maske arka plan verisi ile AND'leneceğinden beyaz kısım arka planla aynı kalacak, siyah kısım ise arka planda siyah bir delik oluşturacaktır. Maskenin monokrom olması bellekte kaplayacağı yer açısından daha mantıklıdır. Fakat bit BitBLT IP Çekirdeğinin tüm renkleri desteklemesi nedeniyle maskeler tüm RGB moda göre hazırlanmalıdır.



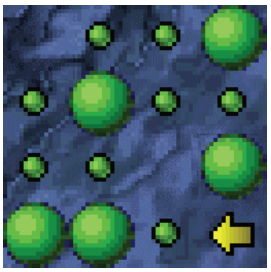
Şekil 3.13. Arka plan üzerine maskeli blit yardımı ile nesnenin yerleştirilmesi

Şekil 3.13'de örnek olarak bir arka plan ve Şekil 3.11'deki nesnenin yerleştirilmesi sonucu elde edilmek istenen görüntü görülmektedir. Birinci BitBLT'de arka plan maske ile Şekil 3.14'te görüldüğü gibi AND'lenir.



Şekil 3.14. Yerleştirilecek nesnenin maskesi ile AND'lenmiş arka plan görüntüsü

İkinci bit BLT'de ise orijinal nesne Şekil 3.15'teki gibi arka plan ile OR'lanır.



Şekil 3.15. Maske ile AND'lenen arka planın nesne ile OR'lanması ve son görünüm

3.4 Alfa Karıştırma (Alpha Composition)

Kompozit, iç içe geçmiş imajlar oluşturabilmek için grafik nesnelerinin renk ve geometrik biçim bilgilerinin yanı sıra yarı saydamlık ya da ters bakış açısı ile kaplama (coverage) oranı bilgisine ihtiyaç duyulur. Kompozit imaj oluşturabilmek için bir piksel üzerinde birden fazla nesnenin görüntülenme hakkına sahip olabilmesini sağlamak gerekir. Bunun için grafik nesnesinin her bir pikseli için bir kaplama ya da saydamlık bilgisi bulundurulmalıdır. Bu bilgi renk kanalları üzerinde saklanamayacağından RGB renk kanalları haricinde ayrı bir kanala ihtiyaç duyulur. İşte bu kanala *alfa kanalı* denmiştir. Aynı zamanda literatürde *matte kanalı* olarak da geçer. İlk olarak A.R. Smith tarafından 1970'de ortaya atılan bu yöntem daha sonra 1984'de Porter & Duff tarafından formülleştirilmiş ve geliştirilmiştir. RGB renk modeline alfa kanalının eklenmesiyle ARGB renk modeli elde edilir [19].

Bir nesnenin bir piksel verisinin kaplama oranını anlayabilmek için bir pikseli birim alana sahip bir dikdörtgen olarak düşünelim. Bu pikselin A renginde ve α kaplama oranına sahip bir nesne tarafından kaplanması demek, bu dikdörtgen pikselin $\%\ \alpha$ kadarının A rengine boyanması demektir. Bu durumda pikselin bütününün rengi αA olur. Daha sonra B rengine ve β kaplama oranına sahip bir nesnenin bu pikselin üzerine getirildiğini düşünürsek ($B\ over\ A$) yeni nesnenin bu piksele katılımı βB kadar olur. β kaplama oranına sahip olmak demek aynı zamanda ($1\ -\ \beta$) oranında geçirgenliğe sahip olmak demektir. Bu durumda

pikselin eski rengini (*1 - β*) oranında görebiliriz. Sonuç olarak pikselin son renk değeri şu şekilde hesaplanabilir:

$$(\beta B + (1 - \beta)\alpha A) \tag{3.13}$$

3.4.1 Önceden çarpılmış alfa (premultiplied alpha)

A renginde ve α kaplama oranına sahip nesnenin üzerine *B* renginde ve β kaplama oranına sahip nesnenin getirilmesi ile oluşan kompozit renk değeri (3.13)'te elde edilmişti. Bu ifade düzenlendiğinde (3.14)'teki kompozit renk değeri elde edilir.

$$C' = \beta B + \alpha A = \beta B + \alpha A - \beta\alpha A \tag{3.14}$$

Denklem (3.14)'te 3 adet çarpma işlemi vardır. Nesnenin her pikseli için düşünüldüğünde bu oldukça maliyetli bir işlemdir. Renk değeri alfa kanalı ile daha önceden çarpılmış olsaydı formül (3.15)'deki gibi olurdu.

$$= += + - \beta \tag{3.15}$$

Burada üssü ile gösterilmiş semboller alfa kanalı ile önceden çarpılmış renk değerleridir. Görüldüğü gibi bu durumda çarpma işlemi teke inmektedir. Yani nesnelerde pikselleştirme işlemi yaparken renk değerlerinin alfa ile çarpılarak oluşturulması avantaj sağlar. Buradaki tek problem pikselin gerçek renk değerinin elde edilmesi için alfa değerine bölme gereksinimidir. Tamsayı ile ifade edilen renk değerlerinde bu bilgi kaybı anlamına gelir. Fakat sonraki kısımda görüleceği gibi önceden alfa ile çarpılmamış renk değerlerinin yol açtığı "Alfa ile İkinci Kez Karıştırma" problemi daha maliyetlidir.

3.4.2 Alfa ile ikinci kez karıştırma problemi

Denklem (3.14) ve (3.15)'te görüldüğü gibi hem önceden çarpılmamış alfa değerlerine göre oluşturulan formülün hem de önceden çarpılmış alfa değerleri ile oluşturulan formülün sol tarafında , yani alfa ile çarpılmış renk değeri, bulunmaktadır. Çünkü önceden alfa ile çarpılmamış piksel değerlerini

karıştırdığımızda elde edilen sonuç alfa ile karıştırılmış piksel değeridir. Bu durumda yeni piksel değerinin üzerine gelen başka bir nesneye ait renk değerini karıştırmak istediğimizde aynı formülü kullanamayız. Yapılması gereken, gerçek renk değerlerini elde etmek için renk kanallarını alfa kanalına bölmektir. Bu da tamsayı değerlerde veri kaybı ve ekstra işlem maliyeti demektir. Bu nedenle bu tez çalışmasında BitBLT IP Çekirdeği, alfa karıştırma operasyonunda önceden çarpılmış renk değerleri ile çalışacak şekilde gerçeklenmiştir [20].

3.4.3 Alfa çarpımının yakınsak olarak gerçeklenmesi

ARGB ifadesinde A (Alfa), renk kanalları gibi 8 bit ile temsil edilir ve [0, 255] aralığındadır. Sıfır değeri tamamen geçirgen anlamına gelir ve piksel üzerinde herhangi bir kaplaması olmaz; 255 ise tamamen mat (opaque) olduğunu gösterir. Renk değerini Alfa ile orantılamak için ihtiyacımız olan ifade:

$$rA/255 \qquad r\text{: Renk } [0{,}255], \quad A\text{: Alfa } [0{,}255] \tag{3.16}$$

Denklem (3.16)'daki ifadeyi tam sayılarla elde edebilmek için bölme yapmadan, nümerik yakınsamalarla daha temel dijital ifadelere dönüştürmek gerekmektedir. Bunun için (3.17)'deki Mclaurin serisinden faydalanılabilir:

$$= a + ax + a + a + \ldots \qquad < 1 \tag{3.17}$$

O halde denklem (3.16), (3.18)'deki gibi ifade edilebilir:

$$\equiv , \quad t = \tag{3.18}$$

Yukarıdaki Mclauren serisine göre:

$$= 1 + t + + \cdots \tag{3.19}$$

O halde (3.19), yakınsamayla aşağıdaki gibi ifade edilebilir,

$$\equiv + \tag{3.20}$$

Dijital ifadeyle,

$$\equiv rA \gg 8 + rA \gg 16 \tag{3.21}$$

O halde, renk kanallarının alfa ile dijital yollarla çarpımı (3.22)'deki şekliyle efektif bir şekilde gerçeklenebilir.

($A_{\text{„}}$,

) (3.22)

BitBLT IP Çekirdeği bu yakınsak hesap yöntemini kullanarak hedef bit haritasındaki RGB değerlerini Alfa değeri ile çarpılmış hale getirebilmektedir.

4. GRAFİK SİSTEMİ TASARIMI

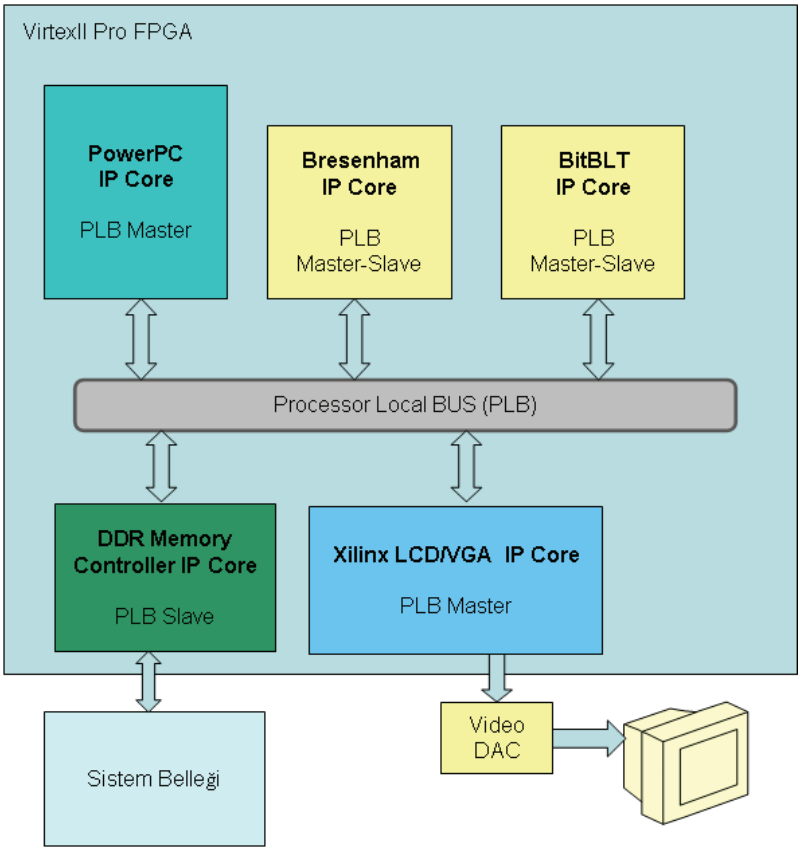
Hedeflenen grafik sistemi tasarımında, iki boyutlu grafik oluşum süreçleri FPGA yongasında gerçeklenmiştir. Tasarımda tek yongada sistem (SOC) oluşturma prensipleri gözetilmiştir. Bu bağlamda, FPGA yongası içerisinde işlemci çekirdeği ve buna bağlı çevresel birimler oluşturulmuştur. Grafik sistemi tasarımı, donanım ve yazılım olmak üzere iki ana başlık altında düşünülebilir.

Donanım tasarımında, bazı grafik algoritmalarını çalıştırmakla görevli IP Çekirdekleri oluşturulmuştur. FPGA yongasında, grafik ve işlemci IP Çekirdeklerinin veri yolları ile birbirine bağlanması ve işlemcide çalışan yazılım içerisinden grafik birimlerine erişim sağlanmıştır. Yazılım tasarımında ise yonga içerisindeki işlemcide çalışacak olan, grafik birimlerini sürmekle görevli sürücü fonksiyonları ve grafik API'leri oluşturulmuştur.

Pikselleştirme ve pikselleştirme sonrası grafik oluşum aşamaları FPGA içerisinde çeşitli çevresel birimler tarafından gerçeklenir. Bresenham'ın doğru çizme algoritmaları VHDL dilinde gerçeklenerek Bresenham IP Çekirdeği oluşturulmuştur. Pikselleştirme sonrası grafik oluşum aşamalarından BitBLT ve Alfa Karıştırma operasyonları ve son olarak görüntüleme cihazı üzerinde imajın oluşturulması PLB'ye bağlı çevresel birimler tarafından gerçeklenir. Bu çevresel birimler sistemin adres uzayında kendi adres alanlarına sahiptir. Adreslenmiş yazmaçları aracılığı ile çevresel birimlere komutlar gönderilir ve durum bilgileri de bu yazmaçlar aracılığı ile gözlenebilir. Sistemin topolojisi Şekil 4.1'de görülmektedir.

Grafik oluşturma sürecinde pikselleştirme ve pikselleştirme sonrası operasyonlara göre daha az maliyetli aşamalar olan geometrik modelleme, model dönüşümleri ve görüntü dönüşümleri işlemci içerisinde gerçeklenebilir. Kullanılan FPGA modeli olan Virtex 2 Pro'da, PowerPC IP Çekirdeği gömülü olarak bulunmaktadır [14]. PowerPC IP Çekirdeği, diğer çevresel birimlere Xilinx EDK ile birlikte sunulan PLB (Processor Local Bus) IP Çekirdeği aracılığı ile komutlarını gönderir. Yine EDK'nın sunduğu arayüzlerden olan PLB IPIF (IP Interface) aracılığı ile çevresel birimler birbirleriyle haberleşebilir ve işlemciye kesme (interrupt) gönderebilirler.

Çevresel birimler PLB ile Xilinx firmasının EDK ile birlikte sunduğu PLB IPIF aracılığı ile haberleşir. PLB IPIF, kullanıcı IP Çekirdekleri ile PLB 64 Bit Standardı arasında çift yönlü bir ara yüz sağlar ve aynı zamanda kullanıcı kodlarını veri yolu mimarisinden izole eder. PLB tarafında master ve slave arayüzlerini gerçekleyen PLB IPIF geri uçta IP Çekirdekleri için Xilinx IPIC (IP Interconnect) arayüzünü sağlar. Xilinx IPIC arayüzü kullanıcıya master ve slave arayüzleri ile birlikte PLB IPIF'ın sunduğu bazı servislerden faydalanmak için standart sinyaller sunar. Sunulan bu standart sinyaller sayesinde IP Çekirdekleri OPB'ye hiçbir değişiklik yapmadan taşınabilir [18]. PLB, PLB IPIF ve IP Çekirdeklerinin PLB'ye iliştirilmesi sonraki bölümlerde anlatılmıştır.



Şekil 4.1. Grafik sisteminin topolojisi

4.1 Processor Local Bus (PLB)

PLB veri yolu mimarisi IBM firmasının geliştirdiği Tek-Yongada-Sistem (System-On-a-Chip (SOC)) mimarisi olan CoreConnect standardının bir parçasıdır. CoreConnect mimarisi, yongadaki çekirdekler, makro elemanlar ve özel amaçlı mantık blokları (custom logic) arasında veri transferinin gerçekleştirilebilmesi için üç farklı veri yolu tipi sunmaktadır:

- Processor Local Bus (PLB)
- On-Chip Peripheral Bus (OPB)
- Device Control Register (DCR) Bus

PLB veri yolu, işlemci çekirdeği, harici bellek arayüzü ve DMA gibi yüksek performans gerektiren elamanların yonga içerisinde haberleşmesi için tasarlanmıştır. OPB ise yavaş çevresel birimler için daha uygundur. Yüksek performans gerektirmeyen çevresel birimlerin OPB veri yoluna bağlanması ile PLB üzerindeki trafiğin azaltılması sağlanır. DCR veri yolu işlemci, işlemci yazmaçları ve slave mantık blokları arasındaki veri transferlerinin gerçekleştirilmesi için geliştirilmiştir [17].

Bu tez çalışmasında FPGA yongası içerisinde gerçeklenen grafik sisteminin bütün birimleri PLB veri yoluna iliştirilmiştir. Çünkü bu çalışmada tasarlanan ve hazır olarak kullanılan bütün IP Çekirdeklerinin sistem belleğine erişimi gerekmektedir. Tasarlanan IP Çekirdeklerinde gerçeklenen grafik algoritmalarının çıktılarını çerçeve arabelleği üzerinde görebilmek, dolayısıyla sistem belleğine hızlı bir şekilde erişebilmek için yüksek hızlı ve az gecikmeli (low-latency) PLB veri yolu tercih edilmiştir.

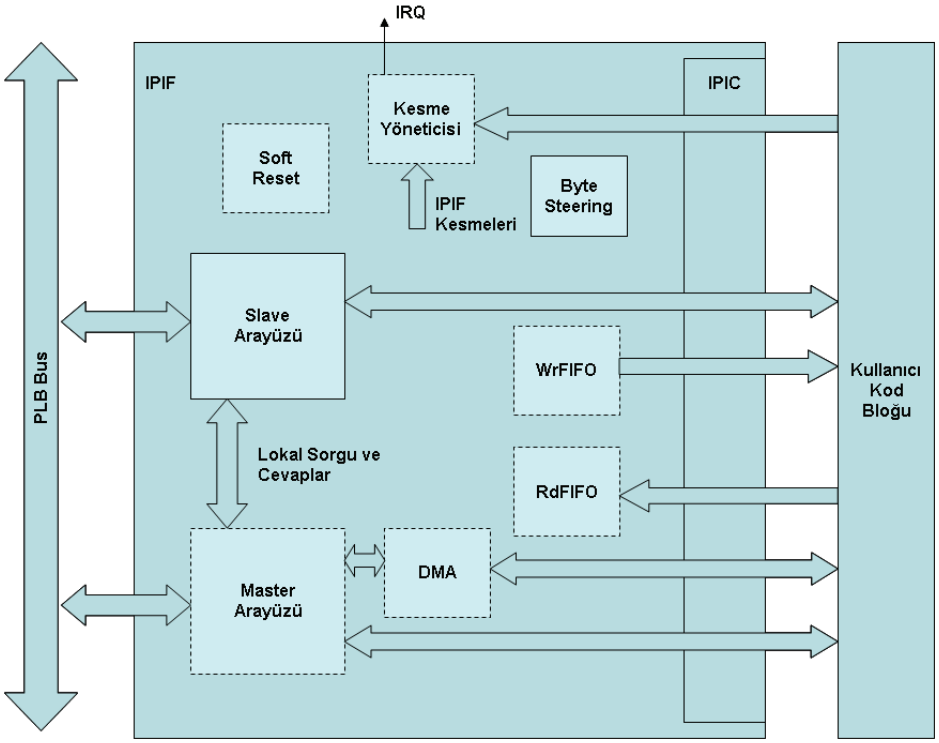
PLB mimarisinin birden fazla master ve slave iliştirilmeye (master-slave atachment) uygun olması, bayt seçme (byte enable) opsiyonu ile farklı veri uzunluklarını (8,16, … , 64 bit) destekleyen birimlerin sistemde yer alabilmesine olanak sağlaması, eşzamanlı yazma okuma işlemini desteklemesi ve kilitlenmeden kaçınma (deadlock avoidance) özelliği grafik sisteminin tasarlanmasında kolaylık sağlamıştır. Çoklu master ilişiği sayesinde, PowerPC IP Çekirdeği ile birlikte birden fazla grafik IP Çekirdeğinin master olarak tasarlanabilmesi sağlanmıştır. PowerPC IP Çekirdeği de dahil olmak üzere 32 bitlik verilerle çalışabilen

sistemdeki IP Çekirdekleri, bayt seçme opsiyonu ile PLB veri yoluna iliştirilebilmiştir.

Grafik algoritmaları VHDL dilinde veri yolu mimarisinden bağımsız olarak gerçeklenmiştir ve FPGA yongasında test edilmeden önce ModelSim simülatör programında test edilmiştir. IP Çekirdekleri içerisinde kullanıcı-özel amaçlı mantık bloğu (user-custom logic) olarak nitelendirebileceğimiz bu HDL kod blokları, farklı veri yollarını destekleyen IP Çekirdekleri içerisine taşınabilir. IP Çekirdeklerini PLB veri yoluna iliştirmek için tasarlanan kod blokları, algoritmalardan bağımsız olarak gerçeklenmiş, bu sayede tasarımın yenilenebilir olması sağlanmıştır. IP Çekirdeklerinin PLB veri yolu üzerinden haberleşmelerinin sağlanması sonraki kısımda anlatılmıştır.

4.2 IP Çekirdeklerinin Veri Yoluna Bağlanması

Tasarlanan grafik sistemini FPGA yongasında gerçeklemek için EDK ortamı kullanılmıştır. EDK ortamını kullanarak Virtex II FPGA içerisinde gömülü olan PowerPC IP Çekirdeği, Xilinx firması tarafından geliştirilen ücretsiz IP Çekirdekleri ve bu çalışmada gerçeklenen grafik IP Çekirdeklerinin PLB veri yoluna bağlanması mümkün olmuştur. EDK, geliştirilen HDL kod bloklarını IP Çekirdeği halinde, sistemdeki veri yoluna bağlı çevresel birimler olarak kullanabilmek için PLB IPIF (PLB IP Interface) adında bir standart sunmaktadır. PLB IPIF, PLB veri yolu ile kullanıcı IP Çekirdekleri arasında bir arayüz sunmaktadır. PLB IPIF'ın kullanıcı kodu ile arayüzü oluşturan sinyal tanımlamaları da ayrıca Xilinx IPIC (IP Interconnect) adında bir standart altında toplanmıştır. IPIC sinyal tanımlamaları OPB IPIF'ta da aynı olduğundan, kullanıcı kodları değişiklik yapılmaksızın PLB ya da OPB ile kullanılabilmektedir. Kullanıcının, IPIC sinyallerini sürerek PLB veri yoluna slave, ya da master-slave olarak bağlanabilmesi sağlanmıştır. IPIF aynı zamanda kullanıcıya DMA, FIFO, Soft Reset ve Kesme Yönetimi (Interrupt Controller) gibi bazı servisler sunmaktadır. Kullanıcı, yine IPIC sinyalleri aracılığı ile bu servisleri kullanabilir. Şekil 4.2'de PLB IPIF'ın blok diagramı görülmektedir. Burada kesikli çizgi ile çerçevelenmiş bloklar opsiyonel bloklardır [18].



Şekil 4.2. PLB IPIF blok diyagramı [18]

PLB IPIF arayüzü ve servisleri, VHDL *generic* tanımlamaları ile ihtiyaca göre özelleştirilebilir. EDK'nın "Create-Import Peripheral" adındaki arayüzü ile kullanılacak servisler ve IPIC sinyalleri seçilebilmektedir. Bu arayüz, gereken seçimler yapıldıktan sonra iki adet VHDL dosyası oluşturur. Örneğin eklenecek IP Çekirdeğinin ismi *MyIPCore* ise oluşturulan dosyaların isimleri "*MyIPCore.vhd"* ve "*UserLogic.vhd"* olacaktır. "*MyIPCore.vhd"* dosyası içerisinde, seçilen servislere göre oluşturulmuş PLB IPIF kodları bulunur. "*UserLogic.vhd"* ise *MyIPCore* biriminin altında yer alan, *UserLogic* adındaki alt birimin şablonunu içerir. Kullanıcı kodları bu dosya içerisine eklenir. Daha sonra yine aynı EDK arayüzü ile oluşturulan IP Çekirdeği sisteme eklenir. Eklenen IP Çekirdeği sistemin adres uzayı içerisinde bir adres alanına sahip olur. Aynı zamanda, CPU'dan IP Çekirdeğinin adres alanına veri yazma-okuma yapabilmeyi sağlayacak C dilindeki sürücü fonksiyonları da oluşturulmuş olur [22].

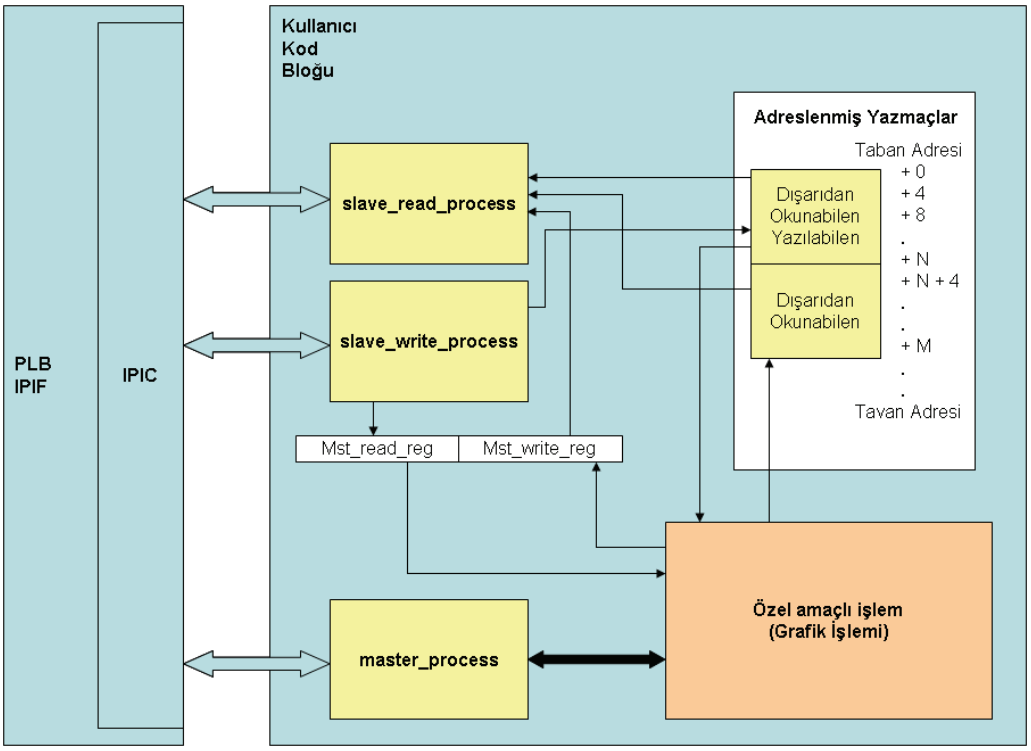
Grafik sisteminde gerçeklenen grafik IP Çekirdeklerinin hepsi hem slave hem de master'dır. Slave arayüzü ile CPU tarafından erişilebilmeleri, master arayüzü ile de IP Çekirdeklerinin sistem belleğine erişebilmeleri sağlanmıştır.

PLB IPIF master-slave arayüzü ile kullanıcı kodunun ilişiğinin kurulabilmesi için üç adet VHDL *process* (süreç) gerçeklenmiştir:

- *slave_read_process*
- *slave_write_process*
- *master_process*

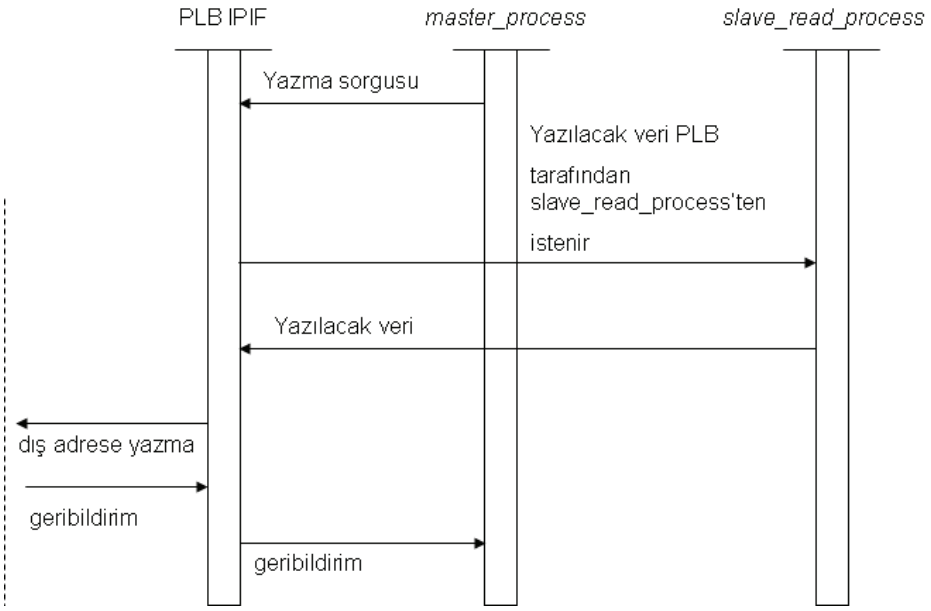
slave_read_process ve *slave_write_process* kodları, ilgili sinyal tanımlamaları ile birlikte, slave olarak çalışması istenen IP Çekirdeklerinin kullanıcı koduna eklenmiştir ve bu sayede bu IP Çekirdekleri diğer IP Çekirdekleri tarafından erişilebilir hale getirilmiştir. Bu nedenle bu iki VHDL işlemi, sistemdeki grafik IP Çekirdeklerinde yer almaktadır. Bu VHDL işlemlerinin görevi aynı zamanda IP Çekirdeğinde adres haritalı yazmaçlar oluşturmaktır. Yazmaçlar kod içerisinde tanımlanan 32 bitlik sinyal dizileridir. Dizilerin boyutu kod içerisindeki sabitler değiştirilerek ayarlanabilir. Bu yazmaçlar sayesinde IP Çekirdeklerine CPU tarafından kontrol edilebilme özelliği kazandırılmıştır. *slave_read_process*, PLB'den gelen okuma isteklerini karşılar ve yazmaç bilgilerinin diğer birimlere aktarılabilmesini sağlar. *slave_write_process* ise PLB'den gelen yazma isteklerini karşılar ve yazmaçlara diğer birimlerden veri girilebilmesini sağlar.

İki tip yazmaç tasarlanmıştır. Yazmaçların bir kısmı CPU tarafından hem okunabilir hem yazılabilir, IP Çekirdeği içerisinde ise sadece okunabilir yazmaçlardır. CPU tarafından komutlar bu yazmaçlar aracılığı ile gönderilir. Yazmaçların diğer kısmı ise IP Çekirdeği içerisinde hem okunabilir hem yazılabilir ancak CPU tarafından sadece okunabilir yazmaçlardır. Bu yazmaçlar, IP Çekirdeğinin durum bilgilerinin CPU tarafından gözlenebilmesi için gereklidir. Yazmaçlarda bu ayrımın yapılmasının sebebi, bir sinyalin birden fazla kaynak tarafından değiştirilebilmesinin mümkün olmamasıdır. Şekil 4.3'te yazmaçların işlevi ve kaynakları görülmektedir. Bu şekilde görüldüğü gibi dışarıdan okunabilen ve yazılabilen yazmaçlara sadece *slave_write_proces,* dışarıdan sadece okunabilen yazmaçlara özel amaçlı grafik işlemi atama yapabilmektedir. Grafik işlemleri için hazırlanan API'lerde bu yazmaçların adresleri tanımlanmıştır. Aynı tanımlamalar IP Çekirdeği içerisinde de tanımlanarak grafik IP Çekirdeğinin arayüzleri oluşturulmuştur.



