İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

TEMEL GÖRÜNTÜ İŞLEME ALGORİTMALARININ FPGA ÜZERİNDE GERÇEKLENMESİ

YÜKSEK LİSANS TEZİ

Mevlüt Mert ÇİL

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Elektronik Mühendisliği Yüksek Lisans Programı

MAYIS 2015

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

TEMEL GÖRÜNTÜ İŞLEME ALGORİTMALARININ FPGA ÜZERİNDE GERÇEKLENMESİ

YÜKSEK LİSANS TEZİ

Mevlüt Mert ÇİL
(504121370)

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Elektronik Mühendisliği Yüksek Lisans Programı

Tez Danışmanı: Prof. Dr. ECE OLCAY GÜNEŞ

MAYIS 2015

İTÜ, Fen Bilimleri Enstitüsü'nün 504121370 numaralı Yüksek Lisans Öğrencisi **Mevlüt Mert ÇİL**, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı **"TEMEL GÖRÜNTÜ İŞLEME ALGORİTMALARININ FPGA ÜZERİNDE GERÇEKLENMESİ"** başlıklı tezini aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı:	**Prof. Dr. ECE OLCAY GÜNEŞ** **İstanbul Teknik Üniversitesi**	.....
Jüri Üyeleri:	**Doç. Dr. Mürvet Kırcı** **İstanbul Teknik Üniversitesi**	.....
	Doç. Dr. Kamuran Nur Bekiroğlu **Yıldız Teknik Üniversitesi**	.....

Teslim Tarihi : **04 Mayıs 2015**
Savunma Tarihi : **05 Haziran 2015**

Bugünlere gelmemi sağlayan sevgili aileme ve
üzerimde emeği geçen hocalarıma,

ÖNSÖZ

Tez çalışmamda beni yönlendiren ve bana yardımcı olan değerli hocam Prof. Dr. Ece Olcay GÜNEŞ'e teşekkür eder saygılarımı sunarım.

Mayıs 2015

Mevlüt Mert ÇİL
Elektronik Mühendisi

İÇİNDEKİLER

Sayfa

KISALTMALAR

FPGA : Field Programmable Gate Array
ASIC : Application Specific Integrated Circuit
PAL : Programmable Array Logic
GAL : Generic Array Logic
PLA : Programmable Logic Array
PLD : Programmable Logic Device
EPROM : Erasable Programmable Read Only Memory
EEPROM : Electronically Erasable Programmable Read-Only Memory
EPLD : Erasable Programmable Logic Device
EEPLD : Electrically Erasable Programmable Logic Device
MAX : Multiple Array matriX
LoG : Laplacian of Gaussian
FIR : Finite Impulse Response
PROM : Programmable Read-Only Memory
CLB : Configurable Logic Block
LUT : Lookup Tables
SoC : System on Chip
HDL : Hardware Description Language

ŞEKİL LİSTESİ

Sayfa

TEMEL GÖRÜNTÜ İŞLEME ALGORİTMALARININ FPGA ÜZERİNDE GERÇEKLENMESİ

ÖZET

Projede görüntü işleme algoritmalarının çoğunda uygulanan temel algoritmalar FPGA üzerinde çalıştırılmıştır. Bu algoritmaların başında RGB gri dönüşümünün yapılması, sobel ve prewitt operatorleri, Gauss filtresi, LoG dönüşümü ve Canny kenar belirleme algoritmalarıdır. Buradaki algoritmalar diğer görüntü işleme algoritmalarının temelini oluşturması açısından önemlidir. Daha gelişmiş görüntü işleme algoritmaları çalıştırılırken önce bu algoritmalar uygulanır. Bu durum projede bu algoritmalarının konu olarak alınma amaçlarından biridir. Bu proje sonunda temel algoritmaların FPGA üzerinde çalışabildiği görülmüştür ve böylece bu tür algoritmaların üzerinde çalışabileceği bilgisayar dışında alternatif bir çalışma ortamının olduğu gösterilmiş oldu.

Projede ele alınıp kullanılan algoritmaların temel özelliği bir resimdeki kenarları bulma amacıyla tasarlanmış olmalarıdır. RGB gri dönüşümünün yapılması tipik olarak algoritmaların üzerinde çalışacağı verinin sadeleştirilmesi olarak kabul edilebilir. Bu dönüşüm sayesinde algoritmaların çalışmasında ihtiyaç bulunmayan kısımlar sadeleştirilmiş olur. Sobel ve prewitt operatorleri kenar bulma algoritmalarının en temel çeşitlerinden olup çalışma prensibi matematik alanındaki fonksiyonlardaki türev prensibine dayanır. Resimdeki pikseller birbirleriyle kıyaslandıklarında aralarındaki fark sobel ve prewitt operatorlerinin oluşturulmasını sağlayan temel dayanak noktasıdır. Gauss filtresi ve LoG dönüşümü de resim üzerinde bir filtreleme işlemi olarak düşünülebilir. Gauss filtresi Gauss dağılımı fonksiyonundan faydalanılarak oluşturulmuştur. Gauss filtresi resmin yumuşatılmasını ve bulanıklaştırılmasını işlevini gerçekleştirerek resimdeki gürültünün azaltılmasını sağlar. LoG filtresi ise resimdeki oval bölgeleri bulan bir algoritmadır. Canny algoritması ise yine türev işlemi prensibine dayanarak kenarların belirlenmesini sağlayan algoritmalardan biridir.

