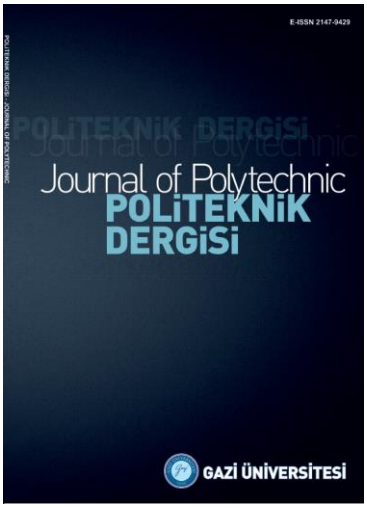


POLİTEKNİK DERGİSİ

JOURNAL of POLYTECHNIC

ISSN: 1302-0900 (PRINT), ISSN: 2147-9429 (ONLINE)
URL: http://dergipark.org.tr/politeknik

Model tabanlı tasarım metotları kullanılarak gerçek zamanlı bir görüntü işleme sisteminin tasarımı ve gerçeklemesi

Design and implementation of a real-time image processing system using model-based design methods

***Yazar(lar) (Author(s))**: Mustafa Yusuf DEMİRCİ [1], İsmail YABANOVA [2]*

ORCID[1]: 0000-0002-5254-2990

ORCID[2]: 0000-0001-8075-3579

Bu makaleye şu şekilde atıfta bulunabilirsiniz(To cite to this article): Demirci M. Y., Yabanova İ., "Model tabanlı tasarım metotları kullanılarak gerçek zamanlı bir görüntü işleme sisteminin tasarımı ve gerçeklemesi", ***Politeknik Dergisi***, 22(4): 827-838, (2019).

Erişim linki (To link to this article): http://dergipark.org.tr/politeknik/archive

DOI: **10.2339/politeknik.423603**

Model Tabanlı Tasarım Metotları Kullanılarak Gerçek Zamanlı Bir Görüntü İşleme Sisteminin Tasarımı ve Gerçeklemesi

Araştırma Makalesi / Research Article

Mustafa Yusuf DEMİRCİ*, İsmail YABANOVA
Fen Bilimleri Enstitüsü, Elektrik – Elektronik Mühendisliği, Afyon Kocatepe Üniversitesi, Türkiye

ÖZ

Bu çalışmada gerçek zamanlı bir görüntü işleme sisteminin tasarımı Zynq mimarisi ve model tabanlı tasarım araçları kullanılarak gerçekleştirilmiştir. Sistemde FPGA ve mikroişlemci birleşiminden oluşan yeni bir mimari olan Zynq-7000 yongasını barındıran bir geliştirme kartı kullanılmış ve bu sisteme uyumlu olan yüksek çözünürlüklü bir kamera ile alınan görüntü üzerinde gri ton dönüşümü, kenar bulma, gürültü giderme ve keskinleştirme filtreleri uygulanarak gerçek zamanlı bir biçimde işlenmiş olan görüntü monitöre aktarılmıştır. Sistemde kullanılan filtre donanım tasarımları HDL kodu yazılmadan Matlab/Simulink ortamında HDL Coder aracı kullanılarak gerçekleştirilmiştir. Kameradan görüntü alan sistemin donanım tasarımı Xilinx firmasının geliştirdiği Vivado Suite isimli araçla yine model tabanlı olarak gerçekleştirilmiştir. Son olarak tasarlanan donanıma uygun yazılımın geliştirilmesi yapılarak sonuçlar ve sistemin kaynak kullanımı incelenmiştir. Yapılan çalışmalar sonucunda sistemin gerçek zamanlı olarak başarıyla çalıştığı görülmüştür.

Anahtar Kelimeler: Görüntü işleme, gerçek zamanlı sistemler, FPGA, Zynq, model tabanlı tasarım.

Design and Implementation of a Real-Time Image Processing System Using Model-Based Design Methods

ABSTRACT

In this study, design of a real-time image processing system was implemented using Zynq architecture and model-based design tools. System uses a development board that includes Zynq-7000 chip, a new architecture which consists of FPGA and microprocessor, and the images were captured with a high-resolution camera compatible with this system. Processed images transferred to the monitor in real-time with applying grayscale conversion, edge detection, noise reduction and sharpening filters. The filter hardware designs used in the system were implemented by using HDL Coder tool in Matlab / Simulink environment without using hand-coded HDL. The hardware design of the system that takes images from the camera has been implemented with the model-based hardware development tool Vivado Suite which developed by Xilinx company. Finally, an appropriate software for the hardware has been developed and results and the resource usage of the system have been analyzed. As a result of the work done, it has been observed that the system works successfully in real time.

Keywords: Image processing, real-time systems, FPGA, Zynq, model-based design.

1. GİRİŞ (INTRODUCTION)

Günümüzde birçok alanda kullanılan görüntü işleme sistemlerinin değişen ihtiyaçlar ve teknolojinin de gelişimiyle birlikte gerçek zamanlı olarak kullanımı ihtiyacı doğmuştur. Özellikle sürüş destek sistemleri, otonom araçlar, uçuş kontrol, güvenlik ve savunma sistemleri gibi alanlarda yoğun olarak kullanılan gerçek zamanlı gömülü sistem tasarımlarında hızdan ziyade istenen işin belirlenmiş olan bir zaman dilimi içerisinde yapılması önceliktir. Bu sebeple zaman kararlılığı ile birlikte öngörülebilirlik özellikleri gerçek zamanlı sistemler için olmazsa olmaz koşullardır [1].

Bu tip gerçek zamanlı görüntü işleme sistemlerde görüntü üzerinde çeşitli filtreler ve öznitelik çıkarma gibi işlemler yapılması gerektiğinden yoğun bir işlem gücüne ihtiyaç duyulmaktadır. Özellikle yüksek çözünürlüklere ihtiyaç duyulan sistemlerde görüntü karesindeki her pikselin ayrı ayrı ve çok kısa bir zaman diliminde işlenmesi gereği mikroişlemci tabanlı klasik sistemleri yetersiz kılmış ve böylece farklı çözüm arayışları ortaya çıkmıştır. Bunlar içerisinde DSP, FPGA ya da GPU tabanlı sistemler olduğu gibi bunların bir arada bulunduğu "yonga üzeri sistem" (SoC-System on Chip)'ler giderek daha yaygın hale gelmiştir. DSP (Dijital Signal Processor – Sayısal Sinyal İşleyici) tabanlı sistemler içerisinde Texas Instruments firmasının geliştirmiş olduğu DaVinci ve OMAP serisi yongalar örnek gösterilebilirken, NXP firmasının S32V34 serisi yongaları gibi işlemci, GPU ve özel görüntü işleyici birimlerinin bir arada bulunduğu sistemlerde bulunmaktadır [2,3,4]. Bu tip sistemler DSP'lerin ve yardımcı işlemcilerin sinyal işlemedeki avantajlarını kullanarak donanım hızlandırıcısı olarak görüntü işleme

