FPGA ÜZERİNDE GÖRÜNTÜ İŞLEME ALGORİTMALARININ GERÇEK ZAMANLI OLARAK GERÇEKLEŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

Muhammed Furkan TAŞDEMİR
Danışman
Doç. Dr. İsmail KOYUNCU
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

Temmuz 2020

Bu tez çalışması 19.FEN.BİL.15 numaralı proje ile Afyon Kocatepe Üniversitesi Bilimsel Araştırma Projeleri Koordinasyon Birimi tarafından desteklenmiştir.

AFYON KOCATEPE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZİ

FPGA ÜZERİNDE GÖRÜNTÜ İŞLEME ALGORİTMALARININ GERÇEK ZAMANLI OLARAK GERÇEKLEŞTİRİLMESİ

Muhammed Furkan TAŞDEMİR

Danışman
Doç. Dr. İsmail KOYUNCU

ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ
ANABİLİM DALI

Temmuz 2020

BİLİMSEL ETİK BİLDİRİM SAYFASI

Afyon Kocatepe Üniversitesi

Fen Bilimleri Enstitüsü, tez yazım kurallarına uygun olarak hazırladığım bu tez çalışmasında;

- Tez içindeki bütün bilgi ve belgeleri akademik kurallar çerçevesinde elde ettiğimi,
- Görsel, işitsel ve yazılı tüm bilgi ve sonuçları bilimsel ahlak kurallarına uygun olarak sunduğumu,
- Başkalarının eserlerinden yararlanılması durumunda ilgili eserlere bilimsel normlara uygun olarak atıfta bulunduğumu,
- Atıfta bulunduğum eserlerin tümünü kaynak olarak gösterdiğimi,
- Kullanılan verilerde herhangi bir tahrifat yapmadığımı,
- Ve bu tezin herhangi bir bölümünü bu üniversite veya başka bir üniversitede başka bir tez çalışması olarak sunmadığımı

beyan ederim.

17/07/2020

Muhammed Furkan TAŞDEMİR

ÖZET

Yüksek Lisans Tezi

FPGA ÜZERİNDE GÖRÜNTÜ İŞLEME ALGORİTMALARININ GERÇEK ZAMANLI OLARAK GERÇEKLEŞTİRİLMESİ

Muhammed Furkan TAŞDEMİR
Afyon Kocatepe Üniversitesi
Fen Bilimleri Enstitüsü
Elektrik-Elektronik Mühendisliği Anabilim Dalı
Danışman: Doç. Dr. İsmail KOYUNCU

Son yıllarda yoğun çalışma yapılan alanlardan birisi olan görüntü işleme, birçok mühendislik alanında başarıyla kullanılmaktadır. Görüntü işleme, girişleri ve çıkışları görüntüler olan işlemleri ve ayrıca bu görüntülerden öznitelikleri çıkartan süreçleri kapsamaktadır. Bu süreçlere, bölütleme, morfolojik işlemler, kenar bulma, köşe bulma ve filtreleme gibi işlemler örnek olarak verilebilir. Bu işlemler için kullanılan Resim veya görüntüler piksellerden oluşmaktadır. Bir VGA (Video Graphics Array (Video Grafik Dizisi)) aracılığı ile aktarılan 640x480 piksel boyutundaki bir görüntü üzerinde işlem yapılabilmesi için 307,200 adet verinin hesaplanması gerekmektedir. Görüntü kalitesi veya boyutu arttığında işlem hacmi de büyümektedir. Bu işlemlerin gerçek zamanlı yapılabilmesi için yüksek hızlı ve paralel işlem yapabilen sayısal platformlara ihtiyaç duyulmaktadır. Sunulan bu çalışmada, köşe algılama algoritmalarından FAST (Features from Accelerated Segment Test (Hızlandırılmış Segment Testinden Özellikler)) ve Harris köşe algılama algoritmaları, kenar bulma algoritmalarından Sobel kenar bulma algoritması, morfolojik işlem algoritmalarından yayma ile aşındırma yöntemleri ve renk değiştirme algoritması FPGA (Field Programmable Gate Array (Alanda Programlanabilir Kapı Dizileri)) çipleri üzerinde çalışmak üzere gerçek zamanlı olarak tasarlanmıştır. Tasarlanan gerçek zamanlı görüntü işleme algoritmalarında Xilinx Vivado Design Suite HLx kullanılmıştır. Gerçek zamanlı görüntüler HDMI (High Defination Multimedia Interface (Yüksek Çözünürlüklü Çoklu-Ortam Arayüzü)) aracığıyla kameradan alınmıştır. Alınan gerçek zamanlı görüntü

verilerinin FPGA üzerinde işlenebilmesi için VHDL (Very High Speed Integrated Circuit Hardware Description Language (Çok Yüksek Hızlı Tümleşik Devre Donanımı Tanımlama Dili)) kullanılmıştır. Yapılan tüm tasarımlar Xilinx Zybo Z7-20 kartı üzerinde gerçeklenmiştir. FPGA-tabanlı tasarımların sonuçlarından elde edilen görüntü verileri HDMI aracılığıyla monitöre aktarılmıştır. Xilinx Zybo Z7-20 FPGA kartı üzerinde çalışmak üzere tasarımı yapılan gerçek zamanlı görüntü işleme algoritmalarından elde edilen sonuçlar sunulmuştur.

2020, xiii + 108 sayfa

Anahtar Kelimeler: Gerçek Zamanlı Görüntü İşleme, FPGA, Vivado HLS, VHDL.

ABSTRACT

M.Sc. Thesis

REAL TIME REALIZATION OF BASIC IMAGE PROCESSING ALGORITHMS ON FPGA

Muhammed Furkan TAŞDEMİR

Afyon Kocatepe University

Graduate School of Natural and Applied Sciences

Department of Electrical-Electronics Engineering

Supervisor: Assoc. Prof. İsmail KOYUNCU

Image processing, which has been one of the areas of intensive work in recent years, has been successfully used in many engineering fields. Image processing includes processes whose inputs and outputs are images, as well as processes that extract attributes from these images. Examples of these processes are segmentation, morphological operations, edge finding, corner finding and filtering. The pictures or images used for these operations consist of pixels. 307,200 pieces of data need to be calculated in order to be able to process an image of 640x480 pixels, which is transferred via a VGA (Video Graphics Array). As image quality or size increases, the transaction volume also increases. In order to carry out these transactions in real time, digital platforms that can perform high speed and parallel transactions are required. In this study presented, FAST (Features from Accelerated Segment Test) and Harris corner detection algorithms, Sobel algorithm from edge detection algorithms, spreading and erosion methods from morphological processing algorithms, and color changing algorithm FPGA (Field Programmable It is designed to run on Gate Array (Field Programmable Gate Arrays) chips in real time. Xilinx Vivado Design Suite HLx was used in the designed real time image processing algorithms. Real-time images were taken from the camera via HDMI (High Definition Multimedia Interface). VHDL (Very High Speed Integrated Circuit Hardware Description Language) was used to process the real-time image data received on FPGA. All the designs were implemented on the Xilinx Zybo Z7-20 card. Image data obtained from the results of FPGA-based designs were transferred to the monitor via

HDMI. The results obtained from real-time image processing algorithms designed to work on Xilinx Zybo Z7-20 FPGA board are presented.

2020, xiii + 108 pages

Keywords: Real Time Image Processing, FPGA, Vivado HLS, VHDL.

TEŞEKKÜR

Mühendis olma hikâyem ilkokul ikinci sınıfta bir tuş darbesiyle koskoca arabanın üstünün nasıl açıldığını merak etmemle başladı. Fikrimse, bu soruya cevap ararken "Öğrenmeyi çok istiyorsan mühendis olmalısın!" uyarısıyla şekillendi. O fikrin şekillenmesinden tam 16 yıl sonra hiç eksilmemiş öğrenme hevesim ile yüksek lisanstan mezun oldum. "İnsanları doktorlar yaşatır, insanlığı ise mühendisler yaşatır" prensibiyle çıktığım bu yolda amacım her zaman insanlığa faydalı olmaktır.

Tez çalışması boyunca maddi ve manevi her türlü desteğini esirgemeyen aynı zamanda tez amacının belirlenmesinde ve tez aşamalarında bilgi birikimini ve tecrübelerini benimle paylaşan çok saygı değer danışman hocam Sayın Doç. Dr. İsmail KOYUNCU'ya en içten samimi duygularımla sonsuz teşekkürlerimi sunarım.

Tez çalışmasının yapılmasında 19.FEN.BİL.15 numaralı proje ile maddi destek sağlayan Afyon Kocatepe Üniversitesi Bilimsel Araştırma Projeleri Koordinasyon birimine teşekkür ederim.

Hayatım boyunca göğüs gerdiğim tüm zorluklar da hiç tereddüt etmeden daima yanımda olan, ne zaman dara düşsem varlıkları ile huzur bulduğum maddi ve manevi desteklerini hiçbir zaman esirgemeyen annem ve kardeşime teşekkürü bir borç bilirim.

Zorlu hayat şartlarından yorulduğum ve tökezlediğim zaman kolumdan tutup kaldıran, maddi ve manevi yanımda olan teyzem Filiz Aydın ve manevi teyzem Gülten Coşkun'a teşekkür ederim.

Muhammed Furkan TAŞDEMİR
AFYONKARAHİSAR, 2020

İÇİNDEKİLER DİZİNİ

sayfa

SİMGELER ve KISALTMALAR DİZİNİ

Simgeler

A	Görüntü işleme algoritması denklem durum değişkenleri
B	Görüntü işleme algoritması denklem durum değişkenleri
C	Görüntü işleme algoritması denklem durum değişkenleri
u	Görüntü işleme algoritması denklem durum değişkenleri
v	Görüntü işleme algoritması denklem durum değişkenleri
w	Görüntü işleme algoritması denklem durum değişkenleri
x	Görüntü işleme algoritması denklem durum değişkenleri
G	Görüntü işleme algoritması denklem durum değişkenleri
σ	Görüntü işleme algoritması denklem durum değişkenleri
y	Görüntü işleme algoritmasında denklem penceresi
I	Piksel yoğunlukları
M	Matris denklemi
z	Tam sayılar kümesi

Kısaltmalar

ASIC	Uygulamaya Özel Tümleşik Devre (Application Specific Integrated Circuit)
FAST	Features from Accelerated Segment Test (Hızlandırılmış Segment Testinden Özellikler)
HDL	Hardware Description Language (Donanım Tanımlama Dili)
HD	High Definition (Yüksek Çözünürlük)
HLS	High Level Synthesis (Yüksek Seviye Sentez)
HPD	Hot Plug Detect (Çalışırken Takmayı Algılama)
IDE	Entegre Geliştirme Ortamı (Integrated Development Environment (IDE))
IP	Intellectual Property (Fikri Mülkiyet)
LUT	Look Up Table (Bakma Tablosu)
MIO	Multiplexed Input/Output (Çoğullamalı Giriş/Çıkış) Programable System (Programlanabilir Sistem)
MMU	Memory Management Unit (Bellek Yönetim Birimi)
MPE	Media Processing Engine (Medya İşleme Motoru)
OCM	Other Chip Memory (Başka Çip Belleği)
PS	Programable System (Programlanabilir Sistem)
PROM	Programmable Read Only Memory (Programlanabilir Salt Okunur Hafıza)
PL	Programable Logic (Programlanabilir Mantık)
SCU	Snoop Control Unit (Snoop Kontrol Ünitesi)
SDK	Software Development Kit (Yazılım Geliştirme Kiti)
SSD	Sum of Squared Differences (Kare Farkların Toplamı)
SRAM	Static Random Access Memory (Sabit Rasgele Erişimli Bellek)
TCL	Team Command Language (Takım Komut Dili)
USB	Universal Serial Bus (Evrensel Seri Veriyolu)
VGA	Graphics Array (Video Grafikleri Dizilimi)

ŞEKİLLER DİZİNİ

sayfa

ÇİZELGELER DİZİNİ

sayfa

RESİMLER DİZİNİ

sayfa

1. GİRİŞ

Son yıllarda görüntü işleme alanında yapılan çalışmalar her geçen gün artmakta ve hemen hemen mühendisliğin bütün alanlarında görüntü işleme tabanlı uygulamalar kullanılmaktadır. Görüntü işleme yöntemleri genel olarak bölütleme (Tombul vd. 2018), eşikleme (Demirci vd. 2019), morfolojik işlemler (Yıldız vd. 2018), kenar bulma (Yiğitbaşı 2014), köşe bulma algoritmaları (Methore vd. 1990) ve filtreleme (Altuntaş vd. 2011) gibi temel işlemlerden oluşmaktadır. Bu işlemler için kullanılan resim veya görüntüler piksellerden oluşmaktadır. Bir Video Grafik Dizisi (Video Graphics Array (VGA)) aracılığıyla aktarılan 640x480 formatındaki görüntünün işlenmesi için 307,200 adet matris verisinin işlenmesi gerekmektedir. Bu verinin gerçek zamanlı bir görüntü olduğu düşünüldüğünde yukarıda ifade edilen işlemler için oldukça fazla Merkezi İşlem Birimi (Central Processing Unit (CPU)) zamanına ihtiyaç duyulmaktadır.

Görüntü işlemenin genel olarak 5 temel kullanım amacı bulunmaktadır. Bunlardan birincisi; yeni oluşturulan ya da üzerinde değişiklik yapılan bir görseli gösterme işlemi olan görselleştirmedir. İkincisi; görüntü kalitesi düşük ya da tahribata uğramış olan görüntüler üzerinde düzeltme işlemleri anlamına gelen görüntü keskinleştirme ve yenilemedir. Üçüncüsü; bir görüntü üzerinden ihtiyaç duyulan kısmın alınması işlemi olan görüntü alımıdır. Dördüncüsü; daha çok makine öğrenmesi yöntemi ile kullanılan desen tanıma işlemidir. Son işlem ise görüntü tanıma olarak adlandırılmaktadır.

Görüntü işleme uygulamaları neredeyse tüm alanlarda uygulanan ve fayda sağlayan bir alandır. Bu uygulamaların kullanım alanları gittikçe artmaktadır. Görüntü işlemenin tıp (Osmanoğlu vd. 2016), algılama sistemleri (Eldem vd. 2017), savunma sanayi (Samtaş vd. 2011), kalite kontrol (Özkan 2012) ve bilgisayarla görme (Koray vd. 2019) gibi kullanım alanları bulunmaktadır.

FPGA (Field Programmable Gate Array (Alan Programlanabilir Kapı Dizileri)) çipleri paralel sinyal işleme, tekrar tekrar programlanabilme, düşük güç tüketimi, hızlı ilk prototip gibi özellikleri sayesinde literatürde oldukça yoğun bir şekilde kullanılan sayısal bir tümleşik devre platformudur. FPGA çipleri üzerinde VHDL, Verilog ve

System C gibi diller kullanılarak tasarımlar yapılabilmektedir.

- **Daha ucuz:** FPGA çipleri paralel bilgisayarlara, süper bilgisayarlara hatta özel tasarlanmış grafik kartlarına göre daha düşük maliyete sahiptirler.

- **Daha hızlı:** FPGA çipleri normal bilgisayarlarla karşılaştırıldığında daha yüksek performans göstermektedirler.

- **Daha esnek:** FPGA çipleri çok kısa sürelerde defalarca yeniden programlanabilmektedirler (Koyuncu 2009).

Sayısal görüntü işleme, resimsel bilgiler kullanılarak amaca ve isteğe yönelik faydalı eklemeler ve çıkarmalar işlemlerinin gerçekleştirilmesidir.

FPGA yeniden programlanabilir hafıza anlamına gelmekte ve yapısal olarak PROM aygıtıdır. FPGA çipleri, paralel işlem kabiliyeti ve yeniden programlanabilmesinden dolayı çokça tercih edilmektedir.

FPGA çipleri üzerinde VHDL, Verilog ve System C gibi diller kullanılarak tasarımlar yapılabilmektedir. FPGA çipleri üretim (Sarma vd. 2009), robotik (Sánchez-Solano 2007), uzay ve havacılık (Dönmez 2019), şifreleme (Garipcan 2017) ve kod çözme (Özbay 2017) gibi alanlarda kullanılmaktadır.

2. LİTERATÜR BİLGİLERİ

Günümüzde sayısal görüntü işleme uygulamaları ile ilgili pek çok alanda çalışmalar yapılmaktadır. Bu çalışma alanlarına tıp bilimleri (Prakash vd. 2014, Hao 2014), ölçme ve enstrümantasyon uygulamaları (Rana vd. 2014, Li vd. 2014), şifreleme bilimleri (Barakat vd. 2014), yer bilimleri (Lu vd. 2014), kontrol (Xiao vd. 2015), yiyecek endüstrisi, kodlama ve kod çözme, yapay sinir ağları, haberleşme örnek olarak verilebilmektedir. Görüntü işleme; normal bir görüntünün, işlenmiş veya değiştirilmiş bir sayısal görüntüye dönüşmesi için kat edilen yol ve yöntemlerdir. Bir görüntüyü daha anlaşılır hale getirmeyi amaçlayan yöntemler; görüntüyü anlamlı alt bölgelere ayırma, kullanılan özellik doğrultusunda alt bölgelerin tanımlanması, tanımlanmış nesnelerin etiketlenmesi ve son olarak da etiket atanan nesnelere bakılarak karar verme ile yorumlama işlemlerinden oluşmaktadır. Görüntüyü anlamlı alt bölgelere ayırma safhasında kenar ve köşe belirleme gibi teknikler kullanılmaktadır. Görüntüde nesnelerin içindeki yansıma ve aydınlatma değişimleri kenarları oluşturmaktadır. Kenarların belirlenmesinde piksel komşulukları önemli bir yere sahiptir. Kenar belirleme için uygulanan yöntemleri, eğim kenar detektörleri, sıfır geçiş (ikincil türev detektörü), gaussion kenar detektörleri (canny) ve vektör sıralama istatistiği ile gerçekleştirilmiş kenar detektörleri olmak üzere dört başlık altında toplamak mümkündür. Eğim kenar detektörlerinden olan Sobel operatöründe, görüntünün yatay ve dikey türevleri çekirdeklerle hesaplanmaktadır. Elde edilen bu sonuçlar ile görüntü içerisindeki her bir noktanın eğim değerleri toplanmaktadır (Yuan-Hui vd. 2006). İkincil türev detektörlerinde, birinci türevdeki tepe noktalar belirlendikten sonra ikinci türev içinde sıfır çaprazlama işleminin yapılmasıyla resim ile ilgili kenar bilgisi elde edilebilmektedir (Gonzales vd. 2002). Canny uygulaması, Gauss ile ayrılmış adım kenarlarının kullanıldığı bir yöntemdir. Bu uygulama kenar belirleme ve gürültü azaltma için kullanılmış bir kenar belirleme detektörüdür (Canny vd. 1986). Sıralama istatistiği, veriler içerisinde en yüksek değeri bulma da ve filtreleme konularında önemli bir rol oynamaktadır. Bu yöntemlere kenar dizeleme (M-ordering), azaltılmış ve toplam dizeleme (R-ordering), kısmi dizeleme (P-ordering) ve şartlı dizeleme (C-ordering) örnek olarak verilebilmektedir (Chung vd. 2006).

Alan Programlanabilir Kapı Dizileri (Field Programmable Gate Array (FPGA)) tasarımcının sistem üzerinde yeniden programlayabileceği tümdevrelerdir. Bu tümdevreler çip üzerinde sistem (System-on-Chip) olarak da adlandırılmaktadırlar. Bu çipler de yapılandırılabilir lojik bloklar, giriş/çıkış blokları ve ara bağlantılar olmak üzere üç ana yapı bulunmaktadır. FPGA çipleri prototip aşamasında Uygulamaya Özel Tümleşik Devreler (Application Specific Integrated Circuit (ASIC)) ile karşılaştırıldığında daha düşük maliyetli ve daha hızlı tasarım gibi çözümler sunmaktadır. Ayrıca paralel işlem yapabilme özelliklerinden dolayı Sayısal Sinyal İşlemci (Digital Signal Processor (DSP)) ile karşılaştırıldığında daha yüksek işlem kapasitesi ve çalışma frekansı gibi üstünlüklerinden dolayı ilgi çekmektedir. Sonuç olarak FPGA çipleri hızlı ve düşük maliyet sunmaları, yeniden programlanabilme ve yüksek frekansta çalışabilme özelliklerinden dolayı yüksek işlem gücü ve performans gerektiren görüntü işleme uygulamalarında tercih edilmektedir (Koyuncu vd. 2014).

Literatürde FPGA-tabanlı görüntü işleme algoritmaları ile çeşitli çalışmalar yapılmıştır. Kumar ve arkadaşlarının yaptıkları çalışmada, 2 boyutlu Sobel operatörü uygulaması FPGA-tabanlı olarak gerçeklenmiştir (Kumar vd. 2013).

Mehra ve arkadaşlarının yaptığı diğer bir çalışmada, 2 boyutlu FPGA-tabanlı Sobel operatörü uygulaması sunulmuş ve sistemin maksimum çalışma frekansı 148 MHz olarak elde edilmiştir (Mehra vd. 2009).

Diğer bir çalışmada, Anusha ve arkadaşları 3 boyutlu Sobel operatörünü FPGA üzerinde gerçekleştirmişlerdir (Anusha vd. 2012).

Karaköse ve arkadaşlarının yaptığı bir çalışmada, FPGA-tabanlı morfolojik işlem algoritması geliştirilerek nesnelerin tespiti kolaylarştırılmıştır (Karaköse vd. 2016).

Aydoğdu ve arkadaşları bu çalışmalarında FPGA üzerine bağladıkları iki kamera yardımıyla köşe tespiti yapmışlardır (Aydoğdu vd. 2013).

Gacar tarafından yapılan bir diğer çalışmada, FPGA tabanlı görüntü işleme ara birimi

üzerinde güvenlik kamerasından görüntülenen görüntüyü bilgisayar monitörü üstünde görüntülenmiştir (Gacar 2009).

Erdoğan tarafından yapılan farklı bir çalışmada, görüntü işleme üzerinde kullanmak üzere geliştirme kartı tasarlanmıştır. Ayrıca bu kart üzerinde görüntü işleme algoritmaları denenmiştir (Erdoğan 2013).

Özçelik tarafından sunulan farklı bir çalışmada, FPGA kartı üzerinde morfolojik işlemler ve konvolüsyon işlemleri ile ten rengi ve kan hücreleri tanıma üzerine çalışılmıştır (Özçelik 2012).

Çil tarafından literatüre sunulan bir diğer çalışmada, FPGA platformu üzerinde konvolüsyon işlemi gerçekleştirilmiştir. Ayrıca elde edilen sonuçlar Matlab sonuçları ile karşılaştırılmıştır (Çil 2015).

Kızılkaya tarafından yapılan bir çalışmada, görüntülerin görüntü işleme yöntemlerinden histogram ve konvolüsyon işlemleri gerçekleştirilmiş ve çalışma sonuçları sunulmuştur (Kızılkaya 2012).

Çelik tarafından yapılan çalışmada, FPGA üzerinde difüzyon ile sınırlı tanecik kümeleşme modeli kullanılarak fraktal bir şeklin görüntüsü elde edilmişitir. Elde edilen iki boyutlu görüntüler üç boyutlu tek bir görüntü haline getirilmiştir (Çelik 2013).

Özalp tarafından yapılan çalışmada, görüntülere konvolüsyon işlemleri uygulanılmış ve elde edilen sonuçlar ekran da gösterilmiştir (Özalp 2018).

Altuncu tarafından yapılan çalışmada ise Zedboard FPGA kartı üzerinde kenar bulma ve filtreleme gibi görüntü işleme uygulamaları gerçekleştirilmiştir (Altuncu 2015).

Sunulan bu çalışmada, literatürde sunulan çalışmalardan farklı olarak öncelikli olarak gerçek zamanlı görüntüler kameradan HDMI (High Definition Multimedia Interface (Yüksek Çözünürlüklü Çoklu-Ortam Arayüzü)) aracılığı ile alınmıştır. İkinci aşamada,

köşe algılama algoritmalarından FAST (Features from Accelerated Segment Test (Hızlandırılmış Segment Testinden Özellikler)) ve Harris köşe algılama algoritmaları, kenar bulma algoritmalarından Sobel kenar bulma algoritması, morfolojik işlem algoritmalarından yayma ile aşındırma yöntemleri ve renk değiştirme algoritması FPGA çipleri üzerinde çalışmak üzere gerçek zamanlı olarak tasarlanmıştır. Kameradan alınan gerçek zamanlı görüntü verilerinin FPGA üzerinde işlenebilmesi için kullanılan FAST ve Harris köşe algılama algoritmaları, Sobel kenar bulma algoritması, yayma ile aşındırma yöntemleri ve renk değiştirme algoritmaları VHDL dilinde kodlanmıştır. FPGA-tabanlı tasarımlar Xilinx Vivado Design Suite programı kullanılarak sentezlenmiş ve test edilmiştir. Üçüncü aşamada, tasarlanan görüntü işleme algoritmaları Zybo Z7-20 geliştirme kartına yüklenmiştir. Dördüncü aşamada, Zybo Z7-20 geliştirme kartından alınan gerçek zamanlı görüntü verileri HDMI aracılığıyla monitöre aktarılmıştır. Son aşamada ise tasarımlardan elde edilen FPGA çip istatistikleri ve performans sonuçları sunulmuştur.

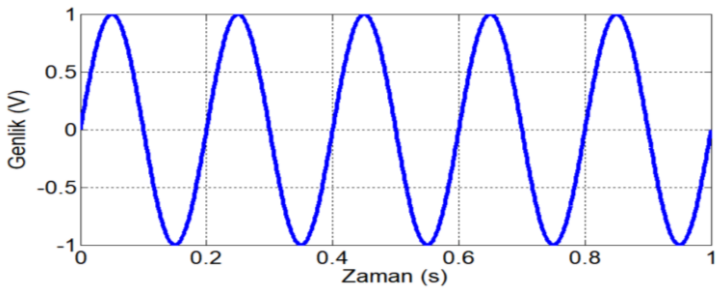
3. MATERYAL ve METOT

Bu başlık altında, yapılan çalışmada içerisinde kullanılan materyaller ve materyalleri kullanırken uyulan ve uygulanan metotlar anlatılmıştır. Anlatılan bu konular üç başlık altında incelenmiştir.

3.1 Görüntü İşleme

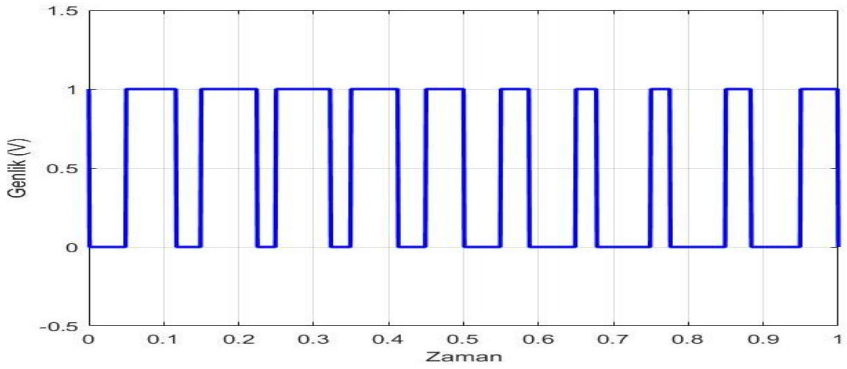
Görüntü işleme kelimesi Almanca kökenli olan Bildbearbeitung kelimesinden türemiş olup resim düzenleme anlamına gelmektedir. Görüntü işleme; sentezlenmiş, ölçülüp kaydedilmiş görüntü verilerinin üzerinde, yine elektronik olan bilgisayar ortamında yazılım programları sayesinde istenilene uygun biçimde değişiklikler yapılmasıdır (İnt. Kyn. 1). Görüntü işleme uygulamaları literatürde iki ana başlık altında incelenmektedir.

- Analog Görüntü: Analog görüntü fonksiyonlarını (f(x,y)) belirli bir maksimum (maks) ve minimum (min) aralığında oluşturan x ve y değerleri, sürekli olarak değişen ve güncellenen gerçek değerlerden oluşmaktadır. Analog bir görüntü veya resmin, bütününden itibaren birimine doğru bakıldığında resmi oluşturan renkler görülebilmektedir. Sayısal görüntüde bütünden birime doğru yaklaştıkça matematiksel ifadeler bulunmaktadır. Bu nedenden dolayı analog bir görüntüyü bilgisayarlar için sinyal haline dönüştürmek gerekmektedir. Analog sinyal tipi sürekli bir sinyaldir ve sinüs dalgaları ile gösterilmektedir. Örneğin bir VGA aracılığıyla kameradan alınan görüntünün bir mikrodenetleyiciye aktarılabilmesi için analog sinyal sayısal sinyale dönüştürülmelidir. Bu sinyal tipinin örneği Şekil 3.1'de gösterilmiştir.



Şekil 3.1 Analog sinyal örneği.

- Sayısal Görüntü: Görüntünün matematiksel formata dönüştürülmüş haline sayısal görüntü denilmektedir. Ayrık zamanlı sinyallerdir ve kare dalga ile gösterilmektedir. Kare dalganın üst noktası olan maksimum noktası 1 değerini alırken alt kısmı olan mininmum noktası 0 değerini almaktadır. Bu görüntü yöntemi sayısal ifadelerden oluştuğu için bilgisayarlar ve yazılımlar tarafından kolaylıkla algılanabilmektedir (İnt. Kyn. 2). Bu sinyal tipinin örneği Şekil 3.2'de verilmiştir.



Şekil 3.2 Sayısal sinyal örneği.

Görüntü İşlemenin genel olarak beş temel amacı aşağıda verilmiştir;

- Görselleştirme – Görünmesi zor nesneleri gözlemleme.
- Görüntü keskinleştirme ve restorasyon – Gürültülü görüntüleri iyileştirme.
- Görüntü alımı – İlgi çekici ve yüksek çözünürlüklü görüntü arama.
- Desen Tanıma – Bir görüntüdeki çeşitli nesneleri tanımlama.
- Görüntü Tanıma – Bir görüntüdeki nesneleri ayırt etme (İnt. Kyn. 3).

Görüntü işleme uygulamaları hemen hemen her alanda uygulanmakta ve bu uygulamaların kullanım alanları gittikçe artmaktadır. Görüntü işleme kullanım alanlarından bazıları şu şekildedir:

- Kriminoloji / Adli Tıp
- Tıbbi Görüntüleme
- Uzaktan Algılayıcı Sistemler
- Askeri Sanayi
- Nakliyat

- Kalite Kontrol
- Sayısal Kamera Görüntüleri
- Morfoloji
- Bilgisayarla Görme

Gri Seviye Dönüşümleri: Gri seviye görüntü dönüşümü herhangi bir görüntüdeki piksel değerlerinin gri seviye değerlere dönüştürülmesidir. Gri seviye görüntü dönüşümü yapabilmek için ihtiyaç duyulan bilgilere göre 3 ayrı sınıfa ayrılabilmektedir.

1-Transformlar (Dönüşümler): Uzaysal dönüşüm ve frekans dönüşümü gibi alanlara dönüşüm yapılarak görüntünün işlenmesi durumudur. Alanlar ile çalışılmasından dolayı bilgi ve birikim gerektirmektedir fakat çok etkili ve verimli algoritmalar bu yöntemler sayesinde çalıştırıldığı için çokça tercih edilmektedir. Görüntüyü negatife çevirme işlemi dönüşümlere örnek verilebilmektedir.

2-Komşuluk İlişkili (Bölgesel) İşlemler: Bu işlem türünde komşu piksellerin durumunu analiz ederek sonuç elde edilmektedir. Bölgesel işlemler bir miktar zor sayılsa da çokça kullanılmaktadır. Bu kullanımlara görüntüyü gri seviye dönüştürme örnek verilebilmektedir. Görüntünün belirli bir pikselinde gri seviyeye dönüştürmek istenildiğinde piksel ve pikselin komşusu olan piksellerin değerleri bilinirse işlemler gerçekleştirilebilmektedir

3-Noktasal İşlemler: Bir pikselin yeni oluşturlan gri seviye piksel değerini, komşusu olan piksel bilgilerine ihtiyaç olmadan elde etme işlemidir. Noktasal işlemler en basit yöntem olmasından dolayı görüntü işleme alanlarında en çok kullanılan yöntemlerden bir tanesidir. Genel olarak elde edilen görüntünün ana işleme girmesine hazırlık yapmak için uygulanan yöntemlerden bir tanesidir (İnt. Kyn. 4).