Projede kullanılan programlama dilleri C# ve Verilog'tur. Matlab ise algoritmaları anlamak ve proje sonuçlarını karşılaştırmak amacıyla yardımcı program olarak kullanılmıştır. Proje bilgisayar ve FPGA kısımlarından oluşmaktadır. Algoritmaların gerçeklendiği temel kısım Verilog ile yazılan FPGA kısmıdır. Bilgisayarda ise FPGA'e giriş verisi göndermek ve çıkış verisi almak için bir basit arayüz programı bulunur. FPGA'e gönderilen orijinal resim üzerinde arayüzde seçilen işlem uygulanır ve bu işlem sonucunun iyi olup olmadığı bilgisayara FPGA'den geri gönderilen resim aracılığıyla kontrol edilir. Elde edilen sonuçlar Matlab programı ile elde edilen sonuçlar ile karşılaştırılarak algoritmaların çalıştığı kanıtlamaktadır.

IMPLEMENTING IMAGE PROCESSING ALGORITHMS ON FPGA

SUMMARY

In this project the basic algorithms that are used in most of the image processing have been implemented on FPGA. These algorithms include RGB to grayscale transformation, sobel and prewitt operators and Gaussian filter, LoG transformation and Canny edge detection algorithm. The important point is that these algorithms form the base of other image processing algorithms. While more advanced algorithms are running, these algorithms are applied in the initial phase. This case is one of the reasons that these algorithms are taken as the subject of the project. At the end of this project it is seen that these basic algorithms can run on FPGA thus showing that it is possible to run this kind of algorithms in an alternative work environment other than a computer.

FPGA is one of the several programming logic devices as of today. The others are simple programmable logic devices, complex programmable logic devices and field programmable interconnect. The main differences among them are the type and the number of the logical blocks they include and how they are connected through using wires.

The common feature of the algorithms that are chosen to be implemented in the project is that they are designed to find the edges in an image. RGB to grayscale transformation can be typically considered to be the simplification on the data size that the algorithms will run on. Thanks to this transformation, the parts, which are not needed for the algorithm while it is running, will be reduced from the total data. Sobel and prewitt operators, which are two of the basic types of the edge detection algorithms, work based on the principle of derivative in mathematics. When the pixels in an image are compared with one another, the difference between them becomes the reference point to form sobel and prewitt operators. Gaussian filter and LoG transformation can be considered as a filtering process on an image. Gaussian filter is formed by the help of the Gaussian distribution function. Gaussian filter provides reduction in the noise in the image by smoothing it and making the image blurred. As for LoG transformation, it is a transformation algorithm that is used to find the blobs in an image. Canny algorithm that is again based on the principle of the mathematical derivative function is one of the algorithms that are used for the edge detection in an image. The reason Canny algorithm is chosen for the project is that it includes the basics what a full fledged edge detection algorithm should have with a minimalistic approach. This makes it suitable for the algorithm to be implemented on an FPGA.

The programming languages, which are used in the project, are C# and Verilog. Verilog is chosen as the hardware definition language for the FPGA due to its similarity to the general purpose programming languages such as C and C++ in syntax making it easier to get used to. However, it is advised that larger system designs should be made using VHDL langauge as VHDL makes it clear that the lines

in the code are working simulataneously and also makes the structure of the architecture easier to see and configure further. Matlab is used as a supportive way in order to make it easier to comprehend the algorithms and to compare the results of FPGA with the results that are acquired from Matlab. The project consists of mainly two sub divisions. One of them is the part that runs on the computer and the other one is the one that runs on the FPGA. The sub part that was implemented on the computer has a simple user interface in order to send input data to the FPGA and read the output data from the FPGA. The user interface is used for choosing the operation that will be carried out on the image that has been sent to the FPGA and after the operation, the result of the operation is checked by the help of the image that has been returned from the FPGA through the user interface. The acquired results will be compared with the results from Matlab to prove that the algorithms are working successfully.

In order to explain the details of the algorithms, RGB to grayscale transformation can be considered as the first step. An image when it is in RGB mode has 24 bits to depict each pixel. The first 8 bits are used for the saturation of the red color, the second 8 bits are used for the saturation of the green color and the last 8 bits are used for the saturation of the blue color. In an image that is in grayscale mode each pixel in the image is represented only by 8 bits that is considered as the saturation between black and white. In order to reduce 24 bits into 8 bits each of the 8 bits are multiplied by a constant. In this process, three constants are needed. The condition is that the sum of the constants should be equal to one. Therefore, these three constants can be regarded as percentage values of the intensity of the three colors that will be transferred into the grayscale. Constants are important because they can make one of the three colors dominant over the others. After trying different constants, it has been found that the most dominant color should be green and the second dominant color should be the red color. This provides the most number of details of the image in RGB mode to be transferred into the grayscale mode. It is also commonly thought that the green color works as the top part of the Gaussian distribution when the human eyesight is considered.

As it has been previously mentioned, sobel and prewitt operators are the results of the derivative operator in mathematics. Using the principle of derivatives, the edges, which consist of pixels where the difference between two pixels gets to the highest value, are emphasized more than the pixels in their proximity. There is a slight difference between sobel and prewitt operators. The only difference between them is the level of noise smoothing. Sobel and prewitt operators can also be divided into two. Each of them can configured to be more effective when finding the horizontal edges by putting the zero vector as the middle row of the filter matrix while it makes vertical edges more clear when the zero vector is taken as the middle column in the filter matrix. The other filter matrices are symmetrical with respect to both axes and to y = x line.