*Sorumlu Yazar (Corresponding Author)
e-posta : mfecidpc@gmail.com

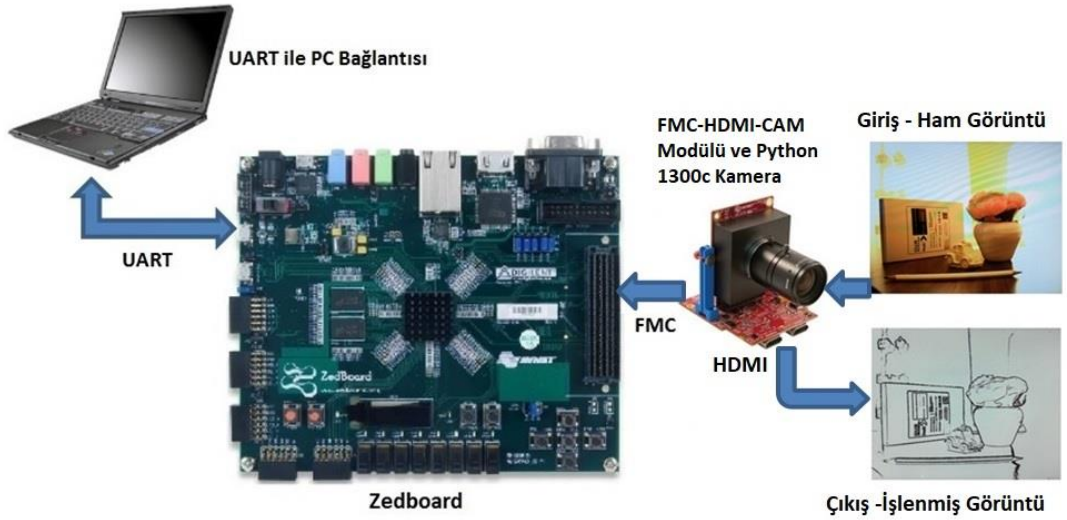
sistemlerinde gerçek zamanlı çalışmayı sağlayabilmektedirler. Bununla birlikte DSP'ler sabit mimariler olduklarından üreticinin destek verdiği kütüphaneler ve arayüzler ile programlanarak nispeten kısıtlı bir alana hitap etmektedirler [5]. Diğer yandan paralel işlem yapmaya daha elverişli yapısı ve düşük güç tüketimi nedeniyle alan programlanabilir kapı dizilerinin (FPGA) bu tip gömülü sistemlerde kullanımı artmıştır. FPGA tabanlı sistemlerde donanım tamamen istenen mimaride programlanabilir. İstenen donanım bir DSP olabileceği gibi tamamen çalışacak sisteme özel oluşturulmuş bir donanım bloğu da olabilmektedir. Ayrıca Xilinx Zynq gibi yeni nesil mimarilerde işlemci ve programlanabilir lojik bloğu birlikte bulunmakta ve bu sistemler tıpkı DSP'ler gibi C dili ile programlanabilmekte, hatta Matlab, Labview gibi model tabanlı geliştirme ortamlarında dahi donanım tasarımı yapılabilmektedir. Bu sebeple önerilen görüntü işleme sisteminde FPGA tabanlı bir sistem tercih edilmiştir. Tüm bu gelişmeler neticesinde mikroişlemci ve FPGA donanımını bir arada kullanan sistemlerin kullanımının görüntü işleme sistemlerinde en iyi performansı verdiği tespit edilmiştir [6]. Bu tip çözümler bilgisayar ve FPGA donanımının bir arada kullanılması ile oluşturulduğunda yüksek güç ve alan tüketimini beraberinde getirdiği için gömülü sistem tasarımlarında bu tip bir kullanım pratik olmamaktadır. Xilinx firması tarafından 2012 yılında tanıtılan Zynq mimarisi aynı yonga içerisinde mikroişlemci ve FPGA donanımını barındıran yapısıyla gömülü sistem piyasasında çok önemli bir açığı kapatmıştır. Bu yonga üzeri sistem mimarisi sayesinde donanım ve yazılımın tamamı tek bir kart üzerinde gerçeklenerek hem yüksek hız, hem de düşük güç ve alan tüketimi gerçekleştirmek mümkün olabilmektedir. Bu nedenle çalışmada Zynq-7000 mimarisine sahip olan Zedboard geliştirme kartı kullanılmıştır. Tüm bu avantajların yanında donanım tasarımının karmaşık olması ve klasik mikroişlemci sistemlerinden farklı yapısı nedeniyle Zynq kullanılarak yapılan tasarımlar uzun bir geliştirme süreci ve farklı disiplinlerden oluşan bir geliştirme ekibi gerektirebilmektedir [7].

İlgili literatür incelendiğinde Zynq mimarisi ile görüntü işleme üzerine yapılan bazı önemli çalışmalar özetlenmiştir. Abdelgawad ve arkadaşları yaptıkları çalışmada Zynq platformu kullanan ZC702 geliştirme kartı üzerinde gerçek zamanlı olarak Canny Kenar bulma algoritmasını uygulamışlardır. Sistemi Vivado HLS ve dahili video kütüphanesini kullanarak tasarlamış ve 1080p çözünürlükte 60 fps hızda çalışabilen bir sistem oluşturmuşlardır [6]. Russell ve Fischaber yaptıkları çalışmada Zedboard geliştirme kartı üzerine yerleştirilen tümleşik bir kamera sistemi ile 1920 x 1080 çözünürlükteki görüntüyü gerçek zamanlı olarak alıp trafik işaretlerinin görüntülerini tanıma ve sınıflandırma yapan bir sistem geliştirmişlerdir. Tüm projeyi OpenCV kütüphanelerini kullanarak altı hafta gibi bir sürede oluşturmuşlar ve böylece platformun hızlı tasarım yapılabilme özelliğini vurgulamışlardır. Geliştirilen sistemde bir trafik işaretini sınıflandırmanın yaklaşık olarak 5 saniye süre aldığı belirtilmiştir [8]. Monson ve arkadaşları yaptıkları çalışmada optik akış algoritmalarını optimize ederek farklı platformlarda yürütüp karşılaştırmalar yapmışlardır. Buna göre aynı algoritma Core i7 işlemcili bir bilgisayarda, Zynq-7000 yongasının yalnızca ARM işlemci biriminde ve son olarak Zynq-7000 mimarisinin programlanabilir lojik birimine özel en iyileştirme yapılarak yürütülmüştür. Sistemin kodu C dilinde Vivado HLS programı kullanılarak sentezlenmiştir. Sonuçlara bakıldığında Zynq üzerinde yürütülen algoritmanın performansı i7 işlemcili bir bilgisayarın performansına yakın olmakla birlikte güç tüketiminin yalnızca bilgisayarın 1/7 si kadar olduğu görülmüştür [9]. Sümer yaptığı çalışmada farklı yüz tanıma metotlarını kullanarak duygu ve yüz ifadesi analizi uygulaması gerçekleştirmiştir. Zynq-7000 platformunu kullanan Zedboard geliştirme kartında OpenCV kütüphanaleri, C++ dili ve gömülü Linux işletim sistemi kullanılarak yapılan çalışmada öfke, mutluluk ve şaşırma ifadeleri sırasıyla %97, %100 ve %97 başarıyla sınıflandırılmış ve yaklaşık saniyede 4 ile 5 kare hızında bir performans elde edilmiştir [10]. Altuncu ve arkadaşları yaptıkları çalışmada bazı görüntü işleme algoritmalarını Zedboard geliştirme kartı üzerinde Verilog donanım tanımlama dilini kullanarak gerçekleştirmişlerdir. Usb üzerinden webcam ile alınan 256 x 256 piksel boyutundaki görüntü üzerinde gerçek zamanlı olarak filtre işlemlerini saniyede yaklaşık 40 kare hızında uygulamayı başarmışlardır [11]. Chhabra ve arkadaşları Zynq-7000 platformunu barındıran XC7Z020-1CLG484 geliştirme kartı üzerinde bir otomatik plaka tanımlama sistemi tasarlamışlardır. Tüm sistem donanımı MATLAB Simulink ve Xilinx System Generator geliştirme ortamı kullanılarak yüksek seviyeli sentez araçları ile tasarlanmış, gerçeklenen uygulamalar içerisinde en verimli olanı tespit edilmiş ve her üç yöntemin de %90'ın üzerinde başarılı olduğu tespit edilmiştir [12]. Shi yaptığı çalışmada Zynq-7000 mimarisine sahip Zedboard üzerinde Sobel kenar belirleme filtresini Vivado HLS aracını kullanarak gerçek zamanlı olarak gerçekleştirmiştir. Görüntü aktarımındaki gecikmeyi önlemek için OpenCV'de yazılmış olan Sobel filtre algoritmasını iyileştirerek daha hızlı çalışmasını sağlamış ve önceki incelediği tasarımlardan %75 daha az kaynak kullanımına rağmen %30 daha fazla performans elde ederek 1080p çözünürlükte maksimum 90,1 fps hıza ulaşabilmiştir [13]. Al-Naqshbandi yaptığı çalışmada Zedboard üzerinde Canny kenar belirleme algoritmasını uygulayıp en iyileştirmesini gerçekleştirmiştir. OpenCV kütüphaneleri kullanılarak yapılan çalışmada Canny kenar belirleme algoritmaları başarıyla uygulanmış ve yüksek seviyeli sentez araçları ile donanım tasarımını gerçekleştirmiştir. Çalışmada algoritmaların performans analizleri yapmış ve yaptığı geliştirmeler sonucunda çekirdek düzeyine 3 kata kadar, uygulama düzeyinde ise 7 kata kadar daha hızlı çalışma sağlandığı belirtmiştir [14].

Yukarıda belirtilen çalışmalar incelendiğinde tasarım süresini azaltmak için kullanılan yüksek seviyeli sentez araçlarının genellikle Vivado HLS isimli C kodu yazılarak donanım tasarımı yapılabilen araç olduğu görülmektedir. Bu çalışmada ise belirtilen tasarım karmaşıklığına çözüm aranarak bir adım daha ileri gidilerek tamamen yüksek seviyeli dillerle gerçek zamanlı bir görüntü işleme sisteminin oluşturulması amaçlanmıştır. Tasarım karmaşıklığın azaltılması maksadıyla donanım tanımlama dilleri kullanımı yerine model tabanlı tasarım yazılımları olan Matlab/Simulink ve bu program üzerinde çalışan HDL Coder, Embedder Coder, Vision HDL Toolbox isimli yüksek seviyeli sentez araçları ile donanım tasarımı yapılmıştır. Sistemin kamera referans tasarımı ve sistem yazılımı ise Xilinx'in kendi araçları olan Vivado Design Suite ve SDK yazılım geliştirme aracı ile oluşturulmuştur. Tasarlanan sistem ile kameradan anlık olarak alınan yüksek çözünürlüklü görüntü gri ton, kenar bulma, gürültü giderme ve keskinleştirme filtrelerine tabi tutularak işlenmiş görüntü gecikme olmaksızın monitöre aktarılmıştır.