Çalışmamızda görüntü işleme alanında çokça tercih edilen iki tane köşe algılama algoritması, iki tane morfolojik işlem, bir tane kenar algılama algoritması ve bir tane de renkli görüntü dönüşüm çalışmalarına yer verilmiştir.

3.1.1 FAST Köşe Algılama Algoritması

Köşe algılama algoritmaları, görüntüde istenilen veriler türlerini algılama istenmeyen veri türlerini ayıklamayı amaçlayan bir yaklaşımdır. Köşe, paralel olmayan iki kenarın kesişimidir. Kenar ise görüntü üzerinde ani piksel yoğunluğunun değişiminden anlaşılmaktadır. Köşeler görüntüde küçük bir alan kaplıyor olsa da görüntü hakkında yorum yapılabilmesini sağlamaktadır. Köşe algılama algoritmaları, hareket algılama, cisim algılama, görüntü kaydetme, video izleme, görüntü bulanıklaştırma, stereo görüntü ve nesne tanımada sıklıkla kullanılmaktadır.

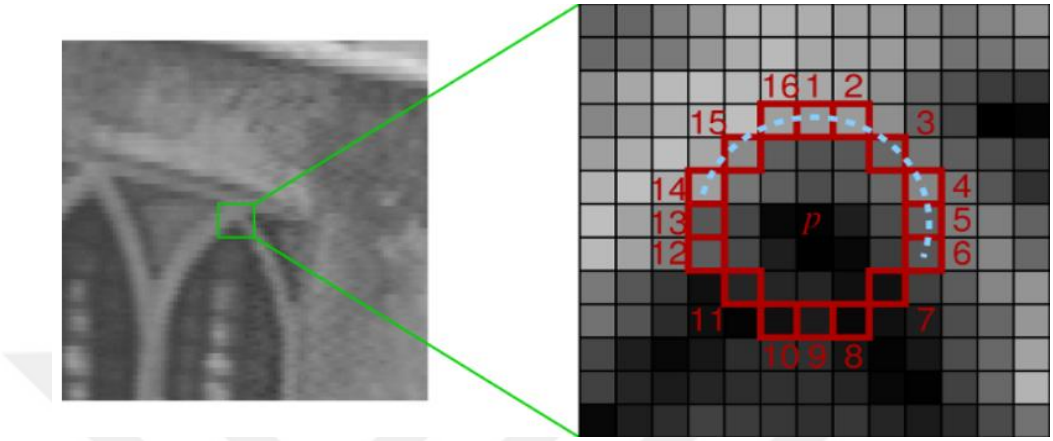
FAST (Features from Accelerated Segment Test (Hızlandırılmış Segment Testinden Özellikler)) bir köşe bulma algoritmasıdır. Bu algoritma Miroslav Trajkovic ve Mark Hadley tarafından 1998 yılında kullanıma sunulmuştur. Bu algoritma işlerken ilk önce köşe adayı bir piksel seçilmektedir. Bu pikseli merkez kabul ederek, etrafında yarıçapı 8 adet piksel olacak şekilde hayali bir çember düşünülmektedir. Bu çemberin çevre uzunluğu üzerinde ki piksellerin yoğunluğu ile köşe adayı pikselin yoğunluklarına bakılmakta ve karşılaştırılmaktadır. Bu karşılaştırmanın sonucunda karşılaştırılan pikseller arasında yoğunluk farkının fazla olduğu görülürse aday noktanın köşe olduğu sonucu çıkarılabilmektedir. FAST köşe bulma algoritmasının yöntemi Şekil 3.1'de verilmiştir. Bu işlemi hızlandımak için çember uzunluğunun üzerindeki tüm pikselleri test etmek yerine bu uzunluk üzerinde ki 1, 5, 9 ve 13 numaralı pikseller ile aday piksel karşılaştırılarak işlem daha da hızlandırılabilmektedir.

Literatürde çoğunlukla Förstner, Harris, LoG, DoG, Wang, SUSAN ve FAST köşe bulma algoritmaları kullanılmaktadır. FAST köşe bulma algoritması literatürde kullanılan diğer algoritmalardan daha hızlı ve doğru köşe tespiti yaptığı için çokça tercih edilmektedir.

FAST köşe algılama algoritmasında üç ana temel nicelik üzerinde durulmaktadır.

- Tutarlılık: Tespit edilen pozisyonlar tutarlı olmalıdır. Gürültü değişimine duyarsız ve aynı noktadan birden fazla görüntü elde edildiğinde hareket olmamalıdır.

- Doğruluk: Köşeler, doğru pozisyonlara mümkün olduğu kadar yakın tespit edilmelidir.
- Hız: Köşe detektörü yeterince hızlı olmak zorundadır (Trajkovic vd. 1998).



Resim 3.1 FAST köşe algılama yöntemi çemberi (Rosten 2006).

3.1.2 Harris Köşe Algılama Algoritması

Harris köşe algılama algoritması 1998 yılında Chris Harris ve Mike Stephens tarafından İngiltere'de sunulmuştur. Harris Köşe Algılama Algoritması ile Moravec köşe algılama yöntemindeki eksiklikler giderilmiş ve daha kesin ve hızlı sonuç vermesi sağlanmıştır. Moravec köşe algılama yönteminde hayali bir kare pencere düşünülmekte ve pencereyi çeşitli yönlerde kaydırarak görüntü yoğunluğundaki ortalama değişimler belirlenmektedir. Bu yöntemde dikkat edilmesi gereken üç önemli nokta vardır:

- Sabit yoğunluk: Pencere hareket ettirildiğinde görülen yoğunlukta belirgin bir değişiklik yoksa bu bölgede kenar ya da köşe yok demektir.
- Kenar tespiti: Pencere hareket ettirildiğinde görülen yoğunluk değişiyor ve temel yoğunluk değişimi satırda ya da temel yoğunluk değişimi sütunda görülüyorsa bu bölgede kenar olduğu anlamına gelmektedir.
- Köşe tespiti: Pencere hareket ettirildiğinde görülen yoğunluk satır ve sütunlar da birlikte değişim gösteriyorsa bu bölgede köşe olduğu anlamına gelmektedir.

Yukarıdaki anlatılan Moravec köşe algılama yönteminde ve eksiklerin giderilmesi

matematiksel bir yönteme tabidir. Denklem (3.1)'de ve denklem (3.2)'de Moravec yönteminin denklemi verilmiştir. Sonrasında ki denklemler yenilikleri ifade etmektedir. Bu denklemde I piksel yoğunluğunu belirtmekte, (x, y) piksel koordinatını vermektedir. E ise yoğunluk değişimini temsil etmektedir. Piksel değişimi ise *u* ve *v* ile ifade edilmektedir. *w* ise hayali pencereyi ifade etmekte ve başlangıç koordinat ifadesi {(1,0), (1.1), (0,1), (-1,1)} olmaktadır.

$$E_{x,y} = \sum_{u,v} w_{u,v} [I_{x+u,y+u} - I_{u,v}]^2 \quad (3.1)$$

$$= \sum_{u,v} w_{u,v} [xX + yY + O(x^2, y^2)]^2 \quad (3.2)$$

Denklem (3.3) ve (3.4)'te (x, y) piksellerinin yoğunluklarının yöntemi gösterilmiştir.

$$X = I \times (-1,0,1) \approx \frac{\partial I}{\partial x} \quad (3.3)$$

$$Y = I \times (-1,0,1)^T \approx \partial I / \partial y \quad (3.4)$$

Pencerenin görüntü üzerinde kaydırılması için denklem (3.5) verilmiştir.

$$\mathrm{E(x,y)} = \mathrm{Ax^2 + 2Cxy + By^2} \quad (3.5)$$

Burada A, B, C bilinmeyenleri pencere ve piksel koordinatlarının karesi çarpılarak bulunmaktadır. Bu işlem denklem (3.6), denklem (3.7) ve denklem (3.8)'de gösterilmiştir.

$$A = X^2 \times w \quad (3.6)$$

$$B = Y^2 \times w \quad (3.7)$$

$$C = (XY)^2 \times w \quad (3.8)$$

Denklem (3.9)'da Moravec yöntemindeki kare pencere yerine Gaussian'daki gibi

dairesel bir pencere kullanılmaktadır.

$$w_{u,v} = exp - (u^2 + v^2)/2\sigma^2 \tag{3.9}$$

E değerinin Moravec yöntemindeki gibi en küçük değeri ile karar vermek yerine kaydırma sonucunda oluşan E değeri üzerinden karar vermek gerekmektedir. Bu işlem denklem (3.10)'da verilmiştir.

$$E(x, y) = (x, y)M(x, y)^T \tag{3.10}$$

Yukarıdaki denklemde M ifadesi 2x2 simetrik bir matristir. Bu matris denklem (3.11)'de verilmiştir.

$$M = \begin{bmatrix} A & C \\ C & B \end{bmatrix} \tag{3.11}$$

Elde edilen sonuçlar için Moravec yöntemindeki gibi üç adet yorum yapılmaktadır;

- Sonuçtaki iki eğri de küçükse diğer bir değişle yerel otokorelasyon işlevi düzse, pencerenin olduğu bölgenin görüntüsü yaklaşık olarak sabit yoğunluktadır.

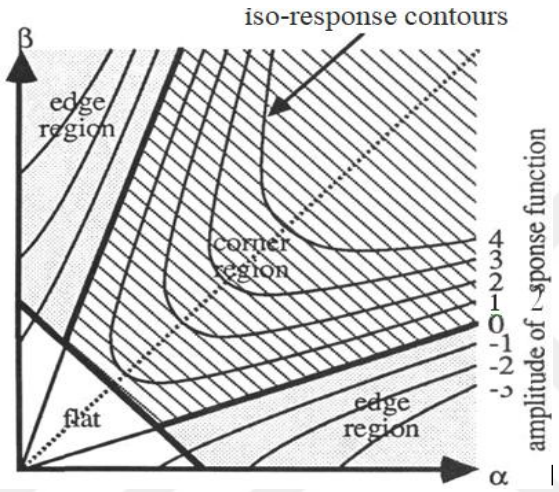
- Eğer bir eğri yüksek ve bir eğri de düşükse diğer bir değişle yerel otokorelasyon fonksiyonu sırt şeklinde ise, o zaman sadece sırt (kenar) boyunca kaymalar E değerinde küçük bir değişikliğe neden olmaktadır. Bu durum bir kenarı ifade etmektedir.

- Her iki eğri de yüksekse, diğer bir değişle yerel otokorelasyon fonksiyonu keskin bir şekilde zirveye ulaşırsa, o zaman herhangi bir yöndeki kayma E değerini de artırmaktadır. Bu durum bir köşeyi ifade etmektedir.

Bu işlemlerin gösterimi Resim 3.2'da verilmiştir.

Temel olarak yorumlandığında şöyle bir sonuç çıkarılabilmektedir. İki kenar tespit edilirse ve o iki kenar birbirine paralel değilse takip edilmelidir bir biri ile kesiştiği

nokta köşe olmaktadır. Harris köşe algılama algoritması, elde edilen görüntülerde bulunan köşelerin tespiti ve özniteliklerini çıkarmak için makine öğrenmesi (Rosten vd. 2006), bulanık mantık (Cuevas vd. 2011) gibi bilgisayarlı görme algoritmalarında kullanılan bir köşe bulma algoritmasıdır. Harris köşe algılama algoritması görüntünün parlaklığı ve rotasyonu gibi değişen durumlar da karalı davranıp aynı sonuçları verebilmektedir. Bu nedenle stereo eşleştirme ve görüntü veritabanı alınında daha sık kullanılmaktadır (Harris vd. 1988).



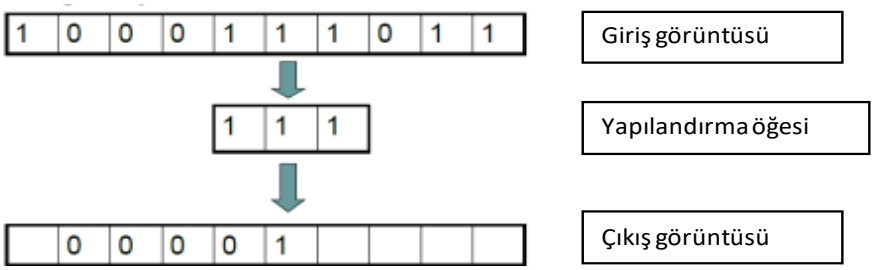
Resim 3.2 Harris köşe bulma algoritma yöntemi.

3.1.3 Aşındırma Yöntemi ile Morfolojik İşlem Algoritması

Aşındırma yöntemi bir morfolojik işlemdir. İkilik sisteme dönüştürülmüş olan imgedeki nesneyi küçültmeye ya da inceltmeye yaramaktadır. Bu işlemin matematiksel izahı denklem (3.13)'de verilmiştir. Aşındırma yöntemi Z^2 uzayında, A ve B kümelerine uygulanmıştır. A işlenecek görseli, B yapılandırma öğesini ve ⊖ ifadesi aşındırma yöntemini temsil etmektedir.

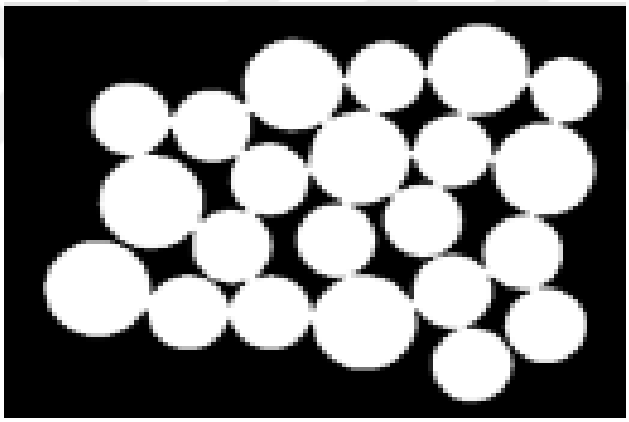
$$A \ominus B = \{z|(B)_z \subseteq A\} \tag{3.12}$$

Yapılandırma öğesinin tamamen uyduğu şekildeki bütün pikseller tespit edilip değiştirilir ve sadece merkez noktadaki değer aynı kalmaktadır. Şekil 3.3'te bu durum gösterilmiştir.

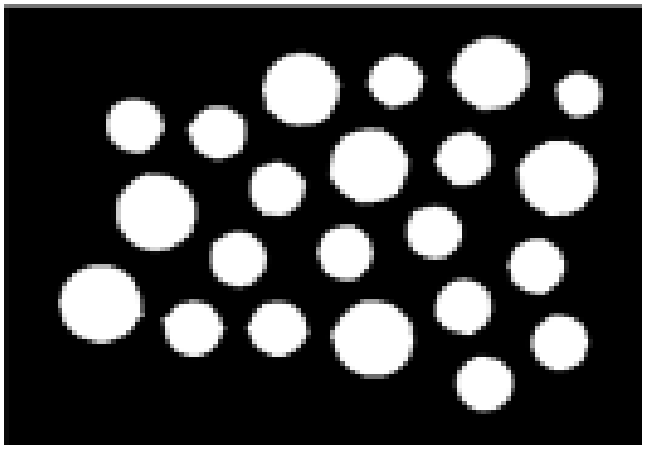


Şekil 3.3 Aşındırma yöntemi akış şeması.

Aşındırma yöntemi tam anlamıyla olmasa da bir bakıma yayma işleminin tersi gibi düşünülebilmektedir. İmge içerisindeki nesneler küçülmekte, delikler genişlemekte ve birbirine bağlı olan nesneler ayrılmaktadır. Resim 3.3'de aşındırma yöntemi uygulanmamış görsel verilmiş olup ve Resim 3.4'de aşındırma işlemi uygulanmış hali verilmiştir.



Resim 3.3 Aşındırma yöntemi uygulanmamış görsel.



Resim 3.4 Aşındırma yöntemi uygulanmış görsel.

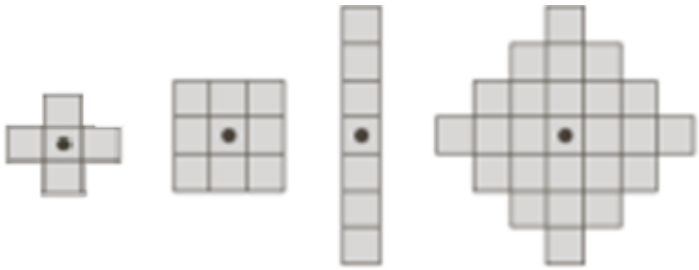
Görüldüğü üzere aşındırma yöntemi uygulanmadan önceki hali ile uygulandıktan sonra ki hali karşılaştırılırsa elde edilen yuvarlaklar aşındırma işleminden sonra küçülmüş ve bundan dolayı birbiriyle olan bağlantıları kesilmiştir.

3.1.4 Yayma Yöntemi ile Morfolojik İşlem Algoritması

Yayma yöntemi ikili sisteme diğer bir değişle binary sisteme dönüştürülmüş olan görüntüde ki nesneyi büyütme veya kalınlaştırma işlemi yapan morfolojik işlemdir. Sayısal bir ifadeyi genişletmek görseli yapısal elemanla kesiştiği bölümler kadar büyütmektir. İşlenecek görselin her bir pikseli, yapısal elemanın merkez noktasına oturtularak genleşme işlemi yapılmaktadır. Bu durumun matematiksel izahı denklem (3.13)'te verilmiştir.

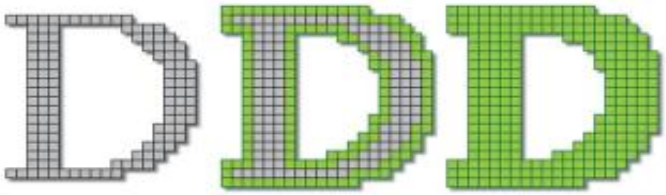
$$A \hexagon B = \{z | (B')_z \cap A \neq \emptyset\} \tag{3.13}$$

Kalınlaştırma işleminin nasıl yapılacağını yapısal eleman belirlemektedir. Genleşme işlemi uygulanmış bir görselde, görsel içerisindeki deliklerin ve boşlukların dolması ve köşe noktasının yumuşaması gözlenmektedir. Bu denklemde A işlenecek olan görseli, B yapı elamanını ve $\hexagon$ ise yayma operatörünü simgelemektedir. Yayma yönteminin aşamaları Resim 3.5'da ve yayma yöntemi uygulanmış hali Resim 3.6'da gösterilmiştir.



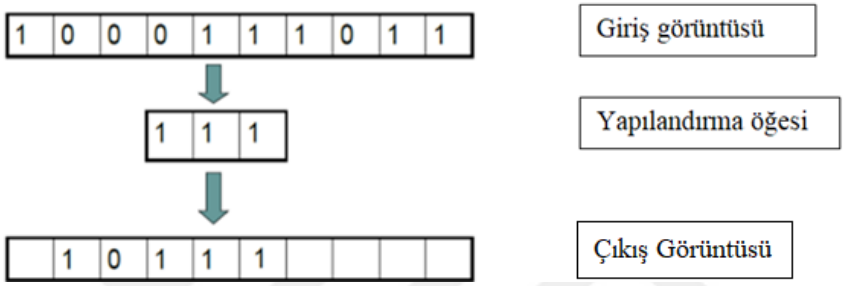
Resim 3.5 Yayma işleminin aşama aşama uygulanması.

Resim 3.5'da görüldüğü üzere merkezi bir piksel belirlenmekte ve yayma işlemi bu pikselin etrafında gerçekleştirilmektedir.



Resim 3.6 Yayma yöntemi uygulaması.

Yapısal öğenin görüntü ile aynı piksel değeri aldığı durumlarda yapısal öğenin tüm değeri görüntünün o kısmına aktarılmaktadır. Şekil 3.4'te bu yöntem gösterilmiştir.



Şekil 3.4 Yayma yöntemi akış şeması.

3.1.5 Sobel Kenar Bulma Algoritması

Sobel filtresi bir kenar bulma algoritmasıdır ve bu algoritma siyah beyaz görüntüye uygulanmaktadır. Dönüştürmek istenilen görüntü ilk önce siyah beyaz formatına dönüştürülmektedir. Sobel filtresi temel olarak [3x3]'lük bir çekirdek (kernel) matrislerden oluşmaktadır. Bu çekirdek matrisler görüntü içerisinde bulunan kenarları bulmak için kullanılmaktadır. Sayısal bir görüntü, bir fonksiyon olarak değerlendirildiğinde, bir nokta üzerindeki 1. türev değerinin, [3x3]'lük komşu olan dört yönde elde edilebilir 1. türev değerlerinin vektör toplamları olmaktadır. Bu yaklaşımla Sobel kenar işleci oluşturulmuştur (Aybar 2008). Şekil 3.5'te [3x3]'lük bir Sobel çekirdeği verilmiştir.

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Şekil 3.5 Sobel kenar bulma algoritması çekirdek ağırlık kaysayıları.

Ani gri seviye değişimlerinin tespit edilmesi kenar tespit edebilmenin en verimli yöntemlerinden birisi sayılmaktadır. Bu yöntemi uygulayabilmek için denklemin bölgesel olarak türevine bakılmaktadır. Kenar belirlenmek istenen görselde, bölgesel olarak türevlere bakıldığında kenar bölgelerinde 1. türev en büyük değere sahip olmaktadır. Buna karşın görüntünün üzerinde 2. türevi alınırsa kenar bölgeleri sıfır değerini almaktadır. Bölgesel olarak görüntünün 1. ve 2. türevleri hesaplanmakta ve böylece kenar tespit edilmektedir. Görsel üzerinde kenar bulmak için 1. Türev kullanıldığında gradyan yöntemi adını almakta 2. türev kullanıldığında ise laplasyen yöntemi adı verilmektedir. Bilindiği üzere bir fonksiyonun 1. türevi o fonksiyonun eğimini vermektedir. Bu yüzden gradyan yönteminin diğer bir adı eğim yöntemidir.

Denklem (3.14)'te görüntünün içerisindeki her bir pikselin eğim değerinin büyüklüğü G olarak ifade edilmektedir ve Şekil 3.5'te verilen Gx yatay ve Gy dikey türev değerleri kullanılarak hesaplanmaktadır (Koyuncu 2015).

$$G = \sqrt{(G_x^2 + G_y^2)} \tag{3.14}$$

İyi bir kenar bulma operatörü için bazı şartlar bulunmaktadır;

- Görüntüdeki kenarları tespit edebilmelidir.
- Gerçek görüntüdeki kenarlar ile algoritma uygulandıktan sonraki bulunan kenarlarının konumları bir biriyle tutarlı olmalıdır.
- Görüntüde içerisinde bulunan kenarın olduğu konumda bir tane kenar belirleyebilmeli ve yapay kenarlar üretilmemelidir.

3.1.6 Renk Değiştirme Algoritması

Görüntü işleme üzerinde görüntülerin tanımlanması için kırmızı, yeşil ve mavi olmak üzere üç ana renk kullanılmaktadır. Görüntüler üzerinde değişiklik yapılabilmesi için bu üç ana renk üzerinde değişiklik yapılması gerekmektedir. Görüntü işleme literatüründe bu renklere RGB denilmekte, kırmızı R (Red), yeşil G (Green), mavi ise B (Blue) ile kısaltılmakta ve toplamsal (additive) renkler olarak tanımlanmaktadır. Bu üç renk görüntü üzerinde üç tane kanala tekabül etmektedir. Bu üç kanal, piksellerle ifade edilmekte, piksellerde sayılar ile temsil edilmektedir. Diğer bir değişle RGB kanallarının her birinin sayısal olarak bir karşılığı bulunmaktadır. Bu sayısal karşılık ikili sistemde en düşük 0 değeri ve en yüksek 1 değeri ile gösterilmektedir. Bu durum sekizli sistemde ise en düşük 0 değeri ve en yüksek 255 değeri ile gösterilmektedir. Görüntü işlemede gri seviye görüntü denilen tek kanallı görseller bulunmaktadır. Bu görseller tek başına renkli bir görüntüyü ifade etmemesine karşın görüntü işleme yöntemlerinde çokça kullanılmaktadır. Üç kanaldan oluşan RGB kanalları bir birinden ayrıştırıldığında her bir kanal gri seviye bir görüntü olmakta ve her kanal 0-255 arasında bir piksel değerine sahip olmaktadır. Renk değiştirme algoritmasında da bu yöntemler kullanılmaktadır. Görüntüden elde edilen RGB değerleri bir birim sola kaydırılarak yeni bir piksel değeri oluşturulmaktadır. Denklem (3.15)'te kırmızı piksel değeri denklem (3.16)'da ki gibi bir birim sola kaydırılarak kırmızı piksel değeri, mavi piksel değerinin yerini alacak ve böylece görüntüdeki kırmızı renk mavi olacaktır. Bu durum yeşil ve mavi renkler içinde aynı olacaktır. Diğer bir değişle yeşil renk yöntem uygulandıktan sonra kırmızı, mavi renk ise yeşil olacaktır.

$$A = [255, 0, 0] \tag{3.15}$$

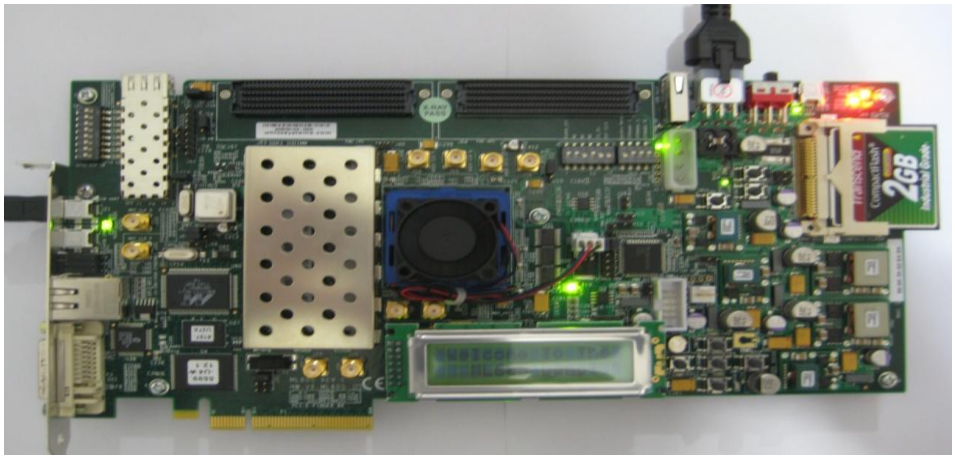
$$A' = [0, 0, 255] \tag{3.16}$$

3.2 Alan Programlanabilir Kapı Dizileri (FPGA)

Bilgisayarın icadından itibaren 1950 yıllarından günümüze kadar tümleşik yapıların tasarımı hızla gelişen bir bilim ve teknoloji alanı olmuştur. Bu teknolojilerden birisi

olan FPGA çipleri 1985 yılında Xilinx firması tarafından geliştirilmiştir. Tarihte programlanabilir olarak üretilen ilk mantık aygıtı PROM (Programmable Read Only Memory (Programlanabilir Salt Okunur Hafıza)) hafızasıdır. Bu terim, bilgi yüklenebilen kalıcı bir bellek anlamına gelmektedir. Farklı birçok PROM türü bir fabrikada toplu olarak (ASIC) ya da kullanıcı (FPGA) tarafından veya sahada programlanabilmektedir. Sahada programlanabilir olanlar FPGA çiplerinin evrimleştiği tiptir. Sahada programlanabilir PROM iki tipte olabilmektedir: EPROM (Erasable Programmable Read Only Memory (Silinebilir ve Programlanabilir Salt Okunur Hafıza)) ve EEPROM (Electronically Erasable Programmable Read-Only Memory (Elektronik olarak Silinebilir ve Programlanabilir Salt Okunur Hafıza)). En yaygın kullanılan EEPROM, kullanıcının belleğin içeriğini silmesine ve birçok kez yeniden programlamasına olanak tanımaktadır (İnt. Kyn. 7). Resim 3.7'de FPGA geliştirme kartı örneği verilmiştir.

FPGA çipleri üzerinde VHDL, Verilog ve System C gibi diller kullanılarak tasarımlar yapılabilmektedir. FPGA çipleri üretim, robotik, uzay ve havacılık, şifreleme ve kod çözme gibi alanlarda kullanılmaktadır.



Resim 3.7 Fpga geliştirme kartı örneği.

3.2.1 FPGA Çiplerinin İç Yapısı

FPGA çipleri tekrar tekrar programlanabilen paralel işlem kabiliyeti olan tümleşik devrelerdir. Bir dizi mantık bloğu ve yönlendirme kanalından oluşmaktadır. İki adet

giriş ve çıkış pedi, bir satırın yüksekliğini veya bir sütunun genişliğini ifade etmektedir. Tüm yönlendirme kanalları aynı genişliğe sahip olmak zorundadır. İlk üretilen FPGA çipi Xilinx tarafından 1985 yılında üretilen XC2064 çipidir ve Resim 3.8'de gösterimiştir (İnt. Kyn. 8). Genel yapısı aşağıda verilen FPGA çipi, programlanabilir üç bileşenden oluşmaktadır.

Resim 3.8 Üretilen ilk FPGA çipi.

3.2.1.1 Yapılandırılabilir Mantıksal Bloklar (Configurable Logic Blocks (CLB))

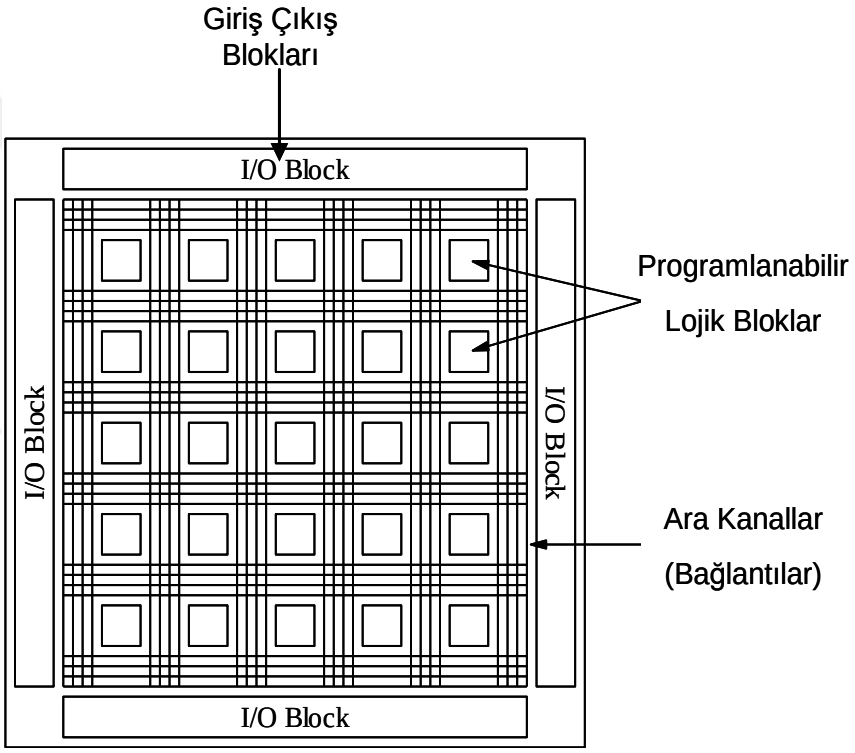
CLB blokları, mantıksal fonksiyonların oluşturulabildiği Look-up table (LUT), tek bitlik bilgilerin saklanabildiği FlipFlop'lar, bilgi akışını yönlendiren multiplekserlar, ve toplayıcıların tanımlanmasında kullanılan "carry-chain" gibi çeşitli elemanlardan oluşmaktadırlar. Oluşturulmak istenen mantıksal devreler çeşitli yazılım araçları sayesinde bir CLB'ye veya parçalara bölünerek birden fazla CLB'ye otomatik olarak uygulanmaktadır.

3.2.1.2 Giriş Çıkış Blokları (Input/Output Blocks (IOB))

IOB'ler FPGA çiplerinin programlanabilir giriş/çıkış terminalleridir. Bu bloklar içinde yer alan pinler isteğe göre giriş, çıkış ya da çift yönlü olarak programlanabilmektedir. Bu bloklar temel görev olarak dış dünya ile çip içindeki donanımın arasında köprü vazifesi görmektedir. FPGA çipinin paket türüne göre bir çipteki IOB sayısı diğer bir değişle pin sayısı 1000 civarındaki sayılara ulaşabilmektedir.