Gaussian filter is applied before sobel and prewitt operators as well as the Canny algorithm to reduce the noise in the image. It works identically to the Gaussian distribution function i.e. bell curve. The image pixels at the center of the Gaussian filter when the filter is being applied on an image are multiplied by the highest value while the pixels which are away from the center of the filter are multiplied by low values. Since all the pixels in the picture are involved in the process, this is valid for all the pixels. The overall result ends up to be slightly circularly blurred images. This

makes it easier for edge detection algorithms to run on the image later as the noise is reduced but the edges are still intact.

LoG transformation is similar to the Gaussian filter in terms of the operations to be applied on the image. Only the coefficients in the filter matrix are replaced with different ones. As it has been mentioned before, LoG transformation is useful for finding the blobs in the image. It can be used to count the heads in an image which includes people. In this example the Gaussian filter will be applied to the image a few times and then when the heads of the people are turned into blobs in the image, LoG transformation can detect them. Since it is also one of the basic algorithms in the field it can be implemented on devices with constraints. For instance, the recently released cameras have a feature of detecting the people from their heads which can be considered as a typical usage area of the LoG transformation.

Canny algorithm is a bit more specialized algorithm when compared to the filters. It has some hysteresis process inside the algorithms which tries to reduce the white noise further by reducing the false positives of the edge pixels. If it didn't have it, the operation would find edges somewhere in the image where there is actually no edge. It is actually an optimization process. If the edge is not clear enough, the algorithm may ignore it. In the project optimal values for the threshold values are found by trying the algorithm.

The implementation of the algorithms are carried through by the help of a pipelined FIR filter. A pipeline structure provides parallelism in the operations and reduces the total amount of time needed for the algorithms to be applied. The optimization is done in order to reduce the total time that the FPGA needs to process the images.

Lastly for the implementation part, the results of the FPGA implementation are compared with the Matlab output to see how the implementation works. The acquired results show us that the algorithms are working as expected. It has also been noticed that FPGA works faster than Matlab.

In order to evaluate the system that is in the project by taking performance, cost and environmental factors into account, since the implementation is done using an FPGA, the performance of the system is lower, the cost is higher and more vulnerable to outside effects when it is compared with a similar system on an ASIC. This project is at the FPGA prototype level which is just before the ASIC design level. If there is a process which can be done on FPGA, then depending on needs there can be made an ASIC version of the circuit. The main reason behind doing an FPGA design before ASIC design is that it is easier to debug the system and correct the errors before starting mass production for the ASIC design. Therefore, there will be some precautions taken in case of possible incremental costs and loss of time during the mass production if there is an error in the logic design.

The usage area of the project can include any application that requires image processing. Advanced image processing algorithms designed by being divided into smaller pieces in terms of functionality further enhances the modularity of the design that can be done on a larger project; therefore, each part of the system can be controlled in a more relaxed manner, and necessary maintenance and changes can be applied. Smaller units based on their positions can be designed in such a way that their cost will be lowered. Another reason that the system is split into smaller sections is to ensure that the system operates faster.

For the future as previously mentioned above, this project has the characteristics of being the initial process for various types of applications can be developed for other various purposes. More functionality than that has been incorporated into the project can be implemented by advancing the system further. An image processing software library can be created for FPGA.

1. GİRİŞ

Projede daha gelişmiş görüntü işleme algoritmaları için temel oluşturan görüntü işleme algoritmaları FPGA üzerinde gerçeklenmiştir. Böylece FPGA'in normalde bilgisayar üzerinde gerçeklenen bu tür algoritmalar için kullanılabileceği görülmüş olup ele alınan algoritmalar için daha basit devreler oluşturulabileceği gösterilmiştir.

1.1. Tezin Amacı

Tezde kullanılan algoritmalar görüntü işleme uygulamalarında kullanılan temel algoritmalardır. Çoğu görüntü işleme algoritmasının parçasını oluşturan bu temel algoritmaların FPGA üzerinde gerçeklenmesi ile hem FPGA üzerinde bir uygulama geliştirmiş olmak hem de bu algoritmalar için alternatif çalışma ortamı üretmek amaçlanmıştır. Böylece ele alınan algoritma için prototip niteliğinde olan FPGA tasarımı daha sonra daha az güç tüketen, maliyeti daha düşük kullanımı daha güvenli ve dayanıklı ASIC'e dönüştürülebilir.