2. DONANIM ALTYAPISI
(HARDWARE SUBSTRUCTURE)

Tasarlanan sistemde Zynq-7000 mimarisine sahip Zedboard geliştirme kartı ile birlikte Avnet firmasının ürettiği FMC-HDMI-CAM modülü ve yine sistemle tam uyumlu olan Python 1300-C kamera modülü kullanılmıştır [15, 16]. FMC, FPGA ara bağlantısı anlamına gelmekte olup, FPGA kartları için yüksek hızlarda çalışabilen standart bir arayüzü tanımlar. Böylece çevre birimleri ile FPGA kartları arasında kolayca bağlantı yapılabilir ve bu arayüze sahip tüm kartlar kullanılabilir [17]. Sistemin bilgisayar ile bağlantısı UART üzerinden sağlanmış olup sistem durumu seri port üzerinden izlenebilmektedir. Bununla birlikte sistem bağımsız olarak çalışmakta ve bilgisayar bağlantısına ihtiyaç duymamaktadır.



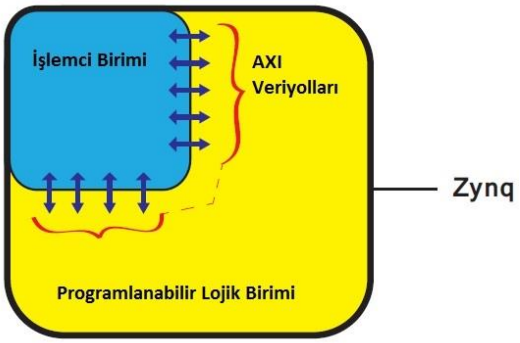
Şekil 1. Tasarlanan Sistemin Genel Şeması (General Schematic of the Designed System)

Zedboard üzerindeki buton ve anahtarlar yardımıyla istenen filtre seçilebilmekte ve diğer parametreler ayarlanabilmektedir. Sistemin genel şeması Şekil 1.de verilmiştir.

2.1 Zynq-7000 Mimarisi (Zynq-7000 Architecture)

Çalışmada kullanılan Zedboard geliştirme kartı Zynq-7000 Z-7020 serisi yonga üzeri sistem ve çevre birimlerinden oluşmaktadır [18]. Zynq-7000, çift çekirdekli ARM Cortex-A9 mimarisine sahip işlemci ile FPGA bileşenlerini aynı yonga içerisinde barındıran Tümüyle Programlanabilir Yonga Üzeri Sistem mimarisidir. Bu sistemde işlemci ile FPGA arasındaki veri aktarımını sağlamak amacıyla yüksek hızlı AXI (gelişmiş genişletilebilir arabirim) veri yolları bulunmaktadır. Böylece bu iki birimin birbirinden bağımsız olmaksızın amaca yönelik olarak bir arada kullanılması olanaklı hale gelmiştir. Zynq-700 AP SoC, temel olarak işlemci birimi (PS- Processing System), FPGA sistemine karşılık gelen bir programlanabilir lojik biriminden (PL - Programmable Logic) ve ara bağlantılardan meydana gelir [19].

Şekil 2.de Zynq mimarisinin basit prensip şeması verilmiştir.



Şekil 2. Zynq-7000 Mimarisi Basit Prensip Şeması (Zynq-7000 Architecture Simple Principle Diagram)

2.2 Sistem Tasarım Akışı (System Design Flow)

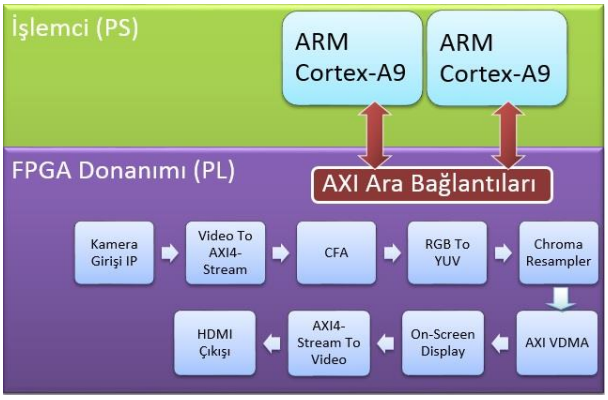
Şekil 3.te görüntü işleyici sisteminin genel tasarım akışı görülmektedir. Sistemin donanım bileşenleri bir araya getirildikten sonra ilk olarak kameradan görüntü alabilen bir referans tasarımın gerçekleştirilmesi gerekmektedir [20, 21]. Kameradan görüntü başarılı bir biçimde alındıktan sonra kullanılacak olan filtrelerin Matlab/Simulink ortamında tasarımı ve simülasyonu yapılarak HDL Coder aracı ile bu filtreler standart IP blokları olarak paketlenmiştir [22]. Bu IP blokları Vivado ortamına dahil edilerek oluşturulmuş olan kamera referans tasarımına entegre edilmiştir. Son olarak sistemin yazılımı Xilinx SDK ortamında yazılarak ve gerekli sürücüler sisteme dahil edilerek sisteme son şekli verilmiştir [23].

Şekil 3. Görüntü İşleyici Sistemi Tasarım Akışı (Image Processor System Design Flow)

2.3 Kamera Referans Tasarımı (Camera Reference Design)

Çalışmada Avnet tarafından üretilen FMC-HDMI-CAM modülü kullanılarak yüksek hızlı FMC arabirimi üzerinden kamera ile bağlantı sağlanmıştır. Görüntü algılayıcı olarak bu modüle uyumlu olan Python-1300c kamera modülü seçilmiştir. Bu modül ON Semiconductor firmasının Python-1300 görüntü algılayıcısına sahip 1280 x 1024 piksel çözünürlükte 210 kare/saniye hıza kadar görüntü aktarma kapasitesine sahiptir [16]. Böylece gerçek zamanlı bir sistem için gerekli olan altyapı oluşturulmuştur.

Oluşturulan referans tasarımın amacı, kullanılan kamera sensöründen gelen ham görüntü üzerinde bazı işlemler yaparak görüntüyü kullandığımız AXI4-Stream veri tipine dönüştürerek daha sonraki işlemler için hazır hale getirmektir. Bu sayede kameradan alınan görüntü kareleri hatasız biçimde ekrana aktarılabilmiştir. Bu işlemler gerçekleştirildikten sonra görüntü işleme için gerekli olan IP bloklarının tasarımı kamera referans tasarımına göre yapılmıştır. Kameradan görüntüyü doğru biçimde alabilmek için "Xilinx Video and Image Processing Pack" ve kamera donanımı için sağlanan 3.parti IP bloklarının indirilerek sisteme dahil edilmesi gerekmektedir [24, 25]. Gerekli işlemler tamamlandıktan sonra sentez ve gerçekleme aşamasına geçilerek sistemin test yazılımı düzenlenerek görüntü başarıyla ekrana yansıtılmıştır. Şekil 4.te kamera referans tasarımının ve kullanılan IP bloklarının sistemdeki durumu görülmektedir.



Şekil 4. Kamera Referans Tasarımı Şematik Gösterimi (Camera Reference Design Schematic Display)

Sistemde kamera üzerinden görüntü öncelikle AXI4-Stream veri tipine dönüştürülmekte, ardından CFA IP bloğu ile eksik renk bileşenleri tekrar oluşturulmaktadır. Daha sonra RGB To YUV bloğu ile görüntü YCbCr formatına dönüştürülmektedir. Sonraki aşamada kullanılan Chroma Resampler IP bloğu ile görüntü 24 bit üzerinden 16 bitlik formata düşürülmüş ve On-Screen Display IP bloğu ile 1280 x 1024 piksel boyutundaki görüntü 1920 x 1080 piksel boyutlarındaki monitöre ortalanarak yerleştirilmiştir. Daha sonra görüntü monitöre aktarılmak üzere tekrar dönüştürülmüş ve FMC-HDMI-CAM modülü üzerindeki HDMI çıkışından monitöre bağlantı yapılmıştır [26,27,28,29,30]. AXI VDMA IP bloğu ile sistem belleği ile doğrudan veri alışverişi sağlanarak sistemin olası darboğazlardan korunması sağlanmıştır [31].