3.2.1.3 Ara Bağlantılar (Interconnections)

Bu birimler hem CLB'ler arasında hem de CLB'ler ile IOB'ler arasında bağlantıları yapılandırmada kullanılmaktadırlar. Programlanabilir olduklarından çok esnek bir yapıya sahiptirler. Bu temel bileşenlerin yanında bazı FPGA çipleri, sayısal sinyal işaret işleme için tasarlanmış özel bloklar, RAM hafıza blokları, hatta işlemci çekirdek üniteleri içerecek şekilde üretilmektedirler (Koyuncu 2011). Şekil 3.6'da bu bağlantılar gösterilmiştir.



Şekil 3.6 FPGA çiplerinin iç yapısı (Koyuncu 2011).

3.2.2 FPGA Pinleri

FPGA pinleri, diğer bir paralel işlemci olan ASIC çiplerinin aksine tekrar tekrar programlanabilmektedir. Bu sayede son kullanıcı istediği bir yenilik ya da istemediği bir yönü değiştirmek için FPGA'yi güncelleyebilmektedir. Bu sebepten dolayı FPGA çipleri üzerinde 2 temel kısım bulunmaktadır.

3.2.2.1 Ayrılmış Pinler (Dedicated Pins):

FPGA çiplerinde bulunan tüm pinlerin %20 ile %30 kadarını oluşturmaktadırlar. Bu pinler, kullanıldıkları alanlara göre üçe ayrılmaktadır;

a. **Güç Pinleri:** Bu kısımda FPGA çiplerinin çalışması için gerekli olan güç ve topraklama pinleri bulunmaktadır.
b. **Konfigürasyon Pinleri:** FPGA çipleri için yazılan programların yüklenmesi için kullanılmaktadır.
c. **Saat Pinleri:** Saat sinyallerinin kullanılması için oluşturulmuş özel alanlardır.

3.2.2.2 Kullanıcı Pinleri (User Pins):

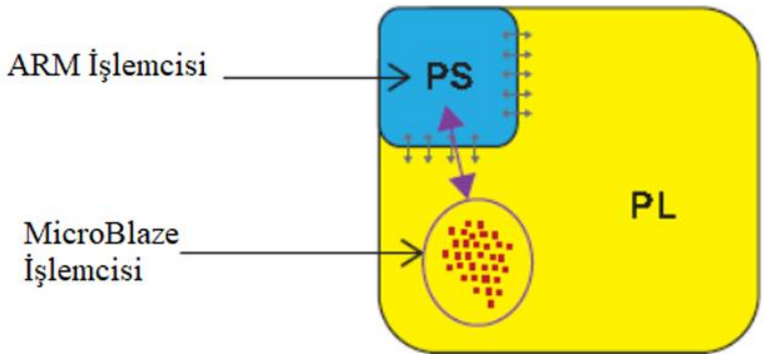
Bunlar kullanıcı tarafından yapılandırılabilen genel yapıda giriş - çıkış pinleridir. Giriş, çıkış ve aynı zamanda giriş ve çıkış olarak üç ayrı bölüme ayrılır. FPGA kartı üzerinde bulunan her bir giriş - çıkış pini aynı zamanda FPGA çiplerinde bir giriş - çıkış hücresine bağlıdır. Giriş - çıkış pin hücrelerinin güçleri harici bir pin olan VCCIO (Collector Collector Voltage Input Output (Giriş Çıkış Kollektör Kolektör Voltajı)) tarafından sağlanmaktadır. Eski FPGA çiplerinde birden fazla VCCIO pini bulunmasına rağmen bütün pinler aynı voltaj tarafından beslenilmekteydi. Son zamanlarda üretilen FPGA çiplerinde ise giriş - çıkışlar gruplara ayrılabilmekte ve bu gruplar farklı voltajlardan beslenebilmektedirler. Böylelikle bir grup giriş - çıkış pinleri 3.3 V ile çalışırken diğer grup giriş - çıkış pinleri de 2.5 V ile çalışabilmektedir (İnt. Kyn. 9).

3.2.3 Zynq-7000 İşlemcisi

Zynq-7000 genişletilebilir işleme platformu 2011 yılında 28 nm silikon yapıda bir çip ile, Xilinx tarafından üretilmiş olup ilk olarak Virtex-7 2000T FPGA kartı ile kullanıcılara sunulmuştur. Zynq-7000 işlemcisi, aynı cihazın içerisinde bulunan yeniden programlanabilir mantığa sahip çift çekirdekli ARM (Acorn RISC Machine (Meşe Palamudu RISC Makina)) Cortex-A9 işlemciyle birlikte çalışmaktadır. FPGA işlemcisi yerine ilk önce bu işlemciden önyükleme yapılmaktadır (İnt. Kyn. 10).

3.2.3.1 İşlemci Sistemi

Tüm Zynq işlemcileri temel olarak çift çekirdekli ARM Cortex-A9 işlemcisi ile çalışmaktadır. Bu işlemciye zor ve karmaşık yapıları kolaylıkla yapabildiği için "zor seviye" bir işlemci denilmektedir. Bu işlemci cihazda optimize edilmiş bir silikon yapıda bulunmaktadır. Xilinx tarafından üretilen MicroBlaze işlemcisi, Zynq-7000 serisi işlemciler ile kıyaslandığında programlanabilir mantık dokusunun öğelerinin birleştirilmesiyle oluşturulan “yumuşak seviye” bir işlemcidir. Genel olarak, yumuşak işlemcilerin avantajı, işlemci örneklerinin sayısının fazlalığı ve tam olarak uygulanmasının daha kolay olmasıdır. Öte yandan, zor işlemci olan Zynq-7000 tabanlı ARM işlemcisinde çok daha yüksek performans elde edebilmektedir. ARM işlemcisiyle Zynq-7000 işlemcisinin birlikte çalışabilmesi için Programlanabilir Mantık (Programmable Logic (PL)) kısmında bir veya daha fazla MicroBlaze yumuşak işlemcisi kullanılabilmektedir. MicroBlaze sistem içindeki belirli düşük seviyeli fonksiyonların koordinasyonunda ve yönetilmesinde rol oynayabilmektedir. Genel olarak Programlanabilir Sistem (Programmable Sistem (PS)) olan ARM Cortex-A9 işlemcisini programlanabilir mantık işlemcisi olan Zynq işlemcisinden bağımsız çalıştırmak işi zorlaştırmaktadır. Bu zorluğu kolaylaştırmak için sistemin gereksinimleri veya kullanılacak alanlara göre MicroBlaze ya da Zynq işlemcileri ARM Cortex-A9 işlemcisi ile beraber kullanılabilmektedir. Kısaca, sistemdeki ARM işlemcisinin varlığı, programlanabilir mantık işlemcilerin kullanımını engellemez hatta birçok uygulama, her iki işlemci türünü beraber kullanarak daha hızlı ve kesin sonuçlar elde edebilmektedir. Şekil 3.7’de FPGA cihazındaki ARM ve MicroBlaze işlemcilerinin çip üzerindeki dağılım alanı gösterilmiştir. Geriye kalan sarı bölge Zynq işlemcisini göstermektedir.

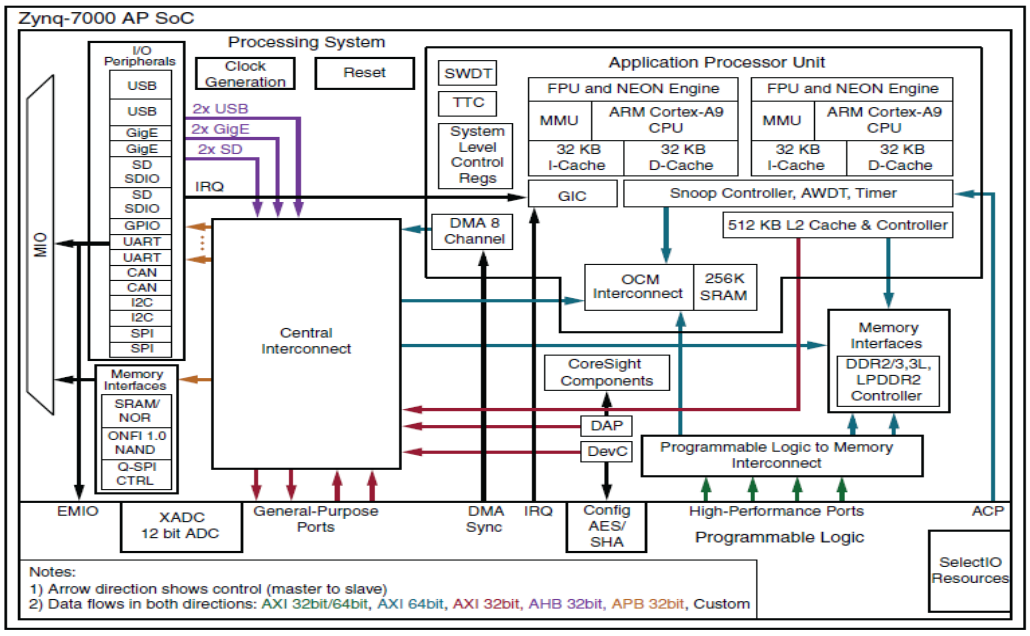


Şekil 3.7 Bir FPGA’deki PS (ARM Cortex-A9) ve Microblaze kısımlarının temsili yerleri.

Zynq işletim sistemi ARM işlemcisi ile birlikte Uygulama İşleme Birimi (Application Processing Unit (APU)) içermektedir. Bu birim çevresel arabirimler, önbellek, bellek arabirimler, ara bağlantı ve saat oluşturma devresini kapsamaktadır.

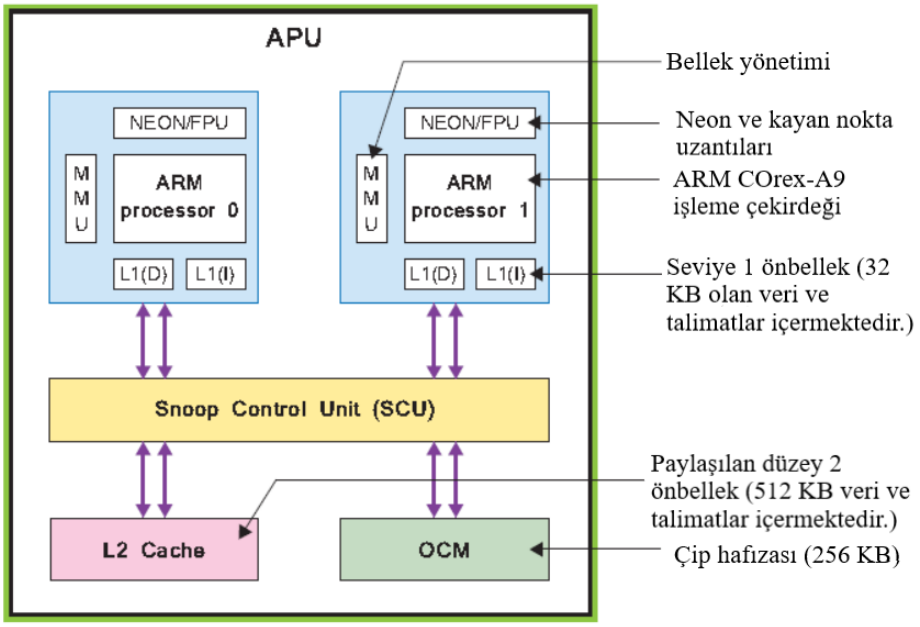
3.2.3.2 Uygulama işleme birimi (APU)

APU öncelikle bir biri ile ilişkili çift çekirdekli ARM işlemci çekirdeğinden oluşmaktadır. Şekil 3.8'de Uygulama İşleme Birimi verilmiştir. Bu birimi, Medya İşleme Motoru (Media Processing Engine (MPE)), Neon ve Kayan Nokta Birimi (Floating Point Unit (FPU)), Bellek Yönetim Birimi (Memory Management Unit (MMU)), önbellek, Başka Çip Belleği (Other Chip Memory (OCM)) ve Snoop Kontrol Ünitesi (Snoop Control Unit (SCU)) oluşturmaktadır. APU biriminin programlanabilen mantık bölümüyle bağlantı kurma ve arayüz oluşturma görevi bulunmaktadır. ARM Cortex-A9 işlemcisi, Zynq işlemcisine bağlı olarak 1 GHz hıza kadar çalışabilme kapasitesine sahiptir. İki ARM çekirdeği 32 KB olan veri kapasitesine ve talimatları iletmek ve saklamak için birinci seviye önbelleğe sahiptir. Bu işlem bağlantıların ve işlemcinin, kaliteli bir performans vermesi adına gerekli olan veri transferleri için gerekli hafızayı oluşturmaktadır.



Şekil 3.8 Uygulama işleme birimi (APU).

Çift çekirdekli ARM Cortex-A9 işlemcisi talimatların ve verilerin direkt olarak ve sorunsuz ulaşılabilmesi için saklama işlemi yapmaktadır. Bu işlemin yapılabilmesi için işlemci içerisinde seviye iki olarak adlandırılan önbellek ve bu önbellikte 512 KB kapasite bulunmaktadır. Ayrıca APU biriminde çip kısmında 256 KB boyutunda OCM belleği ve MMU belleği vardır. MMU belleğinin asıl görevi sanal şekilde çalışan adresler ile fiziksel olarak görüntülenebilen adresler arasında çeviri ve anlaşma yapmaktır. Şekil 3.9'da bu APU biriminin akış şeması verilmiştir.



Şekil 3.9 Uygulama işletim biriminin akış şeması.

3.2.3.3 İşletim Sistemi Arabirimleri

Zynq-7000 işlemcisinin içerisinde bulunan programlanabilen sistemi birimi, hem programlanabilen mantık bileşenleri ile arasında hemde harici bileşenler arasında birçok arabirimlere sahiptir. PS ve harici bileşenler arasındaki iletişim, öncelikle bağlantı sağlayan Çoklu Giriş / Çıkış (Multiple Input Output (MIO)) ile sağlanmaktadır, böylece çevre birimleri ve pinler arasındaki haberleşme gerektiği gibi tamamlanabilmektedir. Bazı belirli bağlantılar PS biriminden harici bağlantılara Genişletilmiş Çoklu Giriş / Çıkış (Extended Multiple Input Output (EMIO)) aracılığıyla da yapılabilmektedir. Bu işlem bilgiyi MIO yerine PL çevre biriminin giriş ve çıkış kaynaklarını kullanarak

paylaşmaktadır. EMIO, hali hazırda bulunandan daha fazla bağlantı noktası gerektiğinde veya PS biriminden PL birimine uygulanan IP bloğu yöntemi ile yapılan bağlantı şekli için kullanılabilmektedir. Zynq-7000 işlemcisinde bulunan arabirimler Çizelge 3.1'de gösterilmiştir.

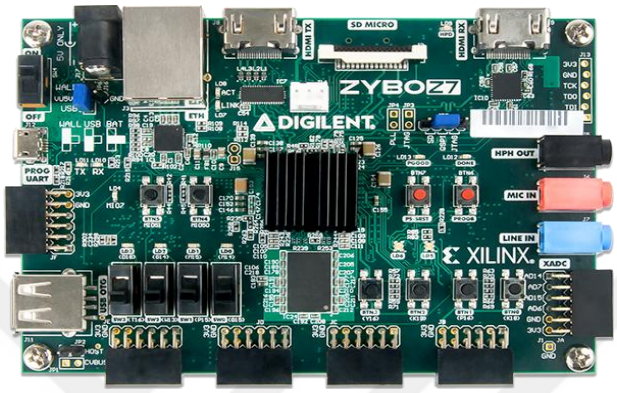
3.2.4 Zybo Z7-20 FPGA Kartı

Zybo Z7-20 kartı, 2012'de piyasaya sürülmüş olup şu anda emekli edilmiş olan Zybo kartının ikinci nesil güncellemesi olan ve Xilinx firması tarafından Zynq-7000 işlemci ailesi üzerine üretilmiş ve kullanıma hazır gömülü, yazılım ve dijital devre geliştirme kartıdır. Zybo Z7-20 kartı üzerindeki Zynq işlemcisi, çift çekirdekli ARM 667 MHz Cortex-A9 işlemcili Xilinx 7 serisine dahil edilmiş olup PL ve PS mantığı beraber entegre edilmiştir.

Çizelge 3.1 Zynq 7000 işlemcili Zybo Z7-20 kartındaki istasyonlar (Zynq Book).

Arabirimler	Açılımı	Görevi
SPI	Seri Çevre Arayüzü (Serial Peripheral Interface)	Master veya slave modunda kullanılabilen seri iletişim için kullanılan standart 4 pinli arabirim yöntemidir.
I2C	Entegre Devre (Inter-Integrated Circuit (IIC))	Seri haberleşme türlerinden senkron haberleşme yöntemidir. En az bir master (usta) bir de slave (köle) cihaz ile kullanılmaktadır.
CAN	Kontrolör Alan Ağı (Controller Area Network)	Haberleşme protokolüdür. En az bir Master ve bir Slave ile çalışmaktadır. ISO 118980-1, CAN 2.0A ve CAN 2.0B standartlarıyla uyumlu veri yolu arabirimi denetleyicisi bulunmaktadır.
UART	Evrensel Asenkron Alıcı / Verici (Universal Asynchronous Receiver Transmitter)	Düşük hızla çalışan ve seri iletişim sağlayan modem arayüzüdür. FPGA kartı ile bilgisayar arasında terminal bağlantıları için kullanılmaktadır.
SD Card	Güvenli Dijital Bellek Kartı (Secure Digital Memory Card)	SD kart belleği ile bağlantı sağlamak için kullanılmaktadır.
USB	Evrensel Seri Veriyolu (Universal Serial Bus) (Zynq book)	FPGA kartı ve ana bilgisayar arasında USB 2.0 bağlantısı sağlanmaktadır.
GigE	Ethernet	10 Mbps, 100 Mbps ve 1 Gbps modlarını destekleyen Ethernet MAC çevre birimidir.

Zybo Z7-20 kartı, görüntü işleme uygulamaları için MIPI CSI-2 uyumlu Pcam konektörü, HDMI girişi, HDMI çıkışı ve yüksek DDR3L bant genişliği de dâhil olmak üzere birçok video işleme özellikleri bulundurmaktadır (İnt. Kyn. 11). Zybo Z7-20 kartı Resim 3.13'te gösterilmiş ve Çizelge 3.2'de Zybo Z7-20 kartının özellikleri listelenmiştir.



Resim 3.9 Zybo Z7-20 Kartı.

a. Yazılım Desteği: Zybo Z7-20 kartı Xilinx firmasının üretmiş olduğu Vivado HLx programları aracılığıyla programlanabilmektedir. Bunlar; Vivado HLx, Vivado ve Vivado HLS (High Level Synthesis) olmak üzere ikiye ayrılmaktadır. Zybo Z7-20 gelişrime kartı Resim 3.9'da verilmiştir.

b. Zynq SoC Mimarisi: Zynq SoC (System on Chip (Çip Üzerindeki Sistem)) iki ayrı alt sisteme ayrılmıştır: Bunlar İşlem Sistemi (PS) ve Programlanabilir Mantık (PL) bölümleridir.

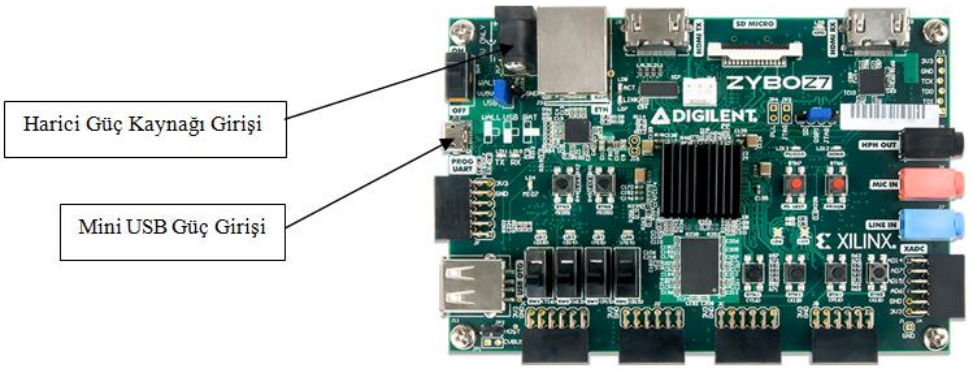
c. Güç Kaynakları: Zybo Z7-20 FPGA mini USB, pil ya da 5 V harici güç kaynağı ve kartı olmak üzere 2 farklı şekilde beslenebilmektedir. Resim 3.10'da bu kaynaklar gösterilmiştir.

d. Zynq Yapılandırması: Zynq-7000 serisi işlemciler, programlanabilir mantık yapısına ve işlemci sistemindeki diğer tüm çipli birimlere master görevi gören işlemci etrafında çalışabilecek şekilde tasarlanmıştır. Bu işlem, bir Birinci Aşama Önyükleyici (First Stage Bootloader (FSBL)) ile programlanabilir mantığı yapılandırmak için bir bit

akışı oluşturmakta ve bu işlem ile kullanıcı uygulaması içeren bir Zynq Önyükleme Görüntüsü ile yükleme ve yürütmeyi sağlamaktadır. Zybo Z7-20 FPGA kartı microSD, Dörtlü SPI Flash ve JTAG olmak üzere üç farklı önyükleme modunu desteklemektedir.

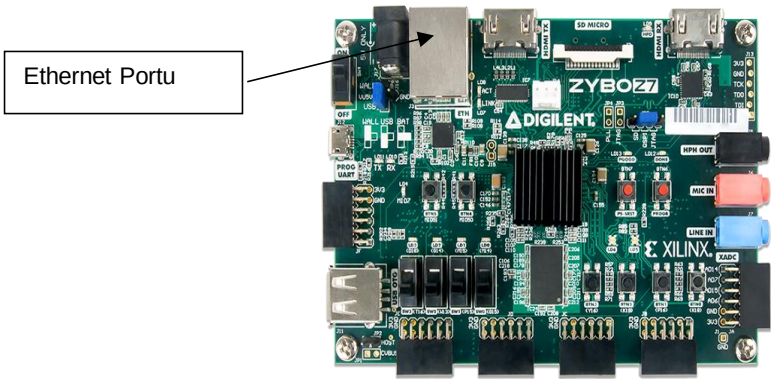
Çizelge 3.2 Zybo Z7-20 kartına genel bakış (İnt. Kyn. 11).

Ürün Özellikleri	Ürün Özelliklerinin Detayları
Zynq İşlemcisi	XC7Z020-1CLG400C
Msps çip üzeri adc	1 adet
Look-up Tables (luts)	53200
Flip-Flops	106400
Block RAM	630 KB
Saat yönetim bölümü	4 adet
Toplam Pmod bağlantı noktası	6 adet
Fan bağlantısı	Opsiyonel Var
Zynq soğutucu	Var
HDMI CEC Desteği	TX ve RX bağlantı noktaları
RGB LED	2 adet



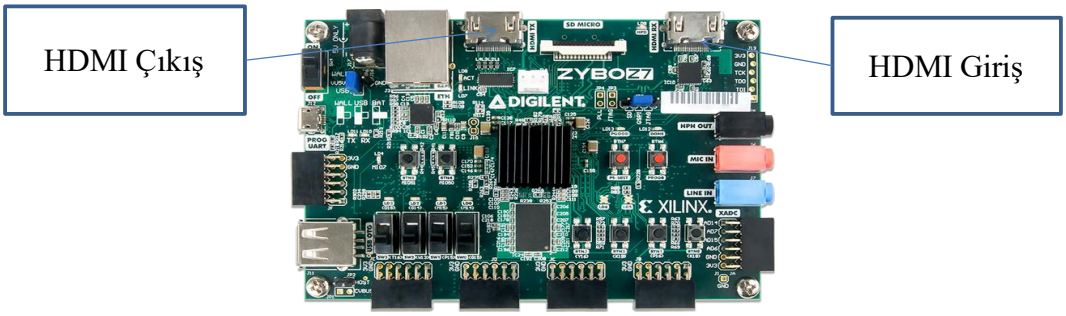
Resim 3.10 Güç girişi kaynakları.

e. Ethernet: Zybo Z7-20 FPGA kartı, ağ bağlantısı için 10/100/1000 Ethernet bağlantı noktası uygulamaktadır. Bunun için Realtek RTL8211E-VL PHY Ethernet kartı kullanılmaktadır. Resim 3.11'de bu giriş gösterilmiştir.



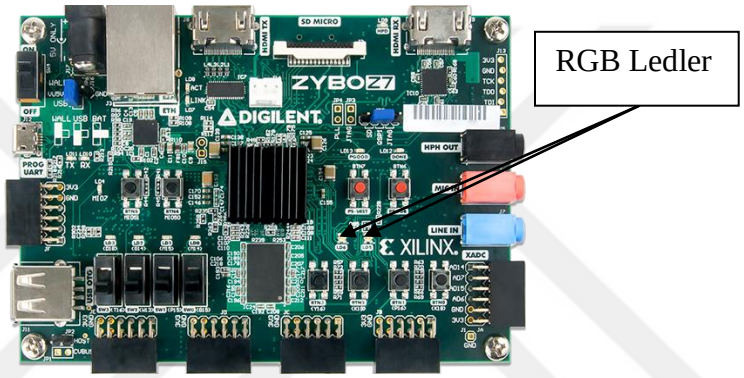
Resim 3.11 Ethernet portu.

f. HDMI: Zybo Z7-20 FPGA kartı, bir adet karttan çıkış sağlayan kaynak portu (HDMI TX (Çıkış)) ve bir adet sisteme akış portu (HDMI RX (Giriş)) içermektedir. HDMI RX bağlantı noktasında kullanılan arabellek, konektör ve FPGA pinleri arasında HDMI akışını ileterek çalışmaktadır. Hem HDMI hem de DVI sistemleri, Zybo Z7-20 FPGA kartının giriş - çıkışları ile doğrudan desteklenen TMDS sinyal standardını kullanmaktadır. Ayrıca, HDMI kaynakları DVI içe akışları ile geriye doğru uyumludur ve bunun tersi de geçerlidir. Diğer bir değişle, Zybo Z7-20 FPGA kartı ile bir DVI monitörünü çalıştırmak veya bir DVI girişininden veri alabilmek için basit bir HDMI girişinden DVI girişine dönüştürücü kullanılabilmektedir. HDMI konektörleri 19 pinden oluşmaktadır. Bu pinler üç farklı veri kanalı, bir diferansiyel saat kanalı, beş GND bağlantı noktası, bir telli Tüketici Elektroniği Kontrolü (Consumer Electronics Control (CEC)) veri yolu içermektedir. Ayrıca I2C veri yolu olan iki telli bir Ekran Veri Kanalı (Display Data Channel (DDC)) veri yolu ve Çalışırken Takmayı Algılama (Hot Plug Detect (HPD)) sinyali ayrıcada 50 mA'ya kadar veri iletebilen 5 V sinyal içermektedir. Resim 3.12'da HDMI giriş ve çıkışı gösterilmektedir.



Resim 3.12 HDMI portları.

g. RGB Ledler: Zybo Z7-20 FPGA kartı iki adet RGB led içermektedir. Her RGB led üç küçük dâhili ledin katotlarını çalıştıran kırmızı, mavi ve yeşil olmak üzere üç giriş sinyaline sahiptir. Bu renklerden birine karşılık gelen sinyali kontrol etmek dâhili ledi aydınlatacaktır. Giriş sinyalleri PL tarafından sinyalleri tersine çeviren bir transistor vasıtasıyla tahrik edilmektedir. Bu yüzden RGB ledleri yakmak için sinyallarin yüksek seviye olarak kontrol edilmesi gerekmektedir. Ledlerin kombinasyonuna bağlı olarak RGB led farklı bir renkte ışık yaymaktadır. Örneğin, kırmızı ve mavi sinyaller yüksek sinyalde, yeşil düşük sinyalde ise RGB led mor renk yayarak yanmaktadır. Ledler, Resim 3.13'te gösterilmektedir.

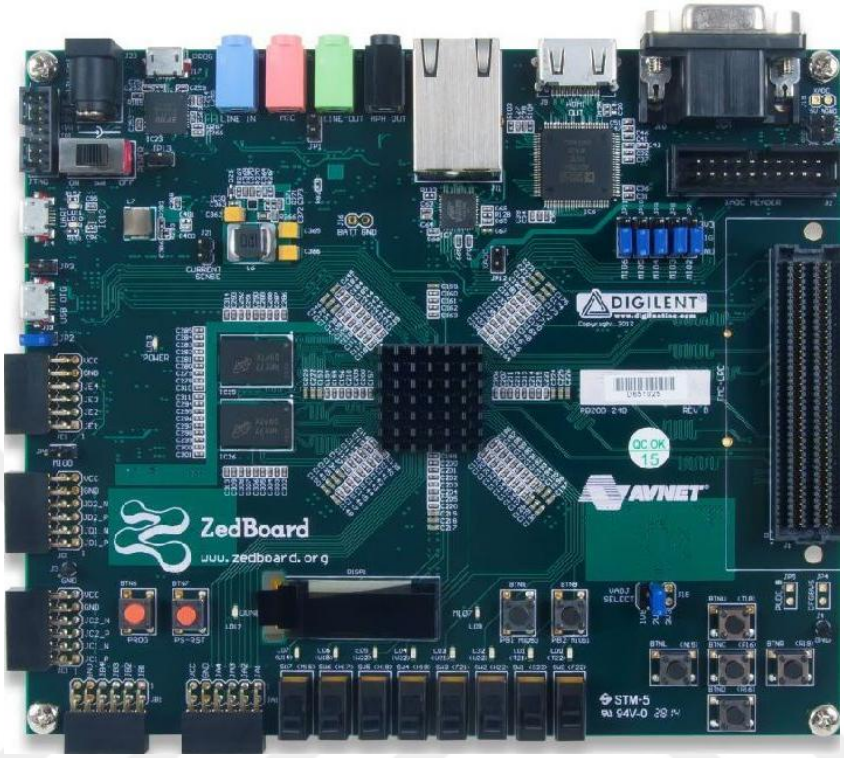


Resim 3.13 RGB ledler.

3.2.5 ZedBoard Zynq-7000 FPGA Geliştirme Kartı

ZedBoard, Xilinx ve Avnet firmaları tarafından üretilen ve içerisinde Zynq-7000 mimarili işlemci bulunan düşük maliyetli bir geliştirme kartıdır. Bu kart Linux, Android, Windows ve diğer OS / RTOS tabanlı işletim sistemleri tarafından programlanabilmektedir. İşlem gücünün yüksek olması, güç tüketimin düşük ve paralellik yapısının olmasından dolayı görüntü işleme, video işleme, aynı anda birden fazla motor sürme gibi çok fazla güç gerektiren çalışmalarda çokça tercih edilmektedir. Tekrar tekrar programlanabilmesi sayesinde ASIC işlemcilerinden birkaç adım öne çıkmaktadır. Xilinx firmasının ürettiği Vivado Design Suite uygulaması ile tasarımlar üretilebilmektedir. ZedBoard FPGA kartına ait donanım görünümü Resim 3.14'de ve Zynq-7000 geliştirme kartının özellikleri ise Çizelge 3.3'te gösterilmiştir. Zedboard FPGA kartının başlıca çalışma alanları şöyledir;

- Video İşleme
- Motor Konrol
- Yazılım Hızlandırma (İnt. Kyn. 12)



Resim 3.14 ZedBoard kartı.

Çizelge 3.3 Zedboard geliştirme kartının özellikleri.

Ürün Özellikleri	Ürün Özelliklerinin Detayları
İşlemci	Çift çekirdekli ARM Cortex A9
Hafıza	512 MB DDR3
SD Kart	Var, 4 GB SD Kart
Giriş / Çıkış Arabirimleri	Programlama ve seri haberleşme için CMOS-UART, 10/100/1G Ethernet, CMOS OTG 2.0, 12-Bit VGA girişi ve VGA çıkışı, ses giriş çıkışı, kulaklık, mikrofon bulunmaktadır.
Ekranlar	128x32 OLED
Anahtarlar ve LED'ler	Programlanabilir mantık ve işleme sistemleri için erişilebilir sıfırlama düğmeleri ve LED'ler vardır.
Saat Çevrim Frekansı	Programlanabilir mantık için 100 Mhz osilatör, İşleme sistemi için 33.333 Mhz osilatör bulunmaktadır.