1.2. Literatür Araştırması

FPGA elektronik tasarım ve programlama alanlarında son zamanlarda önemli bir çalışma ortamı sunmuştur. Matlab ile birlikte çeşitli görüntü işleme algoritmalarının gerçeklenmesinde önemli bir yere sahiptir. Literatürde FPGA üzerindeki kaynakların önceki dönemlerde kısıtlı olması nedeniyle piksel bazlı pencere kaydırma yöntemi ile bazı görüntü işleme algoritmaları gerçeklenebilir olmuştur (Nelson, 2000). Son dönemlerde ise paralel programlama yöntemlerinin gelişmesi ve daha geniş belleklerin daha az yer kaplaması FPGA üzerindeki bellek ve işlemci hızı gibi kısıtlamaları azaltmış ve dolayısıyla hızlı çalışabilen iş hattı yapısına sahip algoritma modelleme yöntemleri gerçeklenmiştir (Greisen, Heinzle, Gross, & Burg, 2011). Değişik görüntü işleme algoritmalarının gerçeklenebildiği bir çalışma ortamı sunan FPGA ile resimler üzerinde filtreler uygulabilir ve resimlerdeki gürültüler azaltılabilir hale gelmiştir (Maurya & Gupta, 2007). Bir diğer uygulama ise resimlerdeki optik akış vektörlerinin bulunması olup söz edilen FPGA'deki donanım

iyileştirmeleri, paralel programlama ve 2 boyutlu filtre matrislerinin tek boyutlu vektörler olarak daha küçük parçalara bölünmesiyle işlem performansında artış elde edilmiştir (Wei, Lee, & Nelson, 2007). FPGA üzerinde çalışmaların devam ettiği bu konuda çalışan insanların ilgi duyduğu güncel elektronik devre tasarım alanlarından birisidir.

2. TEORİK BİLGİ

Teorik bilgiler kısmında proje kapsamında kullanılan araçlar hakkında bilgiler, bu araçlardan biri olan FPGA hakkında açıklamalar ve konu dahilindeki algoritmaların anlatımı yer alacaktır.

2.1. Kullanılan Araçlar Hakkında Bilgiler

Projede kullanılan araçlar temel olarak yazılım ve donanım olarak ikiye ayrılabilir. Donanım kısmında FPGA ve bağlantı kabloları ile bilgisayar yer almaktadır. Yazılım kısmında ise algoritmayı gerçeklemek için kullanılan geliştirme ortamları ile algoritmaların daha iyi kavranıp incelenmesi için kullanılan geliştirme ortamları yer almaktadır. Algoritmaların gerçeklenmesi için kullanılan temel geliştirme ortamları Xilinx ISE Design Suite ile Microsoft Visual Studio'dur. Proje teması bir algoritmayı bir donanım üzerinde gerçekleştirmek olduğu için bu algoritmaları anlamak tasarlama işlemini kolaylaştıran bir adımdır. Bu amaçla algoritmalardan bir kısmı yardımcı olması açısından Matlab'da gerçekleştirilmiştir. Böylece FPGA'da gerçekleştirilen uygulama Matlab'daki ile karşılaştırılabilir hale gelmiştir.

2.2. FPGA Hakkında Bilgiler

Bugün çeşitli programlanabilir mantık mimarileri mevcut olarak bulunmaktadır. Her mimarinin genellikle satıcıya özgü çeşitleri vardır. Başlıca türleri şunlardır:

- Basit Programlanabilir Lojik Cihazlar (SPLDs)
- Kompleks Programlanabilir Lojik Cihazlar (CPLDs)
- Sahada Programlanabilir Kapı Dizileri (FPGAs)
- Sahada Programlanabilir Ara Bağlantılar (FPICs)

Basit programlanabilir lojik cihazlara örnek olarak PAL, GAL, PLA ve PLD verilebilir. Bunlar programlanabilir lojik cihazların en küçükleri ve en ucuzlarındandır. Çoğu SPLD fonksiyonların tanımlanması sırasında sigorta veya EPROM, EEPROM ve FLASH gibi kalıcı hafıza hücreleri anahtarlama işlemi için