Şekil 5.te oluşturulan kamera referans sisteminin çalışma görüntüsü verilmiştir.

Şekil 5. Kamera Referans Tasarımı Sistem Görüntüsü (Picture of The Camera Reference Design System)

2.4 Görüntü İşleyici Sistemi (Image Processor System)

Kamera referans tasarımının ardından kullanılacak görüntü işleme filtrelerinin tasarımları yapılmıştır. Bu bölümde Xilinx ve Mathworks araçlarının bir arada kullanılması gerekmektedir [32, 33, 34]. Sistem tasarımı Simulink ortamında tamamen model tabanlı olarak yapılarak simülasyonları simulink üzerinde yapıldıktan sonra HDL Coder ile otomatik olarak HDL kodu üretilerek bu kodlar kullanılabilir IP paketlerine dönüştürülmüştür. Ayrıca Embedder Coder, Simulink Coder, Matlab Coder ve Vision HDL Toolbox isimli araçların ve bunların Zynq için olan ek paketlerinin de Matlab üzerinde kurulu olması gerekmektedir. Sistem tasarımında Vivado 2016.4 ve Matlab R2017b kullanılmıştır. Simulink üzerinde çalışan sistem akan piksel arayüzü olarak adlandırılan bir tür seri arabirimdir. Bu yöntemde pikseller seri biçimde aktarılır ve veri ile kontrol olmak üzere iki çeşit sinyal üzerinden veri aktarımı gerçekleşir. Bu arayüz AXI4-Stream ile birebir uyumlu olup tasarlanan IP bloklarının kamera referans sistemiyle tamamen uyumlu olarak çalışmasını sağlar [35]. Görüntü işleyici sisteminin genel yapısı Şekil 6.da gösterildiği gibidir.

Şekil 6. Video İşleyici Sistemi Blok Diyagramı (Video Processing System Block Diagram)

Simulink ortamında simülasyon yapılabilmesi için görüntü öncelikle "Kaynak Video" isimli blok üzerinden kareler halinde gelir ve ardından Kareden Piksele Dönüşüm bloğu ile seri hale getirilerek piksel formatına dönüştürülür. HDL Filtre bloğunda ise Zynq üzerinde çalışacak olan filtre sistemi bulunur ve filtreden geçen görüntü tekrar kareler haline dönüştürülerek işlenmiş görüntü "video viewer" bloğundan izlenebilmektedir. Böylece HDL koduna dönüşüm yapılmadan önce sistemin tepkisi gözlemek mümkün olmuştur.

3. SİMULİNK ORTAMINDA SİSTEMİN TASARIMI VE SİMÜLASYONU (DESIGN AND SIMULATION OF THE SYSTEM IN SIMULINK ENVIRONMENT)

Çalışmada kullanılan filtreler Simulink ortamında Vision HDL Toolbox blokları ile ayrı ayrı tasarlandıktan her biri birer alt sistem haline getirilerek öncelikle Matlab üzerinde simülasyonları yapılmış, daha sonra HDL iş akısı danışmanı ile kullanılabilir IP bloklarına dönüştürülmüştür. Başta yapılan dört adet filtre tasarımı sistemde tüm filtrelerin aynı anda aktif edilebilmesi amacıyla birleştirilerek iki adet IP bloğuna indirgenmiştir. Bu sayede tasarım karmaşıklığı artmadan tüm filtreler aynı anda aktif edilebilmiştir.

3.1 Gri ton Dönüştürücü (Grayscale Convertor)

Gri ton dönüştürme filtresi tasarlanırken Vision HDL Toolbox içerisindeki RGB To Intensity bloğu ile işlem yapılmıştır. Buna göre her üç renk değeri sabit katsayılarla çarpılarak gri ton değerleri elde edilmiştir [36]. Eşitlikteki katsayılar referans tasarımda kullanılan YUV formatına ve bu formatın en sık kullanılan kodlama biçimi olan ITU rec.601 standardına uygun olarak seçilmiştir [37, 38].

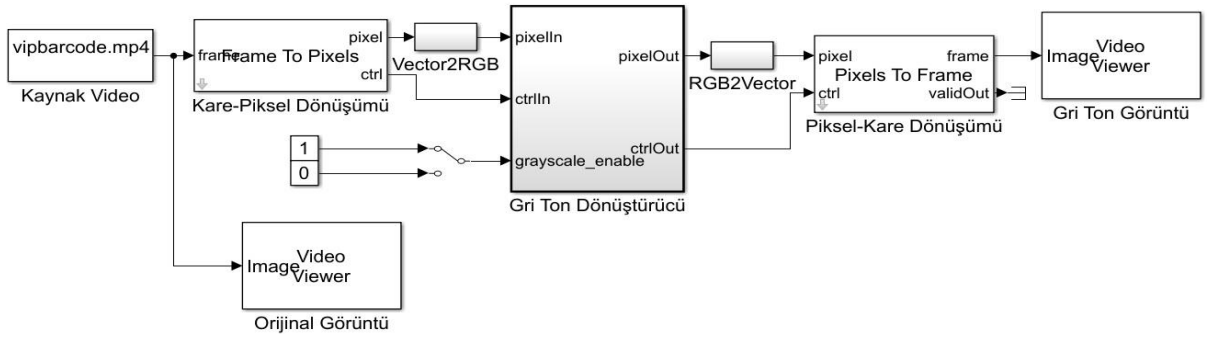
Formül 1.de gri ton değerlerinin bulunması için gereken katsayılar verilmiştir.

$$piksel\ değeri = [0{,}299 \quad 0{,}578 \quad 0{,}114] \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{1}$$

Sistem Şekil 7.de görüldüğü gibi tasarlanmıştır. Simülasyon Simulink ortamında yapıldıktan sonra "Gri_ton_Donusturucu" isimli simulink bloğu içerisinde oluşturulan filtre sistemi HDL Coder ile IP bloğuna dönüştürülmüştür. Filtre IP bloğu tasarımının diyagramı Şekil 8.de verilmiştir. Simülasyon sonucu Şekil 9.da gösterilmiştir.

3.2 Kenar Bulucu (Edge Detector)

Kenar bulma yönteminin amacı görüntüdeki kenar hatlarının belirginleştirilerek öznitelik çıkarımının ve belirli şekillerin algılanmasının kolaylaşmasıdır. Kenar bulucu filtre tasarımında Vision HDL Toolbox üzerindeki Edge Detector bloğu kullanılmıştır [36]. Yöntem olarak Sobel yöntemi seçilmiştir. Sobel



Şekil 7. Gri Ton Dönüştürme Sistemi Blok Diyagramı (Grayscale Conversion System Block Diagram)

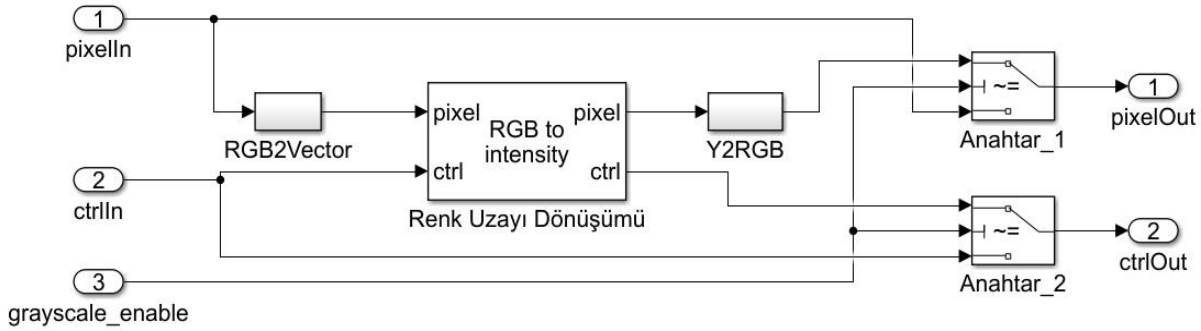
operatöründe gri ton görüntü üzerinde yatay ve dikey olmak üzere iki türevsel matris ile maskeleme yapılarak görüntünün gradyan bileşenleri elde edilir. Elde edilen bileşenlerin ağırlıkları hesaplanarak kenarları bulunmuş görüntü oluşturulur.

$$G = \sqrt{Gx^2 + Gy^2} \quad (4)$$

Gy = Düşey Gradyan Bileşeni

Gx = Yatay Gradyan Bileşeni

Gradyan değerleri Formül 4.e göre birleştirilerek



Şekil 8. Gri Ton Dönüştürücü IP Bloğu (Grayscale Convertor IP Block

Şekil 9. Gri Ton Dönüştürme Simülasyon Sonucu (Grayscale Conversion Simulation Result)

Kenar bulucu bloğu gradyan değerlerini 45 ve 135 derece açılarda belirler ve Formül 2. ile Formül 3.teki matrisleri kullanır [39].