3.2.6 HDMI Çıkışlı Kamera

HDMI, 2003 yılında ses görüntü verilerinin sıkıştırılmadan direkt dijital olarak aktarılabilmesi için geliştirilmiştir. HDMI; blu-ray, HD-DVD disk çalar, bilgisayar, oyun konsolu, dijital uydu alıcısı gibi cihazlara, ses ve görüntü cihazlara ve görüntü göstermek için üretilen cihazlara bağlanmaktadır. 2006 yılında HDTV, kameralar ve dijital fotoğraf makinelerinde de kullanılmaya başlanmıştır. DVI ile geri uyumlu olup, ek olarak ses verilerini de aktarmaktadır. Sürüm 1.4 ile ethernet veri iletişimini de desteklemektedir. Resim 3.15'de HDMI kablosunun bir örneği verilmiştir. Resim 3.16'da bir HDMI kamera ve Resim 3.17'de HDMI kameranın bağlantı bölümü gösterilmiştir.

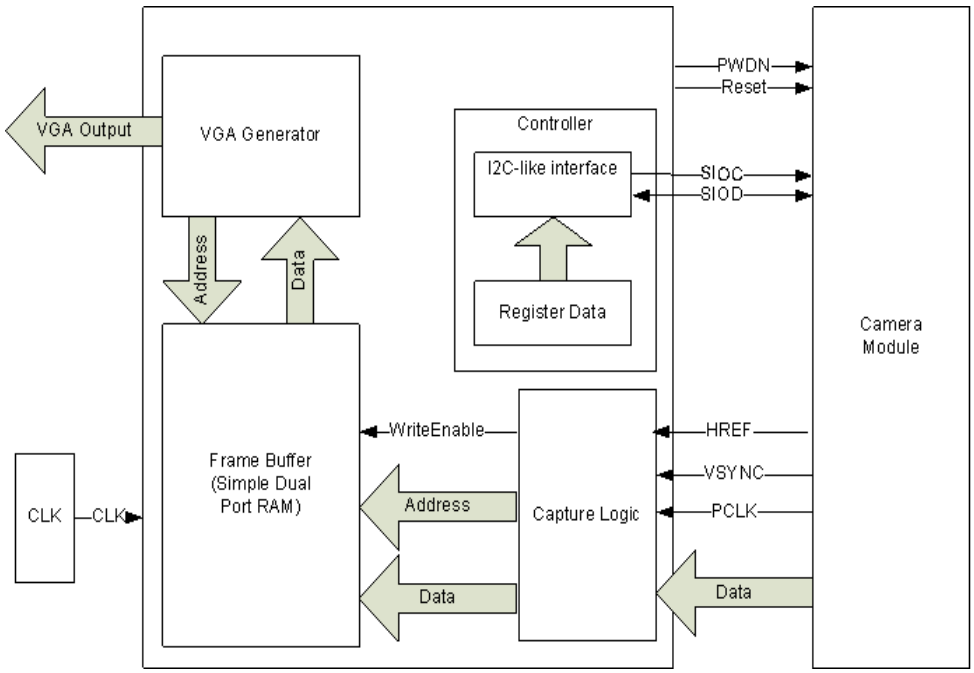
Resim 3.15 HDMI kablosu.

Resim 3.16 HDMI çıkışlı bir kamera.

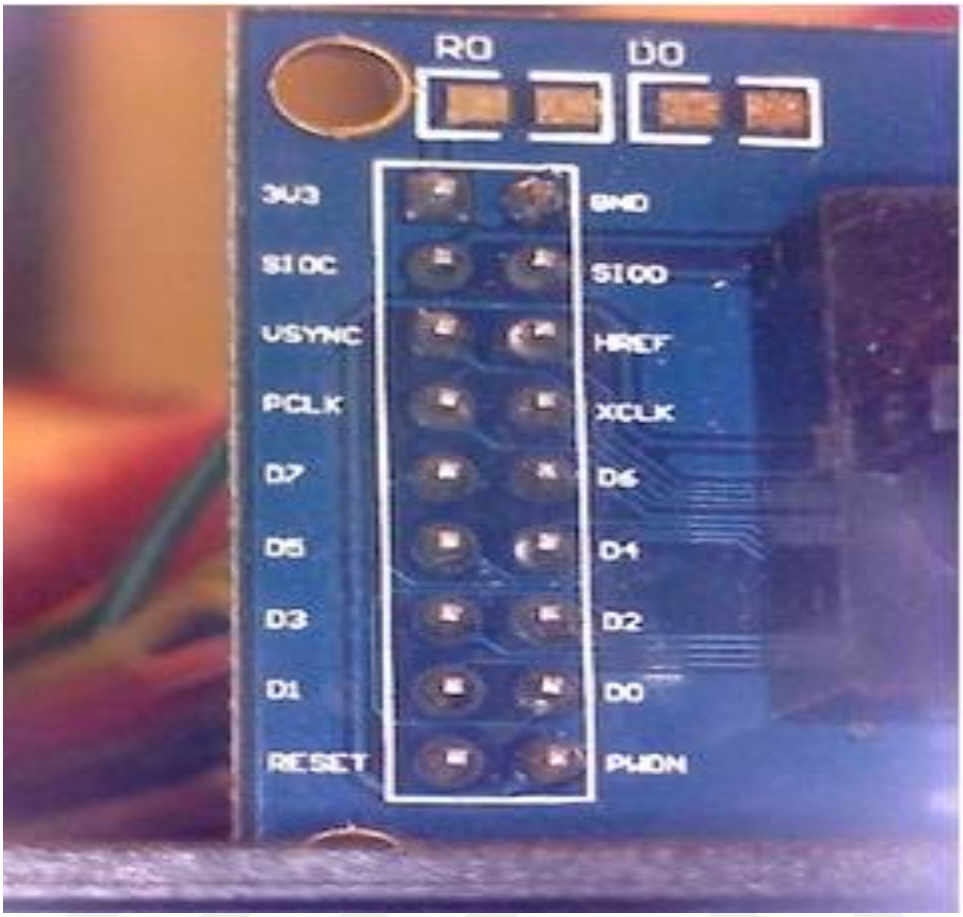
Resim 3.17 Kameranın HDMI bağlantı noktası.

3.2.7 OV7670 Kamera

Averlogic firmasının ürettiği AL422B entegresini kullanmakta olan karmaşık işlemlerde tercih edilebilen bir kamera modülüdür. Üzerinde 16 adet veri pini ve bu 2 adet güç besleme ve toprak pinleri bulunmaktadır. Güç besleme pini 3.3 V ile çalışmaktadır. Diğer pinler kamera modülünden alınan sinyalleri aktarmak ya da kamera modülüne sinyal göndermek için kullanılmaktadır. Şekil 3.10'da kameranın sinyal akış şeması verilmiştir. Resim 3.18'de sinyal pinleri gösterilmiş ve Çizelge 3.4'te sinyal pinlerinin yapısı kısaca anlatılmıştır.



Şekil 3.10 CMOS kameranın iç yapısı.



Resim 3.18 OV7670 Kameranın bağlantı pinleri.

Çizelge 3.4 OV7670 kamerasının giriş çıkış sinyalerinin işlevleri.

İşaretler	Kullanım Yöntemi	Sinyal Aktifliği
3v3	Güç Girişi	
GND	Toprak	
SIOC	Seri Komut Veri Yolu (400 KHz)	
SIOD	Seri Komut Veri Yolu Verileri	
VSYNC	Dikey Senkronizasyon	Aktif Yüksek
HREF	Piksel Örneklemesi İçin CE çıkışı	
PCLK	Piksel Saat	
XCLK	Sistem Saati	
D0-D7	Piksel Verileri	
RESET	Cihaz Sıfırlama	Aktif Düşük
PWDV	Cihaz Gücü	Aktif Yüksek

3.3 Tasarım Uygulamaları

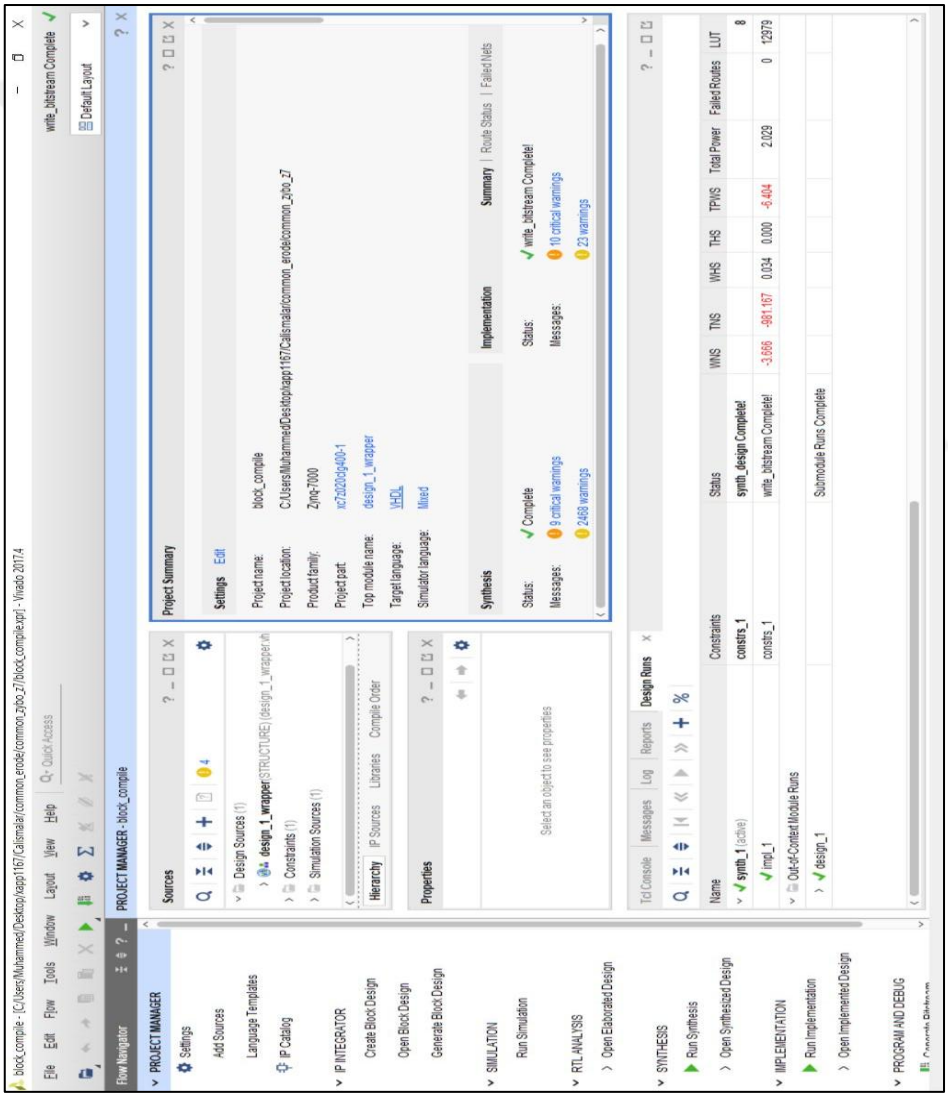
FPGA çipleri kullanıcılar tarafından programlanabilmektedir. Programlama işlemine tasarım denilmektedir ve bu tasarımların yapılabilmesi için FPGA çiplerini üreten Xilinx firması tarafından Vivado HLx adı altında bilgisayar uygulamaları üretilmiştir. Bu tasarım programları bir temel başlığın altında olmak suretiyle üç başlık ile anlatılmıştır.

3.3.1 Vivado Design Suite

Vivado Design Suite, Xilinx tarafından yeni nesil olarak üretilen yine Xilinx üretimi olan FPGA çiplerini programlamak amacıyla üretilmiş bir yazılım paketidir. Vivado Design Suite, eski bir yazılım paketi olan Xilinx ISE yerine piyasaya sürülmüş olup içerisine HDL sentezi ve analizi için geliştirmeler ve ek özellikler eklenmiştir. Vivado Design Suite, tüm tasarım akışının temel olarak yenilenmiş olması, yeni yazım yöntemleri ve teknik gözden geçirmeler yenilikleri ile daha sağlıklı yazılım süreçleri oluşturmaktadır. Gözden geçirenler tarafından "iyi tasarlanmış, çiplere iyi entegre edilmiş ve kullanımı kolay " olarak nitelendirilmektedir. Resim 3.19'da Vivado Design Suite uygulamasının görseli verilmiştir.

Resim 3.19 Vivado Design Suite.

Vivado Design Suite önceki yazılım paketi olan ISE'de olduğu gibi dâhili bir mantık simülatörü içermektedir. Vivado Design Suite ayrıca yaygın olarak kullanılan C ve C++ kodlama dillerini programlanabilir mantığa dönüştüren Vivado HLS uygulaması ile yüksek hızlı sentez imkânı barındırmaktadır. Ayrıca yapılan tasarımların doğrulanması için Elektronik Sistem Seviyesi (Electronic System Level (ESL)) tasarım araçlarını içermektedir. Vivado Design Suite, Xilinx tarafından "veri modeline entegrasyon kolaylığı, algoritmalar ve performans açısından en son sistem" amacı ile tanımlanmıştır. Resim 3.20'de Vivado Design Suite uygulamasının tasarım ekranı verilmiştir.



Resim 3.20 Vivado Design Suite giriş ekranı.

Vivado Design Suite kullanıcıların kullanımına birkaç seçenek ile sunulmaktadır. Uygulama da ücretli lisans ile kullanılan kısımlar olmasının yanı sıra WEB PACK Edition adı verilen sürümü ile de başlangıç seviyesinde ki kullanıcılara tasarımlarını yapabilmeleri için ücretsiz bir yöntem sağlanmaktadır. Bu yöntem tarafından ücretli lisansa ihtiyaç duyulan yöntemler kadar geniş tasarım imkânları sunulmamasına karşın temel kapsamda da olsa tasarımcılara tasarım imkânı sağlanmaktadır. Vivado Design Suite yukarıda bahsedildiği gibi Xilinx ISE uygulamasının yenilenmiş hali olarak sunulmuştur. Resim 3.21'de görseli verilen Xilinx ISE programını Vivado Design Suite ile yenilemek için Xilinx firması büyük bir çalışma yapmıştır. Vivado Design Suite 2012 senesinin nisan ayında tanıtılmıştır. Bu çalışma kapsamında 1000 kişi gibi büyük bir ekiple çalışılmış ve birkaç yıl kadar sürmüştür. Bu çalışma firmaya 200 milyon dolara mal olmuştur.

Resim 3.21 Xilinx ISE amblemi.

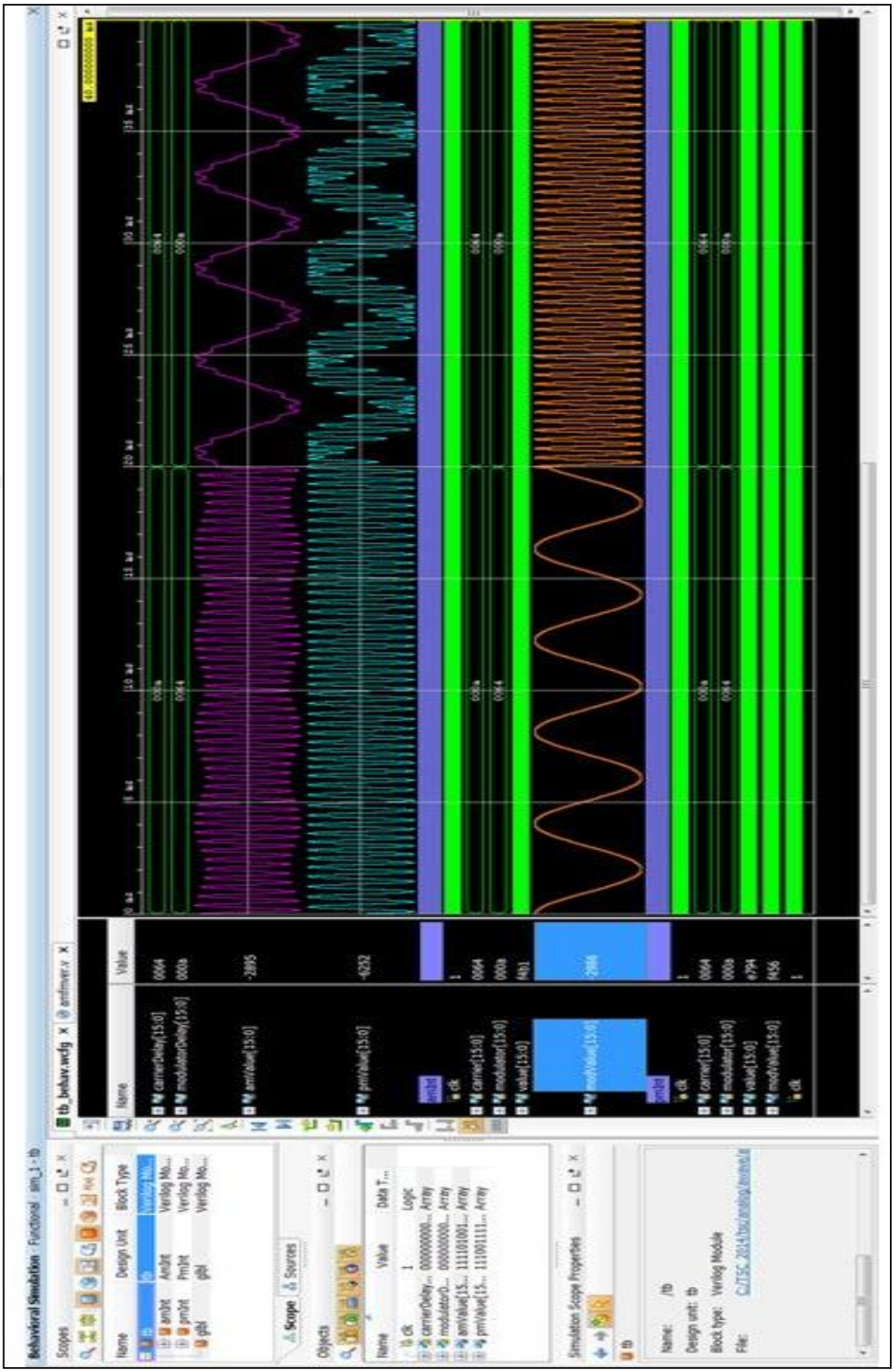
Vivado Design Suite yazılım paketinin içinde bulunan temel tasarım yardımcıları şöyledir;

- **Vivado HLS**, kullanıcılar için zor olabilecek diller olan VHDL ve Verilog gibi dillerin zorluğunu aşmak ve geliştiricinin verimliliğini artırmak için üretilmiş bir tasarım uygulamasıdır. C ve C ++ yaygın olarak kullanılan dilleri ve bu diller ile kullanılan, şablonları, işlevleri, tanımlamaları ve kütüphaneleri kullanılarak tasarım yapılmasını sağlamaktadır. Vivado HLS ile tasarlanan tasarımların paketlenip IP Core olarak Vivado Design Suite içerisine yüklenebilmesi 2014 yılı itibariyle

sağlanmıştır. Vivado HLS içerisinde, birçok farklı firma tarafından üretilen ve yazılım paketlerinde kullanılan Open CL ve Open CV kütüphaneleri bulunmakta ve bu tasarımlar IP Core haline getirilebilmektedir. Resim 3.22'de Vivado HLS giriş ekranı verilmiştir.

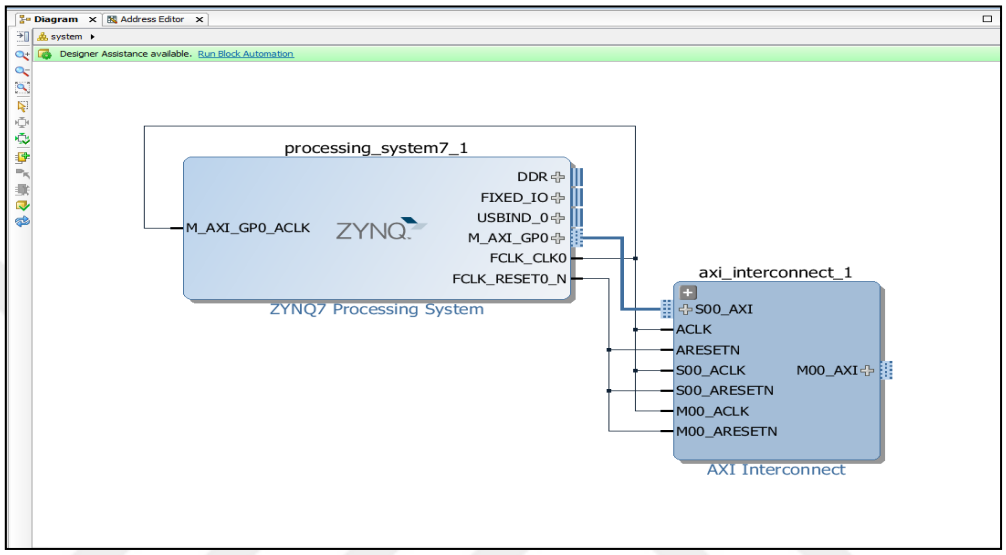
Resim 3.22 Vivado HLS giriş ekranı.

- **Vivado Simülatör**, Vivado Design Suite programının bir bileşeni konumunda olmaktadır. VHDL, Verilog, C ve C++ gibi dilleri ve IP Core'ları destekleyen derlenmiş bir simülatördür. Yapılan tasarım sonuçlarının izlenebilmesini sağlamaktadır. Şekil 3.11'de Vivado simülator ekranı görülmektedir.



Şekil 3.11 Vivado simülatör ekranı.

- **Vivado IP Entegratörü**, tasarımcılar için kullanıma hazır hale getirilmiş Xilinx IP kütüphanesinden hızlı bir şekilde tasarımı oluşturabilmeleri için IP Core kaynağı sağlamaktadır. Entegratör ayrıca, Vivado HLS ve MathWorks Simulink yöntemleri ile tasarımlara imkân sağlamaktadır. Resim 3.23'te IP kütüphanesinden bir örnek bulunmaktadır.



Resim 3.23 Vivado IP entegratörü ile örnek bir tasarım.

- **Vivado TCL mağazası**, Vivado Design Suite eklentilerini geliştirmek için bir sözlük sistemidir. Vivado Design Suite programının kaynaklarına ekleme yapmak ve değiştirmek için kullanılmaktadır ve Takım Komut Dili (Team Command Language (TCL)) anlamına gelmektedir. Vivado Design Suite yazılım paketinin tüm temel işlevleri TCL komut dosyaları aracılığıyla çağrılabilmekte ve kontrol edilebilmektedir.

Xilinx firması tarafından 2018 itibariyle Ultrascale, Ultrascale +, Spartan-7, Virtex-7, Kintex-7, Artix-7 ve Zynq-7000 işlemcileri ile yeni tasarımlar yapabilmek için Vivado Design Suite uygulamasına ihtiyaç duyulmaktadır. Vivado Design Suite 2014 tarihi itibariyle Xilinx firması tarafından üretilen orta ve büyük ölçekli FPGA çiplerini kapsayan bir tasarım uygulamasıdır. Xilinx ISE ise daha küçük ölçekli FPGA çiplerini ve Karmaşık Programlanabilir Mantık Cihazılarını (Complex Programmable Logic Device (CPLD)) kapsamaktadır.

Tüm mikro denetleyici çipler arasında FPGA çipleri, en az güç ile en fazla işlem sağlayan mikro denetleyici olduğu görülmektedir. FPGA çipi için Vivado Design Suite ya da Xilinx ISE ile üretilen ve dikkatlice optimize edilen her algoritma herhangi bir paralel olmayan işlemci ile yapabilecek tasarımlardan daha az gecikme ve daha az güç ile daha hızlı çalışabilmektedir. Xilinx firması tarafından yapılan araştırmalara göre FPGA çipleri Vivado uygulamaları ile CPU çipleri ve GPU çiplerinden 25 kat daha iyi performans göstermektedir (İnt. Kyn. 13).

3.3.1.1 Vivado Yüksek Seviyeli Sentez

Vivado Yüksek Seviyeli Sentez(HLs); C, C ++ ve System C gibi yaygın programlama dillerinin RTL oluşturmaya gerek kalmadan doğrudan FPGA çipleri için IP oluşturulmasını sağlamaktadır. Resim 3.24'de Vivado HLS uygulamasının görseli görülmektedir.

Resim 3.24 Vivado Yüksek Seviyeli Sentez amblemi.

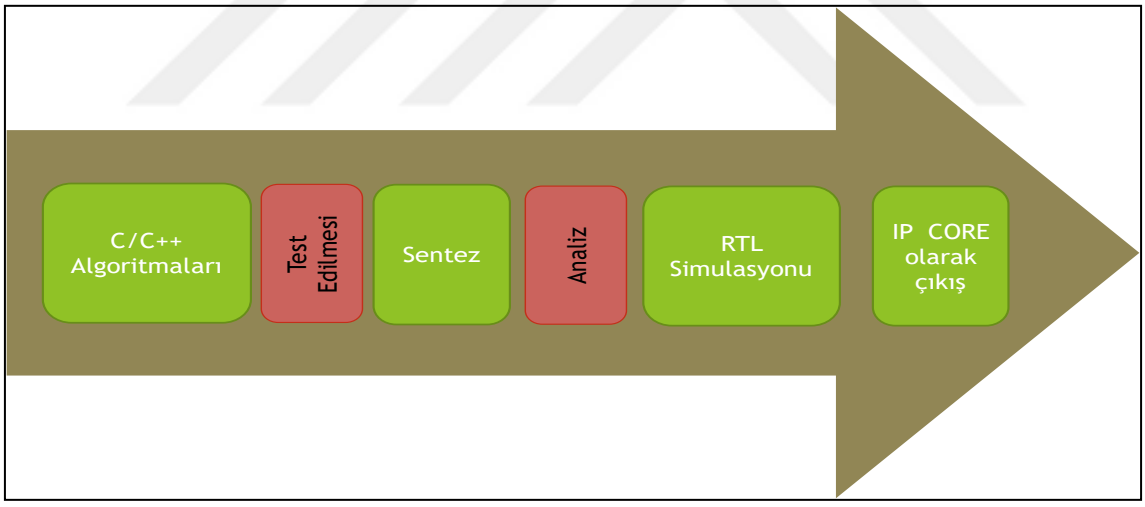
Vivado HLS uygulaması ile algoritmik tanımlama, veri türü belirtimi (tam sayı, sabit nokta veya kayan nokta) ve arabirimlerin (FIFO, AXI4, AXI4-Lite, AXI4-Stream) soyutlanması, rasgele hassas veri türleri, video gibi kütüphaneleri içerisinde barındırdığı

için birçok alanda çalışma yapılabilmektedir (İnt Kyn 14). Çizelge 3.5'te Vivado HLS tasarım uygulamasında bulunan bazı kütüphaneler görülmektedir.

Çizelge 3.5 Vivado Yüksek Seviyeli Sentez kütüphaneleri (İnt Kyn. 14).

Vivado HLS Kütüphanleri	Kütüphane içerikleri
Rasgele Hassas Veri Türleri	Tam sayı ve sabit nokta (ap_cint.h, ap_int.h ve systemc.h)
HLS Akışı	Akış veri yapıları için modeller. En iyi performansı ve alanı elde etmek için tasarlanmıştır (hls_stream.h)
HLS Matematik	C dili için (math.h), C ++ dili için (cmath.h), abs, atan, atanf, atan2, atan2, ceil, ceilf, copysign, copysignf, cos, cosf, coshf, expf, fabs, fabsf, floorf, fmax, fmin, logf, fpclassify, isfinite, isinf, isnan, isnormal, log, log10, modf, modff, recip, recipf, round, rsqrt, rsqrtf, 1/sqrt, signbit, sin, sincos, sincosf, sinf, sinhf, sqrt, tan, tanf, trunk.

Şekil 3.12 ve şekil 3.13'te Vivado HLS uygulamasında tasarlanan yazılımın IP Core, yöntemine dönüşüm aşamaları verilmiştir.

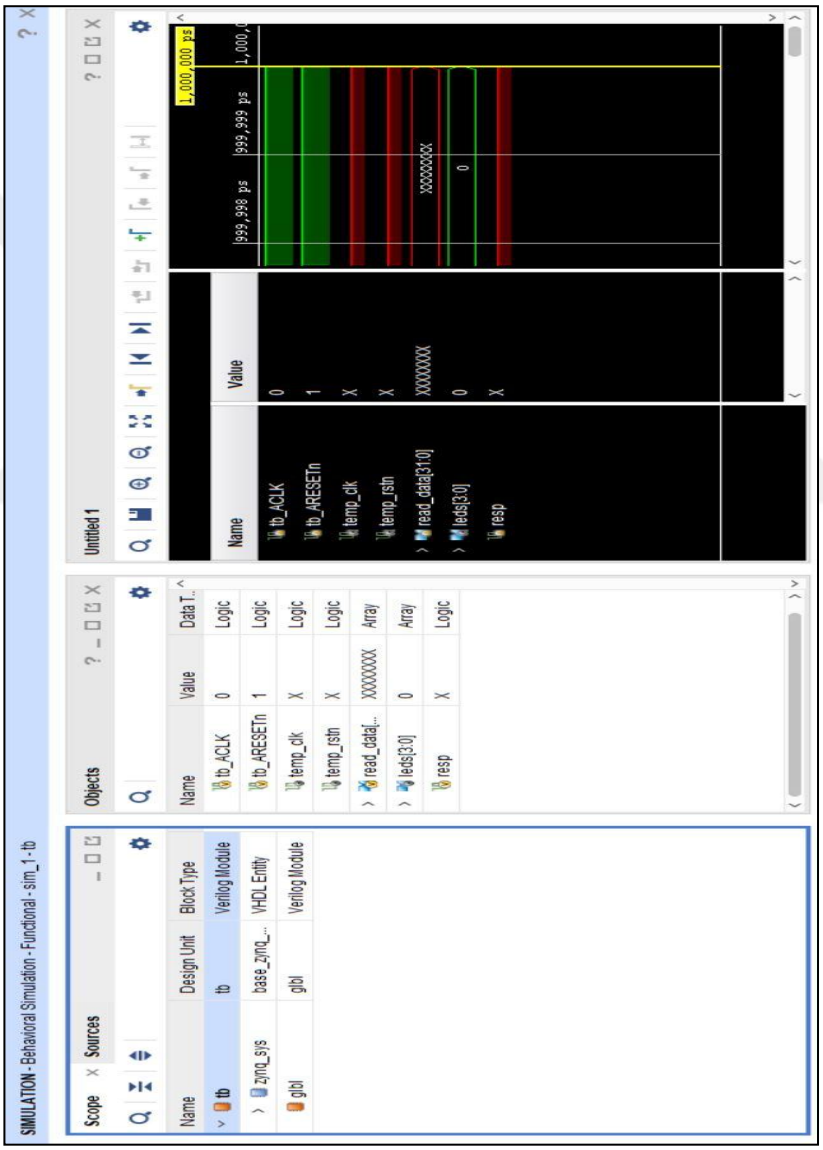


Şekil 3.12 Vivado Yüksek Seviyeli Sentez aşamaları.

Şekil 3.13 Vivado Yüksek Seviyeli Sentez kod dönüşümleri.