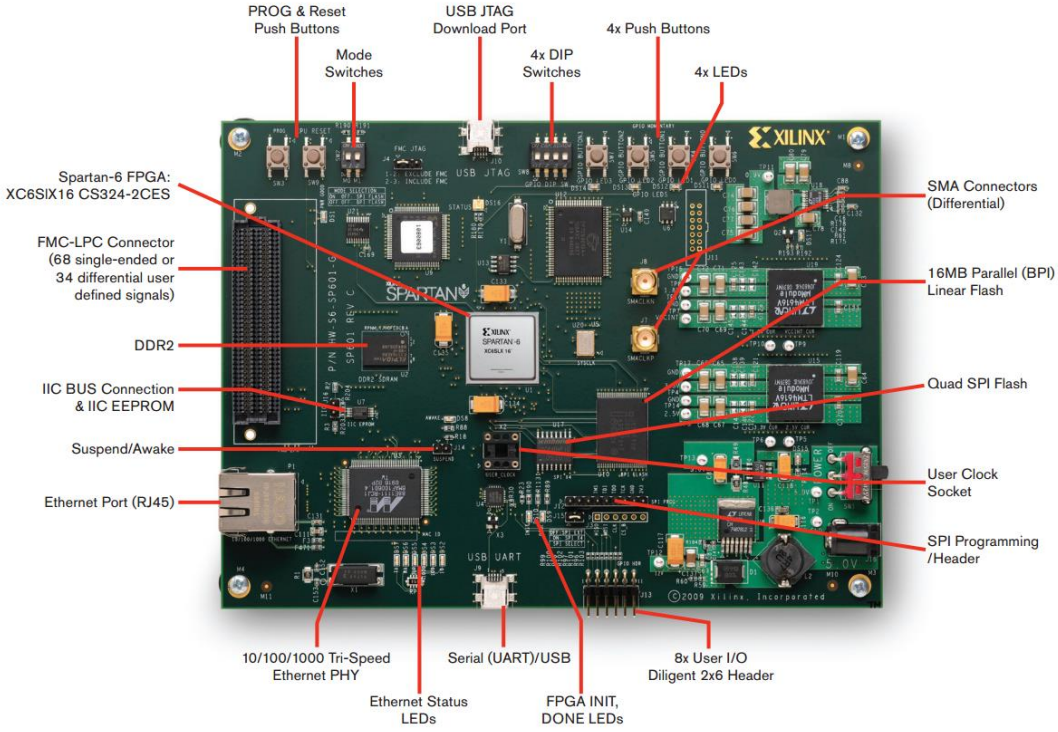
kullanılır. Kompleks programlanabilir lojik cihazlar basit programlanabilir lojik cihazlara oldukça benzerdir sadece daha yüksek kapasiteye sahiptir. Normal bir CPLD 64 adet SPLD'ye denk gelir. CPLD'lerdeki fonksiyon blokları tamamıyla bağlantılı olmayabilir. Genel olarak CPLD'lerde bir programlanabilir anahtar matrisi vasıtasıyla birbirine bağlanmış PAL benzeri lojik bloklar bulunur. CPLD'ler pinden pine olan aktarımda hızlı performansa sahip olduğundan kontrole dayalı tasarımlarda en iyi seçimdir. Yapısındaki giriş yelpazesinin geniş olması karmaşık, ve yüksek performanslı durum makinaları oluşturmalarına uygun olmasını sağlar. FPGA SPLD ve CPLD'lerden farklıdır ve en yüksek lojik kapasiteyi sunarlar. FPGA'de giriş çıkış blokları ile çevirli ve birbirine programlanabilir bağlantılar ile bağlanmış bir dizi lojik blok bulunur. Normal bir FPGA 64 adetten on binlerce adete kadar lojik bloka ve bunlardan çok daha fazla flip-flopa sahip olabilir. Çoğu FPGA pahalı bir işlem olması sebebiyle içerisindeki lojik blokların tamamı arasında bağlantı bulundurmazlar. Bunun yerine gelişmiş yazılımlar PCB'deki otomatik rotalayıcının parçaları yerleştirdiği gibi lojik tasarımı cihaza yerleştirme ve rotalama işlemini yapar. Günümüzde en yüksek yoğunluklu FPGA'ler mikroişlemciler gibi SRAM teknolojisi kullanılarak yapılır. SRAM tabanlı cihazlar doğal olarak sistem içerisinde dahi yeniden programlanabilirler, fakat bunun için bir çeşit harici hafıza kaynağına ihtiyaç duyarlar. Bu hafızada her lojik bloğun işlevini, giriş çıkış bloklarından hangilerinin giriş hangilerinin çıkış olduğunu ve blokların birbirlerine nasıl bağlandığını belirten program yer almaktadır. FPGA yapılandırmayı belirten hafızayı kendi kendine yükler veya harici bir işlemci tarafından bu bilgiler FPGA'e indirilir. FPGA bu işlemi kendi kendine yaparken FPGA bir işlemcinin önyükleme PROM'unu adreslemesi gibi bir byte genişliğindeki PROM'u adresler veya ardışık erişimli seri PROM kullanır. FPIC'lere gelindiğinde ise FPIC bir lojik devresi değil fakat bir programlanabilir kablolama cihazı olarak kabul edilebilir. Programlanması sırasında, FPIC cihazdaki bir pini bir diğerine bağlayarak programlanabilir bağlantılar sağlar. FPIC'ler de FPGA'ler gibi SRAM programlama teknolojisini kullanır.

FPGA teknolojisinin ortaya çıkmasında PROM ve PLD teknolojileri rol oynamıştır. Fakat bunların programlanması donanım tabanlı programlamadır. Xilinx'in kurucu ortaklarından Ross Freeman ve Bernard Vonderschmitt ilk ticari FPGA olan XC2064'ü 1985'de icat etmiştir. XC2064 64 CLB ve 2 LUT'a sahiptir (Xilinx,

2003). 1980'lerde binler mertebesinde olan FPGA'deki kapı sayıları 2000'li yıllarda milyon mertebesine ulaşmıştır.

FPGA tarihsel olarak incelendiğinde ASIC'lere oranla daha yavaş, daha çok güç tüketen ve aynı işlem için daha çok alan gerektiren çözümler sunmuşlardır. Ancak yeni gelişmelerle birlikte güç tüketimi azalan ve daha hızlı çalışan FPGA'ler özellikle ürünün piyasaya sürülmesi, tasarımın kolayca değiştirilebilmesi ve ASIC tasarımında rastlanan karmaşıklığı içermemesi açısından daha çok tercih edilir hale gelmiştir. Yeni üretilen FPGA'lere mikroişlemcilerin ve bazı uygulamalarda gerekli olan analog dijital çeviricilerin eklenmesi ise FPGA'lerin performanslarının yanı sıra kullanım alanlarını da arttırmaktadır. Önceleri yalnızca telekomünikasyon ve ağ yapılandırılmasında kendine yer edinen FPGA'ler günümüzde otomotivden endüstriye sanayinin çeşitli alanlarında kullanılmaktadır ve hatta son kullanıcıya kadar ulaşmıştır.