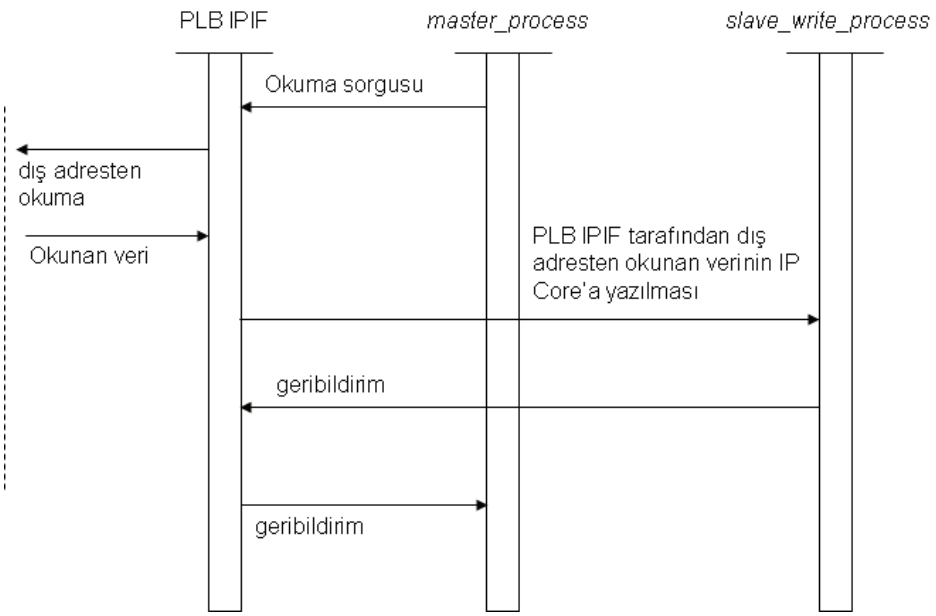
Şekil 4.3. Tasarlanan IP Çekirdeklerinde ortak olan master-slave işlemlerinin görevi ve yazmaçların kaynak ve işlevleri

PLB IPIF'ın master arayüzü tek başına kullanılamaz. Master arayüzü kullanılacaksa slave arayüzü de kullanılmak zorundadır. Dolayısıyla *master_process* ile birlikte *slave_read_process* ve *slave_write_process* de eklenmelidir. IP Çekirdeği içerisinde master read ve write operasyonları dolaylı olarak slave arayüzü ile birlikte yapılmaktadır. IP Çekirdeği içerisinde, PLB IPIF arayüzü kullanılarak yazma ya da okuma yapılacağı zaman, kaynak ve hedef adresler belirtilir. Örneğin kendi adres alanımız dışındaki bir adrese yazma işlemi yapacağımız zaman, kaynak kendi adres alanımız içerisinde olmak durumundadır. Kaynağın adresi, bir IPIC sinyali olan *IP2IP_Addr* portu ile PLB IPIF'a verilir. Hedef adres ise *Bus2IP_Addr* portu ile belirtilir. Kaynağın adresi, hedefin adresi ve yazma komutu verildikten sonra, PLB IPIF *IP2IP_Addr* ile verilen kaynak adresinden okuma yapar. Okumayı yaparken IP Çekirdeğinin slave arayüzü kullanılmış olur. Okumayı yaptıktan sonra hedef adrese yazar ve master arayüzüne geribildirim gönderir. Bu süreç Şekil 4.4'de gösterilmiştir [18].



Şekil 4.4. IP Çekirdeği içerisinde master arayüzü ile dışarıdaki adrese veri yazma

Kendi adres alanımız dışındaki bir adresten okuma yapacağımız zaman yine kaynak ve hedef adres belirtilir. Bu sefer kaynak adres dış alandaki adrestir (*Bus2IP_Addr*). Hedef ise IP Çekirdeğinin kendi adres alanı içerisinde olmalıdır (*IP2IP_Addr*). Okuma komutu verildikten sonra PLB IPIF kaynak adresten okumayı yapar ve IP Çekirdeğinin slave arayüzünü kullanarak *IP2IP_Addr* adresine veriyi yazar. Yazma işlemi tamamlandıktan sonra master arayüzüne geribildirim yapılır. Bu süreç de Şekil 4.5'de gösterilmiştir.



Şekil 4.5. IP Çekirdeği içerisinde master arayüzü ile dışarıdaki adresten veri okuma

4.3 Bresenham IP Çekirdeği

Bölüm 4.2'de anlatılan, grafik IP Çekirdeklerinin PLB'ye iliştirilmesi ile ilgili tasarım, Bresenham IP Çekirdeği için de geçerlidir. Bresenham IP Çekirdeğinin PLB IPIF ile haberleşme şekli ve adres haritalı yazmaçların oluşturulması ile ilgili ayrıntılar Bölüm 4.2'de anlatılmıştır. Bresenham IP Çekirdeğinin blok diagramı Şekil 4.7'de görüldüğü gibidir. Bu şekilde de görüldüğü gibi, adres haritalı yazmaçlar Bresenham IP Çekirdeğinin girdileri ve çıktıları için kullanılmıştır.

Bölüm 3,1'de anlatılan ve Şekil 3.6'da gösterilen Bresenham Algoritması bütün doğrular için genelleştirilmemiştir. Eğer ilklendirme aşamasında majör eksenin hangisi olduğu ve minör eksen üzerinde hangi yönde gidildiği, yani majör eksene göre eğimin işareti belirlenirse algoritma bütün doğrular için geçerli hale getirilebilir. Şekil 4.6'da genelleştirilmiş Bresenham Algoritması gösterilmektedir.

```
function DogruCiz (major_eksen_ilk, minor_eksen_ilk,
                   major_eksen_son, hata_major_shifted,
                   hata_minor_shifted, negatif_yon, major_y)

int hata := 0
int arttirimYonu := 1

if negatif_yon = TRUE then
        arttirimYonu := -1

while major_eksen_ilk <= major_eksen_son

        if major_y = TRUE then
                PikselBoya(minor_eksen_ilk, major_eksen_ilk)
        else
                PikselBoya(major_eksen_ilk, minor_eksen_ilk)

        if(hata >= 0)
                minor_eksen_ilk := minor_eksen_ilk + arttirimYonu
                hata := hata - hata_major_shifted

        hata := hata + hata_minor_shifted
        major_eksen_ilk := major_eksen_ilk + 1
```

Şekil 4.6. Geliştirilmiş Bresenham Algoritması'nın bütün doğrular için tanımlanmış hali

Bütün doğrular için yeniden düzenlenen Şekil 4.6'daki algoritmanın parametreleri şu şekilde belirlenmiştir:

- **major_eksen_ilk**: Çizilecek doğrunun, majör eksen bileşeni küçük olan uç noktasının majör eksen bileşenidir.
- **minor_eksen_ilk**: Çizilecek doğrunun, majör eksen bileşeni küçük olan uç noktasının minör eksen bileşenidir.
- **major_eksen_son:** Çizilecek doğrunun, majör eksen bileşeni büyük olan uç noktasının majör eksen bileşenidir.
- **hata_major_shifted:** Çizilecek doğrunun majör eksen üzerindeki değişim miktarının iki katıdır ((M2 – M1) << 1).
- **hata_minor_shifted:** Çizilecek doğrunun minör eksen üzerindeki değişim miktarının iki katıdır ((m2 – m1) << 1).
- **negatif_yon:** Değeri sıfırdan farklı ise pozitif majör eksen yönünde ilerlerken minör eksen üzerindeki gidişin negatif yönde olduğunu belirtir.

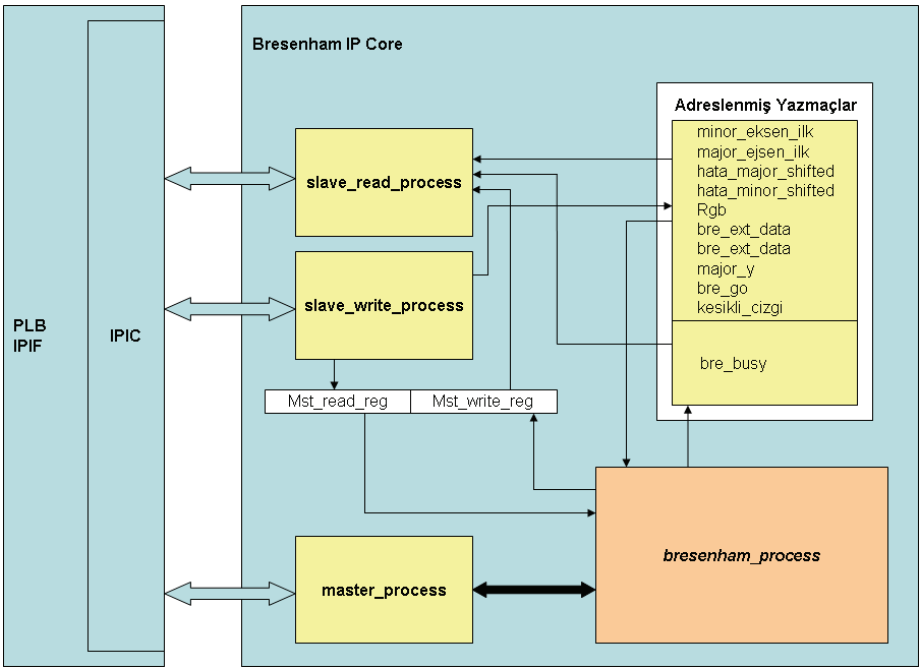
- **major_y:** Değeri sıfırdan farklı ise majör eksenin y ekseni olduğunu belirtir.

Yukarıda listelenen algoritma parametreleri aynı zamanda Bresenham IP Çekirdeğinin girdileridir. IP Çekirdeğinin, çizilecek doğru ile ilgili bilgileri belirten bu girdiler haricinde başka girdileri de vardır:

- **rgb:** Çizilecek doğrunun renk değeridir.
- **bre_go:** Bresenham IP Çekirdeğini tetikleyen girdidir.
- **kesikli_cizgi**: Kesikli çizgi bayrağıdır.
- **bre_ext_data:** Bu girdi kesikli çizgi opsiyonu aktifken kaç piksel aralıklarla kesikli çizgi oluşturulacağını belirtir. Bunun haricinde ileride eklenecek opsiyonlar için ayrılmıştır.

Bresenham IP Çekirdeğinin Çıktıları:

- **bre_busy:** Bresenham IP Çekirdeğinin meşgul olup olmadığını gösterir.

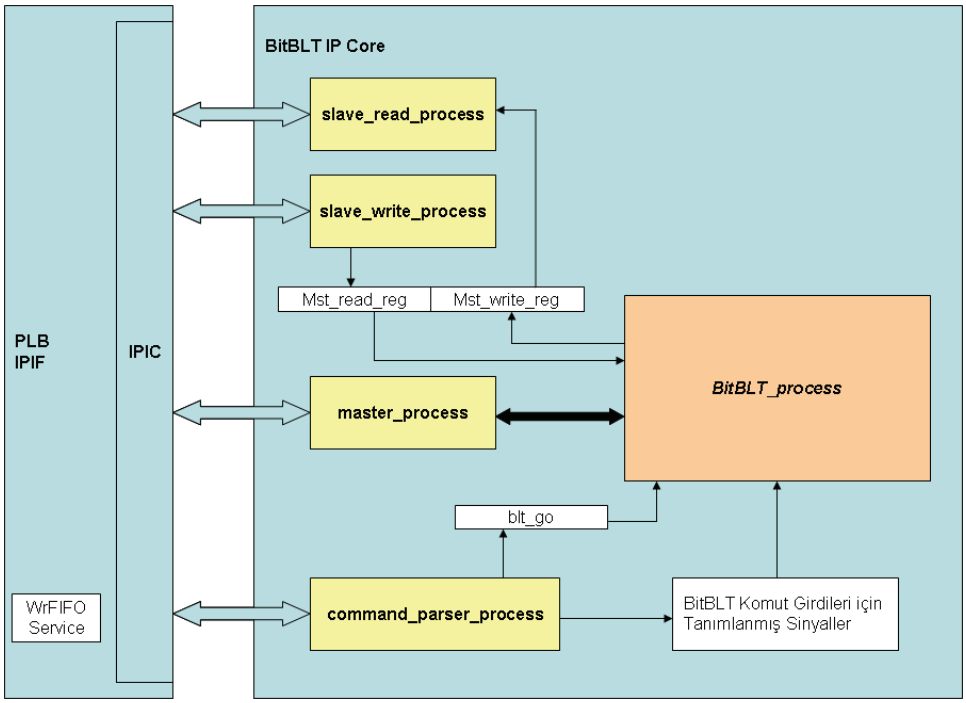


Şekil 4.7. Bresenham IP Çekirdeğinin blok diyagramı

4.4 BitBLT IP Çekirdeği

Pikseller üzerinde yapılacak işlemlerin yükünün fazla oluşu, BitBLT operasyonu için harici bir çevresel birim gerçeklemeyi zorunlu kılmaktadır. [12] BitBLT operasyonunun iyi optimize edilmiş, hızlı birimler tarafından gerçekleştirilmesi grafik sistemi için kritiktir [8]. Bu nedenle BitBLT işlemi, tez çalışmasında gerçeklenen grafik sisteminde yer alan BitBLT IP Çekirdeği aracılığı ile yapılmaktadır. BitBLT IP Çekirdeğinin görevi ve yapabildiği BitBLT operasyonlarının ayrıntıları sonraki kısımda anlatılmıştır.

BitBLT IP Çekirdeği PLB'ye hem master hem de slave arayüzü ile bağlıdır. PLB Slave arayüzü, master arayüzünün slave arayüzü ile birlikte çalışma zorunluluğundan dolayı eklenmiştir. Bu IP Çekirdeğinin slave arayüzü CPU tarafından kullanılmamaktadır. Çünkü komutlar PLB IPIF'ın sağladığı WrFIFO (Write First-In-First-Out) arabelleği ile IP Çekirdeğine iletilmektedir. PLB Master arayüzü ise BitBLT'nin sistem belleğine erişebilmesi için gereklidir. Çünkü görüntülenen piksel verilerinin okunup yazıldığı bölge olan çerçeve arabelleği, sistem belleğindeki herhangi bir bölgede ya da sistemde var olan başka bir çevresel birimin adres uzayı içerisinde tanımlı olabilir. BitBLT IP Çekirdeği için çerçeve arabelleğinin yer aldığı bölge önemli değildir, yani sistem adres uzayı içerisinde olması kâfidir. Bölüm 4.2'de anlatılan, grafik IP Çekirdeklerinin PLB'ye iliştirilmesi ile ilgili tasarım, kısmen BitBLT IP Çekirdeği için de geçerlidir. Farklı olarak adres haritalı yazmaçlar kullanılmamıştır. BitBLT IP Çekirdeğinin PLB IPIF master arayüzü ile haberleşme şekli ile ilgili ayrıntılar için Bölüm 4.2'ye bakılabilir. BitBLT IP Çekirdeğinin blok diyagramı Şekil 4.8'de görülmektedir.



Şekil 4.8. BitBLT IP Çekirdeğinin blok diyagramı

4.4.1 BitBLT komutlarının alınışı

Blok diyagramda (Şekil 4.8) görülen *command_parser_process* işleminin görevi, CPU tarafından gönderilen BitBLT komutlarını WrFIFO'dan alıp *BitBLT_proces*'i tetiklemektir. Bu tasarımda komutlar için FIFO belleğin kullanılmasının sebebi, CPU'nun BitBLT operasyonlarının bitişini beklemeksizin ardı ardına BitBLT komutlarını gönderebilmesini sağlamaktır. İleriki bölümde anlatılan satır tarama yöntemiyle poligon boyama API'sinin tasarlanışında bu özelliğin getirdiği avantajlar vurgulanmıştır. Bu API, poligonların satırlarının taranması işlemi için BitBLT IP Çekirdeği kullanmaktadır. Taranacak satır aralığını hesaplayan API fonksiyonu BitBLT IP Çekirdeğine tarama komutunu verdikten sonra beklemeden bir sonraki satırın aralığını hesaplayıp tarama komutunu ardı ardına göndermektedir. Bu sayede işlemci zamanının az kullanımı açısından avantaj sağlanmıştır.

command_parser_process, bir sonlu durum makinesi (FSM) çalıştırmaktadır. Bu FSM, etkin olmayan durumda (IDLE), tanımlı komut başlığını bekler. Eğer komut başlığı WrFIFO'dan okunursa ardından komut

cümlelerini okumak için oluşturulmuş durumlara geçer ve bu durumlar içerisinde komutlar ilgili sinyallere atanır. Son komut cümlesi alındığında *blt_go* sinyali tetiklenerek *BitBLT_proces* çalıştırılır. Komutların yapısı sonraki kısımda, CPU'dan gönderiliş şekli ise BitBLT sürücü fonksiyonları bölümünde anlatılmıştır.

4.4.2 BitBLT operasyonları ve komutlar

Bu tez çalışmasında tasarlanan BitBLT IP Çekirdeği, kaynak ve hedef arasında Boolean işlemleri yapma, kaynağı hedefe taşıma, kaynak ile hedef arasında alfa karıştırma, hedefi belirli bir renkle doldurma gibi standart BitBLT operasyonları ile birlikte saydamlık, kaynak güncelleme ve dört farklı BitBLT yönü gibi pratik opsiyonlar sunmaktadır.

Saydamlık seçeneği sayesinde bir bit bloğu içerisindeki tüm noktaların, nesneye ait olmadığı durumlarda taşıma ve Booelan işlemlerinin kolaylıkla yapılması sağlanır. Örneğin 'A' harfinin yer aldığı bir bit haritasında harfin iç kısmı geçirgen ise ve taşındığı yerdeki arka planın harfin içinden görünmesi isteniyorsa saydamlık özelliği kullanılabilir. Bunun için kaynak bit haritası üzerinde yok sayılacak ya da var sayılacak tek renk değerinin verilmesi ve saydamlık opsiyonunun seçilmiş olması yeterli olacaktır.

Arka plan güncelleme opsiyonu, tez çalışmasında incelenen BitBLT IP Çekirdeklerinde olmayan bir özelliktir [8, 9, 12]. Bu özellik sayesinde tek BitBLT komutu ile arka plan üzerinde hareketli nesne dolaştırma işlemi mümkün kılınmıştır. Bir kaynak bit haritasının kendisiyle çakışan bir hedef bit haritası üzerine götürülmesi (taşıma, alfa ya da Boolean işlemleri) sırasında, aynı anda kaynağın da arka plan piksel değerleri ile güncellenmesi gerçekleştirilmiştir. Bu sayede hareketli nesne işlemlerinde kaynağı güncellemek için ekstra işlem yapma zorunluluğu ortadan kalkmıştır.

Bit haritası taşıma işleminde bit haritasının neresinden başlanacağı ve hangi yönde ilerleneceği önemlidir. Çünkü hedef ile kaynağın bellek alanlarının çakışması durumunda kaynaktaki verilerin üzerine yazılmaması için yön opsiyonları gerekmektedir. Bu nedenle kaynak bit haritasının başlangıç köşesi IP Çekirdeğinde bir girdi olarak tanımlanmıştır.

BitBLT IP Çekirdeği ile 19 farklı tipte, 184 farklı kombinasyonda BitBLT operasyonu gerçekleştirilebilmektedir. Bu BitBLT operasyonlarının ayrıntıları, IP Çekirdeği girdilerinin operasyonlar üzerindeki etkileri ve bu operasyonlarda BitBLT FSM'nin durum geçişleri Şekil 4.9'da görülmektedir.

4.4.3 BitBLT girdileri

BitBLT komutunun parametreleri şu şekilde belirlenmiştir:

- **argb (32 bit):** Hedef bit haritasının sabit renkle boyanması, kaynağın sabit renkle güncellenmesi, transparan kaynak operasyonunda yok sayılacak ya da dikkate alınacak renk değerinin belirtilmesi için kullanılan girdidir.
- **kaynak_addr (32 bit):** Kaynak bit haritasının adresidir.
- **hedef_addr (32 bit):** Hedef bit haritasının adresidir.
- **arka_plan_addr (32 bit):** Kaynağın arka plan ile güncellemesi durumunda kullanılacak arka plan adresidir.
- **frm_buf_row_size (32 bit):** Bit haritalarının bulunduğu çerçevenin bir satırının kaç piksel içerdiğini belirtir. Bu bilgi, bit haritasında bir sonraki satıra geçerken kaç bayt atlama yapılması gerektiğini hesaplamak için IP Çekirdeği tarafından kullanılır.
- **blt_row_size (32 bit):** Kaynak ve hedef bit haritalarının piksel cinsinden satır uzunluğudur.
- **blt_col_size (32 bit):** Kaynak ve hedef bit haritalarının piksel cinsinden sütun uzunluğudur.
- **kaynak (1 bit):** '0' olması durumunda kaynağın *argb*, '1' olması durumunda ise kaynağın *kaynak_addr*'deki bit haritası olduğunu belirtir.
- **bool (2 bit):** "00" ise üzerine yazma, "01" ise AND, "10" ise OR ve "11" ise XOR operasyonu olduğunu belirtir. *kaynak=0* olduğunda anlamsızdır.
- **transparan (2 bit):** "00" ise kaynağın transparan olmadığını, "01" ise kaynağın transparan olduğunu ve sadece *argb*'de verilen renk

değerinin dikkate alınacağını, “10” ise kaynağın transparan olduğunu ve *argb*’de verilen renk değerindeki piksellerin dikkate alınmayacağını belirtir. *kaynak=0* olduğunda anlamsızdır.

- **alpha (1 bit):** ‘1’ olduğunda kaynağın hedef üzerine getirildiği alfa karıştırma işleminin (*kaynak* ***over*** *hedef*) yapılacağını belirtir. *kaynak=0* veya *bool=“00”* olduğunda anlamsızdır.
- **kose_no (2 bit):** Bu girdi BitBLT operasyonunun yönünü belirtir. Yönler:
  - “00” ise +X, –Y (→, ↑)
  - “01” ise –X, –Y (←, ↑)
  - “10” ise –X, +Y (←, ↓)
  - “11” ise +X, +Y (→, ↓)
- **kaynak_guncelle (2 bit):** “00” ise kaynağın güncellenmeyeceğini, “01” ise kaynağın *argb*’deki renk ile güncelleneceğini, “10” ise kaynağın *arka_plan_addr*’deki renk değeri ile güncelleneceğini belirtir.

4.4.4 BitBLT FSM

Şekil 4.8’deki blok diyagramda görülen *BitBLT_process,* bir sonlu durum makinesinden (FSM) ibarettir. Bu sonlu durum makinesindeki durumlarda yapılan işlemlerin ayrıntıları aşağıdadır. Durum geçiş koşulları ise Şekil 4.10’da görülmektedir.

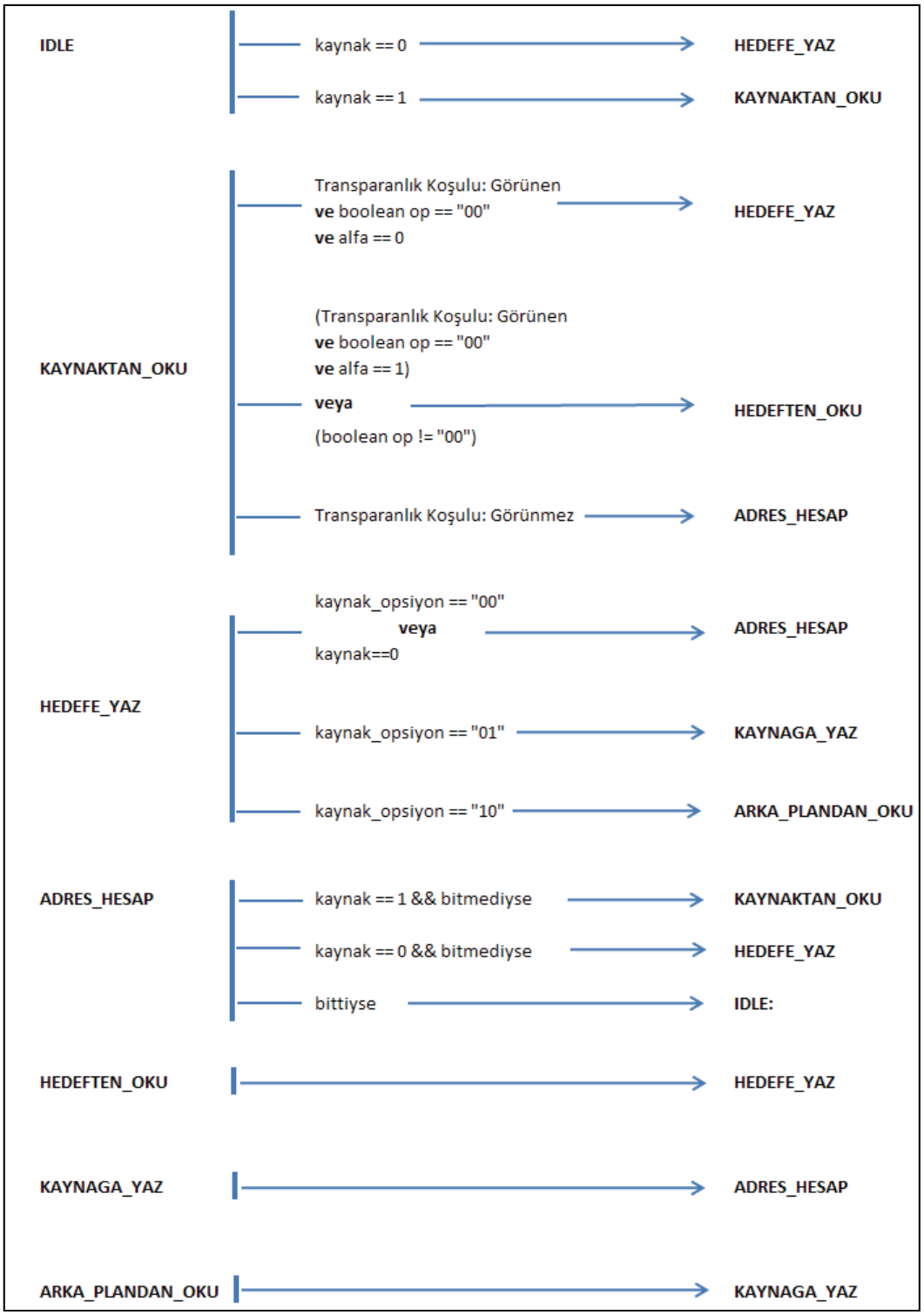
Durumlar:

- **IDLE:** Bu, FSM’nin başlangıç durumudur. Bu durumda iken blok diyagramda da görülen *command_parser_process* tarafından BitBLT komutu (blt_go sinyali aracılığıyla) beklenir. Komut geldiğinde *command_parser_process*’in atadığı sinyaller aracılığı ile komut bilgileri kopyalanır.

- **ADRES_HESAP:** Bu durumda kaynağın, hedefin ve arka planın bir sonraki piksellerinin adresleri hesaplanır. BitBLT operasyonunun bitip bitmediği bu durumda belirlenir. Bitmişse IDLE durumuna dönülür.
- **KAYNAKTAN_OKU:** Kaynak bit haritasının sıradaki pikselinin değeri okunur. Transparanlık koşuluna göre yok sayılacaksa bir sonraki piksel adreslerinin hesaplanması için ADRES_HESAP durumuna geçilir.
- **HEDEFE_YAZ:** BitBLT operasyonunun tipine göre *argb,* kaynağın kendisi, kaynak ile hedef arasında Boolean işlemiyle elde edilen değer ya da alfa karıştırma ile elde edilen değer hedefin sıradaki pikseline yazılır.
- **HEDEFTEN_OKU:** Hedefin sıradaki pikselinin değeri okunur.
- **KAYNAGA_YAZ:** Kaynağın sıradaki pikseline *argb*, ya da arka plandan okunan değer yazılır.
- **ARKA_PLANDAN_OKU:** Arka planın sıradaki pikselinin değeri okunur.

	Girdiler								
Tip No:	**Köşe No**	**Kaynak Opsiyo**	**Alfa**	**Trans-paranlık**	**Boolean Op.**	**Kaynak**	**OPERASYON TİPİ**	**DURUM GEÇİŞLERİ**	Operasyon Sayısı
1	00, 01, 10, 11	X	X	X	X	0	ARGB'deki renk değerini hedefe yaz	IDLE -> {Hedefe Yaz -> Adres Hesap}	4
2	00, 01, 10, 11	00	0	00	00	1	Kaynağı hedefe yaz, kaynağı güncelleme	IDLE -> {Kaynaktan Oku -> Hedefe Yaz -> Adres Hesap}	4
3	00, 01, 10, 11	00	1	00	00	1	Kaynağı hedefe alfa karıştırarak yaz, kaynağı güncelleme	IDLE -> {Kaynaktan Oku -> Hedeften Oku -> Hedefe Yaz -> Adres Hesap}	4
4	00, 01, 10, 11	01	0	00	00	1	Kaynağı hedefe yaz, kaynağı ARGB ile güncelle	IDLE -> {Kaynaktan Oku -> Hedefe Yaz -> Kaynağa Yaz -> Adres Hesap}	4
5	00, 01, 10, 11	01	1	00	00	1	Kaynağı hedefe alfa karıştırarak yaz, kaynağı ARGB ile güncelle	IDLE -> {Kaynaktan Oku -> Hedeften Oku -> Hedefe Yaz -> Kaynağa Yaz -> Adres Hesap}	4
6	00, 01, 10, 11	10	0	00	00	1	Kaynağı hedefe yaz, kaynağı arka plan ile güncelle	IDLE -> {Kaynaktan Oku -> Hedefe Yaz -> Arka Plandan Oku -> Kaynağa Yaz -> Adres Hesap}	4
7	00, 01, 10, 11	10	1	00	00	1	Kaynağı hedefe alfa karıştırarak yaz, kaynağı arka plan ile güncelle	IDLE -> {Kaynaktan Oku -> Hedeften Oku -> Hedefe Yaz -> Arka Plandan Oku -> Kaynağa Yaz -> Adres Hesap}	4
8	00, 01, 10, 11	00	0	01, 10	00	1	Transparan kaynağı hedefe yaz, kaynağı güncelleme	IDLE -> {Kaynaktan Oku -> Hedefe Yaz -> Adres Hesap}, {Kaynaktan oku -> Adres Hesap}	4*2 = 8
9	00, 01, 10, 11	00	1	01, 10	00	1	Transparan kaynağı hedefe alfa karıştırarak yaz, kaynağı güncelleme	IDLE -> {Kaynaktan Oku -> Hedeften Oku -> Hedefe Yaz -> Adres Hesap}, {Kaynaktan Oku -> Adres Hesap}	4*2 = 8
10	00, 01, 10, 11	01	0	01, 10	00	1	Transparan kaynağı hedefe yaz, kaynağı ARGB ile güncelle	IDLE -> {Kaynaktan Oku -> Hedefe Yaz -> Kaynağa Yaz -> Adres Hesap}, {Kaynaktan Oku -> Adres Hesap}	4*2 = 8
11	00, 01, 10, 11	01	1	01, 10	00	1	Transparan kaynağı hedefe alfa karıştırarak yaz, kaynağı ARGB ile güncelle	IDLE -> {Kaynaktan Oku -> Hedeften Oku -> Hedefe Yaz -> Kaynağa Yaz -> Adres Hesap}, {Kaynaktan Oku -> Adres Hesap}	4*2 = 8
12	00, 01, 10, 11	10	0	01, 10	00	1	Transparan kaynağı hedefe yaz, kaynağı arka plan ile güncelle	IDLE -> {Kaynaktan Oku -> Hedefe Yaz -> Arka Plandan Oku -> Kaynağa Yaz -> Adres Hesap}, {Kaynaktan Oku -> Adres Hesap}	4*2 = 8
13	00, 01, 10, 11	10	1	01, 10	00	1	Transparan kaynağı hedefe alfa karıştırarak yaz, kaynağı arka plan ile güncelle	IDLE -> {Kaynaktan Oku -> Hedeften Oku -> Hedefe Yaz -> Arka Plandan Oku -> Kaynağa Yaz -> Adres Hesap}, {Kaynaktan Oku -> Adres Hesap}	4*2 = 8
14	00, 01, 10, 11	00	X	00	01, 10, 11	1	Kaynağı hedefe boolen op. İle yaz. Kaynağı güncelleme	IDLE -> {Kaynaktan Oku -> Hedeften Oku -> Hedefe Yaz -> Adres Hesap}	4*3 = 12
15	00, 01, 10, 11	01	X	00	01, 10, 11	1	Kaynağı hedefe boolen op. İle yaz. Kaynağı ARGB ile güncelle	IDLE -> {Kaynaktan Oku -> Hedeften Oku -> Hedefe Yaz -> Kaynağa Yaz -> Adres Hesap}	4*3 = 12
16	00, 01, 10, 11	10	X	00	01, 10, 11	1	Kaynağı hedefe boolen op. İle yaz. Kaynağı arka plan ile güncelle	IDLE -> {Kaynaktan Oku -> Hedeften Oku -> Hedefe Yaz -> Arka Plandan Oku -> Kaynağa Yaz -> Adres Hesap}	4*3 = 12
17	00, 01, 10, 11	00	X	01, 10	01, 10, 11	1	Transparan kaynağı hedefe boolen op. İle yaz. Kaynağı güncelleme	IDLE -> {Kaynaktan Oku -> Hedeften Oku -> Hedefe Yaz -> Adres Hesap}, {Kaynaktan Oku -> Adres Hesap}	4*2*3 = 24
18	00, 01, 10, 11	01	X	01, 10	01, 10, 11	1	Transparan kaynağı hedefe boolen op. İle yaz. Kaynağı ARGB ile güncelle	IDLE -> {Kaynaktan Oku -> Hedeften Oku -> Hedefe Yaz -> Kaynağa Yaz -> Adres Hesap}, {Kaynaktan Oku -> Adres Hesap}	4*2*3 = 24
19	00, 01, 10, 11	10	X	01, 10	01, 10, 11	1	Transparan kaynağı hedefe boolen op. İle yaz. Kaynağı arka plan ile güncelle	IDLE -> {Kaynaktan Oku -> Hedeften Oku -> Hedefe Yaz -> Arka Plandan Oku -> Kaynağa Yaz -> Adres Hesap}, {Kaynaktan Oku -> Adres Hesap}	4*2*3 = 24
								Toplam Operasyon Sayısı = 184	

Şekil 4.9. BitBLT IP Çekirdeği operasyonları ve durum geçişleri

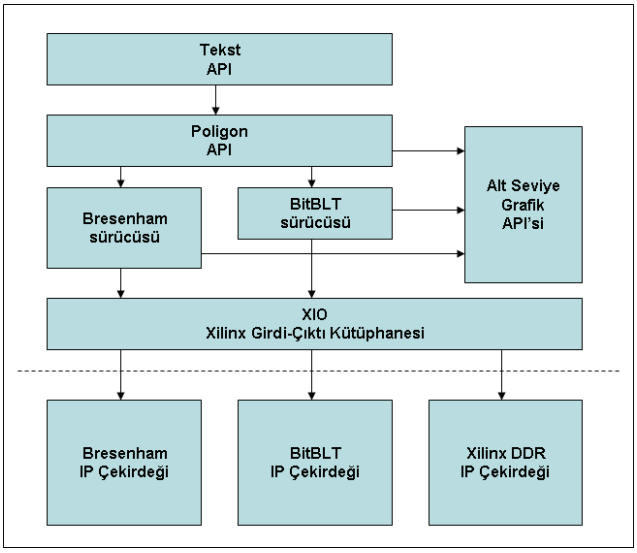


Şekil 4.10. BitBLT IP Çekirdeği durum geçişleri

4.5 Sürücüler ve API'ler

EDK yazılımı, "Create Import Peripheral" arayüzü ile oluşturulan IP Çekirdeği şablonları için bazı temel sürücü fonksiyonları hazırlamaktadır. Bu fonksiyonlar, bazı temel girdi-çıktı fonksiyonları ve IP Çekirdeği için seçilen servislere özel fonksiyonlardır. Örneğin WrFIFO servisi ile birlikte oluşturulan BitBLT IP Çekirdeğinin taslak sürücü fonksiyonları içerisinde yer alan *IPCORE_BITBLT2_mWriteToFIFOIP* fonksiyonu EDK tarafından oluşturulmuş bir fonksiyondur. Bu fonksiyonun görevi BitBLT biriminin WrFIFO belleğine bir veri cümlesi yazmaktır. EDK, IP Çekirdeği şablonu ile birlikte oluşturduğu bu fonksiyonların isimlerini, kullanıcı tarafından IP Çekirdeğine verilen ismi kullanarak türetir. Ancak bu fonksiyonlar yine de hedef IP Çekirdeğinin taban adresini parametre olarak alırlar. Yani özelleştirme aslında sadece ismen yapılmıştır. Bu fonksiyonlar EDK'nın *xio.h* gibi standart kütüphanelerindeki fonksiyonların örten (wrapper) fonksiyonlarıdır. Bu çalışmada oluşturulan sürücü fonksiyonlarında da *xio.h* altındaki *XIo_In32, XIo_Out32* gibi alt seviye fonksiyonlar kullanılmıştır.