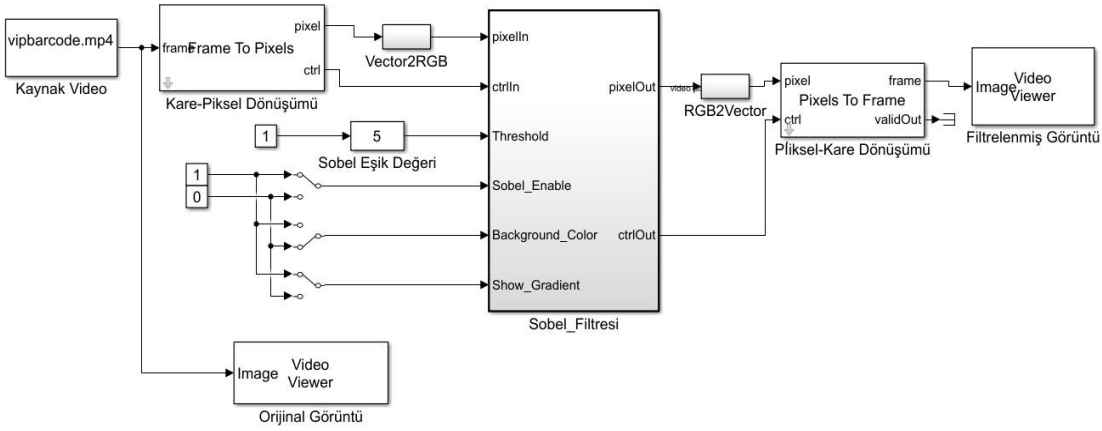
$$Gy = \frac{1}{8}\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \quad (2)$$

$$Gx = \frac{1}{8}\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad (3)$$

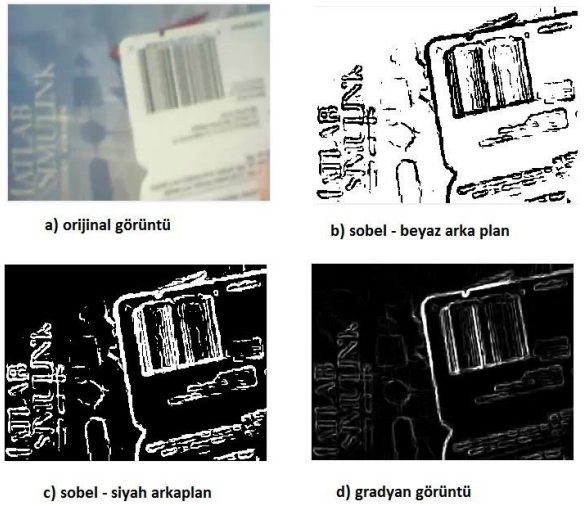
kenarlar elde edilir. Sistemin blok diyagramı Şekil 10.da gösterildiği gibidir. Simülasyon sonucu ise Şekil 11.de verilmiştir.

3.3 Ortanca Filtre (Median Filter)

Ortanca filtre özellikle görüntü üzerindeki tuz-biber gürültüsünün giderilmesinde kullanılan doğrusal olmayan bir filtre çeşididir. Çalışma prensibi 2 boyutlu değişken uzunluktaki bir matris oluşturularak pikseller kendi içerisinde sıralanarak ortanca eleman bulunması şeklindedir. Medyan filtre tasarımında Vision HDL Toolbox üzerindeki Median Filter bloğu kullanılmıştır [36]. Simulink ortamında yapılan simülasyonda filtre etkisinin net olarak görülebilmesi için görüntü üzerine %1 oranında gürültü eklenmiştir. Sistemin blok diyagramı Şekil 12.de verilmiştir. 3 x 3, 5 x 5 ve 7 x 7 komşuluk matrisine göre ortanca filtre sonuçları Şekil 13.te gösterildiği gibidir.



Şekil 10. Kenar Bulucu Sistem Blok Diyagramı (Edge Detector System Block Diagram



Şekil 11. Kenar Bulucu Simülasyon Sonucu (Edge Detector Simulation Result

3.4 Keskinleştirici Filtre (Sharpening Filter)

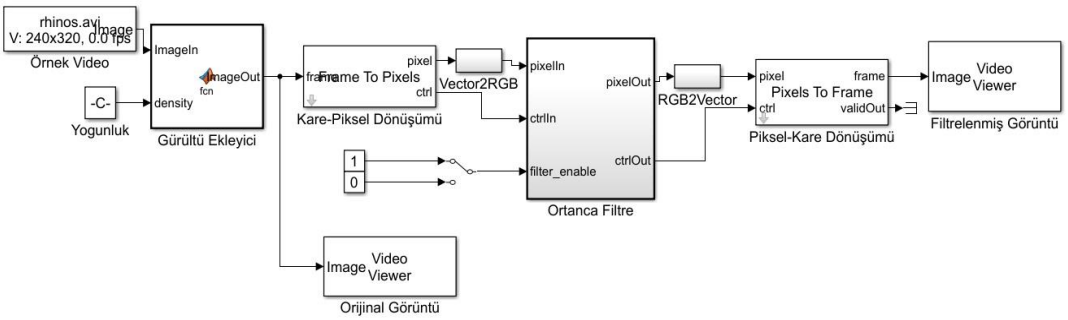
Tasarlanan son sistemde keskinleştirme işlemi yapılmıştır. Bunun için Vision HDL Toolbox altındaki Image Filter bloğu kullanılmıştır [36]. Bu filtrede "fspecial" komutu kullanılarak ikinci derece bir türev fonksiyonu olan laplasyen filtre ile işlem yapılmıştır. Laplasyen filtrenin genel ifadesi Formül 5 ve Formül 6.da gösterilmiştir.

Tasarlanan filtre sisteminde α=0,2 olarak alınmıştır. Ayrıca simulink'in programı konvolüsyon sonucunun daima sıfır olabilmesi için denklem üzerinde yaptığı kaydırma işlemleri sebebiyle filtrede kullanılan konvolüsyon matrisi Formül 7.deki elemanlardan oluşmuştur [39]. Uygulamada keskinleştirici filtre ortanca filtre ile birlikte gerçeklenmiştir. Bunun sebebi keskinleştirici filtrenin ortanca filtrenin meydana getirdiği bulanıklığı giderici etki göstermesidir. Sistemin blok diyagramı Şekil 14.te, filtre sonuçları ise Şekil 15.te verilmiştir.

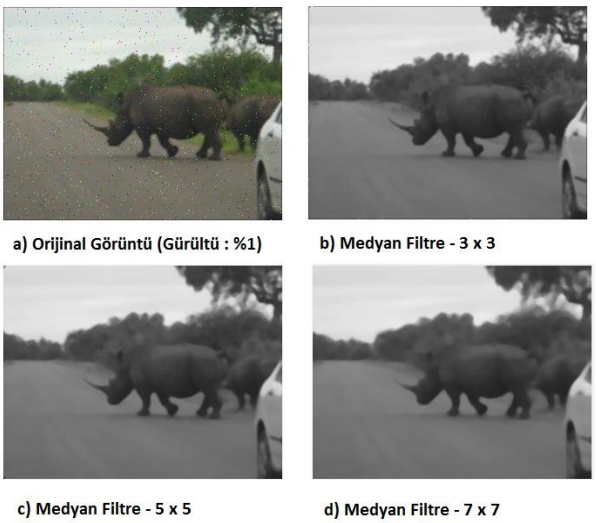
$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \tag{5}$$

$$\nabla^2 = \frac{4}{(\alpha+1)} \begin{bmatrix} \frac{\alpha}{4} & \frac{1-\alpha}{4} & \frac{\alpha}{4} \\ \frac{1-\alpha}{4} & -1 & \frac{1-\alpha}{4} \\ \frac{\alpha}{4} & \frac{1-\alpha}{4} & \frac{\alpha}{4} \end{bmatrix} \tag{6}$$