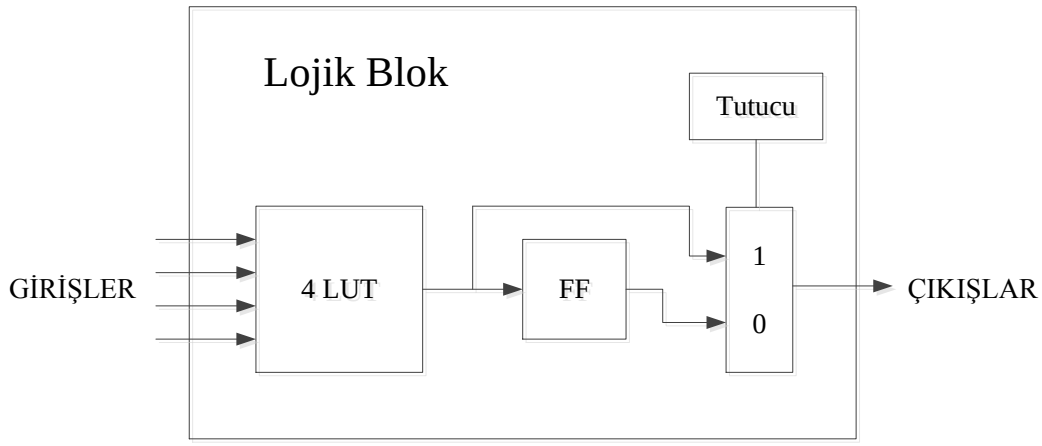
FPGA programlama sonucunda bir elektronik devre oluşur. Bu nedenle bu devrenin kullanılması sırasında sadece giriş uygulanması ve çıkışın alınması anlamını taşır. FPGA mikroişlemciler gibi koşturulacak programı anlamak için bir komut çevrimi bulundurmaz. Bu durum FPGA'deki tasarımların mikroişlemcilere göre daha hızlı çalışmasını sağlar.



Şekil 2.1: FPGA kartı.

Şekil 2.1'de projede kullanılan FPGA'in devre şematiği verilmiştir (Xilinx, 2009). Kullanılan bu FPGA'de 18224 adet saklayıcı, 9112 LUT ve 2176 adet hafıza bağlanma noktası bulunur.

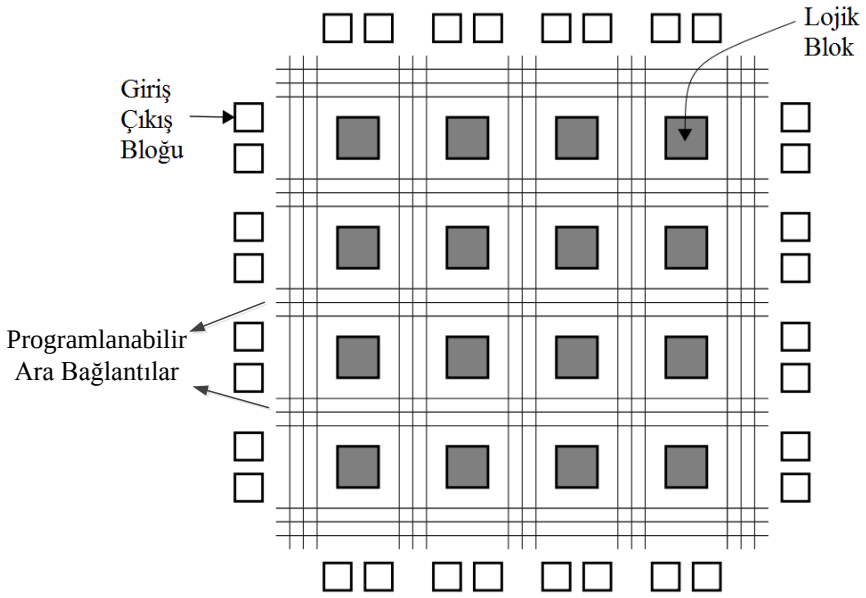
Xilinx tarafından üretilen FPGA'lerde bulunan lojik blok şeması Şekil 2.2'de verilmiştir:



Şekil 2.2: FPGA'deki lojik blok.

Lojik blok boyutları oldukça önemlidir. Geniş lojik bloklar blok içerisinde daha çok lojik gerçekleştirilmesine ve böylece FPGA içerisinde bir fonksiyon gerçeklenirken daha az sayıda lojik blok kullanılmasına olanak tanırken aynı zamanda bir lojik blok FPGA içinde daha çok yer işgal edecektir. Aktif lojik alanı genelde programlanabilir bağlantıların bulunması sebebiyle bütün lojik alandan daha ufaktır. Bütün lojik alan akrif lojik alanı ile programlanabilir bağlantıların yer aldığı alanın toplamıdır. LUT tabanlı FPGA'lerde 4 girişli bir LUT lojik sentezleme ve kullanılan alan açısından en iyi sonuçları vermektedir. Xilinx FPGA'leri SRAM tabanlı lojik bloklar içerir.

FPGA'in mimarisi Şekil 2.3'te verilmiştir. Bu mimari CLB adı verilen sütun ve satırlara matris şekilde dizilmiş ve aralarında bağlantı yolları bulunduran lojik elemanlardan oluşmuştur. Bu simetrik matris FPGA'in dış dünyayla iletişimini sağlayan giriş çıkış blokları ile çevrelenmiştir. Her CLB bir n girişli LUT ve bir çift programlabilir flip floptan oluşmaktadır. Giriş çıkış blokları ayrıca üç durumlu kontrol ve çıkış geçiş hızı gibi fonksiyonları kontrol eder. Ara bağlantılar rotalama için yol sağlar. Birbiri ile komşu lojik elemanlar arasındaki doğrudan ara bağlantılar genel amaçlı ara bağlantılara kıyasla daha az gecikmeye sahiptir.