Bu çalışmada, grafik sisteminde yer alan IP Çekirdeklerine CPU tarafından erişimi sağlayan sürücüler ve bu sürücüleri kullanan üst seviye grafik API'leri C dilinde gerçeklenmiştir. Sürücü ve API'lerin listesi ve hiyerarşisi Şekil 4.11'de görülmektedir.



Şekil 4.11. Sürücüler ve API'ler

4.5.1 Bresenham IP Çekirdeği sürücüsü

Bresenham IP Çekirdeğinin adres haritalı yazmaçlarla nasıl yönetildiği Bölüm 4.3'te anlatılmıştır. Bresenham IP Çekirdeği sürücüsü *_bresenham.h* ve *bresenham.c* dosyalarından oluşmaktadır. *_bresenham.h* dosyasında IP Çekirdeğinin taban adresi, komutların adresleri, opsiyonların bit maskeleri ve *bresenhamDogruCiz* fonksiyonunun prototipi tanımlanmıştır (Şekil 4.12). *bresenham.c* dosyasında ise *bresenhamDogruCiz* fonksiyonu gerçeklenmiştir. Bu fonksiyon çizilecek doğrunun başlangıç ve son noktalarını parametre olarak alır ve Bölüm 4.3'te bütün doğrular için uyarlanan Bresenham algoritmasının girdilerini hesaplar. Daha sonra eğer IP Çekirdeği meşgul ise bekler ve IP Çekirdeği müsait olduğunda hesaplanan parametreleri IP Çekirdeğinin ilgili yazmaçlarına gönderir (Şekil 4.13). Sürücü fonksiyonlarının bütün halleri Ekler kısmında görülebilir.

```
#include "xio.h"


#define BRESENHAM_BASE_ADDR     0xc3a00000

#define NUMBER_OF_REGS              16
#define NUMBER_OF_RO_REGS           4

//Bresenham Register Adresleri
#define BRE_COMMAND_FLAGS           BRESENHAM_BASE_ADDR + 0
#define BRE_EMAJ                    BRESENHAM_BASE_ADDR + 4
#define BRE_EMIN                    BRESENHAM_BASE_ADDR + 8
#define BRE_ESON                    BRESENHAM_BASE_ADDR + 12
#define BRE_HATAMAJ2                BRESENHAM_BASE_ADDR + 16
#define BRE_HATAMIN2                BRESENHAM_BASE_ADDR + 20
#define BRE_RGB                     BRESENHAM_BASE_ADDR + 24
#define BRE_EXT_DATA                BRESENHAM_BASE_ADDR + 28

#define BRE_STATUS_FLAGS            BRESENHAM_BASE_ADDR + NUMBER_OF_REGS * 4 + 0

//Komut Register Maskeleri
#define BRE_GO              0x00000001
#define BRE_NEGATIF_YON     0x00000002
#define BRE_MAJOR_Y         0x00000004
#define BRE_KESIKLI_CIZGI   0x00000008

//Durum Register Maskeleri
#define BRE_BUSY            0x00000001

int bresenhamDogruCiz(struct tNokta nokta1, struct tNokta nokta2,
        unsigned int rgb, int kesikli);
```

Şekil 4.12. Bresenham IP Çekirdeği sürücüsündeki tanımlamalar (_bresenham.h)

```
int bresenhamDogruCiz(struct tNokta nokta1, struct tNokta nokta2,
        unsigned int rgb, int kesikli)
{
    // IP Core Parametreleri
    int hata = 0, \
        hataMajorShifted1, \
        hataMinorShifted1, \
        *majorEksenDegeri, \
        majorSonDeger, \
        *minorEksenDegeri, \
        arttirimYonu = 1, \
        majY = 0;

      // PArametrelerin Hesaplanısı

            |
            |
            |


    //Mesgul ise bekle...
    while(  XIo_In32((XIo_Address)BRE_STATUS_FLAGS) & BRE_BUSY  );
    //Peripherial GO !!
    XIo_Out32((XIo_Address)BRE_EMAJ, *majorEksenDegeri);
    XIo_Out32((XIo_Address)BRE_EMIN, *minorEksenDegeri);
    XIo_Out32((XIo_Address)BRE_ESON, majorSonDeger);
    XIo_Out32((XIo_Address)BRE_HATAMAJ2, hataMajorShifted1);
    XIo_Out32((XIo_Address)BRE_HATAMIN2, hataMinorShifted1);
    XIo_Out32((XIo_Address)BRE_RGB, rgb);

    commandFlags = 0;
    commandFlags |= BRE_GO;
    if(arttirimYonu < 0)
        commandFlags |= BRE_NEGATIF_YON;
    if(majY)
        commandFlags |= BRE_MAJOR_Y;

    if(kesikli)
        commandFlags |= BRE_KESIKLI_CIZGI;

    XIo_Out32((XIo_Address)BRE_EXT_DATA, 4);

    XIo_Out32((XIo_Address)BRE_COMMAND_FLAGS, commandFlags);

    return 1;

}
```

Şekil 4.13. *bresenhamDogruCiz* sürücü fonksiyonu

4.5.2 BitBLT IP Çekirdeği sürücüsü

BitBLT IP Çekirdeğinin WrFIFO komut belleğinin işlevi ve komutların anlamları Bölüm 4.4'te anlatılmıştır. BitBLT sürücü fonksiyonunun görevi, kullanıcının ayarladığı komut verisini tanımlı bir sırayla WrFIFO'ya yazmaktır. BitBLT IP Çekirdeği sürücüsü *_BitBLT.h* ve *BitBLT.c* dosyalarından oluşmaktadır. *_BitBLT.h* dosyasında IP Çekirdeğinin taban adresi, komutların opsiyonlarının belirtildiği bit maskeleri, bir BitBLT komutunun bütününün

tanımlandığı *struct tBitBLT* yapısı ve *bitBLT_go* fonksiyonunun prototipi tanımlanmıştır (Şekil 4.14). *bresenham.c* dosyasında ise *bitBLT_go* fonksiyonu gerçeklenmiştir. Bu fonksiyon *struct tBitBLT* tipindeki komut cümlesinin adresini parametre olarak alır. Fonksiyon, adresi verilen komut cümlelerini Şekil 4.15'te görülen sırada IP Çekirdeğinin WrFIFO belleğine yazar.

```
#include "_grafikOrtak.h"
#include "xio.h"

#define BITBLT_BASE_ADDR        0xc7200000
#define ARKA_PLAN_ADDR_REG      BITBLT_BASE_ADDR + 8
#define BLT_HEADER          0xAA0000AA

#define KAYNAK_ARGB                 0
#define KAYNAK_ADDR                 1
#define BOOL_NON                    0<<1
#define BOOL_AND                    1<<1
#define BOOL_OR                     2<<1
#define BOOL_XOR                    3<<1
#define TRANSPARAN_NON              0<<3
#define TRANSPARAN_ARGB             1<<3
#define TRANSPARAN_NOT_ARGB         2<<3
#define ALPHA                       1<<5
#define SOL_ALT_KOSE                0<<6
#define SAG_ALT_KOSE                1<<6
#define SAG_UST_KOSE                2<<6
#define SOL_UST_KOSE                3<<6
#define KAYNAK_GUNCELLEME           0<<8
#define KAYNAK_GUNCELLE_ARGB        1<<8
#define KAYNAK_GUNCELLE_ADDR        2<<8

struct tBltKomut
{
    unsigned int argb;
    unsigned int kaynak_addr;
    unsigned int hedef_addr;
    unsigned int frm_buf_row_size;
    unsigned int blt_row_size;
    unsigned int blt_col_size;
    unsigned int flags;};

int bitBLT_Go(void* peripAddr, struct tBltKomut* pBlt);
```

Şekil 4.14. BitBLT IP Çekirdeği sürücü tanımlamaları (*_bitBLT.h*)

```
#include "_bitBLT.h"


int bitBLT_Go(void* peripAddr, struct tBltKomut* pBlt)
{
    Xuint32 baseaddr = peripAddr;

    IPCORE_BITBLT2_mWriteToFIFO(baseaddr, BLT_HEADER);
    IPCORE_BITBLT2_mWriteToFIFO(baseaddr, pBlt->argb);
    IPCORE_BITBLT2_mWriteToFIFO(baseaddr, pBlt->kaynak_addr);
    IPCORE_BITBLT2_mWriteToFIFO(baseaddr, pBlt->hedef_addr);
    IPCORE_BITBLT2_mWriteToFIFO(baseaddr, pBlt->frm_buf_row_size);
    IPCORE_BITBLT2_mWriteToFIFO(baseaddr, pBlt->blt_row_size);
    IPCORE_BITBLT2_mWriteToFIFO(baseaddr, pBlt->blt_col_size);
    IPCORE_BITBLT2_mWriteToFIFO(baseaddr, pBlt->flags);

}
```

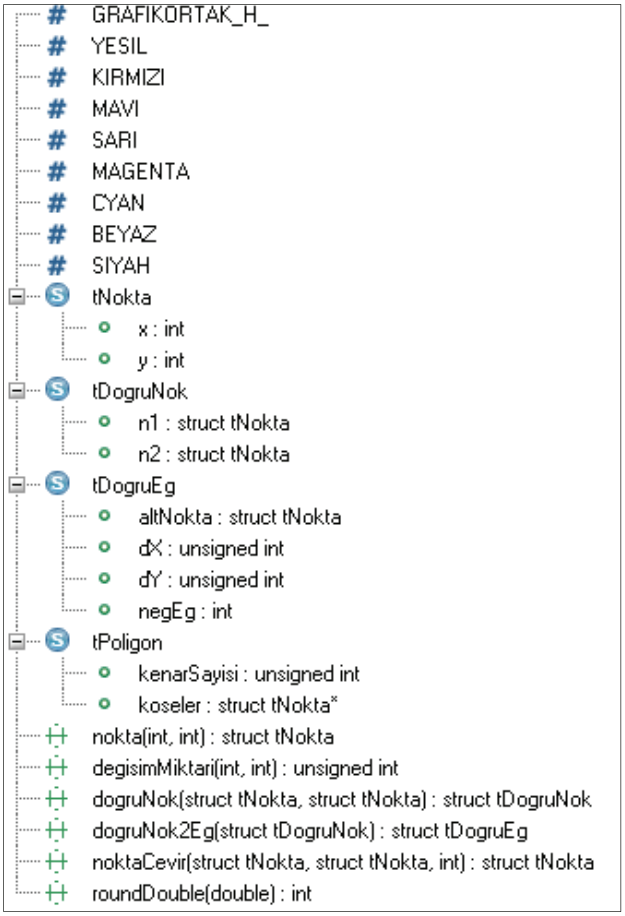
Şekil 4.15. *bitBLT_go* fonksiyonu (*bitBLT.c*)

4.5.3 Alt seviye grafik API'si

Alt seviye grafik API'si, renk, nokta, doğru, poligon gibi düğer üst seviye API'lerin (poligon, text API'leri) ortak kullandığı tanımlamalar, fonksiyonlar ve veri yapılarını içerir. Aynı zamanda, kullanılan Xilinx VGA IP Çekirdeği için gerekli parametreler de bu API'de tanımlanmıştır. *_grafikOrtak.h, grafikOrtak.c, _koordSysXUP.h ve koordSysXUP.c* olmak üzere dört adet dosyadan oluşur.

_grafikOrtak.h dosyasının içeriği Şekil 4.16'da görülmektedir. Burada görüldüğü gibi doğrular iki farklı veri yapısı ile tanımlanmıştır. Veri yapısının ikincisinde (*struct tDogruEg*) ilkinden farklı olarak bir adet nokta, doğrunun eksenler üzerindeki değişim miktarları ve eğiminin işaret bilgisi yer almaktadır. Bu veri yapısı poligon boyama API'si tarafından kullanılmıştır. *_grafikOrtak.c* dosyasında ise şekilde görülen fonksiyonlar gerçeklenmiştir.

_koordSysXUP.h ve *koordSysXUP.c* dosyaları piksel ifadelerinden adres ifadelerine geçişi sağlayan arayüzleri içermektedir. VGA IP Çekirdeğinin tanımladığı çerçeve arabelleğinin adresi ve görünür kısımlarının satır-sütun uzunlukları burada tanımlanmıştır (Şekil 4.17). Buradaki tanımlamalar ve fonksiyonlar, grafik oluşum süreci içerisindeki koordinat dönüşüm aşamasını gerçeklemektedir.



Şekil 4.16. *_grafikOrtak.h* dosyasının içeriği

```
#ifndef KOORDSYSTEMXUP_H_
#define KOORDSYSTEMXUP_H_

#include "_grafikOrtak.h"



#define frameBufStart   0x07E00000
#define satirUzunlugu   1024
#define sutunUzunlugu   484

#define gorunurSatirUzunlugu    640
#define gorunurSutunUzunlugu    484

unsigned int* koordToAddr(struct tNokta nokta);

void pikselBoyaXUP(struct tNokta nokta, unsigned int rgb);

void ekranTemizleSoft(unsigned int rgb);
```

Şekil 4.17. *_koordSysXUP.h* dosyasındaki tanımlamalar

4.5.4 Poligon API

Bu tez çalışmasında, işlemci ve işlemci tarafından sürülebilen çevresel birimler ile grafik oluşturma yönteminin gösterilmesi için satır tarama yöntemi ile poligon boyama algoritması gerçeklenmiştir. Aynı zamanda poligonların tanımlanış yöntemi de pikselleştirme öncesi modelleme aşamasına örnek olarak gösterilebilir. Bu sayede grafik sisteminin işlevselliği gösterilebilmiştir.

Poligon doldurma işlemi CPU IP Çekirdeği ve BitBLT IP Çekirdeği ile birlikte gerçeklenmiştir. BitBLT IP Çekirdeğinin görevi CPU tarafından kesim noktası belirlenen satırları doldurmaktır. Satırların kenar kesim noktalarının hesaplanışı Bölüm 3.2'de gösterilen algoritmalarla, CPU IP Çekirdeğinde yapılır. Daha sonra BitBLT IP Çekirdeği, belirtilen satır parçalarını boyamakla görevlendirilir. Buradaki operasyon BitBLT IP Çekirdeği açısından sadece hedef bit haritasını verilen ARGB değeri ile doldurmaktır. Bir satır parçası, sütun uzunluğu bir olan bit haritası olarak düşünülebilir. Bu nedenle poligon boyama operasyonu için herhangi bir özelleştirme yapılmadan BitBLT IP Çekirdeği kullanılabilir. Burada Bit BLT IP Çekirdeğinin FIFO komut belleği ile birlikte tasarlanmış olası önemli bir avantaj sağlamıştır. Çünkü CPU'da hesaplanan kesim noktalarına göre belirlenen BitBLT komutları, bir satırın boyanması beklenmeksizin BitBLT IP Çekirdeğinin FIFO komut belleğine atılabilmektedir. Bu sayede işlemcide harcanan zaman minimuma indirilmiştir.

CPU IP Çekirdeğinde çalışan poligon boyama fonksiyonunun (**int poligonBoya(struct tPoligon* pol, void* buffer, unsigned int renk**)) görevi doldurulacak poligonda satır kesim noktalarını hesaplamak ve buna göre BitBLT komutlarını BitBLT IP Çekirdeğinin FIFO komut belleğine yazmaktır. Poligonun kenarları ardışık köşe noktaları ile modellenmektedir. Y değeri en küçük olan köşe noktası taramaya başlanacak satırı belirtir. **poligonBoya** fonksiyonu Bölüm 3.2'de anlatılan ortak köşe problemini ortadan kaldırmak için kenar kısaltma ve yeni nokta oluşturma işlemlerini yaparak kenar kesim noktalarını hesaplar. Bir poligon kenarı Şekil 4.18'de görülen veri yapısı ile temsil edilir.

Alt Nokta		ΔX	ΔY	Negatif Eğim
Xalt	Yalt	|Xalt - Xüst|	|Yalt - Yüst|	0 ya da 1

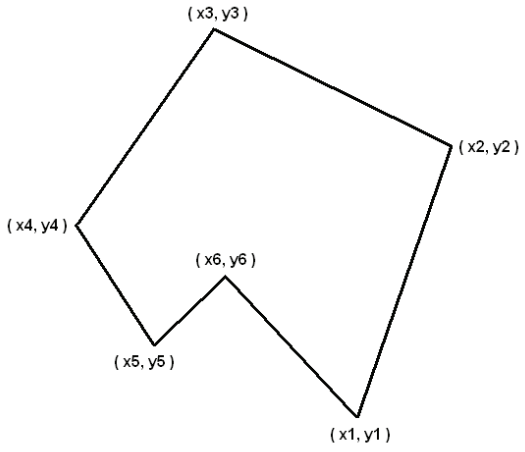
Şekil 4.18. Kenar FIFO Arabelleğine Verilen Bir Kenar Bilgisinin Veri Tipi

Burada *Alt Nokta*, kenarın y değeri daha küçük olan bitim noktasıdır. ΔX ve ΔY ise eksenlerin değişim miktarıdır. Yukarıda anlatılan artımsal hesapta her zaman pozitif değerler kullanıldığından değişim miktarlarının mutlak değeri alınmıştır. Eğimi belli etmek için ise *Negatif Eğim* bayrağı, veri yapısına eklenmiştir. Bu alan *Boolean* tipinde bir alandır ve yalnızca 0 ya da 1 değerlerini alabilir. Poligon iseŞekil 4.19'da görüldüğü gibi ardışık kenar bilgileri ile tanımlanır.

Kenar Sayısı = n	KenarBilgisi-1	KenarBilgisi-2	……..	KenarBilgisi-n

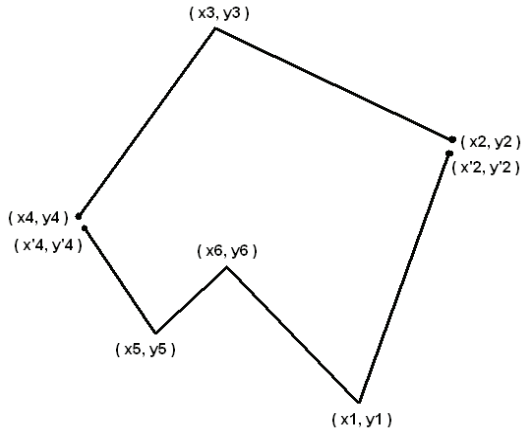
Şekil 4.19. Kenar FIFO arabelleğine girilen verinin yapısı

Şekil 4.20'daki örnek poligon için kenar bilgilerinin nasıl hazırlandığı Şekil 4.22'de gösterilmiştir.



Şekil 4.20. Köşe koordinatları verilen örnek poligon

poligonBoya() fonksiyonu yukarıdaki poligon için ortak köşe problemini gidererekŞekil 4.21'deki poligonu elde eder.



Şekil 4.21. Ortak kenar problemi giderilmiş poligon

Çizelge 4.1. Ortak kenar problemi giderilmiş poligon için üretilen kenar bilgileri

Alt Nokta		ΔX	ΔY	Negatif Eğim	Sayaç
x1	y1	|x1 - x6|	|y1 - y6|	1	0
x1	y1	|x1 - x'2|	|y1 - y'2|	0	0
x5	y5	|x5 - x'4|	|y5 - y'4|	1	0
x5	y5	|x5 - x6|	|y5 - y6|	0	0
x4	y4	|x4 - x3|	|y4 - y3|	0	0
x2	y2	|x2 - x3|	|y2 - y3|	1	0

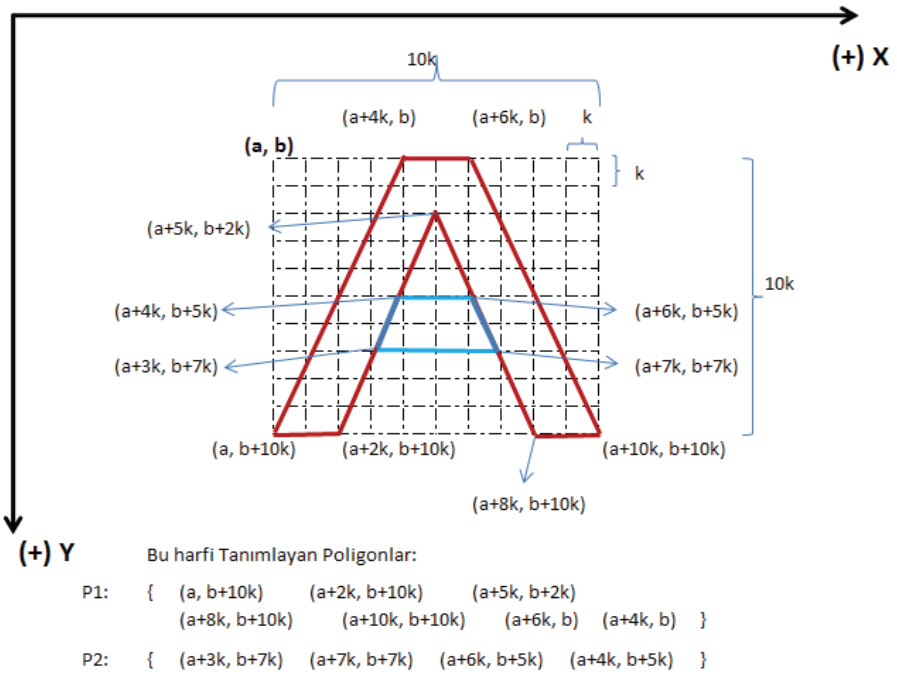
Çizelge 4.1'de görülen tablo poligonBoya fonksiyonunun yukarıdaki poligonun boyanması için hazırlanmış kenar bilgilerinin listesidir. Poligon boyama fonksiyonu bu alanları aynı zamanda değişken olarak kullanır. Alt Nokta X değeri bir önceki tarama satırı için bulunan kesim noktasının X değerini tutar. Her tarama satırında hangi kenarların kesildiği Alt Nokta Y ve ΔY değerlerinden faydalanılarak listede taranır. Satır numarası Y ve ΔY arasında ise bu kenarın listedeki son hesap bilgilerinden yeni kesim noktası hesaplanır. Daha sonra bulunan kesim noktalarının X değerleri küçükten büyüğe sıralanır. Böylece boyanacak satır parçaları belirlenmiş olur. Belirlenen satır parçalarının boyanması için BitBLT IP Çekirdeğinin FIFO komut belleğine gerekli komutlar yazılır. Böylece poligon boyama işleminde CPU IP Çekirdeğinin işlevi tamamlanmış olur.

Poligon API'sinde poligon boyamanın yanı sıra poligon ölçekleyen ve taşıyan, *poligonScale* ve *poligonTasi* gibi fonksiyonlar bulunmaktadır. Bu fonksiyonlar da grafik oluşum süreci içerisindeki model dönüşümleri kısmını temsil etmektedir.

4.5.5 Tekst API

Tekst API, grafik sisteminin işlevselliğini ve grafik ardışık düzeninin bütün aşamalarının gerçeklenişini göstermek amacıyla tasarlanmıştır. Tekst API fonksiyonları, sistemde tasarlanan API'ler içerisinde en üst seviyede olan fonksiyonlardır. Bir alt seviyede Poligon API bulunmaktadır.

Tekst API'de yazı karakterleri poligonlar ile modellenmiştir. Örneğin Şekil 4.22'de 'A' karakterinin iki adet poligon ile modellenişi gösterilmektedir. Bu şekilde görüldüğü gibi her karakter 10x10 çözünürlükte tasarlanmıştır. Karakterin tanımlandığı çerçevedeki her göz 'k' kalınlıkta kabul edilmiştir. 'k' dinamik olarak değiştirilebilen bir değer olarak düşünülmüştür. Bu sayede karakterler istenilen ölçekte çizdirilebilecektir. Aynı zamanda karakterler tanımlanırken köşe noktaları sol üst köşe **(a,b)** referans alınarak belirtilmiştir. Şekilde görüldüğü gibi karakteri ifade eden poligon köşeleri 'a', 'b' ve 'k' değişkenleri ile formülleştirilmiştir. Bu sayede karakterlerin ekran üzerindeki herhangi bir noktada ve herhangi bir ölçekte çizdirilebilmesi mümkündür.



Şekil 4.22. 'A' karakterinin poligonlarla modellenişi

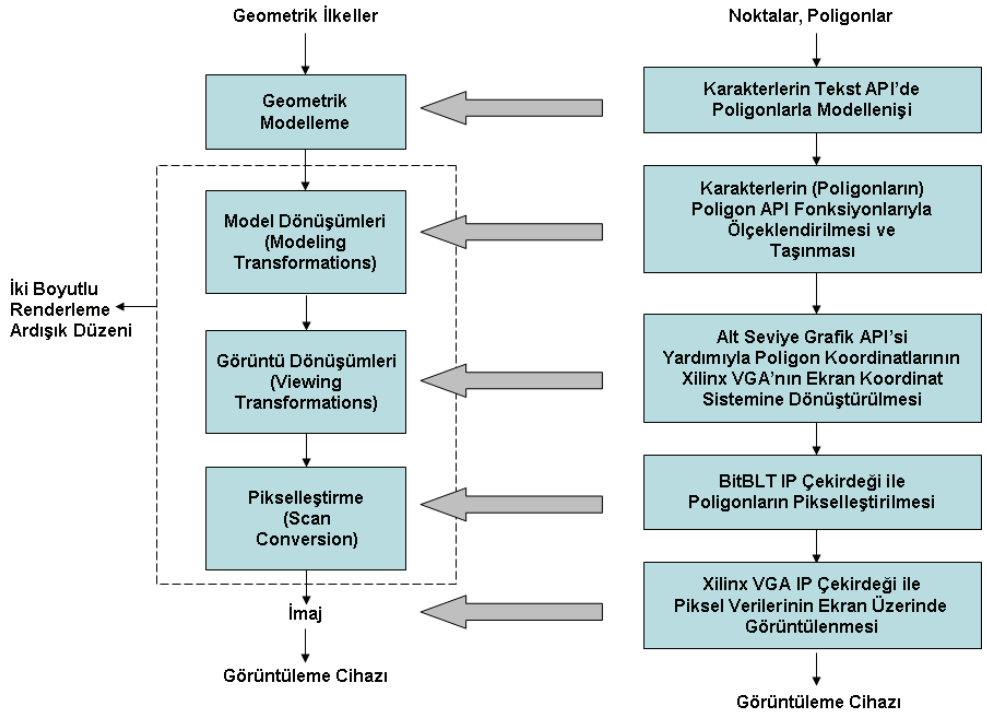
Karakterler API içerisinde statik bir tabloda, a=0, b=0 ve k=1 kabul edilerek tanımlanmışlardır (Şekil 4.23). Birim ölçüde tanımlanan bu poligonlar, Poligon API'de sağlanan poligon taşıma ve ölçeklendirme fonksiyonları ile dönüştürülmektedirler. Bu sayede istenilen ölçek ve konumda karakterler çizdirilebilmektedir.

```
struct tPoligonKarakter
{
    int poligonSayisi;
    struct tPoligon** pols;
};

static struct tPoligonKarakter pchar_dummy = { 0, NULL };

/*********** KARAKTER TABLOSU *************/

// karakter: A
static struct tNokta koseler_A1[7] = { {0, 10}, {2, 10}, {5, 2}, {8, 10},
        {10, 10}, {6, 0}, {4, 0} };
static struct tPoligon pol_A1 = { 7, koseler_A1 };
static struct tNokta koseler_A2[4] = { {3, 7}, {7, 7}, {6, 5}, {4, 5} };
static struct tPoligon pol_A2 = { 4, koseler_A2 };
static struct tPoligon* pols_A[2] = { &pol_A1, &pol_A2 };
static struct tPoligonKarakter pchar_A = { 2, pols_A };

// karakter: B
static struct tNokta koseler_B1[4] = { {1,0}, {4,0}, {4,10}, {1,10} };//4
static struct tPoligon pol_B1 = { 4, koseler_B1 };
static struct tNokta koseler_B2[17] = { {4,0}, {8,0}, {10,2}, {10,4}, {9,5},
        {10,6}, {10,8}, {8,10}, {4,10}, {4,8}, {8,8}, {8,6}, {4,6}, {4,4},
        {8,4}, {8,2}, {4,2}  };//17
static struct tPoligon pol_B2 = { 17, koseler_B2 };
static struct tPoligon* pols_B[2] = { &pol_B1, &pol_B2 };
static struct tPoligonKarakter pchar_B = { 2, pols_B };

// karakter: C
static struct tNokta koseler_C1[22] = { {3,0}, {7,0}, {9,1}, {10,2}, {10,3},
        {8,3}, {6,2}, {4,2}, {2,3}, {2,7}, {4,8}, {6,8}, {8,7}, {10,7},
        {10,8}, {9,9}, {7,10}, {3,10}, {1,9}, {0,8}, {0,2}, {1,1} };//22
static struct tPoligon pol_C1 = { 22, koseler_C1 };
static struct tPoligon* pols_C[1] = { &pol_C1 };
static struct tPoligonKarakter pchar_C = { 1, pols_C };
```

Şekil 4.23. Karakterlerin tanımlanışı

5. SİSTEM GÖSTERİMİ VE TESTİ

Tasarlanan grafik sistemi ile Bölüm 2'de anlatılan iki boyutlu grafik oluşum aşamalarının hepsi gerçeklenebilmiştir. Şekil 4.25'te, yazı karakterlerinin grafik sisteminde gerçeklenişi ve karakter renderleme aşamalarının grafik oluşum aşamaları ile eşleştirilmesi görülmektedir. Noktalar ile poligonlar, poligonlar ile karakterler modellenerek, Geometrik Modelleme aşaması gerçeklenmiştir. Poligon API fonksiyonları ile poligonların ölçeklendirilmesi ve taşınması sağlanmıştır; bu sayede Model Dönüşümü aşaması gerçeklenmiştir. Görüntü Dönüşümü aşamasının gerçeklenmesi için Alt Seviye Grafik API'si tasarlanmıştır. Bu API'deki fonksiyonlar ile poligonların köşe noktalarının koordinatları ekran koordinat sistemine dönüştürülmüştür. Poligonların pikselleştirilmesinde kullanılan BitBLT ve Bresenham IP Çekirdekleri ile Pikselleştirme aşaması gerçeklenmiştir. Son olarak Görüntüleme aşamasının gerçeklenmesi için Xilinx VGA IP Çekirdeği kullanılmıştır.