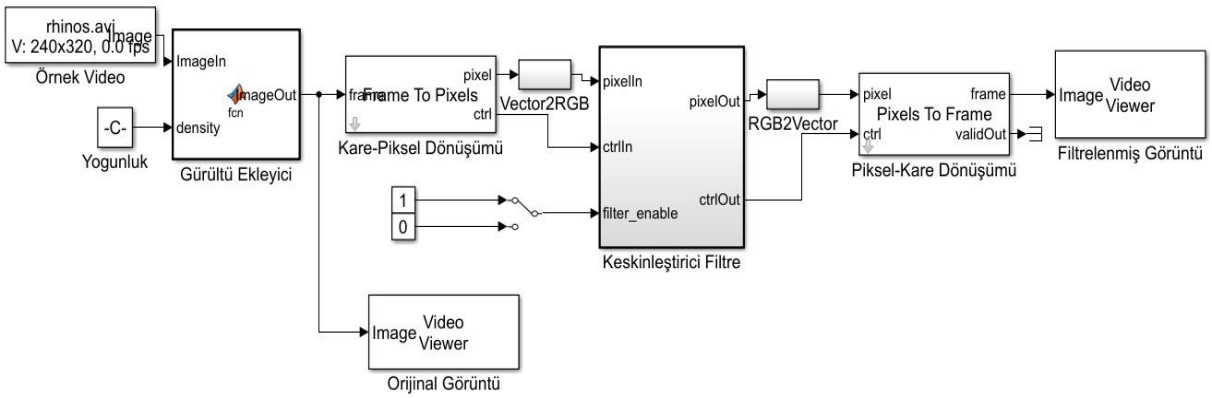
$$\nabla^2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{7}$$



Şekil 12. Ortanca Filtre Sistem Blok Diyagramı (Median Filter Block Diagram

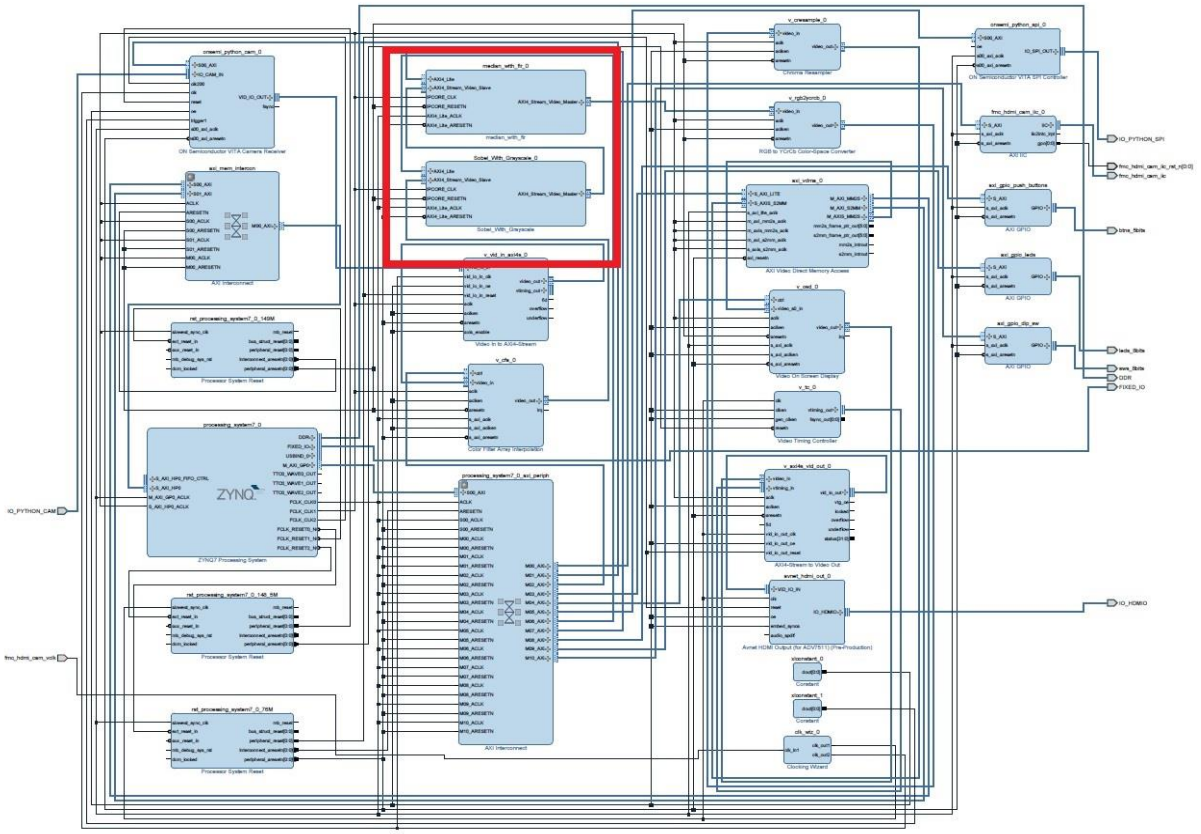


Şekil 13. Ortanca Filtre Simülasyon Sonucu (Median Filter Simulation Result



Şekil 14. Keskinleştirici Filtre Sistem Blok Diyagramı (Sharpening Filter Block Diagram

Şekil 15. Keskinleştirici Filtre Simülasyon Sonucu (Sharpening Filter Simulation Results)



Şekil 16. Tüm Sistem Donanım Tasarımı Blok Şeması (Complete System Hardware Design Block Diagram)

4. TASARIMLARIN SENTEZLENMESİ VE BULGULAR (DESIGN SYNTHESIS AND RESULTS)

Tasarımların simulink üzerindeki simülasyonları tamamlandıktan sonra her bir filtre sistemi ayrı ayrı HDL koduna çevrilmiştir. Bu işlem Matlab/Simulink eklentisi olan HDL Coder ve HDL iş akışı danışmanı vasıtasıyla gerçekleşmiştir. HDL iş akışı danışmanı, Matlab ve Simulink üzerindeki sistemleri HDL koduna çevirme işlemi sırasında gerekli ayarların yapılmasını sağlayan bir arayüzdür. Bu aşamadan sonra standart bir IP bloğuna çevrilen sistem Vivado Suite içerisinde bulunan IP entegre edici sistemi ile kamera referans tasarımına bağlanmıştır. Böylece filtre sistemleri Zedboard üzerinde sırası ile gerçeklenmişlerdir. Daha sonraki çalışmalarda ise sistemin yonga üzerinde daha az alan kaplaması ve yazılımın sadeleşmesi amacıyla tasarlanan IP blokları Simulink üzerinde bir araya getirilerek tekrar tasarım yapılmıştır.

Buna göre gri ton dönüştürme ve kenar bulma sistemleri tek bir IP bloğu haline getirilmiştir. Ortanca ve keskinleştirme filtreleri ise ikinci IP bloğunu oluşturmuşlardır. Gerekli ara bağlantılar tekrar yapılarak sistem tekrar sentezlenerek gerçeklenmiştir. Gerçeklenen son sistemin Vivado Suite üzerindeki blok tasarımı Şekil 16.da verilmiştir. Çalışmada tasarlanan IP blokları şema üzerinde işaretlenmiştir.

4.1 Sentez Sonuçları (Synthesis Results)

Sistemin sentezi ve yonga üzerindeki yerleşimi tamamlandıktan sonraki kaynak kullanımı Çizelge 1.de verilmiştir. Görüldüğü üzere mevcut kaynakların en fazla %35'lik bir kısmı kullanılmıştır. Burada herhangi bir optimizasyon çalışması yapılmadığı da göz önünde bulundurulmalıdır.

Sistemde ana saat hızı 148,50 MHz, kamera ve görüntü işleme bloklarının çalıştığı piksel saat hızı ise 108 MHz olarak belirlenmiştir. Sistemin çalışma hızı Formül 8.deki gibi hesaplanmaktadır. Formülde EYH ekran yenileme hızını temsil ederken toplam yatay piksel ve toplam dikey piksel değerleri ise aktif pikseller ile boşluk pikselleri ve senkronizasyon piksellerinin toplamından oluşur [40].

$$\text{EYH} = \frac{\text{Piksel Saat Frekansı (Hz)}}{(\text{Toplam Yatay Piksel+ Toplam Dikey Piksel})} \quad (8)$$

$$\text{EYH} = \frac{108\,064\,000}{[(1280+29+192+85)\ x\ (1024+0+3+35)]} = 64{,}15 \quad (9)$$

Buna göre görüntü işleme sisteminin 1280 x 1024 çözünürlükte 60 Hz yenileme süresinde çalışabildiği ve böylece saniyede 60 kare görüntü işleyebildiği belirlenmiştir [41]. Bu durum sistemin gerçek zamanlı olarak çalıma şartını sağladığını göstermektedir.