3.3.1.2 Vivado Simülatörü

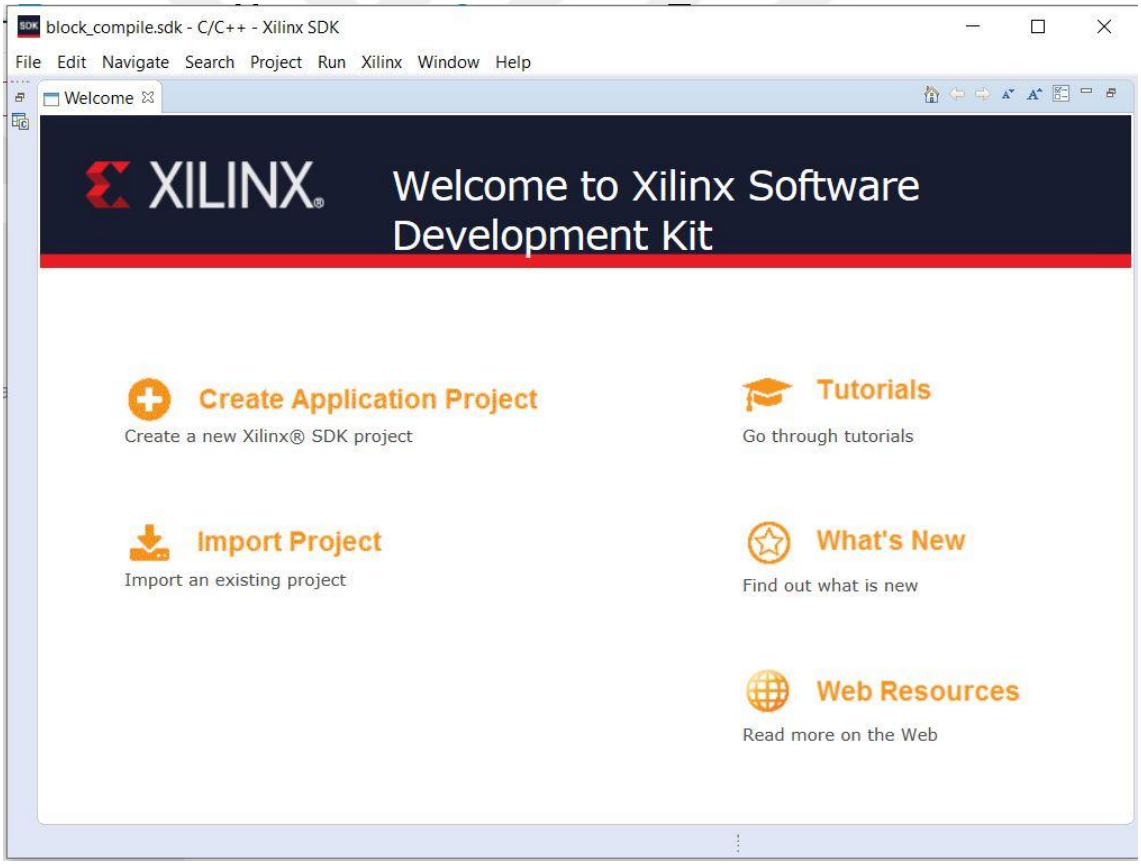
Vivado Simulator; System C, Verilog ve VHDL dillerini destekleyen bir simülatördür. Vivado Simulator, tüm Vivado Design Suite sürümlerinde dâhili olarak ücretsiz bulunmaktadır. Tasarım boyutu, örnekleri veya satır sınırlaması yoktur ve tek bir lisans kullanarak sınırsız sayıda karışık dil simülasyonu örneğini çalıştırmaya izin vermektedir. Resim 3.25'de Vivado simülatörü açılış görseli görülmektedir.



Resim 3.25 Vivado simülatörü açılış görseli.

3.3.1.2 Vivado Yazılım Geliştirme Kiti

Yazılım Geliştirme Kiti (Software Development Kit (SDK)), Xilinx firmasının üretmiş olduğu Zynq, UltraScale, ve MicroBlaze gibi gömülü işlemcileri programlamayı hedefleyen bir Entegre Geliştirme Ortamı (Integrated Development Environment (IDE)) uygulamasıdır. Vivado Design Suite tasarım programının içerisinde bulunmaktadır ve hiçbir lisans ücreti gerektirmemektedir. SDK, Vivado Design Suite ile oluşturulan donanım tasarımlarıyla çalışmaktadır. Eclipse programının açık kaynak standardına ile benzer bir alt yapıya sahiptir. Yaygın olarak kullanılan C ve C++ dilleri ile yazılım imkânı sunulmuştur. Oluşturulan yazılımın uygulanmak istenen mikro denetleyiciye aktarılmadan performans analizini yapabilmektedir. SDK uygulamasının açılış ekranı Resim 3.26'da verilmiştir.



Resim 3.26 Yazılım Geliştirme Kiti açılış ekranı.

4. BULGULAR

Bu çalışmada Zynq 7000 işlemcili Zybo Z7-20 FPGA geliştirme kartı ve Zynq 7000 işlemcili ZedBoard geliştirme kartı ile yapılmış çalışmalar ve bu çalışmaların sonuçları 7 başlık altında verilmiştir.

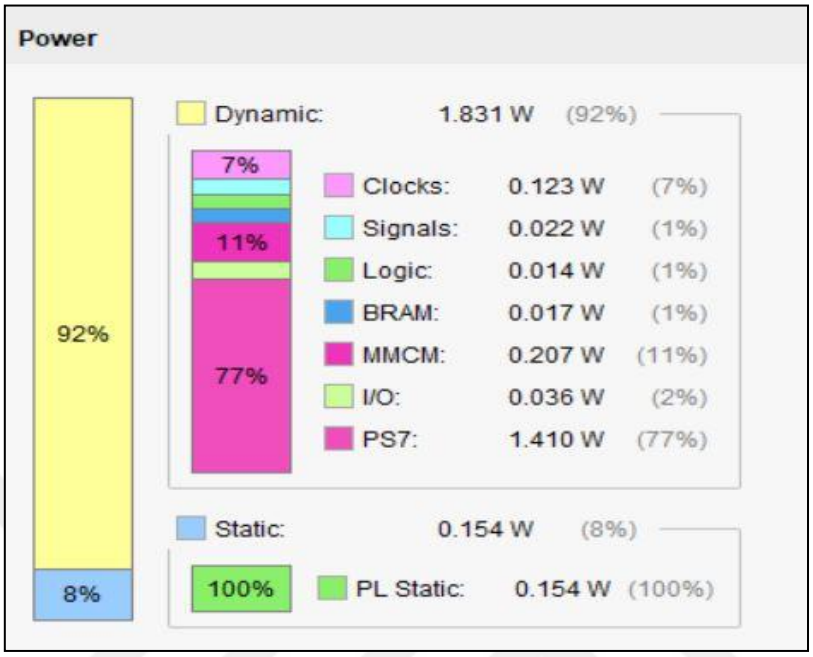
4.1 Gerçek Zamanlı Görüntünün HDMI ve VGA Aracılığı ile Aktarımı

Çalışmanın bu kısmında**,** gerçek zamanlı olarak oluşturulan görüntünün sadece yalın halde ve hiçbir görüntü işleme algoritmasına maruz bırakılmadan HDMI ve VGA kabloları aracılığıyla aktarılması ve monitörden izlenmesi için FPGA çipinde tasarlanmış ve uygulanmıştır. Tasarlanan bu algoritmalada VHDL kullanılmıştır. Bu uygulama ile HDMI ile aktarılan görüntünün oluşturulması için Zynq 7000 işlemcili Zybo Z7-20 FPGA geliştirme kartı kullanılmış ve VGA ile aktarılan görüntü için ise Zynq 7000 işlemcili ZedBoard geliştirme kartı kullanılmıştır. VGA ile HDMI görüntü aktarımları için farklı kartlar kullanılmasının sebebi Zybo Z7-20 kartının VGA çıkışının bulunmamasındır. FPGA çiplerinde gerçek zamanlı görüntünün HDMI ve VGA aracılığı ile aktarımı ile gerçekleştirilen gerçek zamanlı akış uygulaması için Vivado Design Suite programları kullanılmıştır. HDMI aktarımı yapılan görüntünün piksel çözünürlüğü 1280x720 piksel formatındadır. Görüntünün VGA aktarımı için kullanılan piksel çözünürlüğü ise 640x480 piksel formatıdır ve bu görüntünün alınabilmesi için OV7670 kamerası kullanılmıştır.

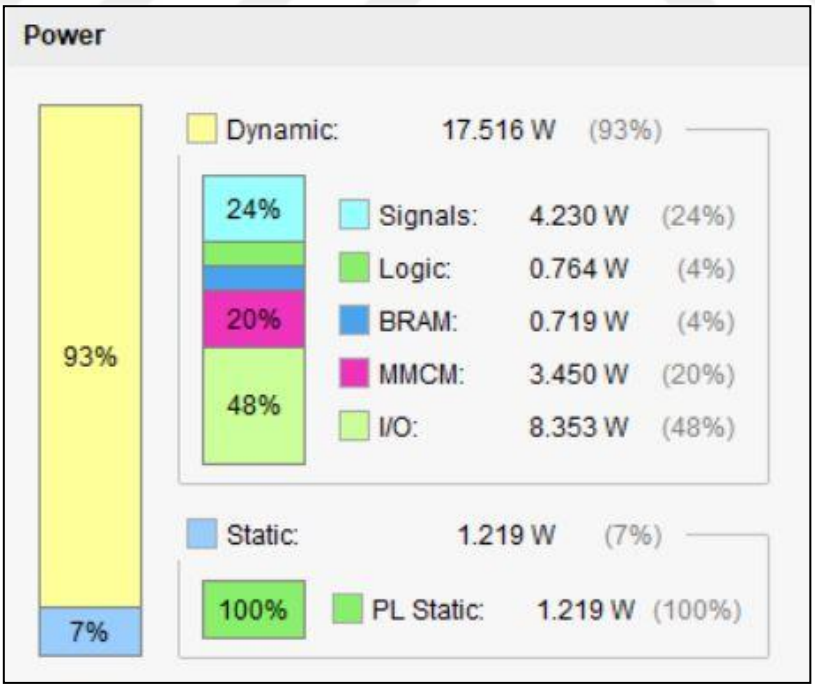
FPGA-tabanlı gerçek zamanlı görüntünün HDMI aracılığıyla aktarılması için Vivado Design Suite programı kullanılmıştır. Zynq 7000 işlemcili Zybo Z7-20 FPGA geliştirme kartı üzerinde yapılan çalışmada, FPGA çipi BRAM %10, FF %17 ve LUT %20 oranında kullanılmıştır. Ayrıca tasarımın çalışması için 1.985 W güce ihtiyaç duyulmaktadır.

FPGA-tabanlı gerçek zamanlı görüntünün VGA aracılığıyla aktarılması için Vivado Design Suite programı kullanılmıştır. Zynq 7000 işlemcili ZedBoard FPGA geliştirme kartı üzerinde yapılan çalışmada, FPGA çipi BRAM %74, FF %1 ve LUT %1 oranında kullanılmıştır. Ayrıca tasarımın çalışması için 17.516 W güce ihtiyaç duyulmaktadır. Bu

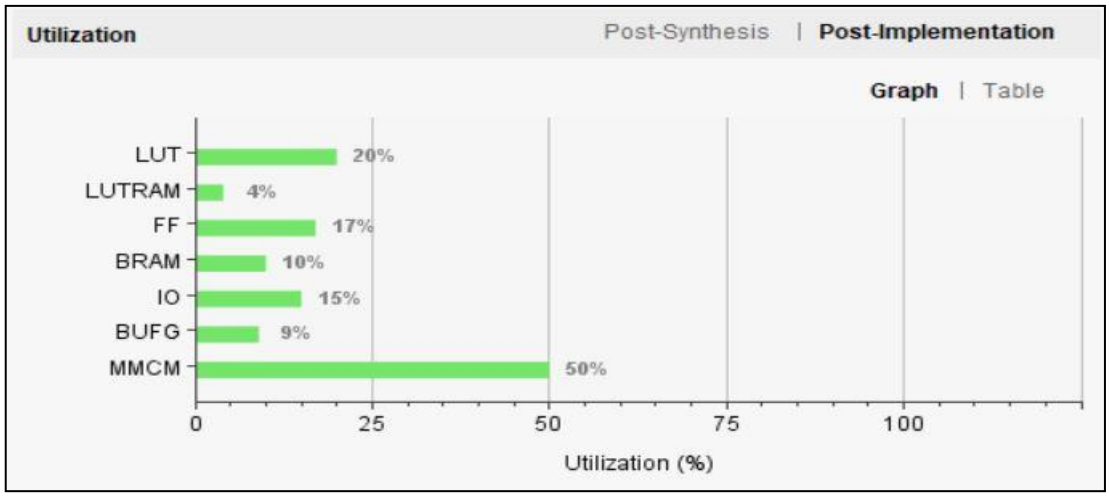
sonuçların güç ihtiyaçlarının görselleri Resim 4.1, Resim 4.2'de ve hafıza kullanımları ise Resim 4.3 ve Resim 4.4'te verilmiştir.



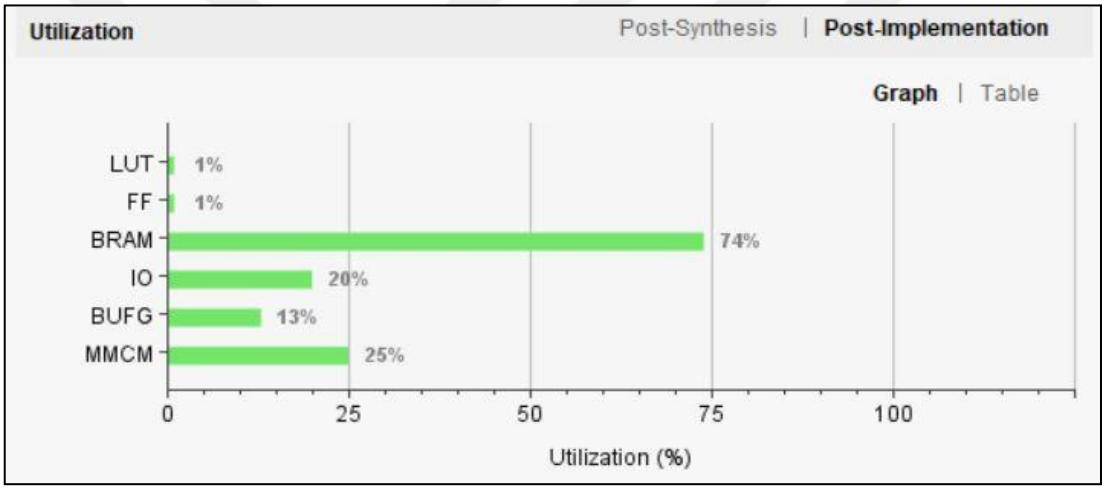
Resim 4.1 Görüntünün HDMI aracılığıyla aktarımı güç istatistikleri.



Resim 4.2 Görüntünün VGA aracılığıyla aktarımı güç istatistikleri.

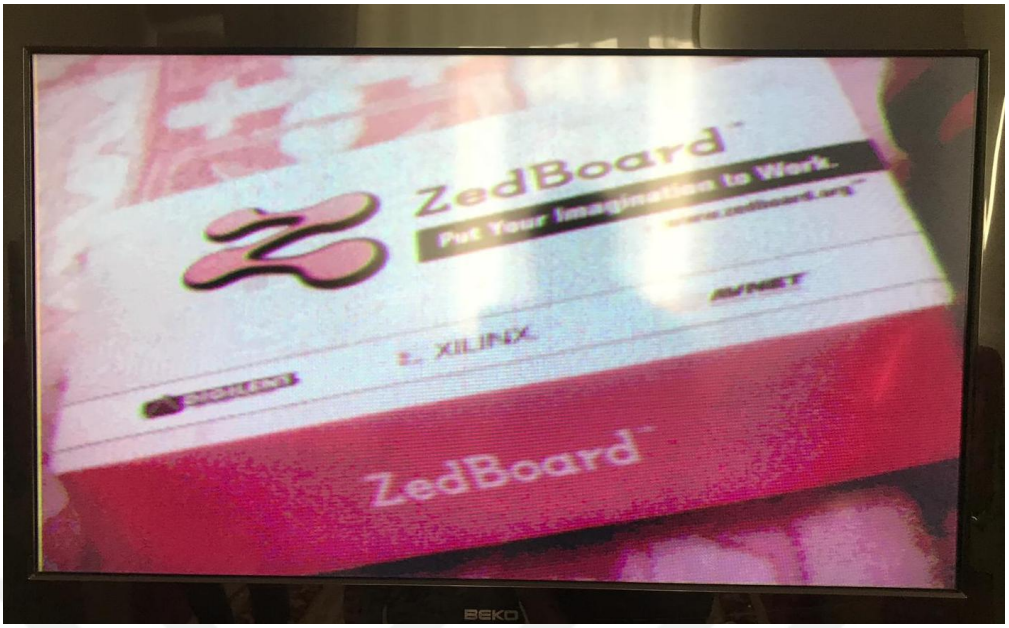


Resim 4.3 HDMI görüntünün çip istatistikleri.



Resim 4.4 VGA görüntünün çip istatistikleri.

Bu sonuçları kıyaslaması yapılırken dikkat edilmesi gereken birkaç husus vardır. Zybo Z7-20 kartı 5 V besleme ile çalışırken Zedboard kartı 12 V besleme ile çalışmaktadır. Sonuçları daha iyi anlayabilmek için yüzdelik oranlara bakılmalıdır. Bu durumda VGA görüntü aktarımı çalışmasının BRAM kullanımı yüksek olması bir dezavantaj olmuş olsada LUT ve FF bölümlerinin düşük seviyede olması bir avantajdır. HDMI görüntünün aktarılması çalışmasında ise genel olarak hafızanın bir miktarı kullanılmış ve istenilen başka işlemlere yeterince hafıza bırakılabilmiştir. Bu çalışmaların ekran görüntüleri Resim 4.5'te gerçek zamanlı görüntünün VGA aracılığıyla aktarımı ve Resim 4.6'da gerçek zamanlı görüntünün HDMI aracılığıyla ekranda görüntülenmesi verilmiştir.



Resim 4.5 Gerçek zamanlı görüntünün VGA aracılığıyla ekranda görüntülenmesi.

Resim 4.6 Gerçek zamanlı görüntünün HDMI aracılığıyla ekranda görüntülenmesi.

4.2 FAST Köşe Algılama Görüntü İşleme Algoritmasının Gerçek Zamanlı Uygulanması

Bu çalışmada, köşe belirleme uygulaması için FAST Köşe Algılama algoritması, FPGA çipi üzerinde donanımsal olarak tasarlanmıştır. Tasarlanan FAST Köşe Algılama

algoritması donanım tanımlama dili olan VHDL ile tasarlanmıştır. Tasarım sırasında kullanılan FAST Köşe Algılama algoritması Vivado HLS (High Level Synthesis) programı ile sentezlenmiş ve IP Core haline getirilmiştir. Vivado SDK uygulanmasından da faydalanılmıştır. Tasarım platformu olarak Xilinx Vivado 2017.4 kullanılmış ve tasarlanan sistem Zybo Z7-20 FPGA çipi için sentezlenmiştir. Elde edilen sonuçlar aşağıda verilmiştir. Resim 4.7'de üzerinde algoritma kullanılmamış görüntü ışık düzeyi düşük görüntü, Resim 4.8'de Matlab uygulamasından alınan sonuçlar verilmiştir. Resim 4.9 ise FPGA üzerinde yapılan çalışma sonuçları verilmiştir ve bu sonuçlar karşılaştırılmıştır.

Resim 4.7 Üzerinde algoritma kullanılmamış ışık düzeyi düşük görüntü.

Resim 4.8 Matlab uygulamasında bulunan fast algoritması ile ve ışık düzeyi düşük ortamda köşe bulunması.

Resim 4.9 FPGA üzerinde fast köşe algılama algoritması ile gece çekilen ve ışık düzeyi düşük ortamda görüntü üzerinde köşe bulunması.

Resim 4.10'da üzerinde algoritma kullanılmamış görüntü ışık düzeyi yüksek görüntü, Resim 4.11'de Matlab uygulamasından alınan sonuçlar ve Resim 4.12 ise FPGA üzerinde yapılan çalışma sonuçları verilmiştir ve bu sonuçlar karşılaştırılmıştır.

Resim 4.10 Üzerinde algoritma kullanılmamış görüntü ışık düzeyi yüksek görüntü.

Resim 4.11 Matlab uygulamasında bulunan fast algoritması ile ve ışık düzeyi yüksek ortamda köşe bulunması.

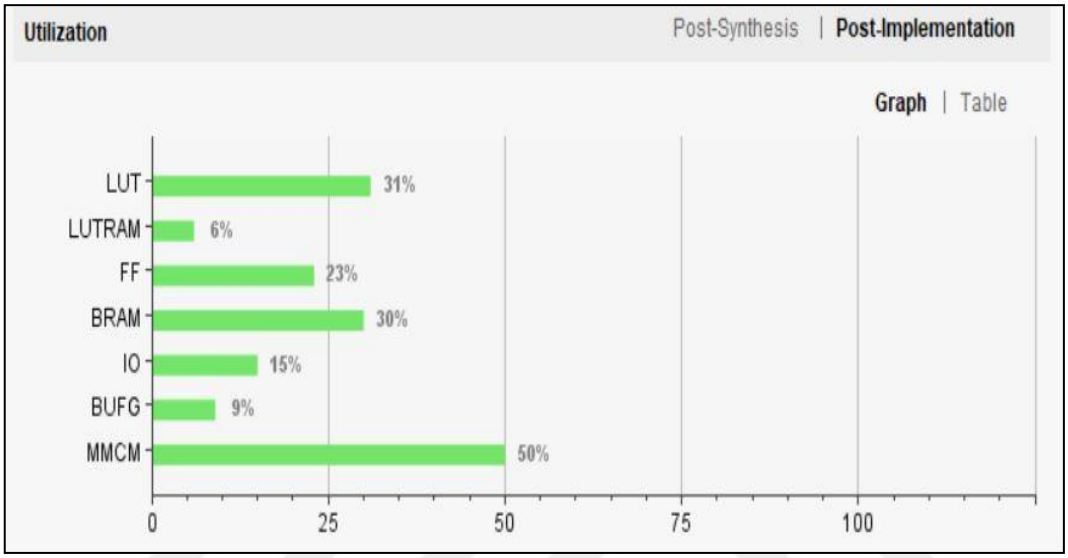
Resim 4.12 FPGA üzerinde fast köşe algılama algoritması ile ışık düzeyi yüksek ortamda görüntü üzerinde köşe bulunması.

FPGA-tabanlı gerçek zamanlı FAST Köşe algılama algoritması ile gerçekleştirilen görüntü işleme uygulaması için Vivado Design Suite ve Vivado High Level Synthesis programları kullanılmıştır. Tasarımda VHDL kullanılmıştır. FAST Köşe algılama algoritması için Open CV kütüphanesinden yararlanılmıştır. Tasarlanan FPGA üzerindeki gerçek zamanlı FAST Köşe algılama algoritması tabanlı görüntü işleme uygulaması kullanılarak 1280x720 piksel formatındaki gerçek zamanlı görüntüler işlenebilmiştir. Zynq 7000 işlemcili Xilinx Zybo Z7-20 FPGA geliştirme kartı üzerinde yapılan çalışmada, FPGA çipi BRAM %30 oranında kullanılmıştır. Ayrıca tasarımın çalışması için 2.145 W güce ihtiyaç duyulmaktadır.

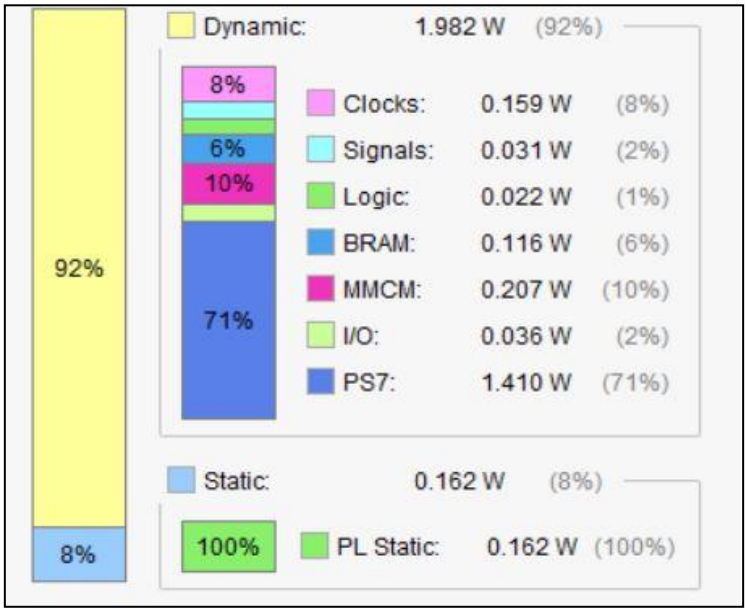
Işık düzeyi düşük görüntünün Matlab ve vivado programlarının sonuçları kıyaslandığında düşük ışıkta FPGA üzerinde uygulanan algoritmanın daha fazla köşe bulduğu görülmektedir. Özellikle görüntüde bulunan sol aşağıdaki evlerin penceresinin etrafına dikkatlice bakıldığında bu fark hemen göze çarpmaktadır. Işık düzeyi yüksek görüntü sonuçlarına bakıldığında ise sonuçların neredeyse bir birine yakın olduğu görülmektedir. Üst kısımdaki beyaz kitaplara bakıldığında FPGA ile yapılan köşe tespitinin bir miktarda olsa daha etkili olduğu görülmektedir. Ayrıca Matlab uygulamasının köşe işaretlemelerinin vivado uygulamasının köşe işaretlemelerine göre

daha büyük boyutta olması göz yanılgısına sebep olabilmektedir. Kıyaslama yapılırken buna dikkat edilmelidir.

FAST Köşe algılama algoritmasının kullandığı çip istatistikleri Resim 4.13'te ve güç istatistikleri 3. 14'te verilmiştir.



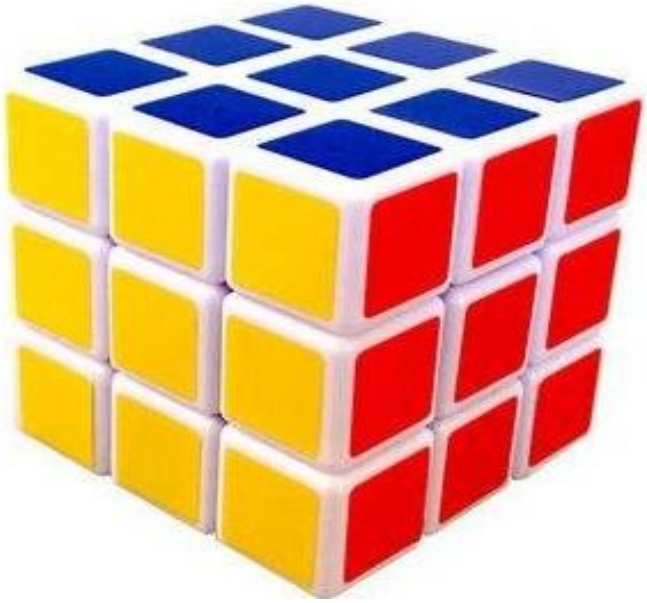
Resim 4.13 FAST Köşe algılama algoritması çip istatistikleri.



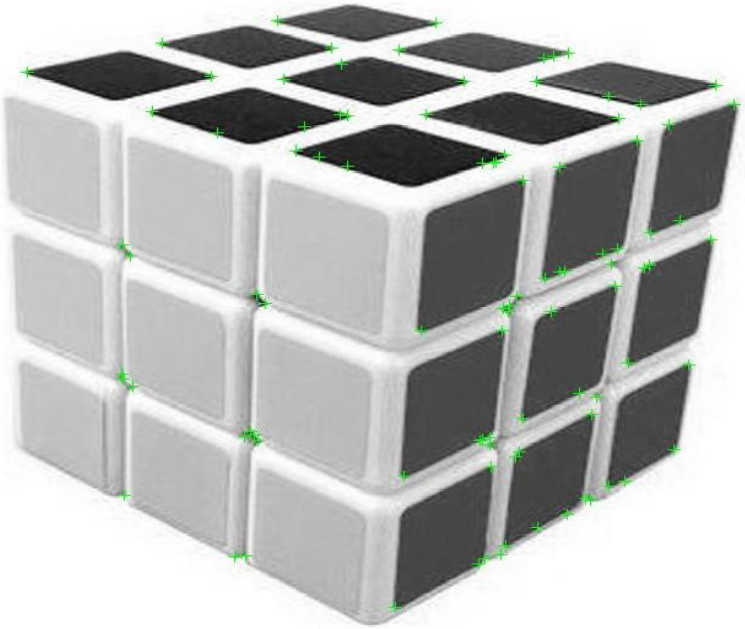
Resim 4.14 FAST Köşe algılama algoritması tasarımının güç istatistikleri.

4.3 Harris Köşe Algılama Görüntü İşleme Algoritmasının Gerçek Zamanlı Uygulanması

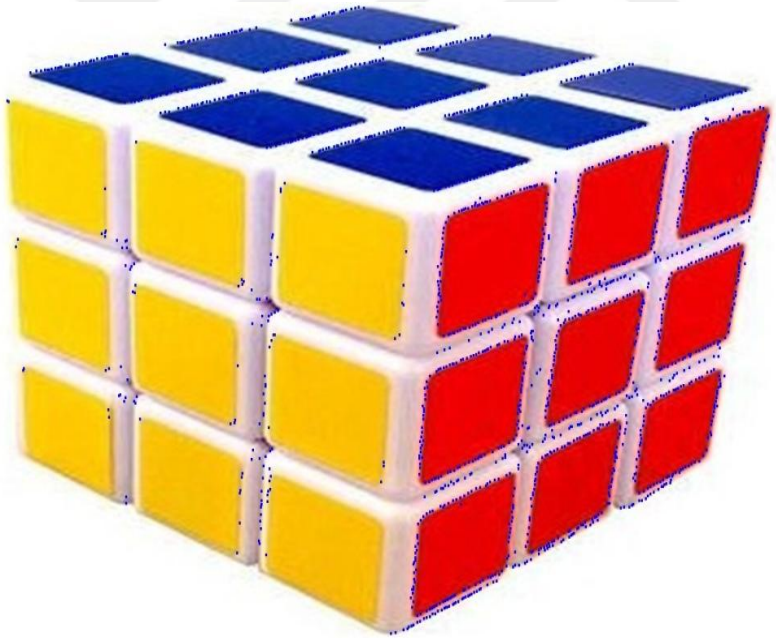
Sunulan bu çalışmada, bir köşe algılama algoritması olan Harris Köşe Algılama Algoritması, FPGA çipinde çalıştırılacak biçimde tasarlanmış ve uygulanmıştır. Bu uygulaması Zynq 7000 işlemcili Xilinx Zybo Z7-20 FPGA geliştirme kartı üzerinde gerçeklenmiştir. Tasarlanan Harris Köşe Algılama görüntü işleme algoritmasının ünitesi donanım tanımlama dili olan VHDL ile kodlanmıştır. FPGA çipinde Harris Köşe Algılama Algoritması ile gerçekleştirilen gerçek zamanlı görüntü işleme uygulaması için Vivado Design Suite, Vivado High Level Synthesis, Vivado SDK programları kullanılmıştır. Bu uygulama için Open CV kütüphanesinden faydalanılmıştır. Tasarlanan görüntü işleme algoritması ile 1280x720 piksel formatındaki görüntüler işlenebilmiştir. Zynq 7000 işlemcili Xilinx Zybo Z7-20 FPGA geliştirme kartı üzerinde yapılan çalışmada, FPGA çipi Blok RAM %48 oranında, LUT %71 oranında, FF ise %42 oranında kullanılmıştır. Ayrıca tasarımın çalışması için 2.651 W güce ihtiyaç duyulmaktadır. Sonuçlar Resim 4.15'te üzerinde algoritma kullanılmayan yalın haldeki görüntü, Resim 4.16'da Matlab uygulamasında haris köşe bulma algoritması ile köşe bulunan küp görüntüleri ve Resim 4.17'de Harris Köşe algılama algoritması kullanılan görüntüleri verilmiştir ve bu sonuçlar karşılaştırılmıştır.



Resim 4.15 Harris Köşe algılama algoritması uygulanmamış rubik küpü.



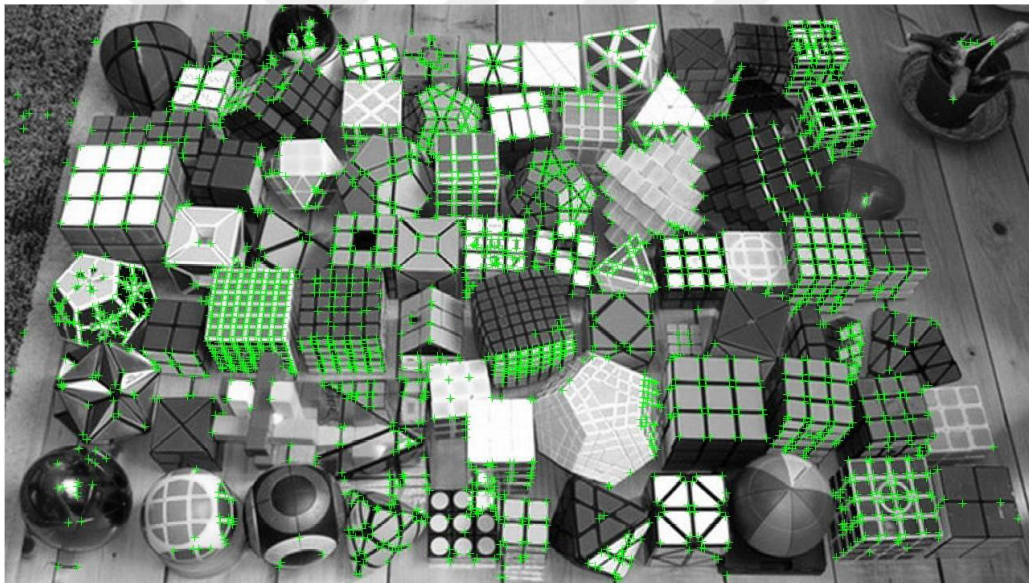
Resim 4.16 Matlab uygulamasında bulunan harris algoritması ile rubik küpü üzerinde köşe bulunması.