Şekil 5.1. Karakter renderleme aşamaları ile iki boyutlu grafik ardışık düzeninin eşleştirilmesi

Grafik sisteminin testi, XUP V2P geliştirme kitinin VGA çıkışına bağlanan bir monitör yardımıyla görsel olarak yapılmıştır. Aşağıdaki şekillerde, FPGA yongasında renderlenen örnek ekran görüntüleri görülmektedir. Bu ekran görüntüleri ile sürücülerin, API'lerin ve IP Çekirdeklerinin işlevleri gösterilmiştir. Aynı zamanda, sabit bir arka plan üzerinde hareketli ve saydam nesne gezdirme işleminin gerçeklenmesi, şekillerdeki görüntüler ile anlatılmıştır.

Şekil 4.26'da Bresenham IP Çekirdeği kullanılarak çizdirilen, kesikli ve sürekli çizgiler, BitBLT IP Çekirdeği ile beyaza boyanan alanlar ve Tekst API fonksiyonları ile yazdırılan "FPGA" yazısı görülmektedir. Bu görüntü oluşturulduktan sonra, çerçeve arabelleğindeki veri bellekteki başka bir bölgeye, arka plan verisi olarak kaydedilmiştir.

Şekil 5.2. Bresenham metodu ile çizdirilen kesikli ve sürekli çizgilerin, Tekst API ile yazdırılan karakterlerin ve BitBLT ile boyanan alanların görülebildiği örnek (arka plan) ekran görüntüsü

Şekil 4.26'daki arka plan kaydedildikten sonra, Poligon API ile yıldız şeklinde ve geçirgen olmayan bir poligon çizdirilmiştir. Bu görüntü Şekil 4.27'de

görülmektedir. Poligon çizdirilirken, renginin alfa değeri 127 olarak verilmiştir. Ancak çizdirme aşamasında alfa karıştırma opsiyonu seçilmediğinden geçirgen görülmemektedir. Bu aşamadan sonra yapılmak istenen, yıldızı başka bir bölgeye taşımak ve hedefteki görüntü ile alfa karıştırma yaparak saydam görüntü elde etmektir. Bu işlem için BitBLT IP Çekirdeği kullanılmıştır. Tek BitBLT komutu ile yıldızın eski bölgesinin arka plan görüntüsü ile güncellenmesi, hedef görüntü ile alfa karıştırma operasyonunun yapılması gerçekleştirilmiştir. Aynı zamanda BitBLT IP Çekirdeğinin saydamlık opsiyonu da kullanılmıştır. Kaynak bit haritasından taşınacak olan piksellerin renk değeri olarak yıldızın renk değeri verilmiştir. Bu sayede kaynak bit haritasında yıldız ile aynı renkte olmayan arka plan görüntüsü taşınmamıştır. Şekil 4.28'de ve Şekil 4.29'da yıldızın alfa karıştırma sonucu kazanmış olduğu saydamlık ve yıldızın ilk bölgesinin arka plan ile güncellenmiş olduğu görülebilmektedir.

Şekil 5.2. Şekil 5.1'deki arka plan görüntüsünün üzerine çizdirilen, yıldız şeklinde, geçirgen olmayan poligonun yer aldığı ekran görüntüsü

Şekil 5.3. Şekil 5.2'de görülen yıldızın saydamlaştırılıp, arka plan üzerindeki başka bir bölgeye taşınması ile elde edilen ekran görüntüsü

Şekil 5.4. Şekil 5.3'te görülen yıldızın başka bir bölgeye taşınması, saydamlığının yakın plandan görünüşü

6. SONUÇ

Bu tezde, işlemci ve grafik IP Çekirdeklerinden oluşan iki boyutlu bir grafik sistemi, FPGA yongasında gerçeklenmiştir. Tezde yapılan çalışma, yazılım ve donanım tasarımı olmak üzere iki temel başlık altında düşünülebilir. Donanım tasarımında CPU IP Çekirdeğine veri yolu ile bağlı IP Çekirdekleri oluşturulmuştur. Yazılım tasarımında ise CPU IP Çekirdeğinde çalışan sürücü ve API fonksiyonları yazılmıştır.

Xilinx firmasına ait EDK ortamında, Virtex2P FPGA yongasında bulunan PowerPC IP Çekirdeğine PLB veri yolu ile bağlı IP Çekirdekleri oluşturmak mümkün olmuştur. Bu sayede işlem yükü açısından maliyetli grafik işlemlerinin, FPGA yongasındaki çevresel birimlerde gerçeklenebileceği gösterilebilmiştir.

Tez çalışmasında öncelikle, IP Çekirdeklerinin PLB üzerinde master ve slave olarak çalışabilmesi için gerekli kodlamalar yapılmıştır. Xilinx IPIC arayüzü ile bağlantı sağlayan kodlar yazılmış ve IP Çekirdeklerinin PLB veri yolunda çift yönlü iletişim yapabilmeleri sağlanmıştır. IP Çekirdeklerinin slave arayüzü CPU IP Çekirdeği tarafından erişilebilmelerini, master arayüzü ise IP Çekirdeklerinin DDR bellek üzerindeki çerçeve arabelleğine erişebilmelerini mümkün kılmıştır.

Bresenham'ın doğru çizme algoritması Bresenham IP Çekirdeği içerisinde gerçeklenmiştir. Bu IP Çekirdeğinde adres haritalı yazmaçlar oluşturulmuştur. CPU IP Çekirdeği üzerinde çalışan ve bu yazmaçlara erişerek Bresenham IP Çekirdeğine komut gönderen sürücü fonksiyonları C dilinde gerçeklenmiştir.

BitBLT ve Alfa Karıştırma operasyonları BitBLT IP Çekirdeği içerisinde gerçeklenmiştir. Bu IP Çekirdeği de sistemde hem master hem de slave olarak tanımlanmıştır. Bu IP Çekirdeği gerçeklenirken PLB IPIF'ın WrFIFO servisinden faydalanılmıştır. Komutlar bu FIFO arabellek aracılığı ile IP Çekirdeğine aktarılmıştır. Bu yapı, satır tarama ile poligon boyama algoritmasının BitBLT IP Çekirdeği ile gerçeklenebilmesini mümkün kılmıştır. İşlemci tarafında hesaplanan kenar kesim noktaları piksel boyama operasyonları beklenmeksizin FIFO belleğe aktarılmış ve CPU tarafındaki işlem minimuma indirilmiştir.

Bresenham ve BitBLT IP Çekirdekleri gerçeklendikten sonra, yazılım tasarımı yapılmıştır. Yazılım tasarımında grafik IP Çekirdeklerini süren sürücü

fonksiyonları, Poligon ve Tekst API'leri yazılmıştır. API fonksiyonları ile pikselleştirme öncesi grafik oluşum aşamaları, sürücüler ve IP Çekirdekleri vasıtasıyla da pikselleştirme ve pikselleştirme sonrası grafik aşamaları gerçeklenmiştir.

Tezde yapılan çalışma geliştirilmeye açıktır. Geliştirme, donanım ve yazılım tasarımında kolayca yapılabilir. Örneğin elips çizme, yumuşatma (antialiasing) ve dokulama gibi başka grafik işlemlerini yapan IP Çekirdekleri gerçeklenip sisteme kolayca eklenebilir. Bu çalışmada IP Çekirdeklerinin PLB'ye bağlanması için yapılan kodlamalar, geliştirilecek diğer IP Çekirdeklerinde de kullanılabilir.

KAYNAKLAR

[1] Singh, S. ve Bellec, P., *Virtual Hardware for Graphics Applications Using FPGAs*, The University of Glasgow Computing Science Dept., Compass Design Automation, Glasgow İngiltere, Antipolis Fransa, 2001.

[2] Chin, L., *FPGA Based Embedded Vision Systems*, Lisans Tezi, The University of Western Australia, Faculty of Engineering, Crawley Avustralya, 2006.

[3] Holten, H., *Design for Scalability in 3D Computer Graphics Architectures*, Yüksek Lisans Tezi, Technical University of Denmark, Computer Science and Technology Informatics and Mathematical Modelling, Copenhagen Danimarka, 2001.

[4] Warner, R., *Real Time 3-d Graphics Processing Hardware Design Using Field-Programmable Gate Arrays*, Lisans Tezi, Pennsylvania State University, Computer Engineering, 1999.

[5] Knutsson, N., *An FPGA-based 3D Graphics System*, Yüksek Lisans Tezi, Linköping Institute of Technology, System Technique Dept., Linköping İsveç, 2005.

[6] Gray, L., Woodson, R., Chau, A. ve Retzlaff, S., *Graphics for the long term*: *An FPGA-based GPU*, Titan Advanced Products & Design Company Teknik Raporu, San Diego, A.B.D., 2005.

[7] Altera Company, *Using FPGAs to Render Graphics and Drive LCD Interfaces*, 2009.
www.altera.com/literature/wp/wp-01100-graphic-lcd-display.pdf

[8] LogiBrics Company, *logiBITBLT IP Core DataSheet*, 2009.
www.logicbricks.com/Documentation/Datasheets/IP/logiBITBLT_hds.pdf

[9] Imagination Company, *POWERVR MBX IP Core DataSheet*, 2008.
theliel.papipapito.com/Doc/PowerVR_%20MBX.pdf

[10] Altera Company, *AN371: Automotive Graphics System Reference Design,* 2004.
www.altera.com/literature/an/an371.pdf

[11] Altera Company, *Avalon LCD Controller*, 2004.
www.altera.com/literature/an/an372.pdf

[12] Altera Company, *Applying Graphics to FPGA-Based Solutions*, 2008. http://www.altera.com/literature/wp/wp-01075-applying-graphics-to-fpga-based-solutions.pdf

[13] Şahin, İ., ve Koyuncu, İ., "Grafik Sistemleri İçin Fpga Cihazlarında Çalışmak Üzere Tasarlanmış Matris Çarpım Motoru", SAÜ. Fen Bilimleri Dergisi, 12, 61-68, 2008.

[14] Xilinx Company, *XUP Virtex-II Pro Development System Hardware Reference Manual*, 2005 www.xilinx.com/univ/XUPV2P/Documentation/ug069.pdf

[15] Hearn, D., Baker, P., Wesley, A., *Computer Graphics,* Prentice-Hall, A.B.D., 1998.

[16] Xilinx Company, *Virtex II Pro Datasheet*, 2007. www.xilinx.com/support/documentation/data_sheets/ds083.pdf

[17] IBM Company, CoreConect Bus Specifications, 2009. https://www01.ibm.com/chips/techlib/techlib.nsf/productfamilies/CoreConnect_Bus_Architecture

[18] Xilinx Company, *PLB IPIF (v2.01a) Product Specification*, EDK Documentation, 2004.

[19] Porter, T., Duff, T., *Compositing Digital Images*, Computer Graphics Project, Lucasfilm Ltd., San Francisco, A.B.D., 1984.

[20] Smith, A., *Image Compositing Fundamentals* , Microsoft Tech, 1995.

[21] Xilinx Company, *XST 7.1 User Guide*, EDK Documentation, 2005

[22] Xilinx Company, *EDK Concept, Tools and Techniques*, EDK Documentation, 2005.

[23] Chu, P., *RTL Hardware Design Using VHDL*, Wiley Interscience, A.B.D., 2005.

[24] Can, T., *METU Ceng 477 Computer Engineering Lecture Notes*, *2 Dimensional Viewing*, METU, 1995.

Ek-1 BitBLT IP Çekirdeği Kodları

jEdit - user_logic.vhd

```
1   ------------ IP Core BitBLT User Logic ------------
2
3
4   library ieee;
5   use ieee.std_logic_1164.all;
6   use ieee.std_logic_arith.all;
7   use ieee.std_logic_unsigned.all;
8
9   library proc_common_v1_00_b;
10  use proc_common_v1_00_b.proc_common_pkg.all;
11
12  entity user_logic is
13    generic
14    (
15     --Kullanıcı parametreleri
16       C_BASEADDR                     : std_logic_vector(0 to 31)     := x"c7200000";
17     --Kullanıcı parametreleri BİTİŞ
18      C_AWIDTH                       : integer              := 32;
19      C_DWIDTH                       : integer              := 64;
20      C_NUM_CS                       : integer              := 1;
21      C_NUM_CE                       : integer              := 2;
22      C_WRFIFO_DWIDTH                : integer              := 32
23    );
24    port
25    (
26    --Test Portları
27       leds_go             : out std_logic_vector(0 to 3);
28     --Test Portları BİTİŞ
29
30      Bus2IP_Clk                     : in  std_logic;
31      Bus2IP_Reset                   : in  std_logic;
32      Bus2IP_Addr                    : in  std_logic_vector(0 to C_AWIDTH-1);
33      Bus2IP_Data                    : in  std_logic_vector(0 to C_DWIDTH-1);
34      Bus2IP_BE                      : in  std_logic_vector(0 to C_DWIDTH/8-1);
35      Bus2IP_Burst                   : in  std_logic;
36      Bus2IP_CS                      : in  std_logic_vector(0 to C_NUM_CS-1);
37      Bus2IP_CE                      : in  std_logic_vector(0 to C_NUM_CE-1);
38      Bus2IP_RdCE                    : in  std_logic_vector(0 to C_NUM_CE-1);
39      Bus2IP_WrCE                    : in  std_logic_vector(0 to C_NUM_CE-1);
40      Bus2IP_RdReq                   : in  std_logic;
41      Bus2IP_WrReq                   : in  std_logic;
42      IP2Bus_Data                    : out std_logic_vector(0 to C_DWIDTH-1);
43      IP2Bus_Retry                   : out std_logic;
44      IP2Bus_Error                   : out std_logic;
45      IP2Bus_ToutSup                 : out std_logic;
46      IP2Bus_RdAck                   : out std_logic;
47      IP2Bus_WrAck                   : out std_logic;
48      Bus2IP_MstError                : in  std_logic;
49      Bus2IP_MstLastAck              : in  std_logic;
50      Bus2IP_MstRdAck                : in  std_logic;
51      Bus2IP_MstWrAck                : in  std_logic;
52      Bus2IP_MstRetry                : in  std_logic;
53      Bus2IP_MstTimeOut              : in  std_logic;
54      IP2Bus_Addr                    : out std_logic_vector(0 to C_AWIDTH-1);
55      IP2Bus_MstBE                   : out std_logic_vector(0 to C_DWIDTH/8-1);
56      IP2Bus_MstBurst                : out std_logic;
57      IP2Bus_MstBusLock              : out std_logic;
58      IP2Bus_MstNum                  : out std_logic_vector(0 to 4);
59      IP2Bus_MstRdReq                : out std_logic;
60      IP2Bus_MstWrReq                : out std_logic;
61      IP2IP_Addr                     : out std_logic_vector(0 to C_AWIDTH-1);
62      IP2WFIFO_RdReq                 : out std_logic;
63      WFIFO2IP_Data                  : in  std_logic_vector(0 to C_WRFIFO_DWIDTH-1);
```