Çizelge 1. Sistemin kaynak kullanım tablosu (System resource utilization table)

Birim	Mevcut	Kullanılan	Kullanım Oranı (%)
Toplam LUT	53200	18646	35,05
Mantıksal LUT	53200	17446	32,79
Hafıza LUT	17400	1200	6,90
Toplam Flip-Flop	106400	24336	22,87
Blok Hafıza	140	35	25,00
Blok DSP	220	29	13,18

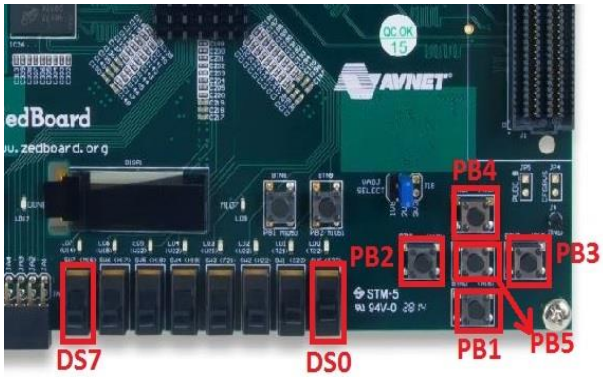
4.2. Sistem Yazılımının Gerçeklenmesi (Implementation of the System Software)

Donanım tasarımının ardından sistem yazılımı donanıma uygun olarak tasarlanmıştır. Tasarım Vivado Suite içerisindeki SDK aracı kullanılarak C dilinde gerçekleştirilmiştir. Yazılımda kullanılan IP bloklarının hazırlanması ve çalıştırılması, giriş-çıkış birimlerinin ayarlanması gibi işlemler yanında filtre parametreleri de ayarlanmıştır. Geliştirilen sistemde filtreler çalışma anında değiştirilebilmekte ve birden fazla filtre aynı anda aktif olabilmektedir. Sistemin kontrolü Zedboard üzerindeki buton ve anahtarlar ile yapılmaktadır. Filtrelerin aktiflik durumu yine Zedboard üzerindeki ledlerden izlenebilmektedir.

Sistem yazılımının akış şeması Şekil 17.de verilmiştir. Ana program döngüsünde filtre seçimi yapılan sistem çalışmaktadır. Burada çoklayıcı mantığı kullanılarak zedboard üzerindeki anahtarların durumuna göre ilgili filtrenin seçimi yapılacak şekilde sistem tasarlanmıştır.

Şekil 18.de görülen Zedboard üzerindeki anahtarlardan DS0 en düşük değerlikli bit, DS7 ise en yüksek değerlikli bit olarak kabul edilerek anahtarların durumlarına göre ilgili filtreyi aktif edecek biçimde sistem tasarlanmıştır.

Şekil 17. Sistem Yazılımı Akış Şeması (System Software Flow Chart)



Şekil 18. Zedboard Üzerindeki Buton ve Led Kontrolleri (Button and Led Controls on Zedboard)

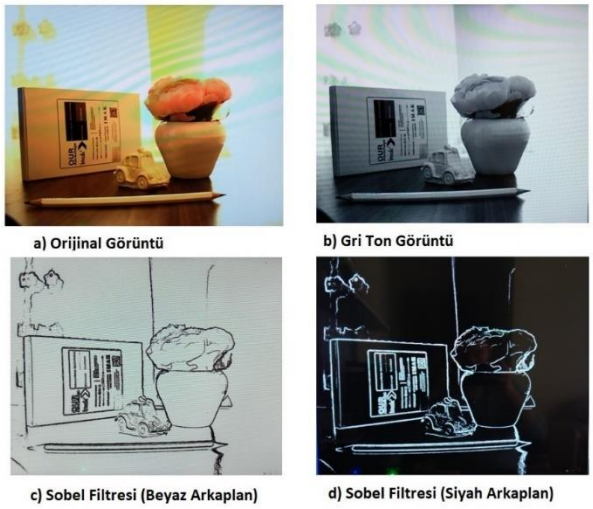
4.3 Çalışmanın Önceki Çalışmalarla Karşılşaştırması (Comparison With The Previous Works)

Yapılan çalışmanın ilgili çalışmalarda olan karşılaştırma tablosu Çizelge 2.de verilmiştir. Buna göre sistemler üzerinde kullanılan geliştirme kartları, görüntü boyutları ve kaynak kullanımları incelendiğinde çalışmada önerilen sistemin [11] numaralı çalışmaya göre performans olarak bariz biçimde üstün olduğu görülmüştür. Sistemin çözünürlüğü [6] numaralı çalışmaya göre daha düşük olmakla birlikte özellikle

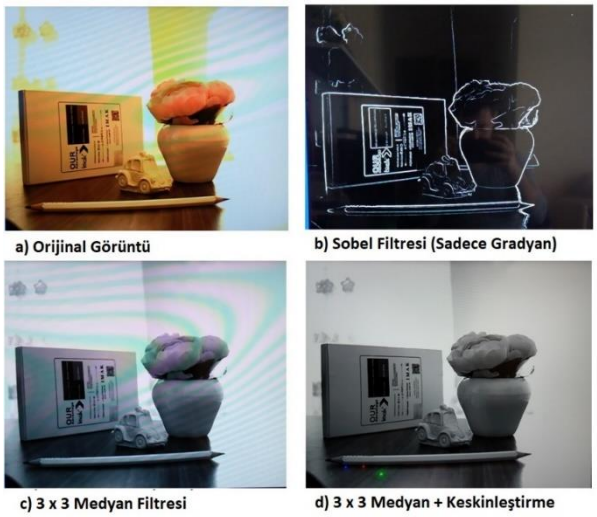
Çizelge 2. Sistemin diğer çalışmalarla olan karşılaştırma tablosu (Comparison table of the system with other studies)

Yayın No.	Sistem Tanımı	Platform	Metot	Görüntü Boyutu	FPS	Blok Hafıza	Toplam LUT	Flip-Flop	LUT Kullanım Oranı (%)
[6]	Canny Kenar Bulma	Zynq-7000 ZC702	Vivado HLS	1920*1080	60	21	34129	26393	65
[11]	Gri ton- Kenar Bulma ve Yumuşatma	Zynq-7000 Zedboard	HDL	256*256	40	97,5	2648	3652	4,98
Önerilen Sistem	Gri Ton-Kenar Bulma-Medyan-Keskinleştirme	Zynq-7000 Zedboard	HDL Coder	1280*1024	60	35	18646	24336	35,05

önerilen sistemdeki blok hafıza kullanımının daha yüksek, ancak toplam LUT ve Flip-Flop kullanımının daha düşük seviyede olduğu görülmektedir. Önerilen sistemde 1080p çözünürlüğe çıkılamamasının sebebi kullanılan kameranın bu çözünürlüğü destekleme-mesidir. Sistemin çalışma görüntüleri Şekil 19 ve Şekil 20.de verilmiştir.



Şekil 19. Gri Ton ve Kenar Bulucu Sonucu (Grayscale Convertor and Edge Detector Result



Şekil 20. Kenar Bulucu, Medyan ve Keskinleştirme Sonucu (Edge Detector, Median and Sharpening Filter Result

5. SONUÇLAR (CONCLUSION)

Bu çalışmada gerçek zamanlı bir görüntü işleme sistemi model tabanlı olarak tasarlanmış ve Zedboard geliştirme kartı üzerinde gerçeklenmiştir. Bu tip sistemlerdeki yüksek işlem gücü ihtiyacı nedeniyle çalışmada kullanılmak üzere Zynq mimarisi seçilmiş ve hem donanım hem de yazılım tasarımı bu sistem üzerinde yapılmıştır. Sistemin donanım ve yazılım tasarımında Xilinx şirketinin Zynq uyumlu olarak geliştirdiği Vivado Suite ve buna bağlı araçlar kullanılmış, görüntü işleyici sistemlerinin tasarımında ise Matlab/Simulink ve Mathworks şirketinin geliştirdiği HDL Coder, Vision HDL Toolbox gibi model tabanlı geliştirme araçları kullanılmıştır. Bu sayede elle HDL kodu yazımına gerek kalmamış tasarım süresi kısalmıştır. Çalışmada kullanılan Zedboard geliştirme kartı üzerinde 1280 x 1024 piksel çözünürlükte 60 Hz hızında çalışabilen bir sistem geliştirilmiştir. Sistemde uygulanan Sobel kenar bulma, ortanca filtre, gri ton dönüştürücü ve keskinleştirici filtre tasarımları kart üzerindeki giriş ve çıkış birimlerinden çalışma anında kontrol edilerek parametreleri değiştirilebilen esnek bir sistem ortaya konmuştur. Yapılan tasarımlar neticesinde sistemin ihtiyaçlara cevap verebildiği, gerçek zamanlı çalışma şartını yerine getirebildiği görülmüştür. Bununla birlikte kaynak kullanımının fazla yüksek olmadığı, ancak HDL Coder aracında ilgili en iyileştirmeler yapıldığı takdirde kullanımın daha da düşebileceği düşünülmektedir

Önerilen tasarım tamamen yeniden kullanılabilir olarak gerçekleştirilmiştir. Bu sayede daha ileri çalışmalarda ve farklı sistemlerde tasarlanan IP bloklarının ya da sistemin tamamının tekrar kullanılması mümkün olabilecektir. Gelecekteki çalışmalarda bu sistem ön işleme birimi olarak kullanarak daha gelişmiş gerçek zamanlı nesne tanıma ve takip uygulamalarının gerçekleştirilebileceği ön görülmektedir.