Resim 4.17 FPGA üzerinde harris köşe algılama algoritması uygulanmış rubik küpü.

Resim 4.18'te Harris köşe algılama algoritması uygulanmamış yalın halde ki rubik küpleri, Resim 4.19'da Matlab üzerinde harris algoritması ile köşe bulunmuş ve 4.20'de Harris köşe algılama algoritması ile rubik küplerinde köşe bulunmuştur ve sonuçları karşılaştırılmıştır.

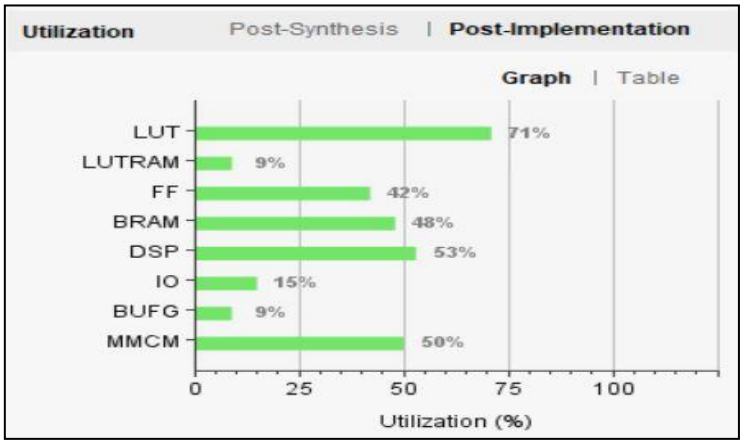
Resim 4.18 Harris köşe algılama algoritması uygulanmamış rubik küpleri.



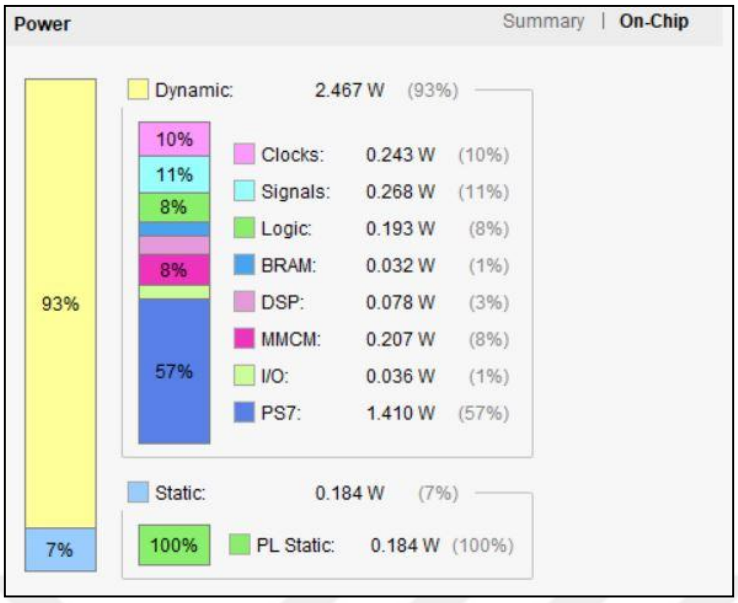
Resim 4.19 Matlab uygulamasında bulunan harris algoritması ile rubik küpleri üzerinde köşe bulunması.

Resim 4.20 FPGA üzerinde harris köşe algılama algoritması ile rubik küpleri üzerinde köşe bulunması.

Resim 4.16 ve Resim 4.17 kıyaslandığında FPGA üzerinde tespit edilen köşeler Matlab programında tespit edilen köşelere göre açıkça daha fazladır. Resim 4.19 ve Resim 4.20 kıyaslandığında yine FPGA ile yapılan köşe tespitinin daha fazla olduğu görülmektedir. Fakat Resim 4.16 ve ve Resim 4.19'a bakıldığında köşe olmayan yerleride işaretleyebildiği görülmektedir. Bu durum bir dezavantaj oluşturmaktadır. Resim 4.21'de sistemin çip istatistikleri verilmiştir. Resim 4.22'de sistemin güç istatistikleri verilmiştir.



Resim 4.21 Harris Köşe algılama algoritması çip istatistikleri.

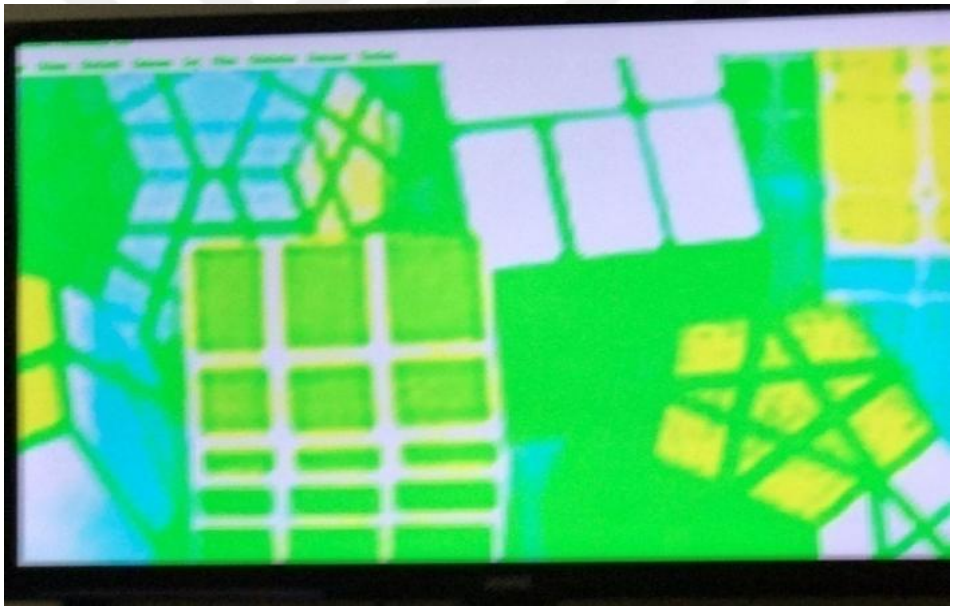


Resim 4.22 Harris Köşe algılama algoritması tasarımının güç istatistikleri.

4.4 Aşındırma Yöntemi ile Morfolojik İşlemlerin Gerçek Zamanlı Uygulanması

Çalışmanın bu kısmın da, morfolojik işlem uygulaması olan Aşındırma Yöntemi görüntü işleme algoritması, FPGA çipi üzerinde donanımsal olarak tasarlanmıştır. Tasarlanan Aşındırma Yöntemi ile görüntü işleme algoritmasının ünitesi donanım tanımlama dili olan VHDL ile kodlanmıştır. Tasarım sırasında kullanılan Aşındırma Algoritması Vivado HLS programı ile sentezlenmiş ve IP Core haline getirilmiştir. Ayrıca Vivado SDK uygulamasından faydalanılmıştır. Tasarım platformu olarak Xilinx Vivado 2017.4 kullanılmış ve tasarlanan sistem Zybo Z7-20 FPGA çipi için sentezlenmiştir. Aşındırma algoritması için Open CV kütüphanesinden yararlanılmıştır. Sonuçlar Resim 4.23'te aşındırma algoritması uygulanmamış görüntü ve Resim 4.24'te aşındırma algoritması uygulanmış görüntü olarak verilmiş ve tartışılmıştır.

Resim 4.23 Aşındırma algoritması uygulanmamış rubik küpleri.

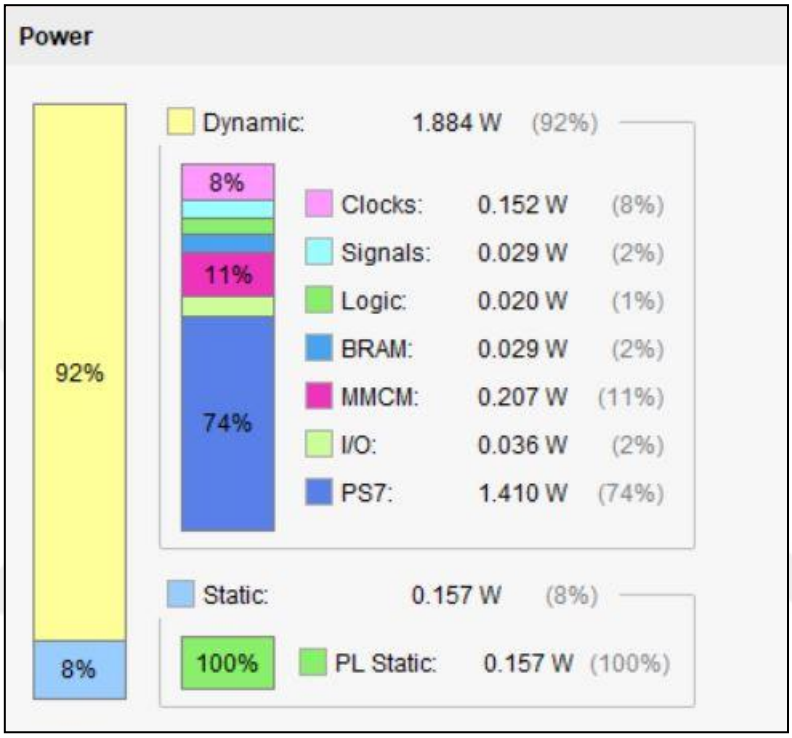


Resim 4.24 Aşındırma algoritması uygulanmış rubik küpleri.

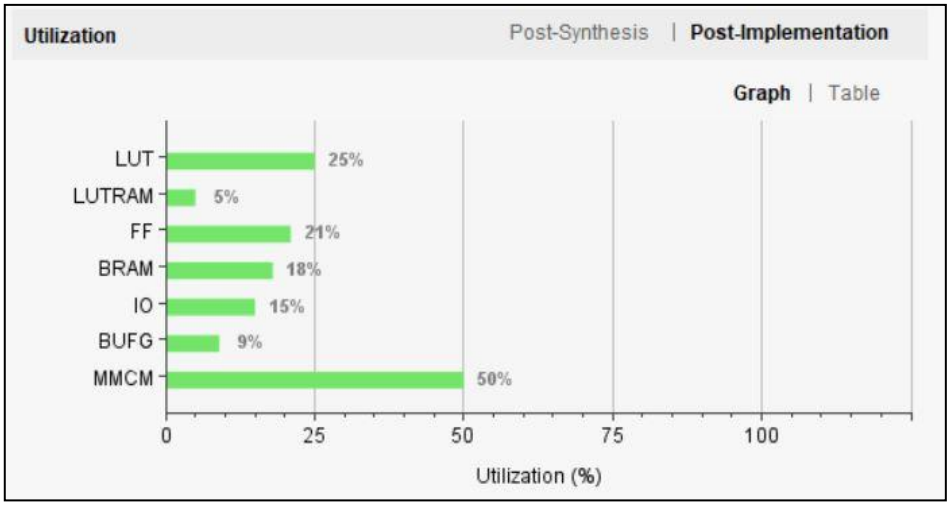
Resim 4.23 ve Resim 4.24'e bakıldığında aşındırma algoritması uygulanmış olan resmin içerisindeki dörtgenlerin arasında ki mesafenin aşınmadan dolayı bir miktar arttığı görülmüştür.

Tasarlanan FPGA üzerindeki gerçek zamanlı aşındırma algoritması tabanlı görüntü işleme uygulaması kullanılarak 1280x720 piksel formatındaki gerçek zamanlı görüntüler işlenebilmiştir. Zynq 7000 işlemcili Xilinx Zybo Z7-20 FPGA geliştirme

kartı üzerinde yapılan çalışmada, FPGA çipi Blok RAM %18 oranında, LUT %25 oranında, FF ise %21 oranında kullanılmıştır. Ayrıca tasarımın çalışması için 2.041 W güce ihtiyaç duyulmaktadır. Resim 4.25'te aşındırma algoritması tasarımının güç istatistikleri ve Resim 4.26'da aşındırma algoritması çip istatistikleri verilmiştir. Çip istatistiklerine bakıldığında hafızanın az bir miktarının kullanıldığı görülmektedir.



Resim 4.25 Aşındırma algoritması tasarımının güç istatistikleri.

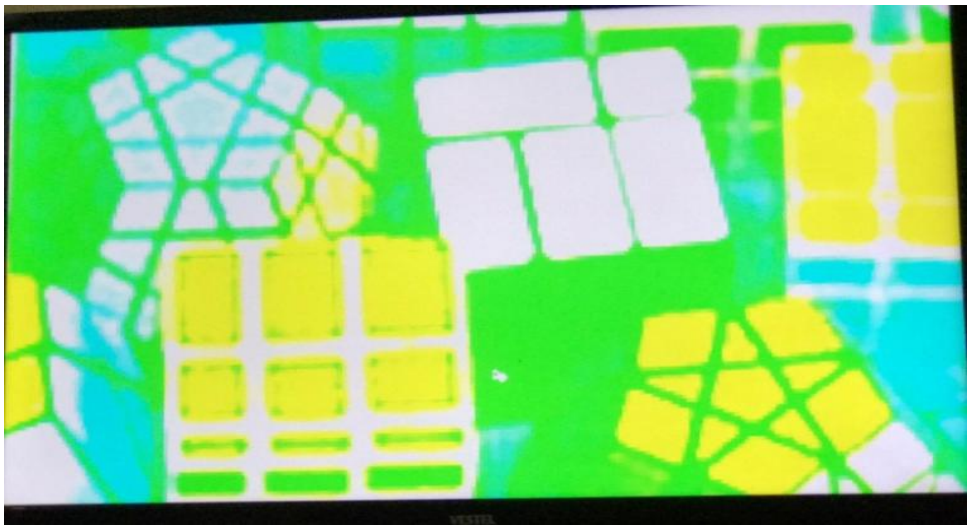


Resim 4.26 Aşındırma algoritması çip istatistikleri.

4.5 Yayma Yöntemi ile Morfolojik İşlemlerin Gerçek Zamanlı Uygulanması

Çalışmanın bu kısmında, morfolojik işlem uygulaması Yayma yöntemi ile görüntü işleme algoritması, FPGA çipi üzerinde donanımsal olarak tasarlanmıştır. Tasarlanan yayma görüntü işleme algoritmasının ünitesi donanım tanımlama dili olan VHDL ile kodlanmıştır. Tasarım sırasında kullanılan yayma algoritması Vivado HLS programı ile sentezlenmiş ve IP Core haline getirilmiştir. Tasarım platformu olarak Xilinx Vivado 2017.4 kullanılmış ve tasarlanan sistem Zybo Z7-20 FPGA çipi için sentezlenmiştir. Yayma algoritması için Open CV kütüphanesinden yararlanılmıştır. Sonuçlar Resim 4.27 ve Resim 4.28'de verilmiştir.

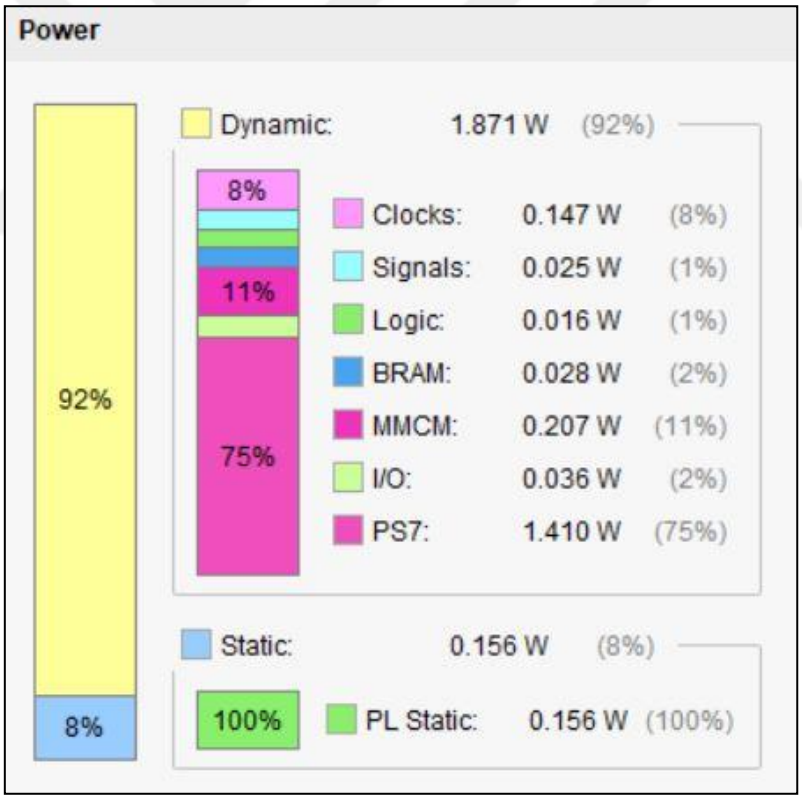
Resim 4.27 Yayma algoritması uygulanmamış rubik küpleri.



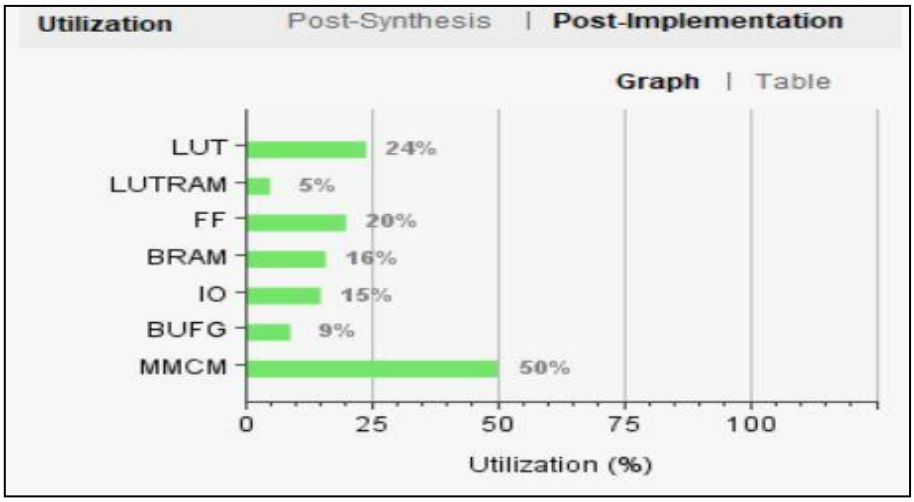
Resim 4.28 Yayma algoritması uygulanmış rubik küpleri.

Resim 4.27 ve Resim 4.28'e bakıldığında yayma algoritması uygulanmış olan resmin içerisindeki dörtgenlerin arasında ki mesafenin yayılmadan dolayı bir azaldığı görülmüştür.

Tasarlanan FPGA üzerindeki gerçek zamanlı yayma yöntemi ile görüntü işleme uygulaması kullanılarak 1280x720 piksel formatındaki gerçek zamanlı görüntüler işlenebilmiştir. Zynq 7000 işlemcili Xilinx Zybo Z7-20 FPGA geliştirme kartı üzerinde yapılan çalışmada, FPGA çipi Blok RAM %16 oranında, LUT %24 oranında, FF ise %20 oranında kullanılmıştır. Ayrıca tasarımın çalışması için 2.036 W güce ihtiyaç duyulmaktadır. Resim 4.29'da yayma algoritması tasarımının güç istatistikleri ve Resim 4.30'da yayma algoritması çip istatistikleri verilmiştir. Çip istatistiklerine bakıldığında hafızanın az bir miktarının kullanıldığı görülmektedir.



Resim 4.29 Yayma algoritması tasarımının güç istatistikleri**.**



Resim 4.30 Yayma algoritması çip istatistikleri**.**

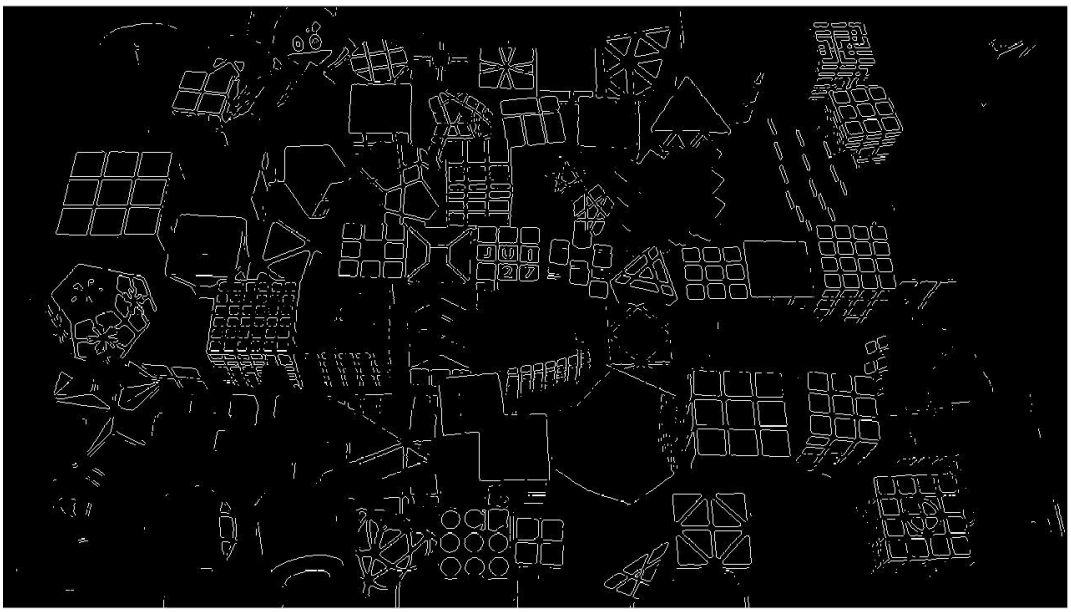
4.6 Sobel Görüntü İşleme Algoritmasının Gerçek Zamanlı Uygulanması

Çalışmanın bu kısmında, kenar belirleme uygulaması olan Sobel Görüntü İşleme Algoritması, FPGA çipi üzerinde donanımsal olarak tasarlanmıştır. Tasarlanan sobel görüntü işleme algoritmasının ünitesi donanım tanımlama dili olan VHDL ile kodlanmıştır.

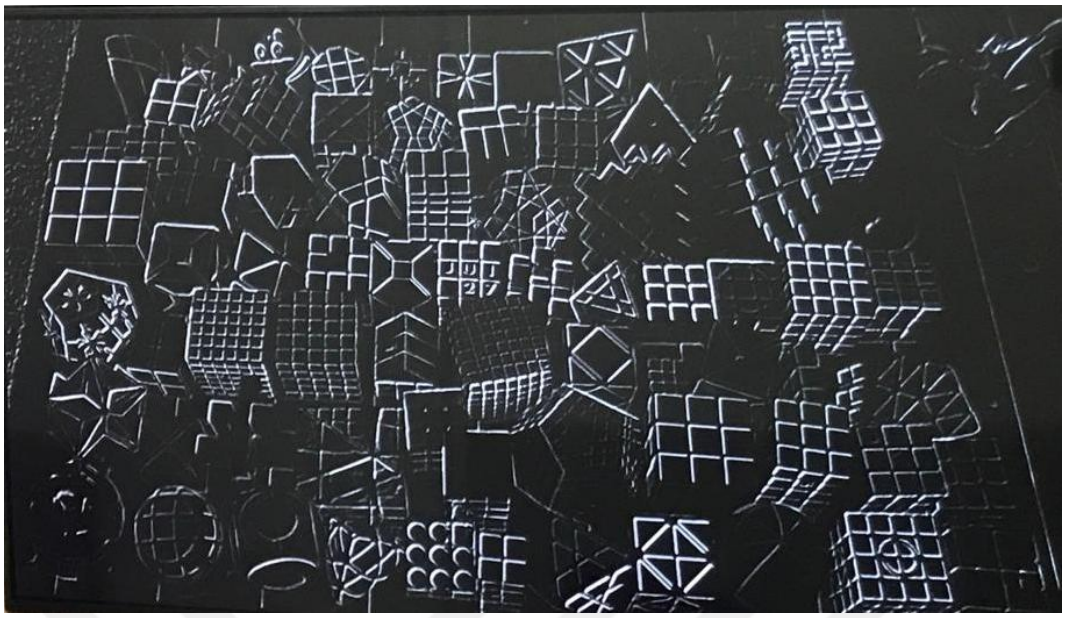
Tasarım sırasında kullanılan sobel görüntü işleme algoritması Vivado HLS programı ile sentezlenmiş ve IP Core haline getirilmiştir. Ayrıca Vivado SDK uygulamasından faydalanılmıştır. Tasarım platformu olarak Xilinx Vivado 2017.4 kullanılmış ve tasarlanan sistem Zybo Z7-20 FPGA çipi için sentezlenmiştir. Tasarımdan elde edilen görüntüler

Resim 4.31'de herhangi bir kenar bulma algoritması uygulanmamış rubik küpleri olarak, Resim 4.32'de Matlab üzerinde sobel kenar algoritması uygulanmış görüntü ve Resim 4.33'te FPGA üzerinde sobel kenar bulma algoritması uygulanmış görüntü verilmiştir ve tartışılmıştır.

Resim 4.31 Herhangi bir kenar bulma algoritması uygulanmamış rubik küpleri.



Resim 4.32 Matlab uygulaması üzerinde sobel algoritması uygulanmış rubik küpleri.

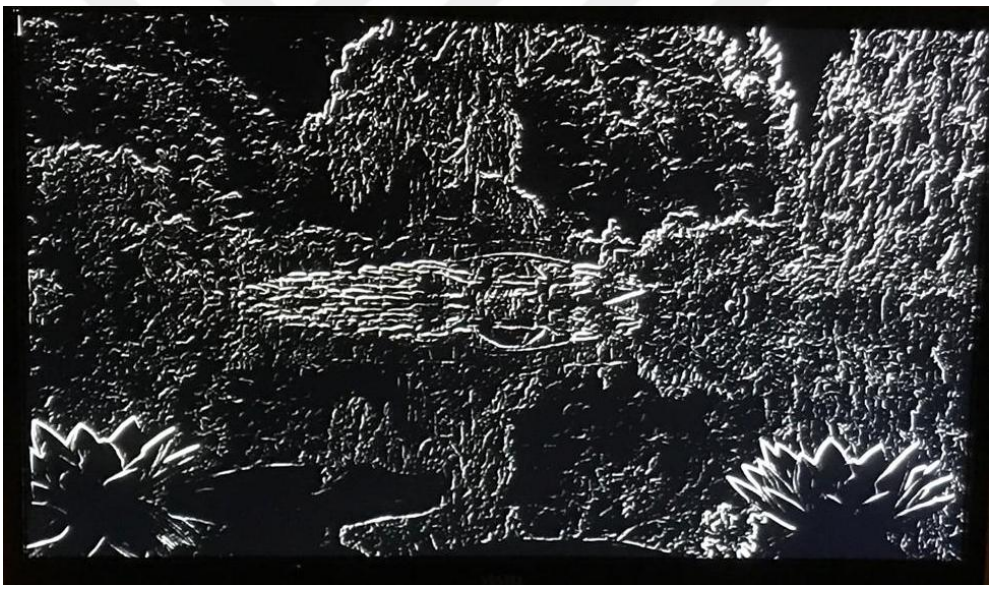


Resim 4.33 FPGA üzerinde Sobel kenar bulma algoritması uygulanmış rubik küpleri.

Resim 4.34'te herhangi bir kenar bulma algoritması uygulanmamış doğa görüntüsü verilmiştir, Resim 4.35'te Matlab uygulaması üzerinde sobel algoritması uygulanmış ve 4.36'da FPGA üzerinde sobel kenar bulma algoritması doğa görüntüsüne uygulanmış ve sonuçları karşılaştırılmıştır.

Resim 4.34 Herhangi bir kenar bulma algoritması uygulanmamış doğa görüntüsü.

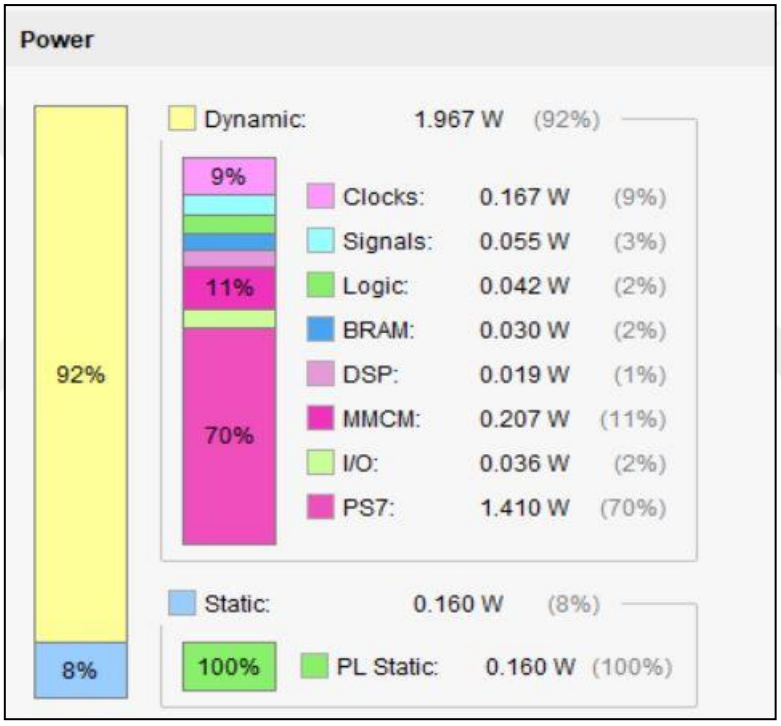
Resim 4.35 Matlab uygulaması üzerinde sobel algoritması uygulanmış rubik küpleri.



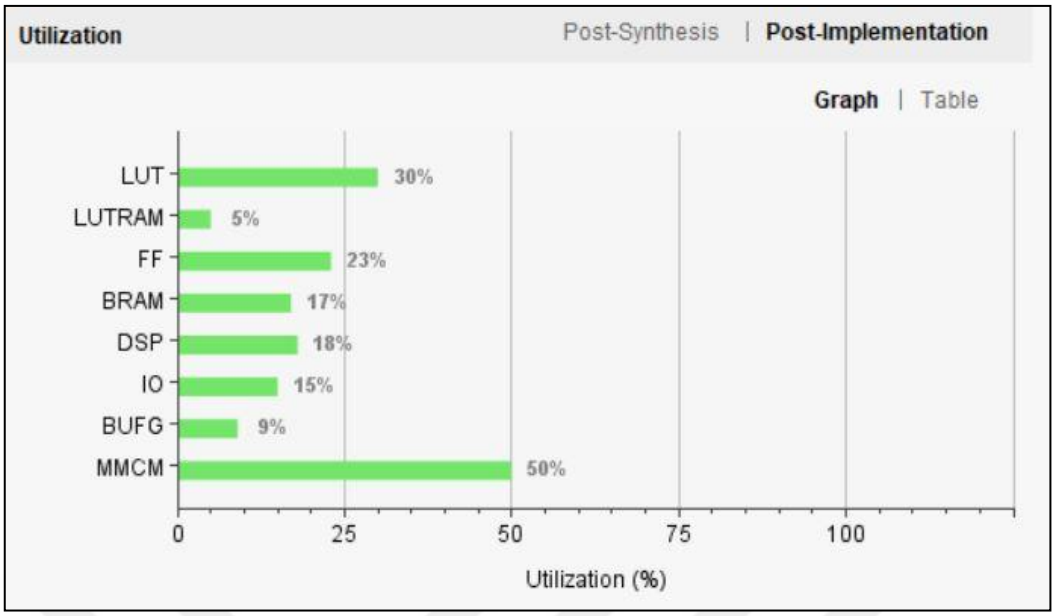
Resim 4.36 FPGA üzerinde Sobel kenar bulma algoritması uygulanmış doğa görüntüsü.