25.11.2010 03:10 :: page 1

```
64      WFIFO2IP_RdAck                : in  std_logic;
65      WFIFO2IP_AlmostEmpty          : in  std_logic;
66      WFIFO2IP_Empty                : in  std_logic
67    );
68  end entity user_logic;
69
70  architecture IMP of user_logic is
71  --#####  Master Sinyalleri  #####--
72    type type_state_mst is (state_mst_idle, state_mst_wait_ack);
73    signal state_mst : type_state_mst;
74    signal mstRdReq : STD_LOGIC;
75    signal mstWrReq : STD_LOGIC;
76
77      -- Master Arayüz Sinyalleri
78    signal mst_go : STD_LOGIC;                      -- IN << Rising Edge Active
79    signal mst_rd_wr : STD_LOGIC;                     -- IN << 0 ise read, 1 ise write
80    signal mst_burst_enable : STD_LOGIC;                -- IN << 1 ise active
81    signal mst_bus_lock_enable : STD_LOGIC;             -- IN << 1 ise active
82    signal mst_num : STD_LOGIC_VECTOR(0 to 4);            -- IN <<
83    signal mst_out_addr : STD_LOGIC_VECTOR(0 to C_AWIDTH-1);    -- IN <<
84    --signal mst_be : STD_LOGIC_VECTOR(0 to C_DWIDTH/8-1);    -- IN <<
85    signal mst_local_addr : STD_LOGIC_VECTOR(0 to C_AWIDTH-1);  -- IN <<
86    signal mstBusy : STD_LOGIC;                     -- OUT >>
87    signal mstSuccess : STD_LOGIC;                      -- OUT >> -- Rising Edge Active
88    signal mstError : STD_LOGIC;                      -- OUT >> -- Rising Edge Active
89
90    signal mst_read_reg : STD_LOGIC_VECTOR(0 to 31);        -- OUT >>
91    signal mst_write_reg : STD_LOGIC_VECTOR(0 to 31);       -- IN <<
92
93
94    --#####  Slave Sinyalleri   #####--
95     constant NUMBER_OF_REGS           : integer          := 16;
96     --constant RO_START_ADDR            : std_logic_vector(0 to 31) := C_BASEADDR + NUMBER_OF_REGS*4;
97     constant NUMBER_OF_RO_REGS        : integer          := 4;
98
99    type type_regs_slave is array(0 to (NUMBER_OF_REGS + NUMBER_OF_RO_REGS - 1 )) of std_logic_vector(0
      to 31);
100   signal regs_slave : type_regs_slave;
101
102   --Slave Read
103     type type_state_rd is (wait_rd_req, count_rd, rd_ack);
104     signal state_rd : type_state_rd;
105   signal next_state_rd : type_state_rd;
106   signal counter_rd : integer;
107   signal count_times_rd : integer;
108   signal ack_rd : STD_LOGIC;
109   signal slave_data_rd : std_logic_vector(0 to C_DWIDTH-1);
110   signal burst_active_rd : STD_LOGIC;
111
112   --Slave Write
113     type type_state_wr is (wait_wr_req, count_wr, wr_ack);
114     signal state_wr : type_state_wr;
115   signal next_state_wr : type_state_wr;
116   signal counter_wr : integer;
117   signal count_times_wr : integer;
118   signal ack_wr : STD_LOGIC;
119   signal burst_active_wr : STD_LOGIC;
120
121
122   -- Komut Alma Sinyaller, Tanımlamalar
123   type COMMAND_SM_TYPE is (   STATE_WAIT_DATA, STATE_INIT, STATE_ARGB, STATE_KAYNAK_ADDR,
124                       STATE_HEDEF_ADDR, STATE_FRM_BUF_ROW_SIZE, STATE_BLT_ROW_SIZE,
125                     STATE_BLT_COL_SIZE, STATE_FLAGS, STATE_BLT_GO);
```

```
126   signal command_state              : COMMAND_SM_TYPE;
127   signal command_state_next         : COMMAND_SM_TYPE;
128   signal ip2wfifo_rdreq_cm          : std_logic;
129
130       -- BLT Komut Girdileri
131     signal ARGB_in :   STD_LOGIC_VECTOR (0 to 31);
132      signal kaynak_addr_in :  STD_LOGIC_VECTOR (0 to 31);
133      signal hedef_addr_in :  STD_LOGIC_VECTOR (0 to 31);
134     signal frm_buf_row_size_in : integer;
135     signal blt_row_size_in : integer;
136     signal blt_col_size_in : integer;
137     --flags
138     signal kaynak_in :  STD_LOGIC;
139      signal boolean_op_in :  STD_LOGIC_VECTOR (0 to 1);
140     signal transparan_in :  STD_LOGIC_VECTOR (0 to 1);
141      signal alpha_in :  STD_LOGIC;
142      signal kose_no_in :  STD_LOGIC_VECTOR (0 to 1);
143     signal kaynak_guncelle_in : STD_LOGIC_VECTOR (0 to 1);
144
145   constant BLT_HEADER : std_logic_vector(0 to 31) := x"AA0000AA";
146
147     --BitBLT Process Sinyaller, Tanımlamalar
148   type blt_state_type is (IDLE, KAYNAKTAN_OKU, HEDEFE_YAZ, HEDEFTEN_OKU, KAYNAGA_YAZ,
149         ARKA_PLANDAN_CKU, ADRES_HESAP);
150
151   signal blt_state        : blt_state_type;
152
153   signal blt_go : STD_LOGIC;  --<<  IN
154   signal blt_busy : STD_LOGIC;  -->> OUT
155
156   signal kaynak_reg : STD_LOGIC;
157   signal boolean_op_reg : STD_LOGIC_VECTOR (0 to 1);
158   signal ARGB_reg : STD_LOGIC_VECTOR (0 to 31);
159   signal kaynak_addr_reg : STD_LOGIC_VECTOR (0 to 31);
160   signal hedef_addr_reg : STD_LOGIC_VECTOR (0 to 31);
161   signal arka_plan_addr_reg : STD_LOGIC_VECTOR (0 to 31);
162   signal transparan_reg : STD_LOGIC_VECTOR (0 to 1);
163   signal alpha_reg : STD_LOGIC;
164   signal kose_no_reg : STD_LOGIC_VECTOR (0 to 1);
165   signal frm_buf_row_size_reg : integer;
166   signal blt_row_size_reg : integer;
167   signal blt_col_size_reg : integer;
168   signal kaynak_guncelle_reg : STD_LOGIC_VECTOR (0 to 1);
169
170   constant arka_plan_addr_regno : integer := 2;
171
172
173   --signal hedef_addr : STD_LOGIC_VECTOR (0 to 31);
174   signal hedef_yaz_data : STD_LOGIC_VECTOR (0 to 31);
175   signal hedef_oku_data : STD_LOGIC_VECTOR (0 to 31);
176
177   --signal kaynak_addr : STD_LOGIC_VECTOR (0 to 31);
178   signal kaynak_yaz_data : STD_LOGIC_VECTOR (0 to 31);
179   signal kaynak_oku_data : STD_LOGIC_VECTOR (0 to 31);
180
181   signal arka_plan_oku_data : STD_LOGIC_VECTOR (0 to 31);
182
183   signal okunacak : STD_LOGIC;
184   signal yazilma_bekle : STD_LOGIC;
185   signal row_indexer : integer;
186
187 begin
188
```

```
189 -- ###################### Bit BLT ######################
190
191
192 BLT_PROC : process( blt_go, Bus2IP_Clk, Bus2IP_Reset ) is
193 variable v_hedef_addr : STD_LOGIC_VECTOR (0 to 31);
194 variable v_kaynak_addr : STD_LOGIC_VECTOR (0 to 31);
195 variable v_arka_plan_addr : STD_LOGIC_VECTOR (0 to 31);
196 variable v_blt_col_size : integer;
197 variable v_hedefe_yaz : STD_LOGIC_VECTOR (0 to 31);
198
199 variable v_alfa : STD_LOGIC_VECTOR (0 to 7);
200 variable v_renk_kaynak : STD_LOGIC_VECTOR (0 to 7);
201 variable v_renk_hedef : STD_LOGIC_VECTOR (0 to 7);
202 variable v_carpim : STD_LOGIC_VECTOR (0 to 15);
203 variable v_alfa_red : STD_LOGIC_VECTOR (0 to 7);
204 variable v_alfa_green : STD_LOGIC_VECTOR (0 to 7);
205 variable v_alfa_blue : STD_LOGIC_VECTOR (0 to 7);
206
207
208
209 begin
210
211     if ( Bus2IP_Clk'event and Bus2IP_Clk = '1' ) then
212      if ( Bus2IP_Reset = '1' ) then
213
214
215        blt_state <= IDLE;
216
217        kaynak_reg <= '0';
218        boolean_op_reg <= (others => '0');
219        ARGB_reg <= (others => '0');
220        kaynak_addr_reg <= (others => '0');
221        hedef_addr_reg <= (others => '0');
222        transparan_reg <= (others => '0');
223        alpha_reg <= '0';
224        kose_no_reg <= (others => '0');
225        frm_buf_row_size_reg <= 0;
226        blt_row_size_reg <= 0;
227        blt_col_size_reg <= 0;
228        kaynak_guncelle_reg <= (others => '0');
229        arka_plan_addr_reg <= (others => '0');
230
231        blt_busy <= '0';
232
233        mst_go <= '0';
234        mst_rd_wr <= '1'; -- 0 ise read, 1 ise write
235        mst_burst_enable <= '0';-- 1 ise active
236        mst_bus_lock_enable <= '0';-- 1 ise active
237        mst_num <= "00001";
238        mst_out_addr <= (others => '0');
239        mst_local_addr <= C_BASEADDR;
240        mst_write_reg <= (others => '0');
241
242        hedef_yaz_data <= (others => '0');
243        hedef_oku_data <= (others => '0');
244
245        kaynak_yaz_data <= (others => '0');
246        kaynak_oku_data <= (others => '0');
247
248        arka_plan_oku_data <= (others => '0');
249
250        okunacak <= '1';
251        yazilma_bekle <= '0';
```

```
252         row_indexer <= 0;
253
254       else--!reset
255         --set default values
256         mst_go <= '0';
257
258         case blt_state is
259           when IDLE =>
260             leds_go <= "0000";
261             blt_busy <= '0';
262             if blt_go = '1' then
263               leds_go <= "0001";
264               blt_busy <= '1';
265
266               kaynak_reg <= kaynak_in;
267               boolean_op_reg <= boolean_op_in;
268               ARGB_reg <= ARGB_in;
269               kaynak_addr_reg <= kaynak_addr_in;
270               hedef_addr_reg <= hedef_addr_in;
271               transparan_reg <= transparan_in;
272               alpha_reg <= alpha_in;
273               kose_no_reg <= kose_no_in;
274               frm_buf_row_size_reg <= frm_buf_row_size_in;
275               blt_row_size_reg <= blt_row_size_in;
276               blt_col_size_reg<= blt_col_size_in;
277               kaynak_guncelle_reg <= kaynak_guncelle_in;
278               arka_plan_addr_reg <= regs_slave(arka_plan_addr_regno);
279
280               row_indexer <= 1;
281
282               if (blt_row_size_in > 0) and (blt_col_size_in > 0) then
283                 if kaynak_in = '0' then  --hedefe ARGB yazılacak
284                   blt_state <= HEDEFE_YAZ;
285                 else
286                   okunacak <= '1';
287                   blt_state <= KAYNAKTAN_OKU;
288                 end if;
289               else
290                 blt_state <= IDLE;
291               end if;
292
293             end if; --blt_go
294
295           when KAYNAKTAN_OKU =>
296             if okunacak = '1' then
297
298               if mstBusy = '0' then
299                 okunacak <= '0';
300                 mst_rd_wr <= '0';--okuuu
301                 mst_out_addr <= kaynak_addr_reg;
302                 mst_go <= '1';
303                 leds_go <= "0010";
304               end if;
305
306             else
307               if (mstSuccess = '1') or (mstError = '1') then
308                 okunacak <= '1';
309                 leds_go <= "0011";
310                 kaynak_oku_data <= mst_read_reg;
311
312                 if (  (transparan_reg = "01") and (mst_read_reg = ARGB_reg) )
313
314                   or
```

```
315
316                 (  transparan_reg = "10" and (mst_read_reg /= ARGB_reg)  )
317                 then
318                 blt_state <= ADRES_HESAP;
319
320               else
321
322                 if (boolean_op_reg = "00") then
323                   if (alpha_reg = '0') then
324                     blt_state <= HEDEFE_YAZ;
325                   else
326                     okunacak <= '1';
327                     blt_state <= HEDEFTEN_OKU;
328                   end if;
329                 else
330                   okunacak <= '1';
331                   blt_state <= HEDEFTEN_OKU;
332                 end if;
333
334               end if;--transparanlık
335
336
337             end if;--mst_busy
338
339           end if;--okunacak
340
341         when HEDEFE_YAZ =>
342
343           if yazilma_bekle = '1'  then
344             if (mstSuccess = '1') or (mstError = '1') then
345               ----------- Next State Logic -----------
346               if (kaynak_guncelle_reg = "00") or (kaynak_reg = '0') then
347                 blt_state <= ADRES_HESAP;
348               elsif kaynak_guncelle_reg = "01" then
349                 blt_state <= KAYNAGA_YAZ;
350               elsif kaynak_guncelle_reg = "10" then
351                 blt_state <= ARKA_PLANDAN_OKU;
352               else
353                 blt_state <= ADRES_HESAP;
354               end if;
355               ----------- Next State Logic ----------- END
356               yazilma_bekle <= '0';
357             end if;
358
359           else
360             ----------- Yazilacak Veriyi Hesapla -----------
361             if kaynak_reg = '0' then  --ARGB yazılacak
362               v_hedefe_yaz := ARGB_reg;
363             else
364
365               if boclean_op_reg = "01" then --AND
366                 v_hedefe_yaz := kaynak_oku_data and hedef_oku_data;
367               elsif boolean_op_reg = "10" then --OR
368                  v_hedefe_yaz := kaynak_oku_data or hedef_oku_data;
369               elsif boolean_op_reg = "11" then  --XOR
370                 v_hedefe_yaz := kaynak_oku_data xor hedef_oku_data;
371               else -- "00": Bool NON
372                 if alpha_reg = '0' then --üzerine yaz
373                   v_hedefe_yaz := kaynak_oku_data;
374                 else --alfa karıştırma
375
376                   v_alfa := kaynak_oku_data(0 to 7);
377                   v_renk_kaynak := kaynak_oku_data(8 to 15); --red
```

```
378                   v_renk_hedef := hedef_oku_data(8 to 15); --red
379                   v_carpim := v_alfa * v_renk_hedef;
380                   if (CONV_INTEGER(v_renk_kaynak) + CONV_INTEGER(v_renk_hedef) -
                      CONV_INTEGER(v_carpim(0 to 7)) ) > 255 then
381                     v_alfa_red := (others => '1');
382                   else
383                     v_alfa_red := v_renk_kaynak + v_renk_hedef - v_carpim(0 to 7);
384                   end if;
385
386                   v_renk_kaynak := kaynak_oku_data(16 to 23); --green
387                   v_renk_hedef := hedef_oku_data(16 to 23); --green
388                   v_carpim := v_alfa * v_renk_hedef;
389                   if (CONV_INTEGER(v_renk_kaynak) + CONV_INTEGER(v_renk_hedef) -
                      CONV_INTEGER(v_carpim(0 to 7)) ) > 255 then
390                     v_alfa_green := (others => '1');
391                   else
392                     v_alfa_green := v_renk_kaynak + v_renk_hedef - v_carpim(0 to 7);
393                   end if;
394
395                   v_renk_kaynak := kaynak_oku_data(24 to 31); --blue
396                   v_renk_hedef := hedef_oku_data(24 to 31); --blue
397                   v_carpim := v_alfa * v_renk_hedef;
398                   if (CONV_INTEGER(v_renk_kaynak) + CONV_INTEGER(v_renk_hedef) -
                      CONV_INTEGER(v_carpim(0 to 7)) ) > 255 then
399                     v_alfa_blue := (others => '1');
400                   else
401                     v_alfa_blue := v_renk_kaynak + v_renk_hedef - v_carpim(0 to 7);
402                   end if;
403
404                   v_hedefe_yaz := v_alfa & v_alfa_red & v_alfa_green & v_alfa_blue;
405
406                 end if;
407
408               end if; --boolean_op_reg...
409
410             end if; --kaynak_reg...
411             ----------- Yazilacak Veriyi Hesapla ----------- END
412
413
414             ----------- Master arayüzü ile yaz -----------
415             if mstBusy = '0' then
416               mst_rd_wr <= '1';--yazzzz
417               mst_out_addr <= hedef_addr_reg;
418               mst_write_reg <= v_hedefe_yaz;
419               mst_go <= '1';
420
421               yazilma_bekle <= '1';
422
423               leds_go <= "0100";
424             end if;
425             ----------- Master arayüzü ile yaz ----------- END
426
427           end if;--yazilma_bekle
428
429
430         when HEDEFTEN_OKU =>
431           if okunacak = '1' then
432
433             if mstBusy = '0' then
434               okunacak <= '0';
435               mst_rd_wr <= '0';--okuuu
436               mst_out_addr <= hedef_addr_reg;
437               mst_go <= '1';
```

```
438                 leds_go <= "0101";
439               end if;
440
441             else
442               if (mstSuccess = '1') or (mstError = '1') then
443                 okunacak <= '1';
444                 hedef_oku_data <= mst_read_reg;
445                 blt_state <= HEDEFE_YAZ;
446                 leds_go <= "0110";
447               end if;
448             end if;
449
450           when KAYNAGA_YAZ =>
451
452             if yazilma_bekle = '1'  then
453               if (mstSuccess = '1') or (mstError = '1') then
454                 blt_state <= ADRES_HESAP;
455                 yazilma_bekle <= '0';
456               end if;
457
458             else
459
460               if mstBusy = '0' then
461                 mst_rd_wr <= '1';--yazzzz
462                 mst_out_addr <= kaynak_addr_reg;
463                 mst_go <= '1';
464
465                 if kaynak_guncelle_reg ="01" then
466                   mst_write_reg <= ARGB_reg;
467                 elsif kaynak_guncelle_reg ="10" then
468                   mst_write_reg <= arka_plan_oku_data;
469                 else
470                    mst_go <= '0';
471                 end if;
472                 yazilma_bekle <= '1';
473
474                 leds_go <= "0111";
475               end if;
476
477             end if;--yazilma_bekle
478
479           when ARKA_PLANDAN_OKU =>
480             if okunacak = '1' then
481
482               if mstBusy = '0' then
483                 okunacak <= '0';
484                 mst_rd_wr <= '0';--okuuu
485                 mst_out_addr <= arka_plan_addr_reg;
486                 mst_go <= '1';
487                 leds_go <= "1000";
488               end if;
489
490             else
491               if (mstSuccess = '1') or (mstError = '1') then
492                 okunacak <= '1';
493                 arka_plan_oku_data <= mst_read_reg;
494                 blt_state <= KAYNAGA_YAZ;
495                 leds_go <= "1001";
496               end if;
497             end if;
498
499           when ADRES_HESAP =>
500             v_hedef_addr := hedef_adcr_reg;
```

```
501             v_kaynak_addr := kaynak_addr_reg;
502             v_arka_plan_addr := arka_plan_addr_reg;
503             v_blt_col_size := blt_col_size_reg;
504
505             leds_go <= "1010";
506
507             --Bir sonraki adresleri hesapla
508             --bitblt bitmişse IDLE'a dön
509             if v_blt_col_size > 0 then
510
511               if row_indexer < blt_row_size_reg then
512
513                 if (kose_no_reg = "00") or (kose_no_reg = "11") then
514                   v_hedef_addr := v_hedef_addr + 4;
515                   v_kaynak_addr := v_kaynak_addr + 4;
516                   v_arka_plan_addr := v_arka_plan_addr + 4;
517                 else
518                   v_hedef_addr := v_hedef_addr - 4;
519                   v_kaynak_addr := v_kaynak_addr - 4;
520                   v_arka_plan_addr := v_arka_plan_addr - 4;
521                 end if;
522
523                 row_indexer <= row_indexer + 1;
524
525               else --satır atla
526                 row_indexer <= 1;
527                 v_blt_col_size := v_blt_col_size - 1;
528
529                 case kose_no_reg is
530                   when "00" =>
531                     v_hedef_addr := v_hedef_addr - ((blt_row_size_reg-1)*4) - (frm_buf_row_size_reg *
                      4);
532                     v_kaynak_addr := v_kaynak_addr - ((blt_row_size_reg-1)*4) - (frm_buf_row_size_reg *
                      4);
533                     v_arka_plan_addr := v_arka_plan_addr - ((blt_row_size_reg-1)*4) -
                      (frm_buf_row_size_reg * 4);
534                   when "01" =>
535                     v_hedef_addr := v_hedef_addr + ((blt_row_size_reg-1)*4) - (frm_buf_row_size_reg *
                      4);
536                     v_kaynak_addr := v_kaynak_addr + ((blt_row_size_reg-1)*4) - (frm_buf_row_size_reg *
                      4);
537                     v_arka_plan_addr := v_arka_plan_addr + ((blt_row_size_reg-1)*4) -
                      (frm_buf_row_size_reg * 4);
538                   when "10" =>
539                     v_hedef_addr := v_hedef_addr + ((blt_row_size_reg-1)*4) + (frm_buf_row_size_reg *
                      4);
540                     v_kaynak_addr := v_kaynak_addr + ((blt_row_size_reg-1)*4) + (frm_buf_row_size_reg *
                      4);
541                     v_arka_plan_addr := v_arka_plan_addr + ((blt_row_size_reg-1)*4) +
                      (frm_buf_row_size_reg * 4);
542                   when "11" =>
543                     v_hedef_addr := v_hedef_addr - ((blt_row_size_reg-1)*4) + (frm_buf_row_size_reg *
                      4);
544                     v_kaynak_addr := v_kaynak_addr - ((blt_row_size_reg-1)*4) + (frm_buf_row_size_reg *
                      4);
545                     v_arka_plan_addr := v_arka_plan_addr - ((blt_row_size_reg-1)*4) +
                      (frm_buf_row_size_reg * 4);
546                   when others =>
547                     null;
548                 end case;
549
550               end if;--row_indexer < (blt_row_size_reg)
551
```

```
552
553             end if;--v_blt_col_size > 0 then
554
555
556             --next_state_logic
557
558
559             if v_blt_col_size <= 0 then
560               blt_state <= IDLE;
561             else
562               if kaynak_reg = '0' then --kaynak ARGD
563                 blt_state <= HEDEFE_YAZ;
564               else
565                 blt_state <= KAYNAKTAN_OKU;
566               end if;
567             end if;
568
569             hedef_addr_reg <= v_hedef_addr;
570             kaynak_addr_reg <= v_kaynak_addr;
571             arka_plan_addr_reg <= v_arka_plan_addr;
572             blt_col_size_reg <= v_blt_col_size;
573
574
575         end case;--case blt_state
576
577       end if; --if !reset
578
579     end if;-- if clk'event
580
581 end process BLT_PROC;
582
583 -- ###################### Bit BLT ###################### END
584
585 -- ###################### Komut Baslatma ######################
586
587 COMMAND_PROC : process( WFIFO2IP_empty, WFIFO2IP_RdAck, Bus2IP_Clk, Bus2IP_Reset ) is
588   begin
589
590     if ( Bus2IP_Clk'event and Bus2IP_Clk = '1' ) then
591
592       if ( Bus2IP_Reset = '1' ) then
593         ip2wfifo_rdreq_cm <= '0';
594            command_state   <= STATE_WAIT_DATA;
595         command_state_next <= STATE_INIT;
596
597
598        --leds_go <= "0110";
599
600       else
601           -- set defaults
602         ip2wfifo_rdreq_cm <= '0';
603        blt_go <= '0';
604
605        case command_state is
606
607             when STATE_WAIT_DATA =>
608
609               if ( WFIFO2IP_empty = '0' ) then
610                 ip2wfifo_rdreq_cm <= '1';
611                 command_state       <= command_state_next;
612               end if;
613
614           when STATE_INIT =>
```

```
615
616             if ( WFIFO2IP_RdAck = '1' ) then
617
618                 if WFIFO2IP_Data = BLT_HEADER then
619                 command_state <= STATE_WAIT_DATA;
620                 command_state_next <= STATE_ARGB;
621                 --leds_go <= "1110";
622               else
623                 command_state <= STATE_WAIT_DATA;
624                 command_state_next <= STATE_INIT;
625                 --leds_go <= "1101";
626               end if;
627
628                 end if;
629
630             when STATE_ARGB =>
631             if ( WFIFO2IP_RdAck = '1' ) then
632               command_state <= STATE_WAIT_DATA;
633               command_state_next <= STATE_KAYNAK_ADDR;
634               ARGB_in <= WFIFO2IP_Data;
635               --leds_go <= "1100";
636                 end if;
637
638           when STATE_KAYNAK_ADDR =>
639             if ( WFIFO2IP_RdAck = '1' ) then
640               command_state <= STATE_WAIT_DATA;
641               command_state_next <= STATE_HEDEF_ADDR;
642               kaynak_addr_in <= WFIFO2IP_Data;
643               --leds_go <= "1011";
644                 end if;
645
646           when STATE_HEDEF_ADDR =>
647             if ( WFIFO2IP_RdAck = '1' ) then
648               command_state <= STATE_WAIT_DATA;
649               command_state_next <= STATE_FRM_BUF_ROW_SIZE;
650               hedef_addr_in <= WFIFO2IP_Data;
651               --leds_go <= "1010";
652                 end if;
653
654           when STATE_FRM_BUF_ROW_SIZE =>
655             if ( WFIFO2IP_RdAck = '1' ) then
656               command_state <= STATE_WAIT_DATA;
657               command_state_next <= STATE_BLT_ROW_SIZE;
658               frm_buf_row_size_in <= CONV_INTEGER(WFIFO2IP_Data);
659               --leds_go <= "1001";
660                 end if;
661
662           when STATE_BLT_ROW_SIZE =>
663             if ( WFIFO2IP_RdAck = '1' ) then
664               command_state <= STATE_WAIT_DATA;
665               command_state_next <= STATE_BLT_COL_SIZE;
666               blt_row_size_in <= CONV_INTEGER(WFIFO2IP_Data);
667               --leds_go <= "1000";
668                 end if;
669
670           when STATE_BLT_COL_SIZE =>
671             if ( WFIFO2IP_RdAck = '1' ) then
672               command_state <= STATE_WAIT_DATA;
673               command_state_next <= STATE_FLAGS;
674               blt_col_size_in <= CONV_INTEGER(WFIFO2IP_Data);
675               --leds_go <= "0111";
676                 end if;
677
```

```
678           when STATE_FLAGS =>
679             if ( WFIFO2IP_RdAck = '1' ) then
680               command_state <= STATE_BLT_GO;
681                 kaynak_in <= WFIFO2IP_Data(31);
682                  boolean_op_in <= WFIFO2IP_Data(29 to 30);
683                  transparan_in <= WFIFO2IP_Data(27 to 28);
684                  alpha_in <= WFIFO2IP_Data(26);
685                  kose_no_in <= WFIFO2IP_Data(24 to 25);
686                 kaynak_guncelle_in <= WFIFO2IP_Data(22 to 23);
687                 --leds_go <= "0110";
688                 end if;
689
690           when STATE_BLT_GO =>
691             if blt_busy = '0' then
692
693               --leds_go <= "0000";
694
695               blt_go <= '1';
696               command_state <= STATE_WAIT_DATA;
697               command_state_next <= STATE_INIT;
698             end if;
699
700             when others =>
701                command_state <= STATE_WAIT_DATA;
702             command_state_next <= STATE_INIT;
703           end case;
704
705       end if;--!reset
706
707     end if;--if clk event
708
709
710   end process COMMAND_PROC;
711
712   IP2WFIFO_RdReq <= ip2wfifo_rdreq_cm;
713
714   -- ###################### Komut Baslatma ###################### END
715
716
717   -- ###################### Master ######################
718     master_process : process( Bus2IP_Reset, Bus2IP_Clk, mst_go )
719     begin
720
721       if(Bus2IP_Reset = '1') then
722
723           IP2Bus_Addr                    <= (others => '0');
724            IP2Bus_MstBE                   <= (others => '0');
725            IP2Bus_MstBurst                <= '0';
726            IP2Bus_MstBusLock              <= '0';
727            IP2Bus_MstNum                  <= (others => '0');
728            IP2IP_Addr                     <= C_BASEADDR;--(others => '0');
729           mstRdReq               <= '0';
730           mstWrReq               <= '0';
731           mstBusy                  <= '0';
732           mstSuccess               <= '0';
733           mstError               <= '0';
734
735           state_mst <= state_mst_idle;
736
737           --leds_2 <= "1111";
738
739
740       elsif (Bus2IP_Clk'event and Bus2IP_Clk = '1') then
```

```
741       --set defaults
742       mstSuccess                <= '0';
743       mstError                  <= '0';
744
745
746       case state_mst is
747
748         when state_mst_idle =>
749           mstBusy                 <= '0';
750
751           if mst_go = '1' then
752             mstBusy                 <= '1';
753
754             --leds_2 <= "1100";
755
756             if mst_out_addr(29 to 31) = "000" then
757               IP2Bus_MstBE <= "11110000";
758             else
759               IP2Bus_MstBE <= "00001111";
760             end if;
761
762             --Read----------------------------
763             if mst_rd_wr = '0' then
764               mstRdReq <= '1';
765
766             --Write---------------------------
767             else
768               mstWrReq <= '1';
769             end if;
770
771             IP2Bus_MstBurst <= mst_burst_enable;
772             IP2Bus_MstBusLock <= mst_bus_lock_enable;
773             IP2Bus_Addr <= mst_out_addr;
774             IP2IP_Addr <= mst_local_addr;
775
776             if mst_burst_enable = '1' then
777               IP2Bus_MstNum <= mst_num;
778             else
779               IP2Bus_MstNum <= "00001";
780             end if;
781
782             state_mst <= state_mst_wait_ack;
783
784           end if; --mst_go
785
786
787
788         when state_mst_wait_ack =>
789           if (Bus2IP_MstLastAck = '1') or (Bus2IP_MstError = '1') or
790              (Bus2IP_MstRetry = '1') or (Bus2IP_MstTimeOut = '1')
791              then
792
793             if (Bus2IP_MstLastAck = '1') then
794               --led_left <= led_left(1) & led_left(0);
795               mstSuccess              <= '1';
796               --leds_2 <= "1110";
797             elsif (Bus2IP_MstError = '1') then
798               mstError                <= '1';
799               --leds_2 <= "1101";
800             elsif  (Bus2IP_MstRetry = '1') then
801               mstError                <= '1';
802               --leds_2 <= "1100";
803             elsif (Bus2IP_MstTimeOut = '1') then
```

```
804               mstError                  <= '1';
805               --leds_2 <= "1011";
806             else
807               mstError                  <= '1';
808               --leds_2 <= "1010";
809             end if;
810
811             state_mst <= state_mst_idle;
812             IP2Bus_MstBurst <= '0';
813             IP2Bus_MstBusLock <= '0';
814             mstRdReq <= '0';
815             mstWrReq <= '0';
816
817
818           end if;
819
820       end case;
821
822     end if;
823
824   end process;
825
826   IP2Bus_MstRdReq <= mstRdReq;
827   IP2Bus_MstWrReq <= mstWrReq;
828
829   -- ---------------------- Master ---------------------- SON
830
831
832   -- ---------------------- Slave Write ----------------------
833   slave_write_process : process( Bus2IP_Reset, Bus2IP_Clk, Bus2IP_WrReq )
834   variable reg_addr : integer;
835   variable data : std_logic_vector(0 to 31);
836   variable addr : std_logic_vector(0 to 31);
837   begin
838
839     if(Bus2IP_Reset = '1') then
840
841         --For Test
842         --led_left <= "01";
843         --rd_data <= x"AABBCCDDEEFF1122";
844         --For Test END
845
846         state_wr <= wait_wr_req;
847         next_state_wr <= wait_wr_req;
848         counter_wr <= 0;
849         count_times_wr <= 0;
850         ack_wr <= '0';
851         burst_active_wr <= '0';
852
853         regs_slave <= (others => (others => '0'));
854
855
856     elsif (Bus2IP_Clk'event and Bus2IP_Clk = '1') then
857     --Set Defaults
858
859       case state_wr is
860         when wait_wr_req =>
861           if Bus2IP_WrReq = '1' then
862
863             if Bus2IP_BE(0 to 3) = "1111" then
864               data := Bus2IP_Data(0 to 31);
865             else
866               data := Bus2IP_Data(32 to 63);
```

```
867             end if;
868
869             addr := Bus2IP_Addr;
870             reg_addr := CONV_INTEGER(Bus2IP_Addr(24 to 31));
871
872             if burst_active_wr = '0' then
873               if Bus2IP_Burst = '1' then
874                 ack_wr <= '1';
875                 burst_active_wr <= '1';
876               else
877                 state_wr <= count_wr;
878                 next_state_wr <= wr_ack;
879                 count_times_wr <= 4;
880               end if;
881
882             else
883               if Bus2IP_Burst = '0' then
884                 state_wr <= wr_ack;
885                  next_state_wr <= wr_ack;
886                  burst_active_wr <= '0';
887               else
888               end if;
889
890             end if;
891
892             if mstRdReq = '1' then --mst reg'e koyalım
893               mst_read_reg <= data(0 to 31);
894             else --reg'e koyalım
895
896               if (reg_addr / 4) <= (NUMBER_OF_REGS - 1) then --readonly alana koyamayız
897
898                 for index in 0 to (NUMBER_OF_REGS - 1) loop
899                   if index = (reg_addr / 4) then
900                     regs_slave(index) <= data;
901                   end if;
902                 end loop;
903
904               end if;
905
906             end if;
907
908           end if;
909
910         when count_wr =>
911           if counter_wr >= count_times_wr then
912             state_wr <= next_state_wr;
913             counter_wr <= 0;
914           else
915             counter_wr <= counter_wr + 1;
916           end if;
917
918
919         when wr_ack =>
920           if ack_wr = '1' then
921             ack_wr <= '0';
922             state_wr <= wait_wr_req;
923             --next_state_wr <= wait_wr_req;
924           else
925             ack_wr <= '1';
926           end if;
927
928
929       end case;
```

```
930
931       end if;
932
933     end process; --slave_write_process
934
935     IP2Bus_WrAck <= ack_wr;
936
937      -- ###################### Slave Write ######################---SON
938
939
940      -- ###################### Slave Read ######################
941
942     slave_read_process : process( Bus2IP_Reset, Bus2IP_Clk, Bus2IP_RdReq )
943     variable addr : std_logic_vector(0 to C_AWIDTH-1);
944     variable reg_addr : integer;
945     begin
946
947       if(Bus2IP_Reset = '1') then
948           --For Test
949           --led_right <= "10";
950           slave_data_rd <= (others => '0');--x"ABCDEF12ABCDEF12";
951           --For Test END
952           state_rd <= wait_rd_req;
953           next_state_rd <= wait_rd_req;
954           counter_rd <= 0;
955           count_times_rd <= 0;
956           ack_rd <= '0';
957           burst_active_rd <= '0';
958
959
960       elsif (Bus2IP_Clk'event and Bus2IP_Clk = '1') then
961         case state_rd is
962           when wait_rd_req =>
963             if Bus2IP_RdReq = '1' then
964               addr := Bus2IP_Addr;
965               reg_addr := CONV_INTEGER(addr(24 to 31));
966
967               if mstWrReq = '1' then --mst datayı koyalım
968                 slave_data_rd <= mst_write_reg & mst_write_reg;
969               else --reçister data koyalım
970                 for index in 0 to (NUMBER_OF_REGS + NUMBER_OF_RO_REGS - 1) loop
971                   if (4*index) = reg_addr then
972                     slave_data_rd <= regs_slave(index) & regs_slave(index);
973                   end if;
974                 end loop;
975               end if;
976
977               if burst_active_rd = '0' then
978
979                 if Bus2IP_Burst = '1' then
980                   burst_active_rd <= '1';
981
982                 else
983                   state_rd <= count_rd;
984                   next_state_rd <= rd_ack;
985                   count_times_rd <= 4;
986
987                 end if;
988
989               else
990                 if Bus2IP_Burst = '0' then
991                   state_rd <= rd_ack;
992                    next_state_rd <= rd_ack;
```

```
993                   burst_active_rd <= '0';
994
995                 else
996                   ack_rd <= '1';
997                 end if;
998               end if;
999
1000
1001            end if;
1002
1003          when count_rd =>
1004            if counter_rd >= count_times_rd then
1005              state_rd <= next_state_rd;
1006            end if;
1007            counter_rd <= counter_rd + 1;
1008
1009          when rd_ack =>
1010            if ack_rd = '1' then
1011              ack_rd <= '0';
1012              state_rd <= wait_rd_req;
1013              --led_right <= led_right xor "11";
1014              --next_state_rd <= wait_rd_req;
1015            else
1016              ack_rd <= '1';
1017            end if;
1018
1019        end case;
1020
1021      end if;
1022
1023    end process;
1024
1025    IP2Bus_RdAck <= ack_rd;
1026    IP2Bus_Data <= slave_data_rd;
1027
1028    -- ###################### Slave Read Code ######################---SON
1029
1030
1031    --Default Dummy Signals
1032    IP2Bus_Error       <= '0';
1033    IP2Bus_Retry       <= '0';
1034    IP2Bus_ToutSup     <= '0';
1035
1036  end IMP;
```