KISALTMALAR (ABBREVIATIONS)

AXI	Advanced eXtensible Interface (Gelişmiş genişletilebilir Arayüz)
DSP	Digital Signal Processing (Sayısal Sinyal İşleme)
FMC	FPGA Mezzanine Card (FPGA Ara Kartı)
FPGA	Field Programmable Gate Array (Alan Programlanabilir Kapı Dizisi)
FPS	Frames Per Second (Saniyedeki Kare Sayısı)
GPU	Graphics Processing Unit (Grafik İşleyici Ünite)
HDL	Hardware Description Language (Donanım Tanımlama Dili)
HDMI	High Definition Multimedia Interface (Yüksek Çözünürlüklü Çoklu ortam Arayüzü)
HLS	High Level Synthesis (Yüksek Seviyeli Sentez)
IP	Intellectual Property (Akıllı Özellik)
LUT	Look Up Table (Başvuru Çizelgesi)
RGB	Red Green Blue (Kırmızı Yeşil Mavi)
SDK	Software Development Kit (Yazılım Geliştirme Aracı)
SoC	System On - Chip (Yonga Üzeri Sistem)
VDMA	Video Direct Memory Access (Video Doğrudan Bellek Erişimi)
YUV/YCbCr	Luminance, Chrominance Blue, Chrominance Red (Parlaklık, Kırmızı Renklilik, Mavi Renklilik)

KAYNAKLAR (REFERENCES)

[1] Kahraman, E., Ünal, V., "Gerçek Zamanlı Gömülü Sistem ve Yazılım Tasarımı'nda ASELSAN Yaklasımı", ***EMO - III.Ulusal Yazılım Mühendisliği Sempozyumu UYMS***, Ankara, (2007).

[2] http://processors.wiki.ti.com/index.php/DaVinci, Erişim (01.10.2018)

[3] https://en.wikipedia.org/wiki/OMAP, Erişim (01.10.2018)

[4] https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/s32-automotive-platform/vision-processor-for-front-and-surround-view-camera-machine-learning-and-sensor-fusion-applications:S32V234, Erişim (01.10.2018)

[5] https://en.wikipedia.org/wiki/Digital_signal_processor, Erişim (01.10.2018)

[6] Abdelgawad, H. M., Safar, M., ve Wahba, A. M., "High level synthesis of canny edge detection algorithm on Zynq platform", ***Int. J. Comput. Electr. Autom. Control Inf.*** *Eng*, 9(1): 148-152, (2015).

[7] Crockett, L. H., Elliot, R. A., Enderwitz, M. A., & Stewart, R. W., "*The Zynq Book: Embedded Processing with the Arm Cortex-A9 on the Xilinx Zynq-7000 All Programmable Soc"*, **Strathclyde Academic Media**, 1st Edition, Glasgow, Scotland, UK. (2014).

[8] Russell, M., Fischaber, S., "OpenCV based road sign recognition on Zynq", ***Industrial Informatics (INDIN), 11th IEEE International Conference***, 596-601, (2013).

[9] Monson, J., Wirthlin, M., ve Hutchings, B. L., "Implementing high-performance, low-power FPGA-based optical flow accelerators in C", ***Application-Specific Systems, Architectures and Processors (ASAP), IEEE 24th International Conference***, 363-369, (2013).

[10] Sümer, Ö., "An Embedded Design And Implementation Of A Facial Expression Recognition System", ***Yüksek Lisans Tezi***, İstanbul Teknik Üniversitesi, Fen Bilimleri Enstitüsü, İstanbul, (2014).

[11] Altuncu, M. A., Güven, T., Becerikli, Y., ve Şahin, S., "Real-time system implementation for image processing with hardware/software co-design on the Xilinx Zynq platform", ***International Journal of Information and Electronics Engineering***, 5(6): 473, (2015).

[12] Chhabra, S., Jain, H., ve Saini, S., "FPGA based hardware implementation of automatic vehicle license plate detection system", ***Advances in Computing, Communications and Informatics (ICACCI), IEEE 2016 International Conference***, 1181-1187, (2016).

[13] Shi, Z., "Rapid Prototyping of an FPGA-Based Video Processing System", ***Yüksek Lisans Tezi***, Virginia Tech - Virginia Polytechnic Institute and State University, Computer Engineering, Blacksburg, Virginia, (2016).

[14] Al-Naqshbndi, S., "Hardware Acceleration of Computer Vision Application", ***Yüksek Lisans Tezi,*** Delft University of Technology, Faculty of Electrical Engineering, Mathematics and Computer Science, Netherlands, (2016).

[15] "HDMI Input/Output FMC Module with Camera Interface Hardware Guide", Rev. 1.0, Avnet Inc., (2015).

[16] "ON Semiconductor PYTHON 1300-C CAMERA MODULE Hardware User Guide", Ver. 1.0, Avnet Inc., (2015).

[17] https://www.xilinx.com/products/boards-and-kits/fmc-cards.html, Erişim (24.09.2018).

[18] "Zedboard Hardware User's Guide", Ver. 7.0, Avnet Inc., (2014).

[19] "Zynq-7000 All Programmable SoC Data Sheet: Overview (DS190)", Ver. 1.11, Xilinx Inc., (2017).

[20] "FMC-HDMI-CAM + PYTHON-1300-C Frame Buffer Design", Ver. 2015.2, Avnet Inc., (2015).

[21] "FMC-HDMI-CAM + PYTHON-1300-C Vivado HLS Reference Design", Ver. 2015.4, Avnet Inc., (2016).

[22] "HDL Coder Getting Started Guide", version 3.11 R2017b, Mathworks Inc., (2017).

[23] "Zynq-7000 All Programmable SoC Software Developers Guide (UG821)", Ver. 12.0, Xilinx Inc., (2015).

[24] "Xilinx Video and Image Processing Pack", https://www.xilinx.com/products/intellectual-property/ef-di-vid-img-ip-pack.html, Erişim (11.05.2018).

[25] "Avnet HDL Github Repository", https://github.com/Avnet/hdl/tree/master/IP, Erişim (11.05.2018).

[26] "Color Filter Array Interpolation v7.0 LogiCORE IP Product Guide (PG002)", Xilinx Inc., (2015).

[27] "Video On-Screen Display v6.0 LogiCORE IP Product Guide (PG010)", Xilinx Inc., (2015).

[28] "RGB to YCrCb Color-Space Converter v7.1 LogiCORE IP Product Guide. (PG013)", Xilinx Inc., (2015).

[29] "Video Timing Controller v6.1 LogiCORE IP Product Guide. (PG016)", Xilinx Inc., (2017).

[30] "Chroma Resampler v4.0 LogiCORE IP Product Guide. (PG012)", Xilinx Inc., (2015).

[31] "AXI Video Direct Memory Access v6.2 LogiCORE IP Product Guide. (PG020)", Xilinx Inc., (2016).

[32] "Matlab Online Documentation : Model Design for AXI4-Stream Video Interface Generation",https://www.mathworks.com/help/supportpkg/xilinxzynq7000/ug/model-design-for-axi4-stream-video-interface-generation.html, Erişim (11.05.2018).

[33] "Matlab Online Documentation : Getting Started with AXI4-Stream Video Interface in Zynq Workflow", https://www.mathworks.com/help/hdlcoder/examples/getting-started-with-axi4-stream-video-interface-in-zynq-workflow.html#hdlcoder_product-hdlcoder_ip_core_axi4_video, Erişim (11.05.2018).

[34] "Matlab Online Documentation : Design Video Processing Algorithms for HDL in Simulink", https://www.mathworks.com/help/visionhdl/gs/design-video-processing-algorithms-for-hdl-in-simulink.html, Erişim (11.05.2018).

[35] "Matlab Online Documentation : Streaming Pixel Interface", https://www.mathworks.com/help/visionhdl/ug/streaming-pixel-interface.html, Erişim (11.05.2018).

[36] "Vision HDL Toolbox Reference", version 1.5 R2017b, Mathworks Inc., (2017).

[37] https://en.wikipedia.org/wiki/Rec._601, Erişim (25.09.2018)

[38] https://en.wikipedia.org/wiki/Grayscale, Erişim (25.09.2018)

[39] "Image Processing Toolbox Reference", version 10.1 R2017b, Mathworks Inc., (2017).

[40] "Display Timing Calculation For Fujitsu MB86R01 SoC", https://www.fujitsu.com/downloads/MICRO/fme/displaycontrollers/an-mb86r01-display-timing-calc-rev1-20.pdf, Erişim (11.05.2018).

[41] http://www.epanorama.net/faq/vga2rgb/calc.html, Erişim (26.09.2018)