Resim 4.35 ve Resim 4.36 kıyaslandığında FPGA üzerinde işlenmiş olan görüntüde ki kenarların daha belirgin ve daha canlı olduğu görülmektedir. Matlab uygulamasında bulunan kenarların çokça keskin ve daha düzensiz olduğu fark edilmektedir. Bu durum ise görselin içerdiği cisimlerin anlaşılırlığını azaltmıştır. FPGA-tabanlı gerçek zamanlı Sobel Kenar Bulma Algoritması ile gerçekleştirilen görüntü işleme uygulaması için Vivado Design Suite ve Vivado HLS programları kullanılmıştır. Sobel Kenar Bulma Algoritması için Open CV kütüphanesinden yararlanılmıştır. Tasarlanan FPGA

üzerindeki gerçek zamanlı Sobel Kenar Bulma Algoritması tabanlı görüntü işleme uygulaması kullanılarak 1280x720 piksel formatındaki gerçek zamanlı görüntüler işlenebilmiştir. Zynq 7000 işlemcili Xilinx Zybo Z7-20 FPGA geliştirme kartı üzerinde yapılan çalışmada, FPGA çipi BRAM %17 oranında, LUT %30 oranında, FF ise %23 oranında kullanılmıştır. Ayrıca tasarımın çalışması için 2.127 W güce ihtiyaç duyulmaktadır. Sonuçlar Resim 4.37, Resim 4.38'de verilmiştir. Resim 4.37'de Sobel Kenar Bulma Algoritması tasarımının güç istatistikleri ve Resim 4.38'de Sobel Kenar Bulma Algoritması çip istatistikleri verilmiştir. Çip istatistiklerine bakıldığında hafızanın az bir miktarının kullanıldığı görülmektedir.



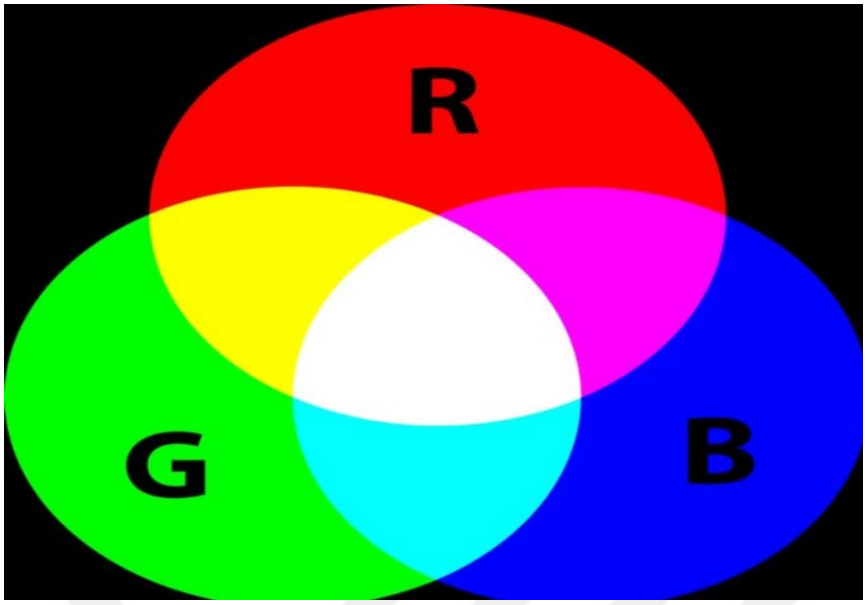
Resim 4.37 Sobel Kenar Algılama algoritması tasarımının güç istatistikleri.



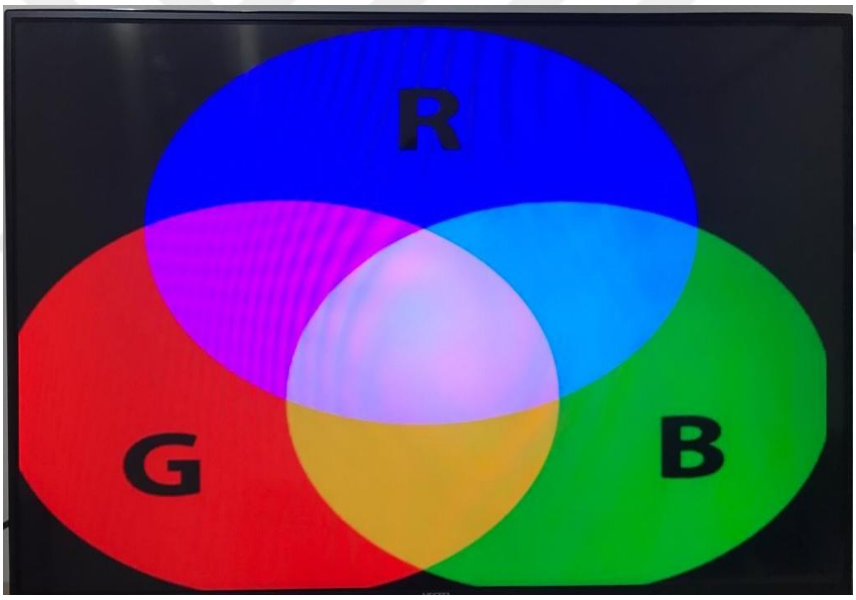
Resim 4.38 Sobel Kenar Algılama algoritmasının çip istatistikleri.

4.7 Renk Değiştirme Yöntemi ile Görüntü İşleme Algoritmasının Gerçek Zamanlı Uygulanması

Çalışmanın bu kısmın da, Renk Değiştirme Yöntemi algoritması, FPGA çipi üzerinde donanımsal olarak tasarlanmıştır. Tasarlanan Renk Değiştirme Yöntemi algoritması ünitesi donanım tanımlama dili olan VHDL ile kodlanmıştır. Tasarım sırasında kullanılan Renk Değiştirme Yöntemi Algoritması Vivado HLS programı ile sentezlenmiş ve IP Core haline getirilmiştir. Tasarım platformu olarak Xilinx Vivado 2017.4 kullanılmış ve tasarlanan sistem Zybo Z7-20 FPGA çipi için sentezlenmiştir. Tasarımdan elde edilen sonuçlar Resim 4.39'da üzerinde renk değiştirme yöntemi uygulanmamış rgb görseli ve Resim 4.40'da üzerinde renk değiştirme yöntemi uygulanmış rgb görseli olarak verilmiştir. Resim 4.41'de üzerinde renk değiştirme yöntemi uygulanmamış lale tarlası görseli ve Resim 4.42'de üzerinde renk değiştirme yöntemi uygulanmış lale tarlası görseli olarak verilmiş ve sonuçlar tartışılmıştır.



Resim 4.39 Üzerinde renk değiştirme yöntemi uygulanmamış rgb görseli.



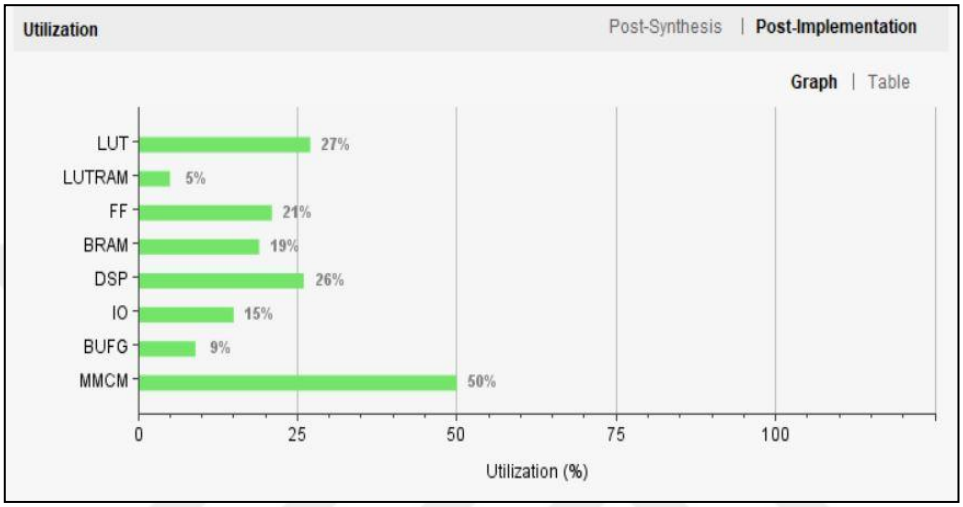
Resim 4.40 Üzerinde renk değiştirme yöntemi uygulanmış rgb görseli.

Resim 4.41 Üzerinde renk değiştirme yöntemi uygulanmamış lale tarlası.

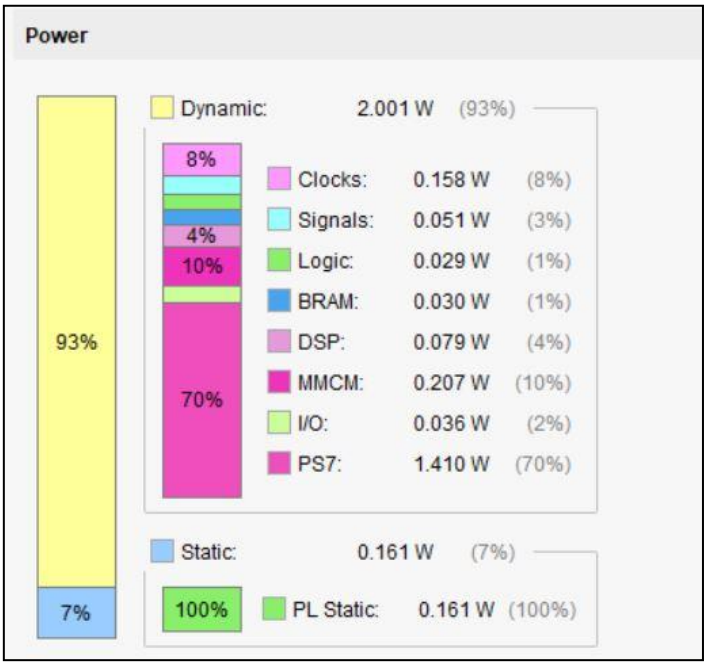
Resim 4.42 Üzerinde renk değiştirme yöntemi uygulanmış lale tarlası.

FPGA-tabanlı gerçek zamanlı Renk Değiştirme Yöntemi ile Görüntü İşleme Algoritması ile gerçekleştirilen görüntü işleme uygulaması için Vivado Design Suite ve Vivado High Level Synthesis programları kullanılmıştır. Değiştirme yöntemi ile görüntü işleme algoritması için Open CV kütüphanesinden yararlanılmıştır. Tasarlanan FPGA üzerindeki gerçek zamanlı değiştirme yöntemi ile görüntü işleme algoritması tabanlı görüntü işleme uygulaması kullanılarak 1280x720 piksel formatındaki gerçek zamanlı görüntüler işlenebilmiştir. Zynq 7000 işlemcili Xilinx Zybo Z7-20 FPGA geliştirme kartı üzerinde yapılan çalışmada, FPGA çipi BRAM %19, FF %21, LUT

%27 oranında kullanılmıştır. Ayrıca tasarımın çalışması için 2.163 W güce ihtiyaç duyulmaktadır. Resim 4.43 Renk değiştirme yöntemi ile görüntü işleme algoritmasının çip istatistikleri ve Resim 4.44'te renk değiştirme yöntemi ile görüntü işleme algoritmasının güç istatistikleri verilmiştir. Çip istatistiklerine bakıldığında hafızanın az bir miktarının kullanıldığı görülmektedir.



Resim 4.43 Renk değiştirme yöntemi ile görüntü işleme algoritmasının çip istatistikleri.



Resim 4.44 Renk değiştirme yöntemi ile görüntü işleme algoritmasının güç istatistikleri.

5.TARTIŞMA VE SONUÇ

Sunulan bu tez çalışmasında, FAST ve Harris köşe algılama algoritmaları, Sobel kenar bulma algoritması, morfolojik işlem algoritmalarından yayma ile aşındırma yöntemleri ve renk değiştirme algoritması FPGA çipleri üzerinde çalışmak üzere gerçek zamanlı olarak tasarlanmıştır.

FPGA çiplerinde çalışmak üzere tasarlanan gerçek zamanlı görüntü işleme algoritmalarında Vivado Design Suite, Vivado High Level Synthesis, Vivado SDK programları kullanılmıştır. Tasarımlar için Open CV kütüphanesinden faydalanılmıştır. Gerçek zamanlı görüntüler HDMI aracığıyla kameradan alınmıştır. HDMI aracığıyla aktarılan görüntü boyutları 1280x720 piksel formatındadır. Alınan görüntü verilerinin FPGA üzerinde işlenebilmesi için VHDL dili kullanılmıştır.

Yapılan tüm tasarımlar Xilinx Zybo Z7-20 geliştirme kartı üzerinde gerçekleştirilmiştir. FPGA-tabanlı tasarımların sonuçlarından elde edilen görüntü verileri HDMI aracılığıyla monitöre aktarılmıştır.

Tez çalışmasında ilk aşamada, gerçek zamanlı görüntü kamera aracılığı ile alınarak Xilinx Zybo Z7-20 geliştirme kartına aktarılmıştır. Burada gerçek zamanlı görüntü üzerinde herhangi bir işlem yapılmadan HDMI aracılığıyla görüntü verileri mönitörde görüntülenmiştir. Zynq 7000 işlemcili Zybo Z7-20 FPGA geliştirme kartı üzerinde yapılan çalışmada BRAM %10, FF %17 ve LUT %20 oranında çip kaynağı kullanılmıştır. Ayrıca tasarımın çalışması için 1.985 W güce ihtiyaç duyulmaktadır.

Çalışmanın ikinci aşamasında, kamera aracılığıyla alınan 1280x720 piksel formatındaki gerçek zamanlı görüntüler üzerinde FPGA-tabanlı FAST köşe algılama algoritması uygulaması gerçekleştirilmiştir. Zynq 7000 işlemcili Zybo Z7-20 FPGA geliştirme kartı üzerinde yapılan çalışmada BRAM %30, FF %23, LUT %31 oranında çip kaynağı kullanılmıştır. Tasarımın çalışması için 2.145 W güce ihtiyaç duyulmaktadır. Bu algoritma Vivado HLS programı ile IPCore oluşturularak elde edilmiştir. FAST köşe algılama algoritması sonuçları, Matlab programının sonuçlarıyla karşılaştırılmıştır.

FAST köşe algılama algoritmasının düşük ışıkta ve yüksek ışıkta bulduğu köşeler Matlab programının düşük ışıkta ve yüksek ışıkta bulduğu köşelerden daha fazla sayıda ve daha kesin olmaktadır.

Çalışmanın üçüncü aşamasında, kamera aracılığıyla alınan 1280x720 piksel formatındaki gerçek zamanlı görüntüler üzerinde FPGA-tabanlı Harris köşe algılama algoritması uygulaması gerçekleştirilmiştir. Zynq 7000 işlemcili Zybo Z7-20 FPGA geliştirme kartı üzerinde yapılan çalışmada, BRAM %48 oranında, LUT %71 oranında, FF ise %42 oranında çip kaynağı kullanılmıştır. Tasarımda 2.651 W güce ihtiyaç duyulmaktadır. FPGA-tabanlı Harris köşe algılama algoritması Vivado HLS programı ile IPCore yapısına dönüştürülmüştür. Harris köşe algılama algoritmasının küplerin üzerinde bulduğu köşeler Matlab uygulamasının bulduğu köşelerden daha fazla olmaktadır. Bu durum aynı görüntü üzerindeki ışık değişimlerinde daha belirgin olmakta ve Harris köşe bulma algoritması daha iyi sonuç vermektedir. Bu çalışma değerlendirilirken Matlab uygulamasının köşe işaretçisinin Harris köşe bulma algortimasının köşe işaretçisinden daha büyük olması sebebiyle göz yanılgısına sebep olabilmektedir. Bu durum göz ardı edilmemelidir. Fakat FPGA-tabanlı Harris köşe algılama algoritmasının sonuçlarına bakıldığında bazı köşe olmayan noktaları işaretlediği görülmektedir. Bu durum yapılan tasarım için bir dezavantaj teşkil etmektedir. İleriki çalışmalarda FPGA-tabanlı Harris köşe algılama algoritması geliştirilebilir.

Çalışmanın dördüncü aşamasında, kamera aracılığıyla alınan 1280x720 piksel formatındaki gerçek zamanlı görüntüler üzerinde FPGA-tabanlı aşındırma algoritması uygulaması gerçekleştirilmiştir. Zynq 7000 işlemcili Zybo Z7-20 FPGA geliştirme kartı üzerinde yapılan çalışmada, FPGA çipi Blok RAM %18 oranında, LUT %25 oranında, FF ise %21 oranında kullanılmıştır. Tasarlanan bu algoritmanın çalışması için 2.041 W güce ihtiyaç duyulmaktadır. Sonuçlarda görüldüğü gibi aşındırma işlemi başarıyla uygulanmıştır.

Çalışmanın beşinci aşamasında, Zynq 7000 işlemcili Zybo Z7-20 FPGA geliştirme kartı üzerinde kameradan alınan 1280x720 piksel formatındaki gerçek zamanlı görüntüler

için FPGA-tabanlı yayma algoritması uygulaması başarılı bir şekilde gerçekleştirilmiştir. FPGA çipi BRAM %16 oranında, LUT %24 oranında, FF ise %20 oranında kullanılmıştır. Tasarım 2.027 W güce ihtiyaç duymaktadır.

Çalışmanın altıncı aşamasında, Zynq 7000 işlemcili Zybo Z7-20 FPGA geliştirme kartında, kameradan alınan 1280x720 piksel formatındaki gerçek zamanlı görüntüler kullanılarak FPGA-tabanlı sobel kenar bulma algoritması uygulaması başarılı bir şekilde gerçekleştirilmiştir. Yapılan çalışmada, tasarım tarafından 2.127 W güce ihtiyaç duyulmaktadır. FPGA çipi BRAM %17 oranında, LUT %30 oranında, FF ise %23 oranında kullanılmıştır. Matlab uygulamasının sonuçları ile kıyaslandığında FPGA-tabanlı sobel kenar bulma algoritması kenarları daha kesin ve doğru biçimde göstermektedir. Ayrıca Matlab uygulamasının sonuçlarında kenarlar bir miktar tahribata uğramıştır. FPGA-tabanlı sobel kenar bulma algoritması görüntüde olan kenarları doğru ve tahribata uğratmadan tespit etmiştir. Bu algoritma literatürde uygulanan diğer algoritmalardan daha fazla piksel yoğunluğu işleyebilmektedir. Piksel yoğunluğu baz alındığında daha az BRAM kullanımı olduğu görülmektedir.

Çalışmanın yedinci aşamasında, kameradan alınan 1280x720 piksel formatındaki gerçek zamanlı görüntüler üzerinde FPGA-tabanlı renk değiştirme algoritması uygulaması gerçekleştirilmiştir. Zynq 7000 işlemcili Xilinx Zybo Z7-20 FPGA geliştirme kartı üzerinde yapılan çalışmada, FPGA çipi BRAM %19 oranında, LUT %27 oranında, FF ise %21 oranında kullanılmıştır. Ayrıca tasarımın çalışması için 2.163 W güce ihtiyaç duyulmaktadır. Sonuçlarda görüldüğü gibi değiştirme algoritması başarıyla uygulanmıştır.

İleriki çalışmalarda HDMI kamera ve PCAM 5C olmak üzere iki kamera kullanılarak elde edilecek görüntüler ile mesafesi tahmini uygulamaları gerçekleştirilebilir. Ayrıca bu çalışmada kullanılan algoritmaların daha hızlı ve daha hassas sonuçlar üretebilmesini sağlayacak farklı yeni algoritmalar geliştirilebilir.

6. KAYNAKLAR

Altuncu M A, 2015, Temel Görüntü İşleme Algoritmalarının Gerçek Zamanlı Olarak FPGA ile Gerçeklenmesi, Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı, Yüksek Lisans Tezi, 46s, Kocaeli.

Altuntaş C, Çorumluoğlu Ö, 2011, Filtreleme Yöntemi İle Digital Görüntü Zenginleştirme ve Örnek Bir Yazılım, Teknik-Online Dergi, 10, 99-107.

Anusha G, Prasad T J, Narayana D S, 2012, Implementation of SOBEL Edge Detection on FPGA, International Journal of Computer Trends and Technology, 3, 472–475.

Aybar E, 2008, Sobel İşleci Kullanılarak Renkli Görüntülerde Kenar Bulma, Afyon Kocatepe Üniversitesi Fen Bilimleri Dergisi, 8, 205-217.

Aydoğdu M F, Demirci M F, Kasnakoğlu C, 2013, Pipelining Harris Corner Detection with a Tiny FPGA for a Mobile Robot, Proceeding of the IEEE International Conference on Robotics and Biomimetics, 15 Eylül, China, 978-1-4799-2744-9.

Barakat M L, Mansingka A S, Radwan A G, Salama K N, 2014, Hardware Stream Cipher with Controllable Chaos Generator for Colour Image Encryption, Image Processing, 8, 33–43.

Canny J F, 1986, A Computational Approach to Edge Dedection, IEEE Transactions on Pattern Analysis and Machine Intelligence, 6, 679–698.

Chris H, Mike S, 1998, A Combined Corner and Edge Detector, Alvey Vision Conference, Eylül, United Kingdom, 23.1-23.6.

Chung C K, Wang W J, 2006, A Novel Edge Detection Metod on The Maximizing Objective Function, Pattern Recognition, 40, 609–618.

Crockett L H, Elliot R A, Enderwitz M A, Stewart R W, 2014, The Zynq Book, Strathclyde Academic Media, 484s, Glasgow, Scotland, UK.

Cuevas E, Zaldıvar D, Pérez-Cısneros M, Sánchez E, Ramírez-Ortegón M, 2011, Robust Fuzzy Corner Detector, Intelligent Automation and Soft Computing, 17, 415-429.

Çelik A R, 2013, Görüntü İşleme Algoritmalarının FPGA Donanımı Üzerinde Gerçeklenmesi, Kahramanmaraş Sütçü İmam Üniversitesi, Fen Bilimleri Enstitüsü Elektrik-Elektronik Mühendisliği Anabilim Dalı, Yüksek Lisans Tezi, 65s, Kahramanmaraş.

Çil M M, 2015, Temel Görüntü İşleme Algoritmalarının FPGA Üzerinde Gerçeklenmesi, İstanbul Teknik Üniversitesi Fen Bilimleri Enstitüsü Elektronik ve Haberleşme Mühendisliği Anabilim Dalı, Yüksek Lisans Tezi, 65s, İstanbul.

Demirci R, Okur Ü, 2019, Renkli Görüntülerin Ortalama Tabanlı Çok Seviyeli Eşiklenmesi, Düzce Üniversitesi Bilim ve Teknoloji Dergisi, 7, 664-676.

Dönmez A, 2019, Havacılık Uygulamaları İçin Kısmi Dinamik Donanım Şekillendirme Ve Üçlü Modül Yedekleme Kullanılarak Hata Toleranslı Fpga Tasarımı, Hacettepe Üniversitesi, Fen Bilimleri Enstitüsü, Yüksek Lisans Tezi, 124s, Ankara.

Eldem A, Eldem H, Palalı A, 2017, Görüntü İşleme Teknikleriyle Yüz Algılama Sistemi Geliştirme, Bitlis Eren Üniversitesi Fen Bilimleri Dergisi, 6, 44-48.

Erdoğan K, 2013, Görüntü İşleme Uygulamaları İçin FPGA Geliştirme Kartı Tasarımı ve Gerçekleştirilmesi, Selçuk Üniversitesi Fen Bilimleri Enstitüsü Elektrik-Elektronik Mühendisliği Anabilim Dalı, Yüksek Lisans Tezi, 77s, Konya.

Gacar A, 2009, FPGA Tabanlı Görüntü Işleme Arabirimi, Selçuk Üniversitesi Fen bilimleri Fakültesi Elektrik-Elektronik Mühendisliği Anabilim Dalı, Yüksek Lisans Tezi, 76s, İzmir.

Garipcan A M, Erdem E, Tuncer T, 2017, Donanım Tabanlı Trivium Akış Şifreleme Algoritmasının FPGA Ortamında Gerçekleştirilmesi. Fırat Üniversitesi Mühendislik Bilimleri Dergisi, 29, 119-130.

Gonzales R C, Woods R E, 2002, Digital Image Processing, Prentice-Hall, 722, New Jersey.

Hao F, Guoping Q, Jie S, Ilyas M, 2014, A Novel Polar Space Random Field Model for the Detection of Glandular Structures, IEEE Transactions on Medical Imaging, 33, 764–776.

Karaköse M, Baygın M, Aydın İ, Sarımaden A, Akın E, 2016, Endüstriyel Sistemlerde Arkaplan Çıkarımı Tabanlı Hareketli Nesne Tespiti ve Sayılması için Yeni Bir Yaklaşım, MSU Fen Bilimleri Dergisi, 4, 373-381.

Kızılkaya R, 2012, Implementation Of Image Processing Algorithms On Fpga Demonstration Board, Dokuz Eylül University Graduate School of Naturel and Applied Sciences, Yüksek Lisans Tezi, 69s, İzmir.

Koray C, Sümer E, 2019, Satranç Oyunu İçin Bilgisayarla Görme Tabanlı Hamle Algılama ve Yorumlama Sistemi, Uludağ University Journal of The Faculty of Engineering, 24, 299-316.

Koyuncu I, Çetin Ö, Katırcıoğlu F, Tuna M, 2015, Edge Dedection Application With FPGA Based Sobel Operator, 978-1-4799-4874-1/14

Koyuncu I, Özcerit A T, Pehlivan I, 2014, Implementation of FPGA-based Real Time Novel Chaotic Oscillator, Nonlinear Dynamics, 75, 1–2, 49–59.

Kumar S, Prabat P, 2013, FPGA Implementation of Image Segmentation By Using Edge Detection Based On Sobel Edge Operator, International Journal of Research in Engineering and Technology, 2, 198–203.

Li L, Gong M, Chui Y H, Schneider M, 2014, A MATLAB-based Image Processing Algorithm For Analyzing Cupping Profiles of Two-Layer Laminated Wood Products, Measurement, 53, 234 – 239.

Lu W, Lifan Z, Guoan B, Chunru W, Lei Y, 2013, Enhanced ISAR Imaging by Exploiting the Continuity of the Target Scene, IEEE Transactions on Geoscience and Remote Sensing, 52, 5736 - 5750.

Mehra R, Rupinder V, 2012, Area Efficient FPGA Implementation of Sobel Edge Detector for Image Processing Applications, International Journal of Computer Applications, 5, 7–11.

Methore R, Nichani S, Ranganathan N, 1990, Corner Detection, Pattern Recognition, 23, 1223-1233.

Osmanoğlu U Ö, Mutlu F, Gürsoy H, Şanlısoy S, 2016, Görüntü İşleme ve Analizinin Tıpta Kullanımı ve Bir Uygulama, Osmangazi Tıp Dergisi, 41, 6-16.

Özalp R, 2018, Çip Üzerinde Sistem Mimarili FPGA Kullanarak Gerçek Zamanlı Görüntü İşleme Algoritmalarının Gerçekleştirilmesi, Fırat Üniversitesi Fen Bilimleri Enstitüsü Mekatronik Mühendisliği Anabilim Dalı, Yüksek Lisans Tezi, 82s, Elazığ.

Özbay B, 2017, Viterbi Kod Çözücünün Güç Etkin Mimari Tasarımı ve FPGA Gerçeklemesi, Maltepe Üniversitesi, Fen Bilimleri Enstitüsü, Yüksek Lisans Tezi, 74s, İstanbul.

Özçelik M F, 2012, Görüntü İşleme Algoritmalarının FPGA Üzerinde Gerçeklenmesi, Gazi Üniversitesi Bilişim Enstitüsü Bilgisayar Bilimleri Anabilim Dalı, Yüksek Lisans Tezi, 63s, Ankara.

Özkan H, 2012, Hayvansal Üretim Endüstrisinde Görüntü İşleme Tabanlı Gerçek Zamanlı Bir Kalite Kontrol Uygulaması, Selçuk Üniversitesi, Fen Bilimleri Enstitüsü, Yüksek Lisans Tezi, 87s, Konya.

Prakash J, Dehghani H, Pogue B W, Yalavarthy P K, 2014, Model-Resolution-Based Basis Pursuit Deconvolution Improves Diffuse Optical Tomographic Imaging, IEEE Transactions on Medical Imaging, 33, 891–901.

Rana K B, Agrawal G D, Mathur J, Puli U, 2014, Measurement of Void Fraction İn Flow Boiling of Zno Water Nanofluids Using Image Processing Technique, Nuclear Engineering and Design, 270, 217–226.

Rosten E, Drummond T, 2006, Machine Learning for High-Speed Corner Detection, European Conference on Computer Vision, United Kingdom, 430-443.

Samtaş G, Gülesin M, 2011, Sayısal Görüntü İşleme ve Farklı Alanlardaki Uygulamaları, Electronic Journal of Vocational Colleges, 2, 85-97.

Sánchez-Solano S, Cabrera A J, Baturone I, Moreno-Velo F J, Brox M, 2007, FPGA İmplementation Of Embedded Fuzzy Controllers For Robotic Applications, IEEE Transactions on Industrial Electronics, 54, 1937-1945.

Sharma S, Chen W, 2009, Using model-based design to accelerate FPGA Development For Automotive Applications, SAE International Journal of Passenger Cars-Electronic and Electrical Systems, 2, 150-158.

Tombul H, Kavzoğlu T, 2018, Nesne Tabanlı Görüntü Analizinde Görüntü Bölütleme Yaklaşımları ve Bölütleme Kalitesinin Analizi, Harita Dergisi, 160, 12-23.

Xiao Y, Dong S, 2015, Multilevel-Based Topology Design and Cell Patterning With Robotically Controlled Optical Tweezers, IEEE Transactions on Control System Technology, 23, 176–185.

Yıldız G, Yıldız D, 2018, Morfolojik İşlemler ve Kenar Algılama Yöntemler Vasıtasıyla Beyin Tümör Yeri Tespiti ve Tümör Alan Hesabının Yapılması, International Journal of Multidisciplinary Studies and Innovative Technologies, 2, 39-42.

Yiğitbaşı E D, 2012, Yapay Arı Kolonisi Optimizasyonu ile Kenar Bulma, Selçuk Üniversitesi, Fen Bilimleri Enstitüsü, Yüksek Lisans Tezi, 96s, Konya.

Yuan-Hui, Y, Chang C C, 2006, A New Edge Detection Approach Based On İmage Context Analysis, Image And Vision Computing, 24, 1090–1102.

İnternet Kaynakları

1) https://tr.wikipedia.org/wiki , 25.05.2020

2) https://www.linkedin.com/in/ecem-çakmak/, 25.05.2020

3) http://udentify.co/, 20.05.2020

4) https://guraysonugur.aku.edu.tr/category/goruntu-isleme/, 14.04.2020

5) www.Matlab.com, 15.04.2020

6) www.Matlab.com, 21.03.2020

7) http://filebox.vt.edu/users/tmagin/history.htm, 03.01.2020

8) http://www.eecg.toronto.edu/~vaughn/challenge/fpga_arch.html, 22.02.2020

9) http://fpganedir.com/FPGA/fpga_yapisi.php, 14.02.2020

10) https://www.xilinx.com/publications/archives/xcell/Xcell79.pdf, 16.05.2020

11) www.reference.digilentinc.com, 18.05.2020

12) http://zedboard.org/product/zedboard, 19.05.2020

13) https://www.eejournal.com/article/20141118-datacenter/, 23.05.2020

14) https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html, 29.05.2020

15) www.gelecegiyazanlar.turkcell.com.tr, 23.04.2020

ÖZGEÇMİŞ

Adı Soyadı : Muhammed Furkan TAŞDEMİR

Doğum Yeri ve Tarihi : KONYA/22.03.1996

Yabancı Dili : İngilizce

İletişim (E-posta) : furkantasdemir123@gmail.com

Eğitim Durumu (Kurum ve Yıl)

Lise	:	Ali Akkanat Anadolu Lisesi , (2010-2014)
Lisans	:	Afyon Kocatepe Üniversitesi, Mekatronik Mühendisliği Bölümü, (2014-2018)
Yüksek Lisans	:	Afyon Kocatepe Üniversitesi, Fen Bilimleri Enstitüsü, Elektrik Elektronik Mühendisliği Anabilim Dalı, (2018-2020)

Projeler ve Akademik Çalışmalar:

Gerçek Zamanlı Görüntünün Fpga Üzerinde İşlenerek Ethernet Üzerinden Veri Aktarımının Gerçeklenmesi, Proje No: 19.FEN.BİL.15, Afyon Kocatepe Üniversitesi BAP Projesi, Araştırmacı, 2019.

Harris Köşe algılama Algoritmasının Fpga Tabanlı Gerçekleştirilmesi, 3rd International Biltek Conference On Science, Technology & Current -Developments, syf 257-264.

Fpga Üzerinde Gerçek Zamanlı Hızlı Köşe algılama Algoritması Tabanlı Görüntü İşleme Uygulaması, 3. Uluslararası Asya Modern Bilimler Kongresi.