Ek-2 Bresenham IP Çekirdeği Kodları

```
1  ---------------- breserham IP COre ----------------
2
3
4  library ieee;
5  use ieee.std_logic_1164.all;
6  use ieee.std_logic_arith.all;
7  use ieee.std_logic_unsigned.all;
8
9  library proc_common_v1_00_b;
10 use proc_common_v1_00_b.proc_common_pkg.all;
11
12
13 entity user_logic is
14   generic
15   (
16     --Kullanıcı parametreleri
17      C_BASEADDR                     : std_logic_vector(0 to 31)     := x"c3a00000";
18    --Kullanıcı parametreleri BİTİŞ
19
20     C_AWIDTH                       : integer              := 32;
21     C_DWIDTH                       : integer              := 64;
22     C_NUM_CS                       : integer              := 1;
23     C_NUM_CE                       : integer              := 2
24   );
25   port
26   (
27       --Test Portları
28       leds_2              : out std_logic_vector(0 to 3);
29     --Test Portları BİTİŞ
30
31     Bus2IP_Clk                     : in  std_logic;
32     Bus2IP_Reset                   : in  std_logic;
33     Bus2IP_Addr                    : in  std_logic_vector(0 to C_AWIDTH-1);
34     Bus2IP_Data                    : in  std_logic_vector(0 to C_DWIDTH-1);
35     Bus2IP_BE                      : in  std_logic_vector(0 to C_DWIDTH/8-1);
36     Bus2IP_Burst                   : in  std_logic;
37     Bus2IP_CS                      : in  std_logic_vector(0 to C_NUM_CS-1);
38     Bus2IP_CE                      : in  std_logic_vector(0 to C_NUM_CE-1);
39     Bus2IP_RdCE                    : in  std_logic_vector(0 to C_NUM_CE-1);
40     Bus2IP_WrCE                    : in  std_logic_vector(0 to C_NUM_CE-1);
41     Bus2IP_RdReq                   : in  std_logic;
42     Bus2IP_WrReq                   : in  std_logic;
43     IP2Bus_Data                    : out std_logic_vector(0 to C_DWIDTH-1);
44     IP2Bus_Retry                   : out std_logic;
45     IP2Bus_Error                   : out std_logic;
46     IP2Bus_ToutSup                 : out std_logic;
47     IP2Bus_RdAck                   : out std_logic;
48     IP2Bus_WrAck                   : out std_logic;
49     Bus2IP_MstError                : in  std_logic;
50     Bus2IP_MstLastAck              : in  std_logic;
51     Bus2IP_MstRdAck                : in  std_logic;
52     Bus2IP_MstWrAck                : in  std_logic;
53     Bus2IP_MstRetry                : in  std_logic;
54     Bus2IP_MstTimeOut              : in  std_logic;
55     IP2Bus_Addr                    : out std_logic_vector(0 to C_AWIDTH-1);
56     IP2Bus_MstBE                   : out std_logic_vector(0 to C_DWIDTH/8-1);
57     IP2Bus_MstBurst                : out std_logic;
58     IP2Bus_MstBusLock              : out std_logic;
59     IP2Bus_MstNum                  : out std_logic_vector(0 to 4);
60     IP2Bus_MstRdReq                : out std_logic;
61     IP2Bus_MstWrReq                : out std_logic;
62     IP2IP_Addr                     : out std_logic_vector(0 to C_AWIDTH-1)
63   );
```

```
64  end entity user_logic;
65
66
67  architecture IMP of user_logic is
68
69    --#####  Master Sinyalleri  #####--
70    type type_state_mst is (state_mst_idle, state_mst_wait_ack);
71    signal state_mst : type_state_mst;
72    signal mstRdReq : STD_LOGIC;
73    signal mstWrReq : STD_LOGIC;
74
75      -- Master Arayüz Sinyalleri
76    signal mst_go : STD_LOGIC;                      -- IN << Rising Edge Active
77    signal mst_rd_wr : STD_LOGIC;                     -- IN << 0 ise read, 1 ise write
78    signal mst_burst_enable : STD_LOGIC;                -- IN << 1 ise active
79    signal mst_bus_lock_enable : STD_LOGIC;             -- IN << 1 ise active
80    signal mst_num : STD_LOGIC_VECTOR(0 to 4);            -- IN <<
81    signal mst_out_addr : STD_LOGIC_VECTOR(0 to C_AWIDTH-1);    -- IN <<
82    --signal mst_be : STD_LOGIC_VECTOR(C to C_DWIDTH/8-1);      -- IN <<
83    signal mst_local_addr : STD_LOGIC_VECTOR(0 to C_AWIDTH-1);  -- IN <<
84    signal mstBusy : STD_LOGIC;                     -- OUT >>
85    signal mstSuccess : STD_LOGIC;                      -- OUT >> -- Rising Edge Active
86    signal mstError : STD_LOGIC;                      -- OUT >> -- Rising Edge Active
87
88    signal mst_read_reg : STD_LOGIC_VECTOR(0 to 31);        -- OUT >>
89    signal mst_write_reg : STD_LOGIC_VECTOR(0 to 31);       -- IN <<
90
91
92    --#####  Slave Sinyalleri   #####--
93     constant NUMBER_OF_REGS            : integer          := 16;
94     --constant RO_START_ADDR             : std_logic_vector(0 to 31) := C_BASEADDR + NUMBER_OF_REGS*4;
95     constant NUMBER_OF_RO_REGS         : integer          := 4;
96
97    type type_regs_slave is array(0 to (NUMBER_OF_REGS + NUMBER_OF_RO_REGS - 1 )) of std_logic_vector(0 to
      31);
98    signal regs_slave : type_regs_slave;
99
100   --Slave Read
101     type type_state_rd is (wait_rd_req, count_rd, rd_ack);
102     signal state_rd : type_state_rd;
103   signal next_state_rd : type_state_rd;
104   signal counter_rd : integer;
105   signal count_times_rd : integer;
106   signal ack_rd : STD_LOGIC;
107   signal slave_data_rd : std_logic_vector(0 to C_DWIDTH-1);
108   signal burst_active_rd : STD_LOGIC;
109
110   --Slave Write
111     type type_state_wr is (wait_wr_req, count_wr, wr_ack);
112     signal state_wr : type_state_wr;
113   signal next_state_wr : type_state_wr;
114   signal counter_wr : integer;
115   signal count_times_wr : integer;
116   signal ack_wr : STD_LOGIC;
117   signal burst_active_wr : STD_LOGIC;
118
119   -- Bresenham Sinyaller, Tan?mlamalar
120   --Sabitler
121     constant BRE_FRAME_BUF_START       : std_logic_vector(0 to 31) := x"07E00000";
122      constant BRE_SATIR_UZUNLUGU        : integer          := 1024;
123     -- Register Sıraları
124     constant BRE_COMMAND_FLAGS         : integer          := 0;
125     constant BRE_EMAJ                : integer          := 1;
```

```
126     constant BRE_EMIN                : integer          := 2;
127     constant BRE_ESON                : integer          := 3;
128     constant BRE_HATAMAJ2            : integer          := 4;
129     constant BRE_HATAMIN2            : integer          := 5;
130     constant BRE_RGB                 : integer          := 6;
131     constant BRE_EXT_DATA            : integer          := 7;
132     --Read Only Register Sıraları
133     constant BRE_STATUS_FLAGS        : integer          := NUMBER_OF_REGS + 0;
134     --Komut Register Flag Sıraları
135     constant BRE_GO               : integer          := 31;
136     constant BRE_NEGATIF_YON         : integer          := 30;
137     constant BRE_MAJOR_Y             : integer          := 29;
138     constant BRE_KESIKLI_CIZGI         : integer          := 28;
139     --Status Register Flag Sıraları
140     constant BRE_BUSY             : integer          := 31;
141
142
143   --Sinyaller
144     type BRESENHAM_SM_TYPE is ( IDLE, BRESENHAM, BOYA );
145
146     signal bresenham_state             : BRESENHAM_SM_TYPE;
147
148     signal bresenham_busy          : std_logic;
149
150     signal  hataMajorShifted1      : std_logic_vector(0 to 31);
151     signal  hataMinorShifted1      : std_logic_vector(0 to 31);
152     signal  majorEksenDegeri       : std_logic_vector(0 to 31);
153     signal  majorSonDeger        : std_logic_vector(0 to 31);
154     signal  minorEksenDegeri       : std_logic_vector(0 to 31);
155     signal  hata               : integer;
156
157     signal  rgb              : std_logic_vector(0 to 31);
158
159     signal  boyanacakNokta_X       : std_logic_vector(0 to 31);
160     signal  boyanacakNokta_Y       : std_logic_vector(0 to 31);
161
162     signal  taramaSayi           : std_logic_vector(0 to 31);
163     signal  taraSayici           : integer;
164     signal  esgec              : std_logic;
165
166 begin
167
168 -- ###################### Bresenham ######################
169
170   regs_slave(BRE_STATUS_FLAGS)(BRE_BUSY) <= bresenham_busy;
171
172 BRESENHAM_PROC : process( regs_slave(BRE_STATUS_FLAGS)(BRE_GO), Bus2IP_Clk, Bus2IP_Reset ) is
173   variable maj_deg : std_logic_vector(0 to 31);
174   variable min_deg : std_logic_vector(0 to 31);
175   variable nokta_x : std_logic_vector(0 to 31);
176   variable nokta_y : std_logic_vector(0 to 31);
177   variable hata_v : integer;
178   begin
179
180     if ( Bus2IP_Clk'event and Bus2IP_Clk = '1' ) then
181      if ( Bus2IP_Reset = '1' ) then
182        bresenham_state   <= IDLE;
183        bresenham_busy    <= '0';
184
185        hataMajorShifted1 <= (others => '0');
186        hataMinorShifted1 <= (others => '0');
187        majorEksenDegeri  <= (others => '0');
188        majorSonDeger   <= (others => '0');
```

```
189         minorEksenDegeri  <= (others => '0');
190         hata          <= 0;
191         rgb         <= (others => '0');
192
193         boyanacakNokta_X  <= (others => '0');
194         boyanacakNokta_Y  <= (others => '0');
195
196         taramaSayi      <= (others => '0');
197         taraSayici      <= 0;
198         esgec         <= '0';
199
200         mst_go <= '0';
201         mst_rd_wr <= '1'; -- 0 ise read, 1 ise write
202         mst_burst_enable <= '0';-- 1 ise active
203         mst_bus_lock_erable <= '0';-- 1 ise active
204         mst_num <= "00001";
205         mst_out_addr <= BRE_FRAME_BUF_START;
206         mst_local_addr <= C_BASEADDR;
207         mst_write_reg <= (others => '0');
208       else
209       --set default values
210       mst_go <= '0';
211
212       case bresenham_state is
213         when IDLE =>
214           bresenham_busy    <= '0';
215           if regs_slave(BRE_COMMAND_FLAGS)(BRE_GO) = '1'  then
216             hataMajorShifted1 <= regs_slave(BRE_HATAMAJ2);
217             hataMinorShifted1 <= regs_slave(BRE_HATAMIN2);
218             majorEksenDegeri  <= regs_slave(BRE_EMAJ);
219             majorSonDeger   <= regs_slave(BRE_ESON);
220             minorEksenDegeri  <= regs_slave(BRE_EMIN);
221             hata          <= 0;
222             rgb         <= regs_slave(BRE_RGB);
223             taramaSayi      <= regs_slave(BRE_EXT_DATA);
224             taraSayici      <= 0;
225             esgec         <= '0';
226
227             bresenham_busy    <= '1';
228
229             bresenham_state <= BRESENHAM;
230
231           end if;
232
233         when BRESENHAM =>
234             maj_deg := majorEksenDegeri;
235             min_deg := minorEksenDegeri;
236             hata_v := hata;
237
238             if regs_slave(BRE_COMMAND_FLAGS)(BRE_MAJOR_Y) = '0' then
239               nokta_x := majorEksenDegeri;
240               nokta_y := minorEksenDegeri;
241             else
242               nokta_y := majorEksenDegeri;
243               nokta_x := minorEksenDegeri;
244             end if;
245
246             if maj_deg <= majorSonDeger then
247                 --pPikselBoya(boyanacakNokta, rgb);
248
249               if(hata_v >= 0) then
250                 if regs_slave(BRE_COMMAND_FLAGS)(BRE_NEGATIF_YON) = '1' then
251                   min_deg := min_deg - 1;
```

```
252                 else
253                    min_deg := min_deg + 1;
254                 end if;
255
256                 hata_v := hata_v - CONV_INTEGER(hataMajorShifted1);
257               end if;
258
259               hata_v := hata_v + CONV_INTEGER(hataMirorShifted1);
260
261               maj_deg := maj_deg + 1;
262
263               if regs_slave(BRE_COMMAND_FLAGS)(BRE_KESIKLI_CIZGI) = '1' then
264                 taraSayici <= taraSayici + 1;
265                 if taraSayici >= taramaSayi then
266                   taraSayici <= 0;
267                   esgec <= esgec xor '1';
268                 end if;
269               end if;
270
271               if esgec = '0' then
272                 bresenham_state <= BOYA;
273               else
274                 bresenham_state <= BRESENHAM;
275               end if;
276
277             else
278               bresenham_state <= IDLE;
279
280             end if;
281
282             majorEksenDegeri <= maj_deg;
283             minorEksenDegeri <= min_deg;
284             hata <= hata_v;
285             boyanacakNokta_X <= nokta_x;
286             boyanacakNokta_Y <= nokta_y;
287
288         when BOYA =>
289           if mstBusy = '0' then
290
291             mst_out_addr <= BRE_FRAME_BUF_START +
292                       (
293                         ( 4 * BRE_SATIR_UZUNLUGU * CONV_INTEGER(boyanacakNokta_Y) )
294                         +
295                         4 * CONV_INTEGER(boyanacakNokta_X)
296                       );
297
298             mst_write_reg <= rgb;
299
300             mst_go <= '1';
301             bresenham_state <= BRESENHAM;
302
303           end if;
304
305         when others =>
306            null;
307
308
309       end case;--bresenham state macihne
310
311     end if;--!reset
312
313   end if;--if clk event
314
```

```
315
316 end process BRESENHAM_PROC;
317
318 -- ###################### Bresenham ###################### SON
319
320
321
322 -- ###################### Master ######################
323   master_process : process( Bus2IP_Reset, Bus2IP_Clk, mst_go )
324   begin
325
326     if(Bus2IP_Reset = '1') then
327
328         IP2Bus_Addr                     <= (others => '0');
329          IP2Bus_MstBE                    <= (others => '0');
330          IP2Bus_MstBurst                 <= '0';
331          IP2Bus_MstBusLcck               <= '0';
332          IP2Bus_MstNum                   <= (others => '0');
333          IP2IP_Addr                      <= C_BASEADDR;--(others => '0');
334         mstRdReq                <= '0';
335         mstWrReq                <= '0';
336         mstBusy                   <= '0';
337         mstSuccess                <= '0';
338         mstError                <= '0';
339
340         state_mst <= state_mst_idle;
341
342         leds_2 <= "1111";
343
344
345     elsif (Bus2IP_Clk'event and Bus2IP_Clk = '1') then
346       --set defaults
347       mstSuccess                <= '0';
348       mstError                  <= '0';
349
350
351       case state_mst is
352
353         when state_mst_idle =>
354           mstBusy                 <= '0';
355
356           if mst_go = '1' then
357             mstBusy                 <= '1';
358
359             leds_2 <= "1100";
360
361             if mst_out_addr(29 to 31) = "000" then
362               IP2Bus_MstBE <= "11110000";
363             else
364               IP2Bus_MstBE <= "00001111";
365             end if;
366
367             --Read----------------------------
368             if mst_rd_wr = '0' then
369               mstRdReq <= '1';
370
371             --Write---------------------------
372             else
373               mstWrReq <= '1';
374             end if;
375
376             IP2Bus_MstBurst <= mst_burst_enable;
377             IP2Bus_MstBusLock <= mst_bus_lock_enable;
```

```
378             IP2Bus_Addr <= mst_out_addr;
379             IP2IP_Addr <= mst_local_addr;
380 
381             if mst_burst_enable = '1' then
382               IP2Bus_MstNum <= mst_num;
383             else
384               IP2Bus_MstNum <= "00001";
385             end if;
386 
387             state_mst <= state_mst_wait_ack;
388 
389           end if; --mst_go
390 
391 
392 
393         when state_mst_wait_ack =>
394           if (Bus2IP_MstLastAck = '1') or (Bus2IP_MstError = '1') or
395             (Bus2IP_MstRetry = '1') or (Bus2IP_MstTimeOut = '1')
396             then
397 
398             if (Bus2IP_MstLastAck = '1') then
399               --led_left <= led_left(1) & led_left(0);
400               mstSuccess                <= '1';
401               leds_2 <= "1110";
402             elsif (Bus2IP_MstError = '1') then
403               mstError                  <= '1';
404               leds_2 <= "1101";
405             elsif  (Bus2IP_MstRetry = '1') then
406               mstError                  <= '1';
407               leds_2 <= "1100";
408             elsif (Bus2IP_MstTimeOut = '1') then
409               mstError                  <= '1';
410               leds_2 <= "1011";
411             else
412               mstError                  <= '1';
413               leds_2 <= "1010";
414             end if;
415 
416             state_mst <= state_mst_idle;
417             IP2Bus_MstBurst <= '0';
418             IP2Bus_MstBusLock <= '0';
419             mstRdReq <= '0';
420             mstWrReq <= '0';
421 
422 
423           end if;
424 
425       end case;
426 
427     end if;
428 
429   end process;
430 
431   IP2Bus_MstRdReq <= mstRdReq;
432   IP2Bus_MstWrReq <= mstWrReq;
433 
434   -- ---------------------- Master ---------------------- SON
435 
436 
437   -- ---------------------- Slave Write ----------------------
438   slave_write_process : process( Bus2IP_Reset, Bus2IP_Clk, Bus2IP_WrReq )
439   variable reg_addr : integer;
440   variable data : std_logic_vector(0 to 31);
```

```
441   variable addr : std_logic_vector(0 to 31);
442   begin
443
444     if(Bus2IP_Reset = '1') then
445
446         --For Test
447         --led_left <= "01";
448         --rd_data <= x"AABBCCDDEEFF1122";
449         --For Test END
450
451         state_wr <= wait_wr_req;
452         next_state_wr <= wait_wr_req;
453         counter_wr <= 0;
454         count_times_wr <= 0;
455         ack_wr <= '0';
456         burst_active_wr <= '0';
457
458         regs_slave <= (others => (others => '0'));
459
460
461     elsif (Bus2IP_Clk'event and Bus2IP_Clk = '1') then
462     --Set Defaults
463       regs_slave(BRE_COMMAND_FLAGS)(BRE_GO) <= '0';
464
465       case state_wr is
466         when wait_wr_req =>
467           if Bus2IP_WrReq = '1' then
468
469             if Bus2IP_BE(0 to 3) = "1111" then
470               data := Bus2IP_Data(0 to 31);
471             else
472               data := Bus2IP_Data(32 to 63);
473             end if;
474
475             addr := Bus2IP_Addr;
476             reg_addr := CONV_INTEGER(Bus2IP_Addr(24 to 31));
477
478             if burst_active_wr = '0' then
479               if Bus2IP_Burst = '1' then
480                 ack_wr <= '1';
481                 burst_active_wr <= '1';
482               else
483                 state_wr <= count_wr;
484                 next_state_wr <= wr_ack;
485                 count_times_wr <= 4;
486               end if;
487
488             else
489               if Bus2IP_Burst = '0' then
490                 state_wr <= wr_ack;
491                  next_state_wr <= wr_ack;
492                  burst_active_wr <= '0';
493               else
494               end if;
495
496             end if;
497
498             if mstRdReq = '1' then --mst reg'e koyalım
499               mst_read_reg <= data(0 to 31);
500             else --reg'e koyalım
501
502               if (reg_addr / 4) <= (NUMBER_OF_REGS - 1) then --readonly alana koyamayız
503
```

```
504                 for index in 0 to (NUMBER_OF_REGS - 1) loop
505                   if index = (reg_addr / 4) then
506                     regs_slave(index) <= data;
507                   end if;
508                 end loop;
509
510               end if;
511
512             end if;
513
514           end if;
515
516         when count_wr =>
517           if counter_wr >= count_times_wr then
518             state_wr <= next_state_wr;
519             counter_wr <= 0;
520           else
521             counter_wr <= counter_wr + 1;
522           end if;
523
524
525         when wr_ack =>
526           if ack_wr = '1' then
527             ack_wr <= '0';
528             state_wr <= wait_wr_req;
529             --next_state_wr <= wait_wr_req;
530           else
531             ack_wr <= '1';
532           end if;
533
534
535       end case;
536
537     end if;
538
539   end process; --slave_write_process
540
541   IP2Bus_WrAck <= ack_wr;
542
543    -- ###################### Slave Write ######################---SON
544
545
546    -- ###################### Slave Read ######################
547
548   slave_read_process : process( Bus2IP_Reset, Bus2IP_Clk, Bus2IP_RdReq )
549   variable addr : std_logic_vector(0 to C_AWIDTH-1);
550   variable reg_addr : integer;
551   begin
552
553     if(Bus2IP_Reset = '1') then
554         --For Test
555         --led_right <= "10";
556         slave_data_rd <= (others => '0');--x"ABCDEF12ABCDEF12";
557         --For Test END
558         state_rd <= wait_rd_req;
559         next_state_rd <= wait_rd_req;
560         counter_rd <= 0;
561         count_times_rd <= 0;
562         ack_rd <= '0';
563         burst_active_rd <= '0';
564
565
566     elsif (Bus2IP_Clk'event and Bus2IP_Clk = '1') then
```

```
567       case state_rd is
568         when wait_rd_req =>
569           if Bus2IP_RdReq = '1' then
570             addr := Bus2IP_Addr;
571             reg_addr := CONV_INTEGER(addr(24 to 31));
572 
573             if mstWrReq = '1' then --mst datayı koyalım
574               slave_data_rd <= mst_write_reg & mst_write_reg;
575             else --register data koyalım
576               for index in 0 to (NUMBER_OF_REGS + NUMBER_OF_RO_REGS - 1) loop
577                 if (4*index) = reg_addr then
578                   slave_data_rd <= regs_slave(index) & regs_slave(index);
579                 end if;
580               end loop;
581             end if;
582 
583             if burst_active_rd = '0' then
584 
585               if Bus2IP_Burst = '1' then
586                 burst_active_rd <= '1';
587 
588               else
589                 state_rd <= count_rd;
590                 next_state_rd <= rd_ack;
591                 count_times_rd <= 4;
592 
593               end if;
594 
595             else
596               if Bus2IP_Burst = '0' then
597                 state_rd <= rd_ack;
598                  next_state_rd <= rd_ack;
599                  burst_active_rd <= '0';
600 
601               else
602                 ack_rd <= '1';
603               end if;
604             end if;
605 
606 
607           end if;
608 
609         when count_rd =>
610           if counter_rd >= count_times_rd then
611             state_rd <= next_state_rd;
612           end if;
613           counter_rd <= counter_rd + 1;
614 
615         when rd_ack =>
616           if ack_rd = '1' then
617             ack_rd <= '0';
618             state_rd <= wait_rd_req;
619             --led_right <= led_right xor "11";
620             --next_state_rd <= wait_rd_req;
621           else
622             ack_rd <= '1';
623           end if;
624 
625       end case;
626 
627     end if;
628 
629   end process;
```

```
630
631   IP2Bus_RdAck <= ack_rd;
632   IP2Bus_Data <= slave_data_rd;
633
634   -- ###################### Slave Read Code ######################---SON
635
636
637   --Default Dummy Signals
638   IP2Bus_Error       <= '0';
639   IP2Bus_Retry       <= '0';
640   IP2Bus_ToutSup     <= '0';
641
642 end IMP;
```