Şekil 2.3: FPGA mimarisi.

FPGA'de tasarım yaparken HDL kullanılır. Temel anlamı donanım tanımlama dili olan bu kavram dahilinde yaygın olarak kullanılan iki adet dil vardır. Bunlar VHDL ve Verilog'tur. VHDL daha çok Avrupa kıtasında yaygın kullanım alanına sahip iken Verilog Amerika kıtasında yaygındır. Verilog dilinin sözdizimi C, C++, Java gibi genel amaçlı bilgisayar programlama dillerine oldukça benzerdir. HDL'in kullanım sırasında bu genel amaçlı bilgisayar programlama dillerinden en büyük farkı alt alta yazılmış birkaç satırın eş zamanlı olarak çalışabilmesidir. Dolayısıyla satır satır çalışan bilgisayar programlama dillerine aşina olan tasarımcıların Verilog'a geçince hata yapmalarının nedenleri arasında bu fark yer alır. VHDL ise donanım tasarım dili olarak nitelendirilmeye daha uygundur. Sözdiziminin bilgisayar programlama dillerinden farklı olması karışıklığı önler fakat aynı zamanda yazılım tasarımının diğer alanlarından gelen geliştiricilerin VHDL'e alışması daha çok zaman alır. Bu tür konular projenin piyasaya sürülme zamanını ve proje sırasındaki iş bölümünü oldukça etkileyen olgulardır.

2.3. Algoritmalar Hakkında Bilgiler

Projede gerçeklenen algoritmalar RGB gri renk dönüşümü yapılması, sobel ve prewitt operatorleri, LoG dönüşümü ve Canny kenar belirleme algoritmasıdır. RGB gri renk dönüşümü diğer algoritmaların parçası olarak ele alınmıştır. Bu bölümde algoritmalar hakkında teorik açıklama yapılacaktır.

Bir resim eğer RGB modda ise resimdeki her piksel 3 farklı byte ile temsil edilir. Kırmızı, yeşil ve mavi renk tonunu belirten bu 3 byte'ın birleşimi o pikselin rengini oluşturur. RGB moddaki resmi griye çevirirken bir piksel 3 byte yerine 1 byte ile temsil edilir hale getirilir. Dolayısıyla bu tek byte pikselin siyah beyaz ton skalasındaki tonunu belirtir ve bu tür byte'lardan oluşan resim gri modda olur.

RGB moddaki bir pikseli gri moda çevirmek için denklemleri oluştururken RGB moddaki resmin her biri 0-255 değerlerinden biri olan R, G ve B isimli 3 byte ile temsil edildiğini kabul edelim. Aşağıdaki denklemler ile RGB gri dönüşümü yapılır (**2.1, 2.2**):

$$x + y + z = 1 \qquad \textbf{(2.1)}$$

$$x * R + y * G + z * B = T \qquad \textbf{(2.2)}$$

x, y ve z katsayıları toplamları 1 olan 3 katsayıdır. Bu sayılar R, G ve B piksel değerleri ile çarpılıp T değeri elde edilir. Bu değer de 0-255 arasında olacaktır ve pikselin gri moddaki değerini verir. x, y ve z'nin alacağı değerlere göre iki farklı griye dönüştürme yöntemi bulunmaktadır. Bunlar ortalama değer yöntemi ve parlaklık yöntemi olarak isimlendirilebilir. Eğer x, y ve z birbirlerine eşit ise bu ortalama değer yöntemi ile griye dönüştürme denilir. Bu yöntemde R, G ve B değerleri grileştirme aşamasında aynı etkiye sahiptir. Eğer x = 0.21, y = 0.72 ve z = 0.07 değerlerinde olursa buna parlaklık yöntemi ile grileştirme denilir. y değeri en büyük değer olacağından RGB modlu resimdeki yeşil tonunun gri modda etkisi artacaktır. Diğer bir yöntem ise düzleştirme yöntemidir. Bu yöntemde T gri değeri **(2.3)** denklemi yardımı ile hesaplanır:

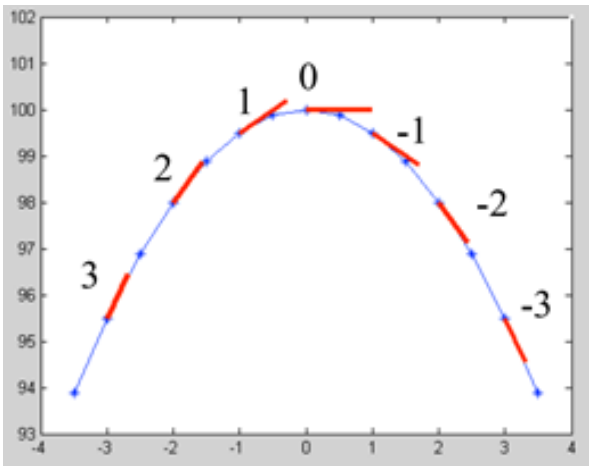
$$(maks(R,G,B) + \min(R,G,B))/2 = T \qquad \textbf{(2.3)}$$

R, G ve B değerlerinin en büyüğü ile en küçüğünün ortalaması alınarak T gri değeri bulunur. Bu yöntem birbirinden uzak değerleri birbirine yakınlaştıran bir yöntem olduğundan çıkan sonuç yumuşatılmış bir sonuçtur.