EKLER

EK 1 Görüntü Akışı Sağlayan Blok Dizaynların SDK Dinamik Saat C Kodu.

```
#include "dynclk.h"
#include "xil_io.h"
#include "math.h"
u32 ClkCountCalc(u32 divide)
{
u32 output = 0;
u32 divCalc = 0;

divCalc = ClkDivider(divide);
if (divCalc == ERR_CLKDIVIDER)
output = ERR_CLKCOUNTCALC;
else
output = (0xFFF & divCalc) | ((divCalc << 10) & 0x00C00000);
return output;
}

u32 ClkDivider(u32 divide)
{
u32 output = 0;
u32 highTime = 0;
u32 lowTime = 0;

if ((divide < 1) || (divide > 128))
return ERR_CLKDIVIDER;
if (divide == 1)
return 0x1041;

highTime = divide / 2;
```

EK 1 (Devam) Görüntü Akışı Sağlayan Blok Dizaynların SDK Dinamik Saat C Kodu.

```
if (divide & 0b1) //if divide is odd
{
lowTime = highTime + 1;
output = 1 << CLK_BIT_WEDGE;
}
else
{
lowTime = highTime;
}

output |= 0x03F & lowTime;
output |= 0xFC0 & (highTime << 6);
return output;
}

u32 ClkFindReg (ClkConfig *regValues, ClkMode *clkParams)
{
if ((clkParams->fbmult < 2) || clkParams->fbmult > 64 )
return 0;

regValues->clk0L = ClkCountCalc(clkParams->clkdiv);
if (regValues->clk0L == ERR_CLKCOUNTCALC)
return 0;

regValues->clkFBL = ClkCountCalc(clkParams->fbmult);
if (regValues->clkFBL == ERR_CLKCOUNTCALC)
return 0;

regValues->clkFBH_clk0H = 0;
```

EK 1 (Devam) Görüntü Akışı Sağlayan Blok Dizaynların SDK Dinamik Saat C Kodu.

```
regValues->divclk = ClkDivider(clkParams->maindiv);
if (regValues->divclk == ERR_CLKDIVIDER)

return 0

regValues->lockL = (u32) (lock_lookup[clkParams->fbmult - 1] & 0xFFFFFFFF);

regValues->fltr_lockH = (u32) ((lock_lookup[clkParams->fbmult - 1] >> 32) & 0x000000FF);
regValues->fltr_lockH |= ((filter_lookup_low[clkParams->fbmult - 1] << 16) & 0x03FF0000);

return 1;
}

void ClkWriteReg (ClkConfig *regValues, u32 dynClkAddr)
{
Xil_Out32(dynClkAddr + OFST_DYNCLK_CLK_L, regValues->clk0L);
Xil_Out32(dynClkAddr + OFST_DYNCLK_FB_L, regValues->clkFBL);
Xil_Out32(dynClkAddr + OFST_DYNCLK_FB_H_CLK_H, regValues->clkFBH_clk0H);
Xil_Out32(dynClkAddr + OFST_DYNCLK_DIV, regValues->divclk);
Xil_Out32(dynClkAddr + OFST_DYNCLK_LOCK_L, regValues->lockL);
Xil_Out32(dynClkAddr + OFST_DYNCLK_FLTR_LOCK_H, regValues->fltr_lockH);
}

double ClkFindParams(double freq, ClkMode *bestPick)
{
double bestError = 2000.0;
double curError;
```

EK 1 (Devam) Görüntü Akışı Sağlayan Blok Dizaynların SDK Dinamik Saat C Kodu.

```
double curClkMult;
double curFreq;
u32 curDiv, curFb, curClkDiv;
u32 minFb = 0;

u32 maxFb = 0;

freq = freq * 5.0;
bestPick->freq = 0.0;
for (curDiv = 1; curDiv <= 10; curDiv++)
{
minFb = curDiv * 6; //This accounts for the 100MHz input and the 600MHz minimum VCO
maxFb = curDiv * 12; //This accounts for the 100MHz input and the 1200MHz maximum VCO
if (maxFb > 64)
maxFb = 64;
curClkMult = (100.0 / (double) curDiv) / freq; //This multiplier is used to find the best clkDiv value for each FB value

curFb = minFb;
while (curFb <= maxFb)
{
curClkDiv = (u32) ((curClkMult * (double)curFb) + 0.5);
curFreq = ((100.0 / (double) curDiv) / (double) curClkDiv) * (double) curFb;
curError = fabs(curFreq - freq);
if (curError < bestError)
{
bestError = curError;
bestPick->clkdiv = curClkDiv;
```

EK 1 (Devam) Görüntü Akışı Sağlayan Blok Dizaynların SDK Dinamik Saat C Kodu.

```
bestPick->fbmult = curFb;
bestPick->maindiv = curDiv;
bestPick->freq = curFreq;

curFb++;
}

bestPick->freq = bestPick->freq / 5.0;
bestError = bestError / 5.0;
return bestError;
}

void ClkStart(u32 dynClkAddr)
{
Xil_Out32(dynClkAddr + OFST_DYNCLK_CTRL, (1 << BIT_DYNCLK_START));
while(!(Xil_In32(dynClkAddr + OFST_DYNCLK_STATUS) & (1 <<
BIT_DYNCLK_RUNNING)));

return;
}

void ClkStop(u32 dynClkAddr)
{
Xil_Out32(dynClkAddr + OFST_DYNCLK_CTRL, 0);
while((Xil_In32(dynClkAddr + OFST_DYNCLK_STATUS) & (1 <<
BIT_DYNCLK_RUNNING)));

return;
}
```

EK 2 Görüntü Akışı Sağlayan Blok Dizaynların SDK Dinamik Saat H Kodu.

```
typedef struct {
u32 clk0L;
u32 clkFBL;
u32 clkFBH_clk0H;
u32 divclk;
u32 lockL;
u32 fltr_lockH;
} ClkConfig;
typedef struct {
double freq;
u32 fbmult;
u32 clkdiv;
u32 maindiv;
} ClkMode;
static const u64 lock_lookup[64] = {
0b0011000110111110100011111010010000000001,
0b0011000110111110100011111010010000000001,
0b0100001000111110100011111010010000000001,
0b0101101011111110100011111010010000000001,
0b0111001110111110100011111010010000000001,
0b1000110001111110100011111010010000000001,
0b1001110011111110100011111010010000000001,
0b1011010110111110100011111010010000000001,
0b1100111001111110100011111010010000000001,
0b1110011100111110100011111010010000000001,
0b1111111111111000010011111010010000000001,
0b1111111111110011100111111010010000000001,
0b1111111111101110111011111010010000000001,
0b1111111111101011110011111010010000000001,
0b1111111111101000101011111010010000000001,
```

EK 2 (Devam) Görüntü Akışı Sağlayan Blok Dizaynların SDK Dinamik Saat H Kodu.

0b11111111111001110001111110100100000000001,
0b11111111111000111111111110100100000000001,
0b11111111111000100110111110100100000000001,
0b11111111111000001101111110100100000000001,
0b11111111110111110100111110100100000000001,
0b11111111110111011011111110100100000000001,
0b11111111110111000010111110100100000000001,
0b11111111110110101001111110100100000000001,
0b11111111110110010000111110100100000000001,
0b11111111110110010000111110100100000000001,
0b11111111110101110111111110100100000000001,
0b11111111110101011101111110100100000000001,
0b11111111110101011101111110100100000000001,
0b11111111110101000101111110100100000000001,
0b11111111110101000101111110100100000000001,
0b11111111110100101100111110100100000000001,
0b11111111110100101100111110100100000000001,
0b11111111110100101100111110100100000000001,
0b11111111110100010011111110100100000000001,
0b11111111110100010011111110100100000000001,
0b11111111110100010011111110100100000000001,
0b11111111110011111010111110100100000000001,
0b11111111110011111010111110100100000000001,
0b11111111110011111010111110100100000000001,
0b11111111110011111010111110100100000000001,
0b11111111110011111010111110100100000000001,
0b11111111110011111010111110100100000000001,
0b11111111110011111010111110100100000000001,
0b11111111110011111010111110100100000000001,
0b11111111110011111010111110100100000000001,

EK 2 (Devam) Görüntü Akışı Sağlayan Blok Dizaynların SDK Dinamik Saat H Kodu.

```
0b11111111110011111010111110100100000000001,
0b11111111110011111010111110100100000000001,
0b11111111110011111010111110100100000000001,
0b11111111110011111010111110100100000000001,
0b11111111110011111010111110100100000000001,
0b11111111110011111010111110100100000000001,
0b11111111110011111010111110100100000000001,
0b11111111110011111010111110100100000000001,
0b11111111110011111010111110100100000000001,
0b11111111110011111010111110100100000000001,
0b11111111110011111010111110100100000000001,
0b11111111110011111010111110100100000000001,
0b11111111110011111010111110100100000000001,
0b11111111110011111010111110100100000000001,
0b11111111110011111010111110100100000000001,
0b11111111110011111010111110100100000000001,
0b11111111110011111010111110100100000000001,
0b11111111110011111010111110100100000000001,
0b11111111110011111010111110100100000000001
};
static const u32 filter_lookup_low[64] = {
0b0001011111,
0b0001010111,
0b0001111011,
0b0001011011,
0b0001101011,
0b0001110011,
0b0001110011,
0b0001110011,
0b0001110011,
```

EK 2 (Devam) Görüntü Akışı Sağlayan Blok Dizaynların SDK Dinamik Saat H Kodu.

```
0b0001001011,
0b0001001011,
0b0001001011,
0b0010110011,
0b0001010011,
0b0001010011,
0b0001010011,
0b0001010011,
0b0001010011,
0b0001010011,
0b0001010011,
0b0001010011,
0b0001010011,
0b0001010011,
0b0001010011,
0b0001100011,
0b0001100011,
0b0001100011,
0b0001100011,
0b0001100011,
0b0001100011,
0b0001100011,
0b0001100011,
0b0001100011,
0b0001100011,
0b0001100011,
0b0001100011,
0b0001100011,
0b0001100011,
0b0001100011,
0b0010010011,
0b0010010011,
```

EK 2 (Devam) Görüntü Akışı Sağlayan Blok Dizaynların SDK Dinamik Saat H Kodu.

```
0b0010010011,
0b0010010011,
0b0010010011,
0b0010010011,
0b0010010011,
0b0010010011,
0b0010010011,
0b0010010011,
0b0010100011,
0b0010100011,
0b0010100011,
0b0010100011,
0b0010100011,
0b0010100011,
0b0010100011,
0b0010100011,
0b0010100011,
0b0010100011,
0b0010100011,
0b0010100011,
0b0010100011,
0b0010100011,
0b0010100011,
0b0010100011,
0b0010100011,
0b0010100011
};
u32 ClkCountCalc(u32 divide);
u32 ClkDivider(u32 divide);
u32 ClkFindReg (ClkConfig *regValues, ClkMode *clkParams);
void ClkWriteReg (ClkConfig *regValues, u32 dynClkAddr);
```

EK 3 Görüntü Akışı Sağlayan Blok Dizaynların Ana SDK Kodu.

```
double ClkFindParams(double freq, ClkMode *bestPick);
void ClkStart(u32 dynClkAddr);
void ClkStop(u32 dynClkAddr); #endif /* DYNCLK_H_ */
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "xparameters.h"
#include "xgpio.h"
#include "xvtc.h"
#include "xaxivdma.h"
#include "xaxivdma_i.h"
#include "vga_modes.h"
#include "dynclk.h"
#define DEMO_MAX_FRAME (720*1280)
#define DEMO_STRIDE (1280*3)
#define DISPLAY_NUM_FRAMES 3

XVtc
XVtc_Config *vtc_config,*vtc_config2 ;
XGpio hpd_in;
XAxiVdma vdma;
XAxiVdma_DmaSetup vdmaDMA;
XAxiVdma_Config *vdmaConfig;
ClkConfig clkReg;
ClkMode clkMode;
u32 frameBuf[DISPLAY_NUM_FRAMES][DEMO_MAX_FRAME];
u32 *pFrames[DISPLAY_NUM_FRAMES];

VideoMode video;
int main()
{
```

EK 3 (Devam) Görüntü Akışı Sağlayan Blok Dizaynların Ana SDK Kodu.

```
init_platform();
disable_caches();
XVtc_Timing vtcTiming;
XVtc_SourceSelect SourceSelect;
int Status;
u16 result;
xil_printf("Hello World\n\r");
vtc_config = XVtc_LookupConfig(XPAR_VTC_0_DEVICE_ID);
XVtc_CfgInitialize(&VtcInst, vtc_config, vtc_config->BaseAddress);
vtc_config2 = XVtc_LookupConfig(XPAR_VTC_1_DEVICE_ID);
XVtc_CfgInitialize(&VtcInst2, vtc_config2, vtc_config2->BaseAddress);

//configure and assert the HPD
XGpio_Initialize(&hpd_in, XPAR_AXI_GPIO_0_DEVICE_ID);
XGpio_DiscreteWrite(&hpd_in,1,0x1);
sleep(20);
XGpio_DiscreteWrite(&hpd_in,2,0x1); ///needs time here
video = VMODE_1280x720;
vtcTiming.HActiveVideo = video.width;
vtcTiming.HFrontPorch = video.hps - video.width;
vtcTiming.HSyncWidth = video.hpe - video.hps;
vtcTiming.HBackPorch = video.hmax - video.hpe + 1;
vtcTiming.HSyncPolarity = video.hpol;
vtcTiming.VActiveVideo = video.height;
vtcTiming.V0FrontPorch = video.vps - video.height;
vtcTiming.V0SyncWidth = video.vpe - video.vps;
vtcTiming.V0BackPorch = video.vmax - video.vpe + 1;;
vtcTiming.V1FrontPorch = video.vps - video.height;
vtcTiming.V1SyncWidth = video.vpe - video.vps;
vtcTiming.V1BackPorch = video.vmax - video.vpe + 1;;
```

EK 3 (Devam) Görüntü Akışı Sağlayan Blok Dizaynların Ana SDK Kodu.

```
vtcTiming.VSyncPolarity = video.vpol;
vtcTiming.Interlaced = 0;
memset((void *)&SourceSelect, 0, sizeof(SourceSelect));
SourceSelect.VBlankPolSrc = 1;
SourceSelect.VSyncPolSrc = 1;
SourceSelect.HBlankPolSrc = 1;
SourceSelect.HSyncPolSrc = 1;
SourceSelect.ActiveVideoPolSrc = 1;
SourceSelect.ActiveChromaPolSrc= 1;
SourceSelect.VChromaSrc = 1;
SourceSelect.VActiveSrc = 1;
SourceSelect.VBackPorchSrc = 1;
SourceSelect.VSyncSrc = 1;
SourceSelect.VFrontPorchSrc = 1;
SourceSelect.VTotalSrc = 1;
SourceSelect.HActiveSrc = 1;
SourceSelect.HBackPorchSrc = 1;
SourceSelect.HSyncSrc = 1;
SourceSelect.HFrontPorchSrc = 1;
SourceSelect.HTotalSrc = 1;
XVtc_RegUpdateEnable(&VtcInst2);
XVtc_SetGeneratorTiming(&VtcInst2, &vtcTiming);
XVtc_SetSource(&VtcInst2, &SourceSelect);
XVtc_EnableGenerator(&VtcInst2);
XVtc_Enable(&VtcInst2);
XVtc_EnableDetector(&VtcInst);
XVtc_Enable(&VtcInst);
xil_printf("Video Mode = %i ", result);
xil_printf("\n\r");
for (int i = 0; i < 3; i++)
```

EK 3 (Devam) Görüntü Akışı Sağlayan Blok Dizaynların Ana SDK Kodu.

```
{
pFrames[i] = frameBuf[i];
}
vdmaConfig = XAxiVdma_LookupConfig(XPAR_AXIVDMA_0_DEVICE_ID);
XAxiVdma_CfgInitialize(&vdma, vdmaConfig, vdmaConfig->BaseAddress);
video = VMODE_1280x720;
ClkFindParams(video.freq, &clkMode);
ClkFindReg(&clkReg, &clkMode);
ClkWriteReg(&clkReg, 0x43C20000);
ClkStop(0x43C20000);
ClkStart(0x43C20000);
vdmaDMA.FrameDelay = 0;
vdmaDMA.EnableCircularBuf = 1;
vdmaDMA.EnableSync = 0;
vdmaDMA.PointNum = 0;
vdmaDMA.EnableFrameCounter = 0;
vdmaDMA.VertSizeInput = video.height;
vdmaDMA.HoriSizeInput = (video.width)*3;
vdmaDMA.FixedFrameStoreAddr = 0;
vdmaDMA.FrameStoreStartAddr[0] = (u32) pFrames[0];
vdmaDMA.Stride = (video.width)*3;
XAxiVdma_DmaConfig(&vdma, XAXIVDMA_WRITE, &(vdmaDMA));
Status = XAxiVdma_DmaSetBufferAddr(&vdma,
XAXIVDMA_WRITE,vdmaDMA.FrameStoreStartAddr);
Status = XAxiVdma_DmaStart(&vdma, XAXIVDMA_WRITE);
Status = XAxiVdma_StartParking(&vdma, 0, XAXIVDMA_WRITE);
XAxiVdma_DmaConfig(&vdma, XAXIVDMA_READ, &(vdmaDMA));
XAxiVdma_DmaSetBufferAddr(&vdma,
XAXIVDMA_READ,vdmaDMA.FrameStoreStartAddr);
XAxiVdma_DmaStart(&vdma, XAXIVDMA_READ);
```

```
XAxiVdma_StartParking(&vdma, 0, XAXIVDMA_READ);
while(1) {}
cleanup_platform();
return 0;}
```

EK 4 Görüntü Akışı Sağlayan Blok Dizaynların Ekran Kodu.

```
#ifndef VGA_MODES_H_
#define VGA_MODES_H_
typedef struct {
char label[64]; /* Label describing the resolution */
u32 width; /*Width of the active video frame*/
u32 height; /*Height of the active video frame*/
u32 hps; /*Start time of Horizontal sync pulse, in pixel clocks (active width + H. front porch)*/
u32 hpe; /*End time of Horizontal sync pulse, in pixel clocks (active width + H. front porch + H. sync width)*/
u32 hmax; /*Total number of pixel clocks per line (active width + H. front porch + H. sync width + H. back porch) */
u32 hpol; /*hsync pulse polarity*/
u32 vps; /*Start time of Vertical sync pulse, in lines (active height + V. front porch)*/
u32 vpe; /*End time of Vertical sync pulse, in lines (active height + V. front porch + V. sync width)*/
u32 vmax; /*Total number of lines per frame (active height + V. front porch + V. sync width + V. back porch) */
u32 vpol; /*vsync pulse polarity*/
double freq; /*Pixel Clock frequency*/
} VideoMode;
static const VideoMode VMODE_640x480 = {
.label = "640x480@60Hz",
.width = 640,
.height = 480,
.hps = 656,
.hpe = 752,
.hmax = 799,
.hpol = 0,
.vps = 490,
.vpe = 492,
```

EK 4 (Devam) Görüntü Akışı Sağlayan Blok Dizaynların Ekran Kodu.

```
.vmax = 524,
.vpol = 0,
.freq = 25.0
};
static const VideoMode VMODE_800x600 = {
.label = "800x600@60Hz",
.width = 800,
.height = 600,
.hps = 840,
.hpe = 968,
.hmax = 1055,
.hpol = 1,
.vps = 601,
.vpe = 605,
.vmax = 627,
.vpol = 1,
.freq = 40.0
};
static const VideoMode VMODE_1280x1024 = {
.label = "1280x1024@60Hz",
.width = 1280,
.height = 1024,
.hps = 1328,
.hpe = 1440,
.hmax = 1687,
.hpol = 1,
.vps = 1025,
.vpe = 1028,
.vmax = 1065,
.vpol = 1,
```

EK 4 (Devam) Görüntü Akışı Sağlayan Blok Dizaynların Ekran Kodu.

```
.freq = 108.0};
static const VideoMode VMODE_1280x720 = {
.label = "1280x720@60Hz",
.width = 1280,
.height = 720,
.hps = 1390,
.hpe = 1430,
.hmax = 1649,
.hpol = 1,
.vps = 725,
.vpe = 730,
.vmax = 749,
.vpol = 1,
.freq = 74.25, //74.2424 is close enough
};

static const VideoMode VMODE_1600x900 = {
.label = "1600x900@60Hz",
.width = 1600,
.height = 900,
.hps = 1648,
.hpe = 1680,
.hmax = 1759,
.hpol = 1,
.vps = 903,
.vpe = 908,
.vmax = 925,
.vpol = 0,
.freq = 97.75
};
```

EK 4 (Devam) Görüntü Akışı Sağlayan Blok Dizaynların Ekran Kodu.

```
static const VideoMode VMODE_1920x1080 = {
.label = "1920x1080@60Hz",
.width = 1920,
.height = 1080,
.hps = 2008,
.hpe = 2052,
.hmax = 2199,
.hpol = 1,
.vps = 1084,
.vpe = 1089,
.vmax = 1124,
.vpol = 1,
.freq = 148.5 //148.57 is close enough
};
#endif /* VGA_MODES_H_ */
```

EK 5 FAST Corner Algoritmasının HLS Kodları.

```
#include "top.h"

void fast_corner(AXI_STREAM24& video_in, AXI_STREAM24& video_out) {
   //Create AXI streaming interfaces for the core
#pragma HLS INTERFACE axis port=video_in bundle=INPUT_STREAM
#pragma HLS INTERFACE axis port=video_out bundle=OUTPUT_STREAM
   IMAGE_C2 img_0(MAX_HEIGHT, MAX_WIDTH);
   IMAGE_C2 img_1(MAX_HEIGHT, MAX_WIDTH);
   IMAGE_C2 img_1_(MAX_HEIGHT, MAX_WIDTH);
   IMAGE_C1 img_1_Y(MAX_HEIGHT, MAX_WIDTH);
   IMAGE_C1 img_1_UV(MAX_HEIGHT, MAX_WIDTH);
   IMAGE_C2 img_2(MAX_HEIGHT, MAX_WIDTH);
   IMAGE_C1 mask(MAX_HEIGHT, MAX_WIDTH);
   IMAGE_C1 dmask(MAX_HEIGHT, MAX_WIDTH);
   PIXEL_C2 color(255,0);
#pragma HLS dataflow
#pragma HLS stream depth=20000 variable=img_1_.data_stream
   hls::AXIvideo2Mat(video_in, img_0);
   hls::Duplicate(img_0, img_1, img_1_);
   hls::Split(img_1, img_1_Y, img_1_UV);
   hls::Consume(img_1_UV);
   hls::FASTX(img_1_Y, mask, 20, true);
   hls::Dilate(mask, dmask);
   hls::PaintMask(img_1_, dmask, img_2, color);
   hls::Mat2AXIvideo(img_2, video_out);
}
```

EK 6 Harris Corner Algoritmasının HLS Kodları.

```
#include "image.h"

void harris_accel_hls(AXI_STREAM24& INPUT_STREAM, AXI_STREAM24& OUTPUT_STREAM)
{
#pragma HLS INTERFACE ap_ctrl_none port=return
#pragma HLS INTERFACE axis register port=INPUT_STREAM
#pragma HLS INTERFACE axis register port=OUTPUT_STREAM

IMAGE_RGB img_in(MAX_HEIGHT, MAX_WIDTH);
IMAGE_RGB img_in1(MAX_HEIGHT, MAX_WIDTH);
IMAGE_RGB img_in2(MAX_HEIGHT, MAX_WIDTH);
IMAGE_GRAY img2(MAX_HEIGHT, MAX_WIDTH);
IMAGE_GRAY32F img3(MAX_HEIGHT, MAX_WIDTH);
IMAGE_GRAY img4(MAX_HEIGHT,MAX_WIDTH);
IMAGE_RGB img5(MAX_HEIGHT,MAX_WIDTH);
IMAGE_RGB img6(MAX_HEIGHT,MAX_WIDTH);
PIXEL_C2 color(255,0);

#pragma HLS stream depth=1 variable=img_in.data_stream
#pragma HLS stream depth=1 variable=img6.data_stream

hls::AXIvideo2Mat(INPUT_STREAM, img_in);
hls::Duplicate(img_in, img_in1, img_in2);
hls::CvtColor<HLS_RGB2GRAY>(img_in1, img3);
hls::Harris<5,3,float>(img3, img2, 2, 300);
hls::Dilate(img2, img4);
hls::CvtColor<HLS_GRAY2RGB>(img4, img5);
hls::PaintMask(img_in2, img5, img6, color);
```

EK 6 (Devam) Harris Corner Algoritmasının HLS Kodları.

```
hls::Mat2AXIvideo(img6, OUTPUT_STREAM);
}
```

EK 7 Genişletme Algoritmasının HLS Kodları.

```
#include "top.h"
void image_filter(AXI_STREAM& video_in, AXI_STREAM& video_out) {
#pragma HLS INTERFACE axis port=video_in bundle=INPUT_STREAM
#pragma HLS INTERFACE axis port=video_out bundle=OUTPUT_STREAM
IMAGE_C2 img_0(MAX_HEIGHT, MAX_WIDTH);
IMAGE_C2 img_1(MAX_HEIGHT, MAX_WIDTH);
IMAGE_C2 img_2(MAX_HEIGHT, MAX_WIDTH);
IMAGE_C2 img_3(MAX_HEIGHT, MAX_WIDTH);
IMAGE_C2 img_4(MAX_HEIGHT, MAX_WIDTH);
PIXEL_C2 pix(50, 50);
#pragma HLS dataflow
hls::AXIvideo2Mat(video_in, img_0);
hls::SubS(img_0, pix, img_1);
hls::Scale(img_1, img_2, 2, 0);
hls::Dilate(img_2, img_3);
hls::Mat2AXIvideo(img_3, video_out);
}
```

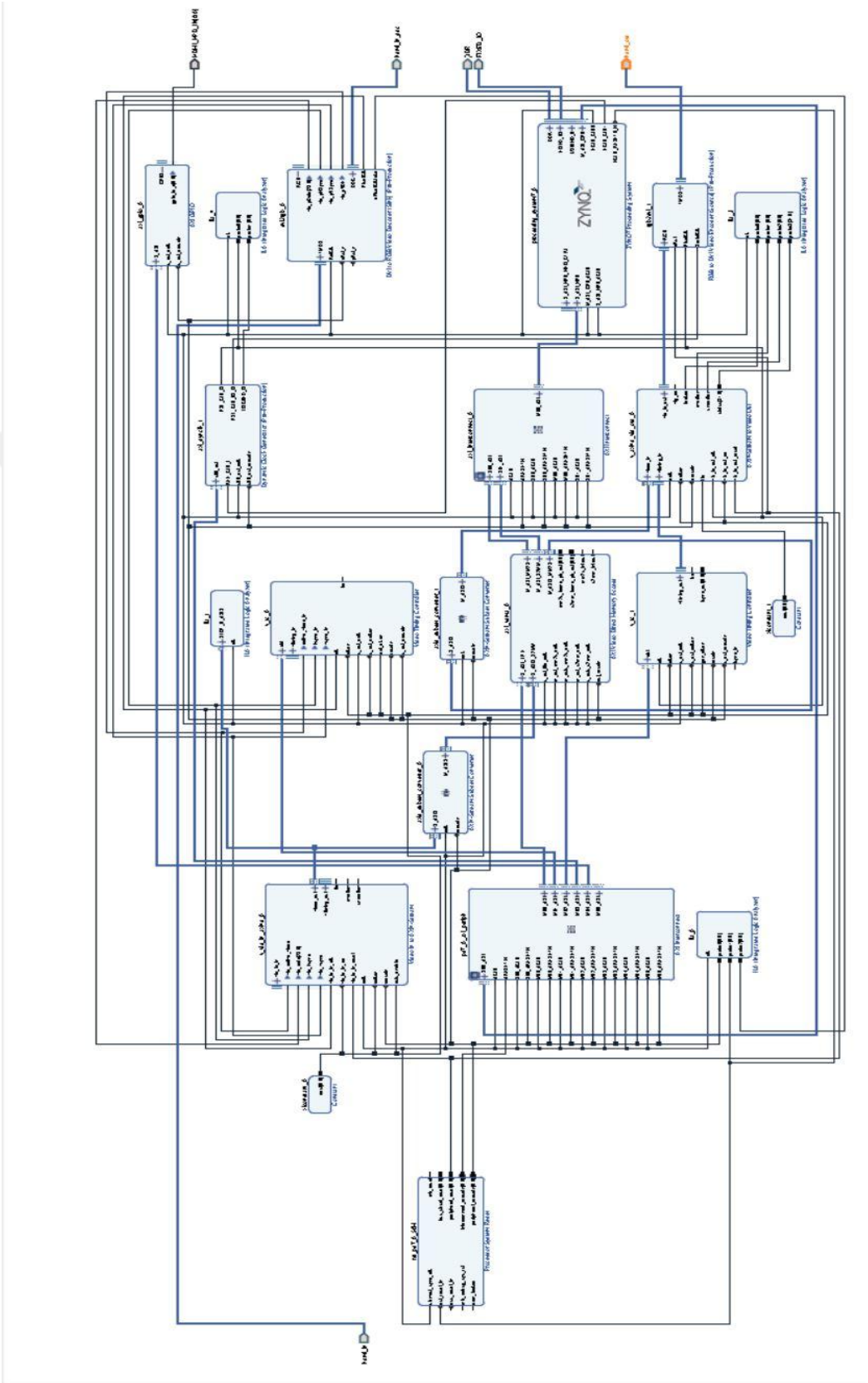
EK 8 Daraltma Algoritmasının HLS Kodları.

```
#include "top.h"
void image_filter(AXI_STREAM& video_in, AXI_STREAM& video_out) {
#pragma HLS INTERFACE axis port=video_in bundle=INPUT_STREAM
#pragma HLS INTERFACE axis port=video_out bundle=OUTPUT_STREAM
IMAGE_C2 img_0(MAX_HEIGHT, MAX_WIDTH);
IMAGE_C2 img_1(MAX_HEIGHT, MAX_WIDTH);
IMAGE_C2 img_2(MAX_HEIGHT, MAX_WIDTH);
IMAGE_C2 img_3(MAX_HEIGHT, MAX_WIDTH);
IMAGE_C2 img_4(MAX_HEIGHT, MAX_WIDTH);
PIXEL_C2 pix(50, 50);
#pragma HLS dataflow
hls::AXIvideo2Mat(video_in, img_0);
hls::SubS(img_0, pix, img_1);
hls::Scale(img_1, img_2, 2, 0);
hls::Erode(img_2, img_3);
hls::Mat2AXIvideo(img_3, video_out);
}
```

EK 9 Sobel Algoritmasının HLS Kodları.

```
#include "sobel.hpp"

void image_filter(AXI_STREAM& INPUT_STREAM, AXI_STREAM&
OUTPUT_STREAM)//, int rows, int cols)
{

#pragma HLS INTERFACE axis port=INPUT_STREAM
#pragma HLS INTERFACE axis port=OUTPUT_STREAM

RGB_IMAGE  img_0(MAX_HEIGHT, MAX_WIDTH);
GRAY_IMAGE img_1(MAX_HEIGHT, MAX_WIDTH);
GRAY_IMAGE  img_2(MAX_HEIGHT, MAX_WIDTH);
GRAY_IMAGE  img_2a(MAX_HEIGHT, MAX_WIDTH);
GRAY_IMAGE  img_2b(MAX_HEIGHT, MAX_WIDTH);
GRAY_IMAGE  img_3(MAX_HEIGHT, MAX_WIDTH);
GRAY_IMAGE  img_4(MAX_HEIGHT, MAX_WIDTH);
GRAY_IMAGE  img_5(MAX_HEIGHT, MAX_WIDTH);
RGB_IMAGE  img_6(MAX_HEIGHT, MAX_WIDTH);

#pragma HLS dataflow
hls::AXIvideo2Mat(INPUT_STREAM, img_0);
hls::CvtColor<HLS_BGR2GRAY>(img_0, img_1);
hls::GaussianBlur<3,3>(img_1,img_2);
hls::Duplicate(img_2,img_2a,img_2b);
hls::Sobel<1,0,3>(img_2a, img_3);
hls::Sobel<0,1,3>(img_2b, img_4);
hls::AddWeighted(img_4,0.5,img_3,0.5,0.0,img_5);
hls::CvtColor<HLS_GRAY2RGB>(img_5, img_6);
hls::Mat2AXIvideo(img_6, OUTPUT_STREAM);
}
```

EK 10 Görüntü Akışını Sağlayan Filtresiz Blok Dizayn.



EK 11 Görüntü Akışını Sağlayan Filtre Kullanılan Blok Dizayn.