Ek-3 Sürücü ve API Kodları

```
1  /*
2   * _bitBLT.h
3   */
4
5  #ifndef _BITBLT_H_
6  #define _BITBLT_H_
7
8
9  #endif /* _BITBLT_H_ */
10
11
12 #include "ipcore_BitBLT2.h"
13
14 #include "_grafikOrtak.h"
15 #include "xio.h"
16
17 #define BITBLT_BASE_ADDR    0xc7200000
18 #define ARKA_PLAN_ADDR_REG    (BITBLT_BASE_ADDR + 8)
19 #define BLT_HEADER      0xAA0000AA
20
21 #define KAYNAK_ARGB         0
22 #define KAYNAK_ADDR         1
23 #define BOOL_NON          0<<1
24 #define BOOL_AND          1<<1
25 #define BOOL_OR           2<<1
26 #define BOOL_XOR          3<<1
27 #define TRANSPARAN_NON        0<<3
28 #define TRANSPARAN_ARGB       1<<3
29 #define TRANSPARAN_NOT_ARGB     2<<3
30 #define ALPHA           1<<5
31 #define SOL_ALT_KOSE        0<<6
32 #define SAG_ALT_KOSE        1<<6
33 #define SAG_UST_KOSE        2<<6
34 #define SOL_UST_KOSE        3<<6
35 #define KAYNAK_GUNCELLEME     0<<8
36 #define KAYNAK_GUNCELLE_ARGB    1<<8
37 #define KAYNAK_GUNCELLE_ADDR    2<<8
38
39 struct tBltKomut
40 {
41   unsigned int argb;
42   unsigned int kaynak_addr;
43   unsigned int hedef_addr;
44   unsigned int frm_buf_row_size;
45   unsigned int blt_row_size;
46   unsigned int blt_col_size;
47   unsigned int flags;
48 };
49
50 int bitBLT_Go(void* peripAddr, struct tBltKomut* pBlt);
51
52 void ekranTemizle(unsigned int rgb);
53
54 int arkaPlanAddrYaz(void* arkaAddr);
```

```
1  /*
2   * bresenham.h
3   *
4   *  Created on: 09.Eyl.2010
5   *      Author: SERDAR
6   */
7
8  #ifndef BRESENHAM_H_
9  #define BRESENHAM_H_
10
11 #include "xio.h"
12
13
14
15
16 //struct bresenhamKomut
17 //{
18 //  unsigned int command_flags;
19 //  unsigned int emaj;
20 //  unsigned int emin;
21 //  unsigned int eson;
22 //  unsigned int hataMaj2;
23 //  unsigned int hataMin2;
24 //  unsigned int rgb;
25 //  unsigned int ext_data;
26 //};
27
28
29 #define BRESENHAM_BASE_ADDR   0xc3a00000
30
31 #define NUMBER_OF_REGS        16
32 #define NUMBER_OF_RO_REGS     4
33
34
35 //Bresenham Register Adresleri
36 #define BRE_COMMAND_FLAGS     BRESENHAM_BASE_ADDR + 0
37 #define BRE_EMAJ          BRESENHAM_BASE_ADDR + 4
38 #define BRE_EMIN          BRESENHAM_BASE_ADDR + 8
39 #define BRE_ESON          BRESENHAM_BASE_ADDR + 12
40 #define BRE_HATAMAJ2        BRESENHAM_BASE_ADDR + 16
41 #define BRE_HATAMIN2        BRESENHAM_BASE_ADDR + 20
42 #define BRE_RGB           BRESENHAM_BASE_ADDR + 24
43 #define BRE_EXT_DATA        BRESENHAM_BASE_ADDR + 28
44
45 #define BRE_STATUS_FLAGS      BRESENHAM_BASE_ADDR + NUMBER_OF_REGS * 4 + 0
46
47
48 //Komut Register Maskeleri
49 #define BRE_GO        0x00000001
50 #define BRE_NEGATIF_YON   0x00000002
51 #define BRE_MAJOR_Y     0x00000004
52 #define BRE_KESIKLI_CIZGI 0x00000008
53
54 //Durum Register Maskeleri
55 #define BRE_BUSY      0x00000001
56
57
58 //unsigned int degisimMiktari(int a1, int a2);
59
60 int bresenhamDogruCizSoft(struct tNokta nokta1, struct tNokta nokta2, unsigned int rgb, void
   (*pPikselBoya)(struct tNokta, unsigned int rgb));
61
62 int bresenhamDogruCiz(struct tNokta nokta1, struct tNokta nokta2, unsigned int rgb, int kesikli);
```

```
63
64 #endif /* BRESENHAM_H_ */
65
66
67
68
69
70
71
72
```

```
1  /*
2   * grafikOrtak.h
3   
4   */
5  
6  #ifndef GRAFIKORTAK_H_
7  #define GRAFIKORTAK_H_
8  
9  
10 #define YESIL 0x0000FF00
11 
12 #define KIRMIZI 0x00FF0000
13 
14 #define MAVI 0x000000FF
15 
16 #define SARI (YESIL | KIRMIZI)
17 
18 #define MAGENTA (MAVI | KIRMIZI)
19 
20 #define CYAN (YESIL | MAVI)
21 
22 #define BEYAZ (KIRMIZI | YESIL | MAVI)
23 
24 #define SIYAH 0x00000000
25 
26 
27 struct tNokta
28 {
29   int x;
30   int y;
31 };
32 
33 struct tDogruNok
34 {
35   struct tNokta n1;
36   struct tNokta n2;
37 };
38 
39 struct tDogruEg
40 {
41   struct tNokta altNokta;
42   unsigned int dX;
43   unsigned int dY;
44   int negEg;
45 };
46 
47 struct tPoligon
48 {
49   unsigned int kenarSayisi;
50   struct tNokta* koseler;
51 };
52 
53 struct tNokta nokta(int x, int y);
54 
55 unsigned int degisimMiktari(int a1, int a2);
56 
57 struct tDogruNok dogruNok(struct tNokta n1, struct tNokta n2);
58 
59 struct tDogruEg dogruNok2Eg(struct tDogruNok dogruNok);
60 
61 struct tNokta noktaCevir(struct tNokta eksen, struct tNokta inn, int teta);
62 
63 int roundDouble(double db);
```

```
64
65 #endif /* GRAFIKORTAK_H_ */
```

```
1  /*
2   * koordSystemXUP.h
3   *
4   *  Created on: 10.Eyl.2010
5   *      Author: SERDAR
6   */
7
8  #ifndef KOORDSYSTEMXUP_H_
9  #define KOORDSYSTEMXUP_H_
10
11 #include "_grafikOrtak.h"
12
13 #define arkaPlanAddr  0x08E00000
14
15 #define frameBufStart 0x07E00000
16 #define satirUzunlugu 1024
17 #define sutunUzunlugu 484
18
19 #define gorunurSatirUzunlugu  640
20 #define gorunurSutunUzunlugu  484
21
22 unsigned int* koordToAddr(struct tNokta nokta);
23
24 unsigned int* koordToArkaAddr(struct tNokta nokta);
25
26 void pikselBoyaXUP(struct tNokta nokta, unsigned int rgb);
27
28 void ekranTemizleSoft(unsigned int rgb);
29
30 #endif /* KOORDSYSTEMXUP_H_ */
```

```
1  /*
2   * poligonBoya.h
3   *
4   */
5
6  #ifndef POLIGONBOYA_H_
7  #define POLIGONBOYA_H_
8
9  #include "_grafikOrtak.h"
10 #include "_bresenham.h"
11 #include "_bitBLT.h"
12 #include "_koordSystemXUP.h"
13
14
15 struct tKenarBilgisi
16 {
17   struct tDogruEg dogru;
18   int counter;
19 };
20
21 struct tKenarDizisi
22 {
23   unsigned int kenarSayisi;
24   struct tKenarBilgisi ilkKenar;
25 };
26
27
28 int kenarDizisiOlustur(void* hedef, struct tPoligon* pol);
29
30 int poligonBoya(struct tPoligon* pol, void* buffer, unsigned int renk);
31
32 int poligonBoyaSoft(struct tPoligon* pol, void* buffer, unsigned int renk, void (*pPikselBoya)(struct
   tNokta, unsigned int rgb));
33
34 int poligonCiz(struct tPoligon* pol, unsigned int renk);
35
36 int poligonCevir(struct tPoligon* pol, struct tNokta eksen, int teta);
37
38 int poligonScale(struct tPoligon* pol, double scale);
39
40 int poligonTasi(struct tPoligon* pol, struct tNokta nokta);
41
42 //void ekranTemizle(unsigned int rgb);
43
44 #endif /* POLIGONBOYA_H_ */
```

```
1  /*
2   * bitBLT.c
3   *
4   */
5
6  #include "_bitBLT.h"
7
8
9  int bitBLT_Go(void* peripAddr, struct tBltKomut* pBlt)
10 {
11   Xuint32 baseaddr = peripAddr;
12
13   IPCORE_BITBLT2_mWriteToFIFO(baseaddr, BLT_HEADER);
14   IPCORE_BITBLT2_mWriteToFIFO(baseaddr, pBlt->argb);
15   IPCORE_BITBLT2_mWriteToFIFO(baseaddr, pBlt->kaynak_addr);
16   IPCORE_BITBLT2_mWriteToFIFO(baseaddr, pBlt->hedef_addr);
17   IPCORE_BITBLT2_mWriteToFIFO(baseaddr, pBlt->frm_buf_row_size);
18   IPCORE_BITBLT2_mWriteToFIFO(baseaddr, pBlt->blt_row_size);
19   IPCORE_BITBLT2_mWriteToFIFO(baseaddr, pBlt->blt_col_size);
20   IPCORE_BITBLT2_mWriteToFIFO(baseaddr, pBlt->flags);
21
22 }
23
24 int arkaPlanAddrYaz(void* arkaAddr)
25 {
26   Xuint32 arka = (Xuint32)arkaAddr;
27
28   XIo_Out32((XIo_Address)(ARKA_PLAN_ADDR_REG), arka);
29
30 }
31
32 void ekranTemizle(unsigned int rgb)
33 {
34   struct tBltKomut bltKomut;
35
36   bltKomut.argb = rgb;
37   bltKomut.blt_col_size = 484;
38   bltKomut.blt_row_size = 640;
39   bltKomut.flags = SOL_UST_KOSE;
40   bltKomut.frm_buf_row_size = 1024;
41   bltKomut.hedef_addr = koordToAddr(nokta(0,0))  ;
42   bltKomut.kaynak_addr = 0;
43
44   bitBLT_Go((void*)BITBLT_BASE_ADDR, &bltKomut);
45
46 //  struct tNokta noktalar[4];
47 //  char buffer[100];
48 //  struct tPoligon pol;
49 //
50 //  pol.kenarSayisi = 4;
51 //  pol.koseler = noktalar;
52 //  noktalar[0] = nokta(0,0);
53 //  noktalar[1] = nokta(640,0);
54 //  noktalar[2] = nokta(640,484);
55 //  noktalar[3] = nokta(0,484);
56 //
57 //  poligonBoya(&pol, buffer, rgb, pikselBoyaXUP);
58
59 }
```

```
1   /*
2    * bresenham.c
3    *
4    *  Created on: 09.Eyl.2010
5    *      Author: SERDAR
6    */
7
8   #include "_grafikOrtak.h"
9   #include "_bresenham.h"
10
11  #include "ipcore_bresenham.h"
12
13
14
15  int bresenhamDogruCizScft(struct tNokta nokta1, struct tNokta nokta2, unsigned int rgb, void
    (*pPikselBoya)(struct tNokta, unsigned int rgb))
16  {
17    int hata = 0, \
18      hataMajorShifted1,
19      hataMinorShifted1, \
20      *majorEksenDegeri, \
21      majorSonDeger, \
22      *minorEksenDegeri, \
23      arttirimYonu = 1;
24
25    unsigned int degisimX2, degisimY2;
26    struct tNokta boyanacakNokta;
27
28    degisimX2 = degisimMiktari(nokta1.x, nokta2.x) << 1;
29    degisimY2 = degisimMiktari(nokta1.y, nokta2.y) << 1;
30
31    if(degisimX2 > degisimY2)
32    {//Majör Eksen X
33      hataMajorShifted1 = degisimX2;
34      hataMinorShifted1 = degisimY2;
35      majorEksenDegeri = &boyanacakNokta.x;
36      minorEksenDegeri = &boyanacakNokta.y;
37
38      if(nokta1.x < nokta2.x)
39      {//nokta1 solda
40
41        *majorEksenDegeri = nokta1.x;
42        majorSonDeger = nokta2.x;
43        *minorEksenDegeri = nokta1.y;
44
45        if(nokta1.y > nokta2.y)
46          arttirimYonu = -1;
47        else
48          arttirimYonu = 1;
49      }
50      else
51      {//nokta2 solda
52
53        *majorEksenDegeri = nokta2.x;
54        majorSonDeger = nokta1.x;
55        *minorEksenDegeri = nokta2.y;
56
57        if(nokta2.y > nokta1.y)
58          arttirimYonu = -1;
59        else
60          arttirimYonu = 1;
61
62      }
```

```
63    }
64    else
65    {//Majör Eksen Y
66      hataMajorShifted1 = degisimY2;
67      hataMinorShifted1 = degisimX2;
68      majorEksenDegeri = &boyanacakNokta.y;
69      minorEksenDegeri = &boyanacakNokta.x;
70
71      if(nokta1.y < nokta2.y)
72      {//nokta1 önce
73
74        *majorEksenDegeri = nokta1.y;
75        majorSonDeger = nokta2.y;
76        *minorEksenDegeri = nokta1.x;
77
78        if(nokta1.x > nokta2.x)
79          arttirimYonu = -1;
80        else
81          arttirimYonu = 1;
82      }
83      else
84      {//nokta2 önce
85
86        *majorEksenDegeri = nokta2.y;
87        majorSonDeger = nokta1.y;
88        *minorEksenDegeri = nokta2.x;
89
90        if(nokta2.x > nokta1.x)
91          arttirimYonu = -1;
92        else
93          arttirimYonu = 1;
94
95      }
96    }
97
98
99    //Bresenham Algoritması
100   while(*majorEksenDegeri <= majorSonDeger)
101   {
102     pPikselBoya(boyanacakNokta, rgb);
103
104     if(hata >= 0)
105     {
106       *minorEksenDegeri = *minorEksenDegeri + arttirimYonu;
107       hata = hata - hataMajorShifted1;
108     }
109
110     hata = hata + hataMinorShifted1;
111
112     *majorEksenDegeri = *majorEksenDegeri + 1;
113   }
114
115   return 1;
116
117 }
118
119
120
121 int bresenhamDogruCiz(struct tNokta nokta1, struct tNokta nokta2, unsigned int rgb, int kesikli)
122 {
123   int hata = 0, \
124     hataMajorShifted1, \
125     hataMinorShifted1, \
```

```
126     *majorEksenDegeri, \
127     majorSonDeger, \
128     *minorEksenDegeri, \
129     arttirimYonu = 1,
130     majY = 0;
131
132   int i;
133
134   unsigned int degisimX2, degisimY2, commandFlags;
135   struct tNokta boyanacakNokta;
136
137   degisimX2 = degisimMiktari(nokta1.x, nokta2.x) << 1;
138   degisimY2 = degisimMiktari(nokta1.y, nokta2.y) << 1;
139
140   if(degisimX2 > degisimY2)
141   {//Majör Eksen X
142     majY = 0;
143
144     hataMajorShifted1 = degisimX2;
145     hataMinorShifted1 = degisimY2;
146     majorEksenDegeri = &boyanacakNokta.x;
147     minorEksenDegeri = &boyanacakNokta.y;
148
149     if(nokta1.x < nokta2.x)
150     {//nokta1 solda
151
152       *majorEksenDegeri = nokta1.x;
153       majorSonDeger = nokta2.x;
154       *minorEksenDegeri = nokta1.y;
155
156       if(nokta1.y > nokta2.y)
157         arttirimYonu = -1;
158       else
159         arttirimYonu = 1;
160     }
161     else
162     {//nokta2 solda
163
164       *majorEksenDegeri = nokta2.x;
165       majorSonDeger = nokta1.x;
166       *minorEksenDegeri = nokta2.y;
167
168       if(nokta2.y > nokta1.y)
169         arttirimYonu = -1;
170       else
171         arttirimYonu = 1;
172
173     }
174   }
175   else
176   {//Majör Eksen Y
177     majY = 1;
178
179     hataMajorShifted1 = degisimY2;
180     hataMinorShifted1 = degisimX2;
181     majorEksenDegeri = &boyanacakNokta.y;
182     minorEksenDegeri = &boyanacakNokta.x;
183
184     if(nokta1.y < nokta2.y)
185     {//nokta1 önce
186
187       *majorEksenDegeri = nokta1.y;
188       majorSonDeger = nokta2.y;
```

```
189       *minorEksenDegeri = nokta1.x;
190
191       if(nokta1.x > nokta2.x)
192         arttirimYonu = -1;
193       else
194         arttirimYonu = 1;
195     }
196     else
197     {//nokta2 önce
198
199       *majorEksenDegeri = nokta2.y;
200       majorSonDeger = nokta1.y;
201       *minorEksenDegeri = nokta2.x;
202
203       if(nokta2.x > nokta1.x)
204         arttirimYonu = -1;
205       else
206         arttirimYonu = 1;
207
208     }
209   }
210
211
212   while(  XIo_In32((XIo_Address)BRE_STATUS_FLAGS) & ERE_BUSY  );
213   //Peripherial GO !!
214   XIo_Out32((XIo_Address)BRE_EMAJ, *majorEksenDegeri);
215   XIo_Out32((XIo_Address)BRE_EMIN, *minorEksenDegeri);
216   XIo_Out32((XIo_Address)BRE_ESON, majorSonDeger);
217   XIo_Out32((XIo_Address)BRE_HATAMAJ2, hataMajorShifted1);
218   XIo_Out32((XIo_Address)BRE_HATAMIN2, hataMinorShifted1);
219   XIo_Out32((XIo_Address)BRE_RGB, rgb);
220
221   commandFlags = 0;
222   commandFlags |= BRE_GO;
223   if(arttirimYonu < 0)
224     commandFlags |= BRE_NEGATIF_YON;
225   if(majY)
226     commandFlags |= BRE_MAJOR_Y;
227
228   if(kesikli)
229     commandFlags |= BRE_KESIKLI_CIZGI;
230
231   XIo_Out32((XIo_Address)BRE_EXT_DATA, 4);
232
233   XIo_Out32((XIo_Address)BRE_COMMAND_FLAGS, commandFlags);
234
235   for(i=0;i<7000;i++);
236
237   return 1;
238
239 }
240
241
242
```

```
1  /*
2   * grafikOrtak.c
3   *
4   *  Created on: 11.Eyl.2010
5   *      Author: SERDAR
6   */
7
8  #include "_grafikOrtak.h"
9  #include <math.h>
10
11 unsigned int degisimMiktari(int a1, int a2)
12 {
13   int fark = a2 - a1;
14   if(fark < 0)
15     fark *= -1;
16   return (unsigned int)fark;
17 }
18
19 struct tNokta nokta(int x, int y)
20 {
21   struct tNokta nok = {x, y};
22   return nok;
23 }
24
25 struct tDogruNok dogruNok(struct tNokta n1, struct tNokta n2)
26 {
27   struct tDogruNok dogru = {n1, n2};
28   return dogru;
29 }
30
31
32
33 struct tDogruEg dogruNok2Eg(struct tDogruNok dogruNok)
34 {
35   struct tDogruEg retval;
36   struct tNokta ustNok;
37
38   if(dogruNok.n1.y >= dogruNok.n2.y)
39   {
40     retval.altNokta = dogruNok.n2;
41     ustNok = dogruNok.n1;
42   }
43   else
44   {
45     retval.altNokta = dogruNok.n1;
46     ustNok = dogruNok.n2;
47   }
48
49   retval.dX = degisimMiktari(dogruNok.n1.x, dogruNok.n2.x);
50   retval.dY = degisimMiktari(dogruNok.n1.y, dogruNok.n2.y);
51
52   if(ustNok.x > retval.altNokta.x)
53   {
54     retval.negEg = 0;
55   }
56   else
57   {
58     retval.negEg = 1;
59   }
60
61   return retval;
62 }
63
```

```
64  struct tNokta noktaCevir(struct tNokta eksen, struct tNokta inn, int teta)
65  {
66    struct tNokta retn;
67    retn.x = (int)((double)eksen.x + ((double)inn.x - (double)eksen.x)*cos((double)teta)
68        - ((double)inn.y - (double)eksen.y)*sin((double)teta));
69
70    retn.y = (int)((double)eksen.y + ((double)inn.x - (double)eksen.x)*sin((double)teta)
71        - ((double)inn.y - (double)eksen.y)*cos((double)teta));
72
73    return retn;
74  }
75
76  int roundDouble(double db)
77  {
78    int ret;
79    double doub;
80
81    ret = (int)db;
82
83    doub = db - (double)ret;
84    if(doub > 0.5) ret++;
85
86    return ret;
87  }
88
89  unsigned int preMultipliedARGB(unsigned int kaplamaYuzdesi, unsigned int rgb)
90  {
91    unsigned int kaplama, alfa, red, green, blue;
92
93    if(kaplamaYuzdesi == 100)
94      kaplama = 100;
95    else
96      kaplama = kaplamaYuzdesi%100;
97
98     alfa = (  kaplama * 255  ) / 100;
99     red = (  kaplama * ((rgb & 0x00FF0000)>>16)  ) / 100;
100    green = (  kaplama * ((rgb & 0x0000FF00)>>8)  ) / 100;
101    blue = (  kaplama * (rgb & 0x000000FF)  ) / 100;
102
103   return (alfa<<24) | (red<<16) | (green<<8) | blue;
104 }
```

```
1  /*
2   * koordSysXUP.c
3   *
4   *  Created on: 10.Eyl.2010
5   *      Author: SERDAR
6   */
7
8  #include "_koordSystemXUP.h"
9
10 unsigned int* koordToAddr(struct tNokta nokta)
11 {
12   unsigned int* retval;
13   retval = (unsigned int*)(4 * satirUzunlugu * nokta.y + 4 * nokta.x + frameBufStart);
14
15   return retval;
16 }
17
18 unsigned int* koordToArkaAddr(struct tNokta nokta)
19 {
20   unsigned int* retval;
21   retval = (unsigned int*)(4 * satirUzunlugu * nokta.y + 4 * nokta.x + arkaPlanAddr);
22
23   return retval;
24 }
25
26
27 void pikselBoyaXUP(struct tNokta nokta, unsigned int rgb)
28 {
29   unsigned int* destAddr;
30   destAddr = koordToAddr(nokta);
31
32   *destAddr  = rgb;
33 }
34
35
36 void ekranTemizleSoft(unsigned int rgb)
37 {
38   unsigned int x, wordnum;
39   unsigned int  *pAddr;
40
41     pAddr   = (unsigned int  *)frameBufStart;
42
43     x = 0;
44     wordnum = 0;
45
46     while (wordnum < 309760)
47     {
48       *(pAddr) = rgb;
49       pAddr++;
50       wordnum++;
51       x++;
52       if (x >= 640){
53        x = 0;
54        pAddr += (1024 - 640);
55        }
56
57     }
58
59 }
60
61
62
```

```
1  /*
2   * poligonBoya.c
3   *
4   *  Created on: 11.Eyl.2010
5   *      Author: SERDAR
6   */
7
8  #include "_poligonBoya.h"
9  //#include "_bitBLT.h"
10
11 int kenarDizisiOlustur(void* hedef, struct tPoligon* pol)
12 {
13   int kcse = 0;
14   int birSonraki;
15   int ikiSonraki;
16   int ucSonraki;
17   int dataLen = 0;
18   struct tNokta* koseler = pol->koseler;
19
20   if(pol->kenarSayisi < 3)
21     return 0;
22
23   *(unsigned int*)(hedef + dataLen) = pol->kenarSayisi;
24   dataLen += sizeof(pol->kenarSayisi);
25
26   while(kose < pol->kenarSayisi)
27   {
28     struct tKenarBilgisi* pKenBil = (struct tKenarBilgisi*)(hedef + dataLen);
29
30     birSonraki = (kose+1)%(pol->kenarSayisi);
31     ikiSonraki = (kose+2)%(pol->kenarSayisi);
32     ucSonraki = (kose+3)%(pol->kenarSayisi);
33
34     pKenBil->dogru = dogruNok2Eg(   dogruNok( koseler[kose], koseler[birSonraki] )   );
35     pKenBil->counter = 0;
36     dataLen += sizeof(struct tKenarBilgisi);
37
38     if
39     (
40       koseler[kose].y < koseler[birSonraki].y
41       &&
42       koseler[ikiSonraki].y > koseler[birSonraki].y
43     )
44     {
45       pKenBil->dogru.dY -= 1;
46       if(pKenBil->dogru.dY == 0)
47       {
48       /*
49        *  Kısaltma yaptığımız kenar yatay oldu!!
50        */
51         koseler[birSonraki].y -= 1;
52       }
53     }
54     else if
55     (
56       koseler[kose].y < koseler[birSonraki].y
57       &&
58       koseler[ikiSonraki].y == koseler[birSonraki].y
59       &&
60       koseler[ucSonraki].y > koseler[ikiSonraki].y
61     )
62     {
63       pKenBil->dogru.dY -= 1;
```

```
64        if(pKenBil->dogru.dY == 0)
65        {
66        /*
67         *  Kısaltma yaptığımız kenar yatay oldu!!
68         */
69          koseler[birSonraki].y -= 1;
70        }
71      }
72      else if
73      (
74        koseler[kose].y < koseler[birSonraki].y
75        &&
76        koseler[ikiSonraki].y == koseler[birSonraki].y
77        &&
78        koseler[ucSonraki].y < koseler[ikiSonraki].y
79      )
80      {
81        //NOPPPP
82      }
83
84
85
86
87      else if
88      (
89        koseler[kose].y > koseler[birSonraki].y
90        &&
91        koseler[ikiSonraki].y < koseler[birSonraki].y
92      )
93      {
94        pKenBil->dogru.dY -= 1;
95        pKenBil->dogru.altNokta.y += 1;
96        if(pKenBil->dogru.dY == 0)
97        {
98        /*
99         *  Kısaltma yaptığımız kenar yatay oldu!!
100        */
101         koseler[birSonraki].y += 1;
102       }
103     }
104     else if
105     (
106       koseler[kose].y > koseler[birSonraki].y
107       &&
108       koseler[ikiSonraki].y == koseler[birSonraki].y
109       &&
110       koseler[ucSonraki].y < koseler[ikiSonraki].y
111     )
112     {
113       pKenBil->dogru.dY -= 1;
114       pKenBil->dogru.altNokta.y += 1;
115       if(pKenBil->dogru.dY == 0)
116       {
117       /*
118        *  Kısaltma yaptığımız kenar yatay oldu!!
119        */
120         koseler[birSonraki].y += 1;
121       }
122     }
123     else if
124     (
125       koseler[kose].y > koseler[birSonraki].y
126       &&
```

```
127       koseler[ikiSonraki].y == koseler[birSonraki].y
128       &&
129       koseler[ucSonraki].y > koseler[ikiSonraki].y
130     )
131     {
132       //NOPPPP
133     }
134
135     kose++;
136   }
137   return dataLen;
138 }
139
140
141 int poligonBoya_1_Soft(struct tPoligon* pol, int baslangicSatiri, int bitisSatiri, void* kenarVeri,
    unsigned int renk, void (*pPikselBoya)(struct tNokta, unsigned int rgb))
142 {
143   int y;
144   int i = 0;
145   int kesilenSayisi = 0;
146   int bcya = 1;
147   int baslamaSatiri;
148   unsigned int kenarSayisi;
149   struct tKenarDizisi* pKenDiz;
150   struct tKenarBilgisi* kenarlar;
151   int kesSira[pol->kenarSayisi]; //Kesme noktalarının sıralanacağı dizi
152   int maxY = 1024;
153
154   pKenDiz = (struct tKenarDizisi*)kenarVeri;
155   kenarSayisi = pKenDiz->kenarSayisi;
156   kenarlar = &(pKenDiz->ilkKenar);
157   baslamaSatiri = baslangicSatiri;
158
159
160   y = baslamaSatiri;
161   while(y < maxY) //Tarama Satirlari Dongusu
162   {
163     int dummy;//FIXME sil
164     int ken = 0; //kesip kesmediğimize bakacağımız kenar indeksi
165
166     if(y == 13)
167       dummy = 0;
168
169     /*
170      * Siralama dizisini büyük bir değerle ilklendirerek
171      * ilgili indekse ilk kez veri koyup koymadığımızı anlayabilelim
172      */
173     i = 0;
174     while(i < kenarSayisi)
175       kesSira[i++] = 0xFFFF;
176
177     kesilenSayisi = 0; //Yeni satıra basladik
178
179     while(ken < kenarSayisi) //Kenar Kontrol Dongusu
180     {
181       //Bu kenarı kesiyor muyuz?
182       if( y >= kenarlar[ken].dogru.altNokta.y
183           &&  y <= (kenarlar[ken].dogru.altNokta.y + kenarlar[ken].dogru.dY)
184           &&  kenarlar[ken].dogru.dY != 0
185           )
186       {//Kesiyoruz
187
188
```

```
189         //Kesim noktaları dizisine sırasına göre yerleştirelim
190         //Kesim noktası altnokta.x değeridir
191         int sira = 0;
192         while(sira < kenarSayisi) //kesim dizisine yerlestirme dongusu
193         {
194           if(kesSira[sira] == 0xFFFF)
195           {//Bu hücreye daha önce değer atanmamış, o halde atayalım
196             kesSira[sira] = kenarlar[ken].dogru.altNokta.x;
197             break;
198           }
199           else if(kesSira[sira] >= kenarlar[ken].dogru.altNokta.x)
200           {
201             /*
202              * Bu hücrede başka değer var
203              * Ve şimdiki kesim noktasından büyük ya da eşit
204              * O halde hepsini bir sağa kaydıralım...
205              */
206             int ind = kenarSayisi - 2;
207             while(ind >= sira)
208             {
209               kesSira[ind + 1] = kesSira[ind];
210               ind--;
211             }
212             kesSira[sira] = kenarlar[ken].dogru.altNokta.x;
213             break;
214           }
215           else
216           {
217             //kesim noktamız bu hücreçekinden büyük
218             //Bir sonrakine bakalım
219           }
220
221           sira++;
222         }//END kesim dizisine yerlestirme dongusu
223
224
225         //Artımsal Hesap yöntemiyle Bir Sonraki satırın kesim noktasını hesaplayalım
226         //Dik doğruysa, x aynı kalır
227         if(kenarlar[ken].dogru.dX != 0)
228         {
229           kenarlar[ken].counter += kenarlar[ken].dogru.dX;
230
231           while(kenarlar[ken].counter >= kenarlar[ken].dogru.dY)
232           {
233             kenarlar[ken].counter -= kenarlar[ken].dogru.dY;
234             if(kenarlar[ken].dogru.negEg)
235               kenarlar[ken].dogru.altNokta.x -= 1;
236             else
237               kenarlar[ken].dogru.altNokta.x += 1;
238           }
239         }
240
241         kesilenSayisi++;
242       }
243
244       ken++;
245     }//END Kenar Kontrol Dongusu
246
247
248 //    if(   kesilenSayisi && ( kesilenSayisi == 1 || (kesilenSayisi/2)*2 != kesilenSayisi )   )
249 //    {
250 //      kesSira[kesilenSayisi-1];
251 //    }
```

```
252
253
254     //Boyama
255     i = 0;
256     while(i < kesilenSayisi)
257     {
258
259       if(boya)
260       {
261         int start = kesSira[i];
262         int end = kesSira[i+1];
263         while(start <= end)
264         {
265           pPikselBoya(nokta(start, y), renk);
266           start++;
267         }
268         boya = 0;
269       }
270       else
271       {
272         boya = 1;
273       }
274       i++;
275     }
276
277
278     y++;
279     if(y > bitisSatiri)
280       break;//boyama bitmis
281   }//END Tarama Satirlari Dongusu END
282
283   return 1;
284 }
285
286
287
288
289
290 int poligonBoya_1(struct tPoligon* pol, int baslangicSatiri, int bitisSatiri, void* kenarVeri, unsigned
    int renk)
291 {
292   int y;
293   int i = 0;
294   int kesilenSayisi = 0;
295   int boya = 1;
296   int baslamaSatiri;
297   unsigned int kenarSayisi;
298   struct tKenarDizisi* pKenDiz;
299   struct tKenarBilgisi* kenarlar;
300   int kesSira[pol->kenarSayisi]; //Kesme noktalarının sıralanacağı dizi
301   int maxY = 1024;
302
303   struct tBltKomut blt;
304
305   pKenDiz = (struct tKenarDizisi*)kenarVeri;
306   kenarSayisi = pKenDiz->kenarSayisi;
307   kenarlar = &(pKenDiz->ilkKenar);
308   baslamaSatiri = baslangicSatiri;
309
310
311   y = baslamaSatiri;
312   while(y < maxY) //Tarama Satirlari Dongusu
313   {
```

```
314     int dummy;//FIXME sil
315     int ken = 0; //kesip kesmediğimize bakacağımız kenar indeksi
316
317     if(y == 13)
318       dummy = 0;
319
320     /*
321      * Siralama dizisini büyük bir değerle ilklendirerek
322      * ilgili indekse ilk kez veri koyup koymadığımızı anlayabilelim
323      */
324     i = 0;
325     while(i < kenarSayisi)
326       kesSira[i++] = 0xFFFF;
327
328     kesilenSayisi = 0; //Yeni satıra basladik
329
330     while(ken < kenarSayisi) //Kenar Kontrol Dongusu
331     {
332       //Bu kenarı kesiyor muyuz?
333       if( y >= kenarlar[ken].dogru.altNokta.y
334           &&  y <= (kenarlar[ken].dogru.altNokta.y + kenarlar[ken].dogru.dY)
335           &&  kenarlar[ken].dogru.dY != 0
336           )
337       {//Kesiyoruz
338
339
340         //Kesim noktaları dizisine sırasına göre yerleştirelim
341         //Kesim noktası altnokta.x değeridir
342         int sira = 0;
343         while(sira < kenarSayisi) //kesim dizisine yerlestirme dongusu
344         {
345           if(kesSira[sira] == 0xFFFF)
346           {//Bu hücreye daha önce değer atanmamış, o halde atayalım
347             kesSira[sira] = kenarlar[ken].dogru.altNokta.x;
348             break;
349           }
350           else if(kesSira[sira] >= kenarlar[ken].dogru.altNokta.x)
351           {
352             /*
353              * Bu hücrede başka değer var
354              * Ve şimdiki kesim noktasından büyük ya da eşit
355              * O halde hepsini bir sağa kaydıralım...
356              */
357             int ind = kenarSayisi - 2;
358             while(ind >= sira)
359             {
360               kesSira[ind + 1] = kesSira[ind];
361               ind--;
362             }
363             kesSira[sira] = kenarlar[ken].dogru.altNokta.x;
364             break;
365           }
366           else
367           {
368             //kesim noktamız bu hücrecekinden büyük
369             //Bir sonrakine bakalım
370           }
371
372           sira++;
373         }//END kesim dizisine yerlestirme dongusu
374
375
376         //Artımsal Hesap yöntemiyle Bir Sonraki satırın kesim noktasını hesaplayalım
```

```
377         //Dik doğruysa, x aynı kalır
378         if(kenarlar[ken].dogru.dX != 0)
379         {
380           kenarlar[ken].counter += kenarlar[ken].dogru.dX;
381
382           while(kenarlar[ken].counter >= kenarlar[ken].dogru.dY)
383           {
384             kenarlar[ken].counter -= kenarlar[ken].dogru.dY;
385             if(kenarlar[ken].dogru.negEg)
386               kenarlar[ken].dogru.altNokta.x -= 1;
387             else
388               kenarlar[ken].dogru.altNokta.x += 1;
389           }
390         }
391
392         kesilenSayisi++;
393       }
394
395       ken++;
396     }//END Kenar Kontrol Dongusu
397
398
399 //    if(   kesilenSayisi && ( kesilenSayisi == 1 || (kesilenSayisi/2)*2 != kesilenSayisi )   )
400 //    {
401 //      kesSira[kesilenSayisi-1];
402 //    }
403
404
405     //Boyama
406     i = 0;
407     while(i < kesilenSayisi && kesilenSayisi > 1)
408     {
409
410       if(boya)
411       {
412         int start = kesSira[i];
413         int end = kesSira[i+1];
414
415         blt.argb = renk;
416         blt.blt_col_size = 1;
417         blt.blt_row_size = end - start + 1;
418         blt.flags = SOL_ALT_KOSE;
419         blt.frm_buf_row_size = satirUzunlugu;
420         blt.hedef_addr = (unsigned int)koordToAddr(nokta(start, y));
421         blt.kaynak_addr = 0;//dummy
422
423         bitBLT_Go((void*)BITBLT_BASE_ADDR, &blt);
424
425 //        while(start <= end)
426 //        {
427 //          pPikselBoya(nokta(start, y), renk);
428 //          start++;
429 //        }
430         boya = 0;
431       }
432       else
433       {
434         boya = 1;
435       }
436       i++;
437     }
438
439
```

```
440     y++;
441     if(y > bitisSatiri)
442       break;//boyama bitmis
443   }//END Tarama Satirlari Dongusu END
444
445   return 1;
446 }
447
448
449
450
451
452 int poligonBoya(struct tPoligon* pol, void* buffer, unsigned int renk)
453 {
454   int i;
455   int baslamaSatiri;
456   int bitisSatiri;
457   unsigned int kenarSayisi;
458   struct tKenarDizisi* pKenDiz;
459   struct tKenarBilgisi* kenarlar;
460
461   if( !kenarDizisiOlustur(buffer, pol) )
462     return 0;
463
464   pKenDiz = (struct tKenarDizisi*)buffer;
465   kenarSayisi = pKenDiz->kenarSayisi;
466   kenarlar = &(pKenDiz->ilkKenar);
467
468   i = 0;
469   baslamaSatiri = 1000000; //Olamayacak kadar büyük bir değer
470   while(i < kenarSayisi)
471   {
472     if(kenarlar[i].dogru.altNokta.y < baslamaSatiri)
473       baslamaSatiri = kenarlar[i].dogru.altNokta.y;
474
475     i++;
476   }
477
478   i = 0;
479   bitisSatiri = -1000000; //Olamayacak kadar küçük bir değer
480   while(i < kenarSayisi)
481   {
482     if(kenarlar[i].dogru.altNokta.y + (int)(kenarlar[i].dogru.dY) > bitisSatiri)
483       bitisSatiri = kenarlar[i].dogru.altNokta.y + kenarlar[i].dogru.dY;
484
485     i++;
486   }
487
488   return poligonBoya_1(pol, baslamaSatiri, bitisSatiri, buffer, renk);
489
490 }
491
492
493
494
495
496 int poligonBoyaSoft(struct tPoligon* pol, void* buffer, unsigned int renk, void (*pPikselBoya)(struct
    tNokta, unsigned int rçb))
497 {
498   int i;
499   int baslamaSatiri;
500   int bitisSatiri;
501   unsigned int kenarSayisi;
```

```
502   struct tKenarDizisi* pKenDiz;
503   struct tKenarBilgisi* kenarlar;
504
505   if( !kenarDizisiOlustur(buffer, pol) )
506     return 0;
507
508   pKenDiz = (struct tKenarDizisi*)buffer;
509   kenarSayisi = pKenDiz->kenarSayisi;
510   kenarlar = &(pKenDiz->ilkKenar);
511
512   i = 0;
513   baslamaSatiri = 1000000; //Olamayacak kadar büyük bir değer
514   while(i < kenarSayisi)
515   {
516     if(kenarlar[i].dogru.altNokta.y < baslamaSatiri)
517       baslamaSatiri = kenarlar[i].dogru.altNokta.y;
518
519     i++;
520   }
521
522   i = 0;
523   bitisSatiri = -1000000; //Olamayacak kadar küçük bir değer
524   while(i < kenarSayisi)
525   {
526     if(kenarlar[i].dogru.altNokta.y + (int)(kenarlar[i].dogru.dY) > bitisSatiri)
527       bitisSatiri = kenarlar[i].dogru.altNokta.y + kenarlar[i].dogru.dY;
528
529     i++;
530   }
531
532   return poligonBoya_1_Soft(pol, baslamaSatiri, bitisSatiri, buffer, renk, pPikselBoya);
533
534 }
535
536
537 int poligonCiz(struct tPoligon* pol, unsigned int renk)
538 {
539   int kose = 0;
540   struct tNokta* koseler = pol->koseler;
541
542   if(pol->kenarSayisi < 3)
543       return 0;
544
545   while(kose < pol->kenarSayisi)
546   {
547     bresenhamDogruCiz(koseler[kose], koseler[(kose+1)%pol->kenarSayisi], renk, 0);
548
549     kose++;
550   }
551
552   return 1;
553 }
554
555 int poligonCevir(struct tPoligon* pol, struct tNokta eksen, int teta)
556 {
557   int kose = 0;
558   struct tNokta* koseler = pol->koseler;
559
560   if(pol->kenarSayisi < 3)
561       return 0;
562
563   while(kose < pol->kenarSayisi)
564   {
```

```
565     koseler[kose] = noktaCevir(eksen, koseler[kose], teta);
566     kose++;
567   }
568 
569   return 1;
570 }
571 
572 int poligonScale(struct tPoligon* pol, double scale)
573 {
574   int kcse = 0;
575   struct tNokta* koseler = pol->koseler;
576   double db_x, db_y;
577 
578   if(pol->kenarSayisi < 3)
579       return 0;
580 
581   while(kose < pol->kenarSayisi)
582   {
583     db_x = (double)koseler[kose].x * scale;
584     db_y = (double)koseler[kose].y * scale;
585 
586     koseler[kose].x = roundDouble(db_x);
587     koseler[kose].y = roundDouble(db_y);
588     kose++;
589   }
590 
591   return 1;
592 }
593 
594 int poligonTasi(struct tPoligon* pol, struct tNokta rokta)
595 {
596   int kcse = 0;
597   struct tNokta* koseler = pol->koseler;
598 
599   if(pol->kenarSayisi < 3)
600       return 0;
601 
602   while(kose < pol->kenarSayisi)
603   {
604     koseler[kose].x += nokta.x;
605     koseler[kose].y += nokta.y;
606     kose++;
607   }
608 
609   return 1;
610 }
611 
612 
```

```
1
2   #include "xparameters.h"
3
4   #include "xutil.h"
5
6   #include "xio.h"
7
8   #include "_koordSystemXUP.h"
9
10  #include "_grafikOrtak.h"
11
12  #include "_bresenham.h"
13
14  #include "_poligonBoya.h"
15  #include "ipcore_BitBLT2.h"
16  //#include "_bitBLT.h"
17
18  #include "text_api.h"
19
20  struct tNokta noktalar[100];
21  char buffer[1000];
22
23  void bekle(int saniye)
24  {
25    int i,j;
26    for(j=0;j<saniye;j++)
27    {
28      for(i=0;i<30000000;i++);
29    }
30  }
31
32  void arkaPlanKopyala(void)
33  {
34    int i;
35    unsigned int *kaynak, *hedef;
36    unsigned int deger;
37
38    kaynak = koordToAddr(nokta(0,0));
39    hedef = koordToArkaAddr(nokta(0,0));
40
41    for(i=0;i<1024*484;i++)
42    {
43      deger = XIo_In32((XIo_Address)kaynak);
44      *hedef = deger;
45
46      kaynak++;
47      hedef++;
48    }
49
50  }
51
52
53  void arka_plan_yap(void)
54  {
55    int dataLen = 0;
56    struct tPoligon pol;
57    struct tBltKomut bltKomut;
58
59    ekranTemizle(SIYAH | 0xFF000000);
60    bekle(1);
61
62    bresenhamDogruCiz( nokta( 20  , 0 ) , nokta(  0 , 20  ) ,YESIL | 0x7D000000 , 0 );
63    bresenhamDogruCiz( nokta( 40  , 0 ) , nokta(  0 , 40  ) ,KIRMIZI | 0x7D000000 , 1 );
```

```
64    bresenhamDogruCiz( nokta( 60  , 0 ) , nokta(  0 , 60  ) ,MAVI | 0x7D000000  , 0 );
65    bresenhamDogruCiz( nokta( 80  , 0 ) , nokta(  0 , 80  ) ,SARI | 0x7D000000  , 1 );
66    bresenhamDogruCiz( nokta( 100 , 0 ) , nokta(  0 , 100 ) ,MAGENTA | 0x7D000000 , 0 );
67    bresenhamDogruCiz( nokta( 120 , 0 ) , nokta(  0 , 120 ) ,CYAN | 0x7D000000  , 1 );
68    bresenhamDogruCiz( nokta( 140 , 0 ) , nokta(  0 , 140 ) ,BEYAZ | 0x7D000000 , 0 );
69    bresenhamDogruCiz( nokta( 160 , 0 ) , nokta(  0 , 160 ) ,YESIL | 0x7D000000 , 1 );
70    bresenhamDogruCiz( nokta( 180 , 0 ) , nokta(  0 , 180 ) ,KIRMIZI | 0x7D000000 , 0 );
71    bresenhamDogruCiz( nokta( 200 , 0 ) , nokta(  0 , 200 ) ,MAVI | 0x7D000000  , 1 );
72    bresenhamDogruCiz( nokta( 220 , 0 ) , nokta(  0 , 220 ) ,SARI | 0x7D000000  , 0 );
73    bresenhamDogruCiz( nokta( 240 , 0 ) , nokta(  0 , 240 ) ,MAGENTA | 0x7D000000 , 1 );
74    bresenhamDogruCiz( nokta( 260 , 0 ) , nokta(  0 , 260 ) ,CYAN | 0x7D000000  , 0 );
75    bresenhamDogruCiz( nokta( 280 , 0 ) , nokta(  0 , 280 ) ,BEYAZ | 0x7D000000 , 1 );
76    bresenhamDogruCiz( nokta( 300 , 0 ) , nokta(  0 , 300 ) ,YESIL | 0x7D000000 , 0 );
77    bresenhamDogruCiz( nokta( 320 , 0 ) , nokta(  0 , 320 ) ,KIRMIZI | 0x7D000000 , 1 );
78    bresenhamDogruCiz( nokta( 340 , 0 ) , nokta(  0 , 340 ) ,MAVI | 0x7D000000  , 0 );
79    bresenhamDogruCiz( nokta( 360 , 0 ) , nokta(  0 , 360 ) ,SARI | 0x7D000000  , 1 );
80    bresenhamDogruCiz( nokta( 380 , 0 ) , nokta(  0 , 380 ) ,MAGENTA | 0x7D000000 , 0 );
81    bresenhamDogruCiz( nokta( 400 , 0 ) , nokta(  0 , 400 ) ,CYAN | 0x7D000000  , 1 );
82    bresenhamDogruCiz( nokta( 420 , 0 ) , nokta(  0 , 420 ) ,BEYAZ | 0x7D000000 , 0 );
83    bresenhamDogruCiz( nokta( 440 , 0 ) , nokta(  0 , 440 ) ,YESIL | 0x7D000000 , 1 );
84    bresenhamDogruCiz( nokta( 460 , 0 ) , nokta(  0 , 460 ) ,KIRMIZI | 0x7D000000 , 0 );
85    bresenhamDogruCiz( nokta( 480 , 0 ) , nokta(  0 , 480 ) ,MAVI | 0x7D000000  , 1 );
86    bresenhamDogruCiz( nokta( 500 , 0 ) , nokta(  20  , 480 ) ,SARI | 0x7D000000  , 0 );
87    bresenhamDogruCiz( nokta( 520 , 0 ) , nokta(  40  , 480 ) ,MAGENTA | 0x7D000000 , 1 );
88    bresenhamDogruCiz( nokta( 540 , 0 ) , nokta(  60  , 480 ) ,CYAN | 0x7D000000  , 0 );
89    bresenhamDogruCiz( nokta( 560 , 0 ) , nokta(  80  , 480 ) ,BEYAZ | 0x7D000000 , 1 );
90    bresenhamDogruCiz( nokta( 580 , 0 ) , nokta(  100 , 480 ) ,YESIL | 0x7D000000 , 0 );
91    bresenhamDogruCiz( nokta( 600 , 0 ) , nokta(  120 , 480 ) ,KIRMIZI | 0x7D000000 , 1 );
92    bresenhamDogruCiz( nokta( 620 , 0 ) , nokta(  140 , 480 ) ,MAVI | 0x7D000000  , 0 );
93    bresenhamDogruCiz( nokta( 640 , 0 ) , nokta(  160 , 480 ) ,SARI | 0x7D000000  , 1 );
94    bresenhamDogruCiz( nokta( 640 , 20  ) , nokta(  180 , 480 ) ,MAGENTA | 0x7D000000 , 0 );
95    bresenhamDogruCiz( nokta( 640 , 40  ) , nokta(  200 , 480 ) ,CYAN | 0x7D000000  , 1 );
96    bresenhamDogruCiz( nokta( 640 , 60  ) , nokta(  220 , 480 ) ,BEYAZ | 0x7D000000 , 0 );
97    bresenhamDogruCiz( nokta( 640 , 80  ) , nokta(  240 , 480 ) ,YESIL | 0x7D000000 , 1 );
98    bresenhamDogruCiz( nokta( 640 , 100 ) , nokta(  260 , 480 ) ,KIRMIZI | 0x7D000000 , 0 );
99    bresenhamDogruCiz( nokta( 640 , 120 ) , nokta(  280 , 480 ) ,MAVI | 0x7D000000  , 1 );
100   bresenhamDogruCiz( nokta( 640 , 140 ) , nokta(  300 , 480 ) ,SARI | 0x7D000000  , 0 );
101   bresenhamDogruCiz( nokta( 640 , 160 ) , nokta(  320 , 480 ) ,MAGENTA | 0x7D000000 , 1 );
102   bresenhamDogruCiz( nokta( 640 , 180 ) , nokta(  340 , 480 ) ,CYAN | 0x7D000000  , 0 );
103   bresenhamDogruCiz( nokta( 640 , 200 ) , nokta(  360 , 480 ) ,BEYAZ | 0x7D000000 , 1 );
104   bresenhamDogruCiz( nokta( 640 , 220 ) , nokta(  380 , 480 ) ,YESIL | 0x7D000000 , 0 );
105   bresenhamDogruCiz( nokta( 640 , 240 ) , nokta(  400 , 480 ) ,KIRMIZI | 0x7D000000 , 1 );
106   bresenhamDogruCiz( nokta( 640 , 260 ) , nokta(  420 , 480 ) ,MAVI | 0x7D000000  , 0 );
107   bresenhamDogruCiz( nokta( 640 , 280 ) , nokta(  440 , 480 ) ,SARI | 0x7D000000  , 1 );
108   bresenhamDogruCiz( nokta( 640 , 300 ) , nokta(  460 , 480 ) ,MAGENTA | 0x7D000000 , 0 );
109   bresenhamDogruCiz( nokta( 640 , 320 ) , nokta(  480 , 480 ) ,CYAN | 0x7D000000  , 1 );
110   bresenhamDogruCiz( nokta( 640 , 340 ) , nokta(  500 , 480 ) ,BEYAZ | 0x7D000000 , 0 );
111   bresenhamDogruCiz( nokta( 640 , 360 ) , nokta(  520 , 480 ) ,YESIL | 0x7D000000 , 1 );
112   bresenhamDogruCiz( nokta( 640 , 380 ) , nokta(  540 , 480 ) ,KIRMIZI | 0x7D000000 , 0 );
113   bresenhamDogruCiz( nokta( 640 , 400 ) , nokta(  560 , 480 ) ,MAVI | 0x7D000000  , 1 );
114   bresenhamDogruCiz( nokta( 640 , 420 ) , nokta(  580 , 480 ) ,SARI | 0x7D000000  , 0 );
115   bresenhamDogruCiz( nokta( 640 , 440 ) , nokta(  600 , 480 ) ,MAGENTA | 0x7D000000 , 1 );
116   bresenhamDogruCiz( nokta( 640 , 460 ) , nokta(  620 , 480 ) ,CYAN | 0x7D000000  , 0 );
117
118
119   bltKomut.argb = preMultipliedARGB(100, BEYAZ);
120   bltKomut.blt_col_size = 30;
121   bltKomut.blt_row_size = 600;
122   bltKomut.flags = SOL_UST_KOSE;
123   bltKomut.frm_buf_row_size = 1024;
124   bltKomut.hedef_addr = koordToAddr(nokta(20,20));
125   bltKomut.kaynak_addr = 0;
126   bitBLT_Go((void*)BITBLT_BASE_ADDR, &bltKomut);
```

```
127
128   bltKomut.argb = preMultipliedARGB(100, BEYAZ);
129   bltKomut.blt_col_size = 30;
130   bltKomut.blt_row_size = 600;
131   bltKomut.flags = SOL_UST_KOSE;
132   bltKomut.frm_buf_row_size = 1024;
133   bltKomut.hedef_addr = koordToAddr(nckta(20,430));
134   bltKomut.kaynak_addr = 0;
135   bitBLT_Go((void*)BITBLT_BASE_ADDR, &bltKomut);
136
137   bltKomut.argb = preMultipliedARGB(100, BEYAZ);
138   bltKomut.blt_col_size = 380;
139   bltKomut.blt_row_size = 30;
140   bltKomut.flags = SOL_UST_KOSE;
141   bltKomut.frm_buf_row_size = 1024;
142   bltKomut.hedef_addr = koordToAddr(nckta(20,50));
143   bltKomut.kaynak_addr = 0;
144   bitBLT_Go((void*)BITBLT_BASE_ADDR, &bltKomut);
145
146   bltKomut.argb = preMultipliedARGB(100, BEYAZ);
147   bltKomut.blt_col_size = 380;
148   bltKomut.blt_row_size = 30;
149   bltKomut.flags = SOL_UST_KOSE;
150   bltKomut.frm_buf_row_size = 1024;
151   bltKomut.hedef_addr = koordToAddr(nckta(590,50));
152   bltKomut.kaynak_addr = 0;
153   bitBLT_Go((void*)BITBLT_BASE_ADDR, &bltKomut);
154 }
155
156
157 int main (void) {
158
159 Xuint32 baseaddr = 0xc7200000;
160   Xuint32 xx;
161   struct tBltKomut bltKomut;
162
163   static struct tNokta koseler_Yildiz1[10]={                    { 51  , 0 } , { 85  , 37  } , { 139 ,
      23  } , { 110 , 68  } , { 138 , 108 } , { 88  , 100 } , { 54  , 138 } , { 50  , 89  } , { 0 , 71  } ,
      { 47  , 52  }
164           };
165   static struct tPoligon pol_Yildiz1 = { 10, koseler_Yildiz1 };
166
167
168   int i,j;
169   unsigned int renk = KIRMIZI;
170   void* addr = (void*)(frameBufStart + (4*1024*100));
171
172   struct tNokta n1;
173   struct tNokta n2;
174
175
176   int dataLen = 0;
177   struct tPoligon pol;
178   unsigned int* kenSay;
179
180   struct tKenarBilgisi* ken1;
181   struct tKenarBilgisi* ken2;
182   struct tKenarBilgisi* ken3;
183   struct tKenarBilgisi* ken4;
184   struct tKenarBilgisi* ken5;
185   struct tKenarBilgisi* ken6;
186
187
```

```
188
189 //   print("-- ekran temizle gir\r\n");
190 //
191 //  ekranTemizle(0x78509040);
192 //  print("-- ekran temizle cik\r\n");
193
194
195   print("-- uc adet blt go basla\r\n");
196
197
198
199     bltKomut.argb = KIRMIZI | 0x78904050;
200     bltKomut.blt_col_size = 100;
201     bltKomut.blt_row_size = 100;
202     bltKomut.flags = SOL_UST_KOSE |  KAYNAK_ADDR ;
203     bltKomut.frm_buf_row_size = 1024;
204     bltKomut.hedef_addr = 0x07E00100 + 1024*4*20;
205     bltKomut.kaynak_adcr = 0x07E00300 + 1024*4*20;
206
207     bitBLT_Go((void*)BITBLT_BASE_ADDR, &bltKomut);
208
209
210     bltKomut.argb = BEYAZ;
211     bltKomut.blt_col_size = 100;
212     bltKomut.blt_row_size = 100;
213     bltKomut.flags = SOL_UST_KOSE |  KAYNAK_ADDR | ALPHA;
214     bltKomut.frm_buf_row_size = 1024;
215     bltKomut.hedef_addr = 0x07E00200 + 1024*4*20;
216     bltKomut.kaynak_adcr = 0x07E00400 + 1024*4*20;
217
218     bitBLT_Go((void*)BITBLT_BASE_ADDR, &bltKomut);
219
220     print("-- uc adet blt go bitti\r\n");
221
222
223 //    bltKomut.argb = CYAN;
224 //    bltKomut.blt_col_size = 300;
225 //    bltKomut.blt_row_size = 300;
226 //    bltKomut.flags = SOL_UST_KOSE;
227 //    bltKomut.frm_buf_row_size = 1024;
228 //    bltKomut.hedef_addr = 0x07E00400 + 1024*4*20;
229 //    bltKomut.kaynak_addr = 1;
230 //
231 //    bitBLT_Go((void*)BITBLT_BASE_ADDR, &bltKomut);
232
233
234
235
236
237 //    pol.kenarSayisi = 4;
238 //    pol.koseler = noktalar;
239 //    noktalar[0] = nokta(100,300);
240 //    noktalar[1] = nokta(300,400);
241 //    noktalar[2] = nokta(250,450);
242 //    noktalar[3] = nokta(150,420);
243 //    dataLen = poligonBoya(&pol, buffer, CYAN);
244
245     print("-- bekle 1e gir\r\n");
246     bekle(1);
247
248     print("-- bekle 1den ciktik\r\n");
249
250     //char_A(nokta(20,20),10, KIRMIZI, 0);
```

```
251
252
253 //
254 //    bekle(1);
255
256
257
258
259     arka_plan_yap();
260
261     print("-- write text\r\n");
262     writeText("FPGA", 4);
263     print("-- write text cikis\r\n");
264
265     arkaPlanKopyala();
266     print("-- arka plan yaptim, kopyaladim\r\n");
267
268
269 //
270
271 //    print("-- bir pol boyuycam\r\n");
272 //    pol.kenarSayisi = 4;
273 //    pol.koseler = noktalar;
274 //    noktalar[0] = nokta(100,300);
275 //    noktalar[1] = nokta(300,400);
276 //    noktalar[2] = nokta(250,450);
277 //    noktalar[3] = nokta(150,420);
278 //    dataLen = poligonBoya(&pol, buffer, 0x0000AA00 | 0x2D000000);
279 //    print("-- bir pol komutu verildi\r\n");
280 //
281 //
282
283
284 //    print("-- bir yildiz boyuycam\r\n");
285 //    dataLen = poligonBoya(&pol_Yildiz1, buffer, preMultipliedARGB(40, 0x0000DABC));
286 //    print("-- bir yildiz komutu verildi\r\n");
287
288 //    print("-- bir blt go yapcam kaynak guncellemeli\r\n");
289 //    bltKomut.argb = preMultipliedARGB(40, 0x0000DAEC);
290 //    bltKomut.blt_col_size = 145;
291 //    bltKomut.blt_row_size = 145;
292 //    bltKomut.flags = SOL_UST_KOSE |  KAYNAK_ADDR | ALPHA | TRANSPARAN_NOT_ARGB | KAYNAK_GUNCELLE_ADDR;
293 //    bltKomut.frm_buf_row_size = 1024;
294 //    bltKomut.hedef_addr = koordToAddr(nokta(300, 200));
295 //    bltKomut.kaynak_addr = koordToAddr(nokta(0, 0));
296 //
297 //    arkaPlanAddrYaz(koordToArkaAddr(nokta(0, 0)));
298 //        print("-- arka addr yazdim\r\n");
299 //
300 //    bitBLT_Go((void*)BITBLT_BASE_ADDR, &bltKomut);
301
302     print("-- bir blt ço gonderdim kaynaga bak!!!\r\n");
303
304    print("-- Exiting main() --\r\n");
305    return 0;
306 }
307
308
309
310
311
312 //
313 //
```

```
314 //
315 //  n1.x = 100;
316 //  n1.y = 100;
317 //  n2.x = 200;
318 //  n2.y = 450;
319 //
320 //
321 //  bresenhamDogruCiz(n1, n2, 0x0000FF00, pikselBoyaXUP);
322 //
323 //  while(1)
324 //  {
325 //    for(i=0;i<500000;i++);
326 //    n1.x = (n1.x + 2) % 640;
327 //    n1.y = (n1.y + 1) % 484;
328 //
329 //    n2.x = (n2.x + 1) % 640;
330 //    n2.y = (n2.y + 2) % 484;
331 //    bresenhamDogruCiz(n1, n2, renk++, pikselBoyaXUP);
332 //  }
333 //
```

```
/*
 * text_api.h
 *
 *  Created on: 03.Kas.2010
 *      Author: SERDAR
 */

#ifndef TEXT_API_H_
#define TEXT_API_H_

#include "_grafikOrtak.h"
#include "_poligonBoya.h"

void writeText(const char* str, int strLen);

#endif /* TEXT_API_H_ */



//int char_dummy(struct tNokta location, unsigned int scale, unsigned int renk, unsigned int option)
//{
//  //dummy
//  return 10*scale; //space
//}
//
//int char_null(struct tNokta location, unsigned int scale, unsigned int renk, unsigned int option)
//{
//  return 0;
//}
//
//int char_space(struct tNokta location, unsigned int scale, unsigned int renk, unsigned int option)
//{
//  return 10*scale;
//}
//
//int char_A(struct tNokta location, unsigned int scale, unsigned int renk, unsigned int option)
//{
//  unsigned int k = scale;
//  int a = location.x;
//  int b = location.y;
//
//  struct tNokta koseler_pol1[7] =
//  {
//      {a, b+10*k},        {a+2*k, b+10*k},          {a+5*k, b+2*k},
//      {a+8*k, b+10*k},          {a+10*k, b+10*k},         {a+6*k, b},     {a+4*k, b},
//
//  };
//
//  struct tNokta koseler_pol2[4] =
//  {
//      {a+3*k, b+7*k},       {a+7*k, b+7*k},       {a+6*k, b+5*k},       {a+4*k, b+5*k},
//
//  };
//
//  struct tPoligon pol1 =
//  {
//      7, &koseler_pol1
//  };
//
//  struct tPoligon pol2 =
//  {
//      4, &koseler_pol2
//  };
```

```
64 //
65 //  struct tPoligon pols[2] =
66 //  {
67 //      pol1, pol2
68 //  };
69 //
70 //
71 //  char_drawer(pols, 2, renk, option);
72 //
73 //  return 10*scale;
74 //}
```

```
1
2   /*
3    * text_api.c
4    *
5    *  Created on: 03.Kas.2010
6    *      Author: SERDAR
7    */
8   #include "text_api.h"
9
10
11  struct tPoligonKarakter
12  {
13    int pcligonSayisi;
14    struct tPoligon** pols;
15  };
16
17  static struct tPoligonKarakter pchar_dummy = { 0, NULL };
18
19  /*********** KARAKTER TABLOSU *************/
20
21  // karakter: A
22  static struct tNokta koseler_A1[7] = { {0, 10}, {2, 10}, {5, 2}, {8, 10},
23      {10, 10}, {6, 0}, {4, 0} };
24  static struct tPoligon pol_A1 = { 7, koseler_A1 };
25  static struct tNokta koseler_A2[4] = { {3, 7}, {7, 7], {6, 5}, {4, 5} };
26  static struct tPoligon pol_A2 = { 4, koseler_A2 };
27  static struct tPoligon* pols_A[2] = { &pol_A1, &pol_A2 };
28  static struct tPoligonKarakter pchar_A = { 2, pols_A };
29
30  // karakter: B
31  static struct tNokta koseler_B1[4] = { {1,0}, {4,0}, {4,10}, {1,10} };//4
32  static struct tPoligon pol_B1 = { 4, koseler_B1 };
33  static struct tNokta koseler_B2[17] = { {4,0}, {8,0}, {10,2}, {10,4}, {9,5},
34      {10,6}, {10,8}, {8,10}, {4,10}, {4,8}, {8,8}, {8,6}, {4,6}, {4,4},
35      {8,4}, {8,2}, {4,2}  };//17
36  static struct tPoligon pol_B2 = { 17, koseler_B2 };
37  static struct tPoligon* pols_B[2] = { &pol_B1, &pol_B2 };
38  static struct tPoligonKarakter pchar_B = { 2, pols_B };
39
40  // karakter: C
41  static struct tNokta koseler_C1[22] = { {3,0}, {7,0}, {9,1}, {10,2}, {10,3},
42      {8,3}, {6,2}, {4,2}, {2,3}, {2,7}, {4,8}, {6,8}, {8,7}, {10,7},
43      {10,8}, {9,9}, {7,10}, {3,10}, {1,9}, {0,8}, {0,2}, {1,1} };//22
44  static struct tPoligon pol_C1 = { 22, koseler_C1 };
45  static struct tPoligon* pols_C[1] = { &pol_C1 };
46  static struct tPoligonKarakter pchar_C = { 1, pols_C };
47
48  // karakter:F
49  static struct tNokta koseler_F1[10]={                    { 0 , 0 } , { 10  , 0 } , { 10  , 2 } , { 3 ,
    2 } , { 3 , 4 } , { 7 , 4 } , { 7 , 6 } , { 3 , 6 } , { 3 , 10  } , { 0 , 10  }
50          };
51  static struct tPoligon pol_F1 = { 10, koseler_F1 };
52  static struct tPoligon* pols_F[1] = { &pol_F1 };
53  static struct tPoligonKarakter pchar_F = { 1, pols_F];
54
55
56  // karakter:P
57  static struct tNokta koseler_P1[4]={                     { 1 , 0 } , { 4 , 0 } , { 4 , 10  } , { 1 , 10
    }
58          };
59  static struct tPoligon pol_P1 = { 4, koseler_P1 };
60  static struct tNokta koseler_P2[10]={                    { 4 , 0 } , { 8 , 0 } , { 10  , 1 } , { 10  ,
    5 } , { 8 , 6 } , { 4 , 6 } , { 4 , 4 } , { 8 , 4 } , { 8 , 2 } , { 4 , 2 }
```

```
61          };
62  static struct tPoligon pol_P2 = { 10, koseler_P2 };
63  static struct tPoligon* pols_P[2] = { &pol_P1, &pol_P2};
64  static struct tPoligonKarakter pchar_P = { 2, pols_P};
65
66  // karakter:G
67  static struct tNokta koseler_G1[20]={                    { 3 , 0 } , { 7 , 0 } , { 10  , 2 } , { 10  ,
    3 } , { 8 , 3 } , { 6 , 2 } , { 4 , 2 } , { 2 , 3 } , { 2 , 7 } , { 4 , 8 } , { 7 , 8 } , { 8 , 7 } , {
    6 , 7 } , { 6 , 5 } , { 10  , 5 } , { 10  , 8 } , { 7 , 10  } , { 3 , 10  } , { 0 , 8 } , { 0 , 2 }
68          };
69  static struct tPoligon pol_G1 = { 20, koseler_G1 };
70  static struct tPoligon* pols_G[1] = { &pol_G1 };
71  static struct tPoligonKarakter pchar_G = { 1, pols_G};
72
73
74
75
76
77  char charDataBuffer[5000];
78
79  int yatayPikselSayisi = gorunurSatirUzunlugu;
80  int dikeyPikselSayisi = gorunurSutunUzunlugu;
81
82  double textScale = 8.5;
83  unsigned int textRenk;// = preMultipliedARGB(100, KIRMIZI);
84  unsigned int textOption;
85
86  #define karakterArasiBosluk (roundDouble(1.3*textScale))
87  #define satirArasiBosluk (roundDouble(5*textScale))
88  #define karakterBoyutu (roundDouble(10*textScale))
89
90  #define cursorHome (nokta(125, 200))
91
92  #define satirKarakterSayisi (   (yatayPikselSayisi-2*cursorHome.x)/(karakterBoyutu+karakterArasiBosluk)
    )
93  #define sutunKarakterSayisi (   (dikeyPikselSayisi-2*cursorHome.y)/(karakterBoyutu+satirArasiBosluk)   )
94
95  //int charNumber = 0;
96  int cursor = 0;
97
98  struct tNokta charPosition(int charNo)
99  {
100   struct tNokta location = cursorHome;
101   int satirSayisi = charNo / satirKarakterSayisi;
102   int satirCharNo = charNo % satirKarakterSayisi;
103
104   location.y += satirSayisi * (karakterBoyutu + satirArasiBosluk);
105   location.x += satirCharNo * (karakterBoyutu + karakterArasiBosluk);
106
107   return location;
108 }
109
110
111 int char_drawer(struct tPoligonKarakter* pchar, struct tNokta location, unsigned int renk, unsigned int
    draw_option)
112 {
113   int pcligonSayisi = pchar->poligonSayisi;
114   struct tPoligon** pols = pchar->pols;
115   struct tPoligon pol;
116   int pcl_index = 0;
117
118
119   while(pol_index < poligonSayisi)
```

```
120   {
121     pol = *pols[pol_index];
122 
123     poligonScale(&pol, textScale);
124     poligonTasi(&pol, location);
125 
126     //TODO drawing options!!!
127     poligonBoya(&pol, charDataBuffer, renk);
128 
129 
130     pol_index++;
131   }
132 
133   return 1;//FIXME
134 
135 }
136 
137 
138 
139 
140 static struct tPoligonKarakter* ascii_pchars[256] =
141 {
142     &pchar_dummy,   //0  NULL
143     &pchar_dummy,   //1
144     &pchar_dummy,   //2
145     &pchar_dummy,   //3
146     &pchar_dummy,   //4
147     &pchar_dummy,   //5
148     &pchar_dummy,   //6
149     &pchar_dummy,   //7
150     &pchar_dummy,   //8
151     &pchar_dummy,   //9
152     &pchar_dummy,   //10
153     &pchar_dummy,   //11
154     &pchar_dummy,   //12
155     &pchar_dummy,   //13
156     &pchar_dummy,   //14
157     &pchar_dummy,   //15
158     &pchar_dummy,   //16
159     &pchar_dummy,   //17
160     &pchar_dummy,   //18
161     &pchar_dummy,   //19
162     &pchar_dummy,   //20
163     &pchar_dummy,   //21
164     &pchar_dummy,   //22
165     &pchar_dummy,   //23
166     &pchar_dummy,   //24
167     &pchar_dummy,   //25
168     &pchar_dummy,   //26
169     &pchar_dummy,   //27
170     &pchar_dummy,   //28
171     &pchar_dummy,   //29
172     &pchar_dummy,   //30
173     &pchar_dummy,   //31
174     &pchar_dummy,   //32  Space
175     &pchar_dummy,   //33
176     &pchar_dummy,   //34
177     &pchar_dummy,   //35
178     &pchar_dummy,   //36
179     &pchar_dummy,   //37
180     &pchar_dummy,   //38
181     &pchar_dummy,   //39
182     &pchar_dummy,   //40
```

```
183     &pchar_dummy,   //41
184     &pchar_dummy,   //42
185     &pchar_dummy,   //43
186     &pchar_dummy,   //44
187     &pchar_dummy,   //45
188     &pchar_dummy,   //46
189     &pchar_dummy,   //47
190     &pchar_dummy,   //48  0
191     &pchar_dummy,   //49  1
192     &pchar_dummy,   //50  2
193     &pchar_dummy,   //51  3
194     &pchar_dummy,   //52  4
195     &pchar_dummy,   //53  5
196     &pchar_dummy,   //54  6
197     &pchar_dummy,   //55  7
198     &pchar_dummy,   //56  8
199     &pchar_dummy,   //57  9
200     &pchar_dummy,   //58  :
201     &pchar_dummy,   //59  ;
202     &pchar_dummy,   //60  <
203     &pchar_dummy,   //61  =
204     &pchar_dummy,   //62  >
205     &pchar_dummy,   //63  ?
206     &pchar_dummy,   //64  @
207     &pchar_A,     //65  A
208     &pchar_B,     //66  B
209     &pchar_C,     //67  C
210     &pchar_dummy,   //68  D
211     &pchar_dummy,   //69  E
212     &pchar_F,   //70  F
213     &pchar_G,   //71  G
214     &pchar_dummy,   //72  H
215     &pchar_dummy,   //73  I
216     &pchar_dummy,   //74  J
217     &pchar_dummy,   //75  K
218     &pchar_dummy,   //76  L
219     &pchar_dummy,   //77  M
220     &pchar_dummy,   //78  N
221     &pchar_dummy,   //79  O
222     &pchar_P,   //80  P
223     &pchar_dummy,   //81  Q
224     &pchar_dummy,   //82  R
225     &pchar_dummy,   //83  S
226     &pchar_dummy,   //84  T
227     &pchar_dummy,   //85  U
228     &pchar_dummy,   //86  V
229     &pchar_dummy,   //87  W
230     &pchar_dummy,   //88  X
231     &pchar_dummy,   //89  Y
232     &pchar_dummy,   //90  Z
233     &pchar_dummy,   //91
234     &pchar_dummy,   //92
235     &pchar_dummy,   //93
236     &pchar_dummy,   //94
237     &pchar_dummy,   //95
238     &pchar_dummy,   //96
239     &pchar_dummy,   //97  a
240     &pchar_dummy,   //98  b
241     &pchar_dummy,   //99  c
242     &pchar_dummy,   //100 d
243     &pchar_dummy,   //101 e
244     &pchar_dummy,   //102 f
245     &pchar_dummy,   //103 g
```

```
246     &pchar_dummy,   //104 h
247     &pchar_dummy,   //105 i
248     &pchar_dummy,   //106 j
249     &pchar_dummy,   //107 k
250     &pchar_dummy,   //108 l
251     &pchar_dummy,   //109 m
252     &pchar_dummy,   //110 n
253     &pchar_dummy,   //111 o
254     &pchar_dummy,   //112 p
255     &pchar_dummy,   //113 q
256     &pchar_dummy,   //114 r
257     &pchar_dummy,   //115 s
258     &pchar_dummy,   //116 t
259     &pchar_dummy,   //117 u
260     &pchar_dummy,   //118 v
261     &pchar_dummy,   //119 w
262     &pchar_dummy,   //120 x
263     &pchar_dummy,   //121 y
264     &pchar_dummy,   //122 z
265     &pchar_dummy,   //123
266     &pchar_dummy,   //124
267     &pchar_dummy,   //125
268     &pchar_dummy,   //126
269     &pchar_dummy,   //127
270     &pchar_dummy,   //128
271     &pchar_dummy,   //129
272     &pchar_dummy,   //130
273     &pchar_dummy,   //131
274     &pchar_dummy,   //132
275     &pchar_dummy,   //133
276     &pchar_dummy,   //134
277     &pchar_dummy,   //135
278     &pchar_dummy,   //136
279     &pchar_dummy,   //137
280     &pchar_dummy,   //138
281     &pchar_dummy,   //139
282     &pchar_dummy,   //140
283     &pchar_dummy,   //141
284     &pchar_dummy,   //142
285     &pchar_dummy,   //143
286     &pchar_dummy,   //144
287     &pchar_dummy,   //145
288     &pchar_dummy,   //146
289     &pchar_dummy,   //147
290     &pchar_dummy,   //148
291     &pchar_dummy,   //149
292     &pchar_dummy,   //150
293     &pchar_dummy,   //151
294     &pchar_dummy,   //152
295     &pchar_dummy,   //153
296     &pchar_dummy,   //154
297     &pchar_dummy,   //155
298     &pchar_dummy,   //156
299     &pchar_dummy,   //157
300     &pchar_dummy,   //158
301     &pchar_dummy,   //159
302     &pchar_dummy,   //160
303     &pchar_dummy,   //161
304     &pchar_dummy,   //162
305     &pchar_dummy,   //163
306     &pchar_dummy,   //164
307     &pchar_dummy,   //165
308     &pchar_dummy,   //166
```

```
309     &pchar_dummy,   //167
310     &pchar_dummy,   //168
311     &pchar_dummy,   //169
312     &pchar_dummy,   //170
313     &pchar_dummy,   //171
314     &pchar_dummy,   //172
315     &pchar_dummy,   //173
316     &pchar_dummy,   //174
317     &pchar_dummy,   //175
318     &pchar_dummy,   //176
319     &pchar_dummy,   //177
320     &pchar_dummy,   //178
321     &pchar_dummy,   //179
322     &pchar_dummy,   //180
323     &pchar_dummy,   //181
324     &pchar_dummy,   //182
325     &pchar_dummy,   //183
326     &pchar_dummy,   //184
327     &pchar_dummy,   //185
328     &pchar_dummy,   //186
329     &pchar_dummy,   //187
330     &pchar_dummy,   //188
331     &pchar_dummy,   //189
332     &pchar_dummy,   //190
333     &pchar_dummy,   //191
334     &pchar_dummy,   //192
335     &pchar_dummy,   //193
336     &pchar_dummy,   //194
337     &pchar_dummy,   //195
338     &pchar_dummy,   //196
339     &pchar_dummy,   //197
340     &pchar_dummy,   //198
341     &pchar_dummy,   //199
342     &pchar_dummy,   //200
343     &pchar_dummy,   //201
344     &pchar_dummy,   //202
345     &pchar_dummy,   //203
346     &pchar_dummy,   //204
347     &pchar_dummy,   //205
348     &pchar_dummy,   //206
349     &pchar_dummy,   //207
350     &pchar_dummy,   //208
351     &pchar_dummy,   //209
352     &pchar_dummy,   //210
353     &pchar_dummy,   //211
354     &pchar_dummy,   //212
355     &pchar_dummy,   //213
356     &pchar_dummy,   //214
357     &pchar_dummy,   //215
358     &pchar_dummy,   //216
359     &pchar_dummy,   //217
360     &pchar_dummy,   //218
361     &pchar_dummy,   //219
362     &pchar_dummy,   //220
363     &pchar_dummy,   //221
364     &pchar_dummy,   //222
365     &pchar_dummy,   //223
366     &pchar_dummy,   //224
367     &pchar_dummy,   //225
368     &pchar_dummy,   //226
369     &pchar_dummy,   //227
370     &pchar_dummy,   //228
371     &pchar_dummy,   //229
```

```
372     &pchar_dummy,   //230
373     &pchar_dummy,   //231
374     &pchar_dummy,   //232
375     &pchar_dummy,   //233
376     &pchar_dummy,   //234
377     &pchar_dummy,   //235
378     &pchar_dummy,   //236
379     &pchar_dummy,   //237
380     &pchar_dummy,   //238
381     &pchar_dummy,   //239
382     &pchar_dummy,   //240
383     &pchar_dummy,   //241
384     &pchar_dummy,   //242
385     &pchar_dummy,   //243
386     &pchar_dummy,   //244
387     &pchar_dummy,   //245
388     &pchar_dummy,   //246
389     &pchar_dummy,   //247
390     &pchar_dummy,   //248
391     &pchar_dummy,   //249
392     &pchar_dummy,   //250
393     &pchar_dummy,   //251
394     &pchar_dummy,   //252
395     &pchar_dummy,   //253
396     &pchar_dummy,   //254
397     &pchar_dummy    //255
398
399 };
400
401 void writeText(const char* str, int strLen)
402 {
403   char* ptr = str;
404   struct tPoligonKarakter* pchar;
405
406   textRenk = preMultipliedARGB(100, KIRMIZI);
407
408   while(strLen--)
409   {
410     pchar = ascii_pchars[(int)(*ptr)];
411
412     char_drawer(pchar, charPosition(cursor++), textRenk, textOption);
413
414     print("-- bir karakter yazdim\r\n");
415
416     ptr++;
417   }
418 }
419
```