Sobel ve prewitt operatörlerinin anlaşılabilmesi için önce bazı açıklamalara ihtiyaç vardır. Gradyant, numerik türev ve konvolüsyon kavramları sırasıyla açıklanacaktır.

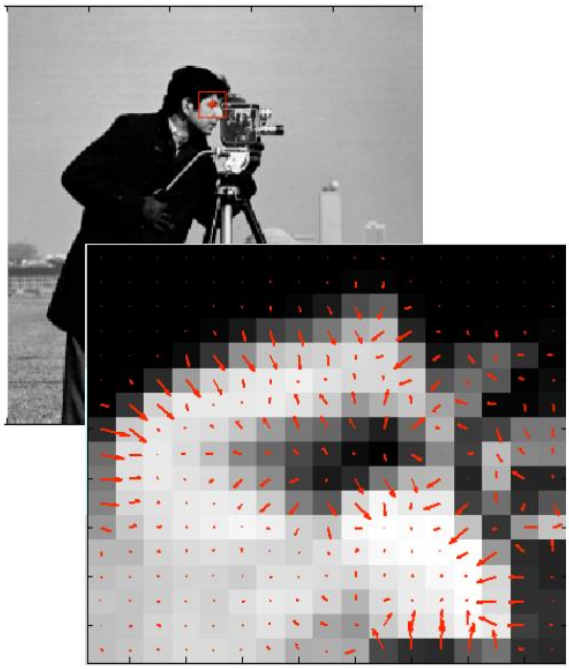
Gradyant bir denklemin bir değişkenine göre bir noktadaki türevi demektir. Dolayısıyla bir f(x) fonksiyonunun x'e göre belirli bir noktadaki türevi gradyantı

verir. Şekil 2.4'de görüldüğü üzere gradyant'ın bir yönü ve büyüklüğü vardır dolayısıyla gradyant vektörü olarak da adlandırılabilir:



Şekil 2.4: Gradyant.

Bir boyutlu türev tek değişkene göre alınan gradyanttır. İki boyutlu türev için iki değişkene ihtiyaç vardır. f(x,y) gibi bir fonksiyonda hem x'e göre hem de y'ye göre türev alınırsa iki boyutlu bir vektör elde edilir. Resim üzerindeki iki boyutlu gradyantlar resmin her pikselinde hesaplanır ve iki piksel arasındaki fark ne kadar büyük olursa gradyant da o kadar büyük olur. Şekil 2.5'te bir örnek bulunmaktadır:



Şekil 2.5: İki boyutlu gradyant örneği.

Numerik türev için Taylor serisinden faydalanılır **(2.4, 2.5)** (Collins & State, 2007a):

$$f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2 f''(x) + \frac{1}{3!}h3f'''(x) + O(h^4) \quad \textbf{(2.4)}$$

$$f(x-h) = f(x) - hf'(x) + \frac{1}{2}h^2 f''(x) - \frac{1}{3!}h3f'''(x) + O(h^4) \quad \textbf{(2.5)}$$

Denklem **(2.4)** düzenlendiğinde **(2.6)** ve **(2.7)** denklemleri elde edilir:

$$f(x+h) - f(x) = hf'(x) + \frac{1}{2}h^2 f''(x) + O(h^3) \quad \textbf{(2.6)}$$

$$\frac{f(x+h) - f(x)}{h} = f'(x) + O(h) \quad \textbf{(2.7)}$$

Denklem **(2.5)** düzenlendiğinde **(2.8)** ve **(2.9)** denklemleri elde edilir:

$$f(x) - f(x-h) = hf'(x) - \frac{1}{2}h^2 f''(x) + O(h^3) \quad \textbf{(2.8)}$$

$$\frac{f(x) - f(x-h)}{h} = f'(x) + O(h) \quad \textbf{(2.9)}$$

Denklem **(2.4)**'ten denklem **(2.5)** çıkarıldığı zaman denklem **(2.10)** elde edilir:

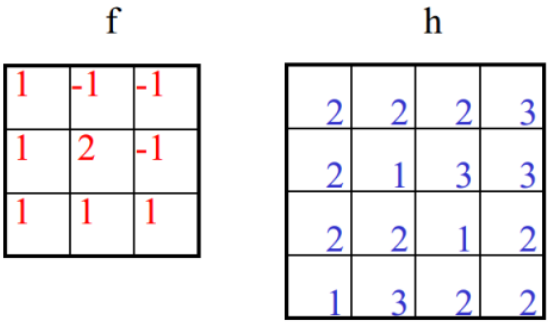
$$\frac{f(x+h) - f(x-h)}{2h} = f'(x) + O(h^2) \quad \textbf{(2.10)}$$

Denklem **(2.10)**'dan yararlanılarak resim içerisindeki pikseller için de türev alma işlemi tanımlanabilir (Collins & State, 2007a):

$$I_x = \frac{dI(x,y)}{dx} = \frac{I(x+1,y) - I(x-1,y)}{2} \quad \textbf{(2.11)}$$

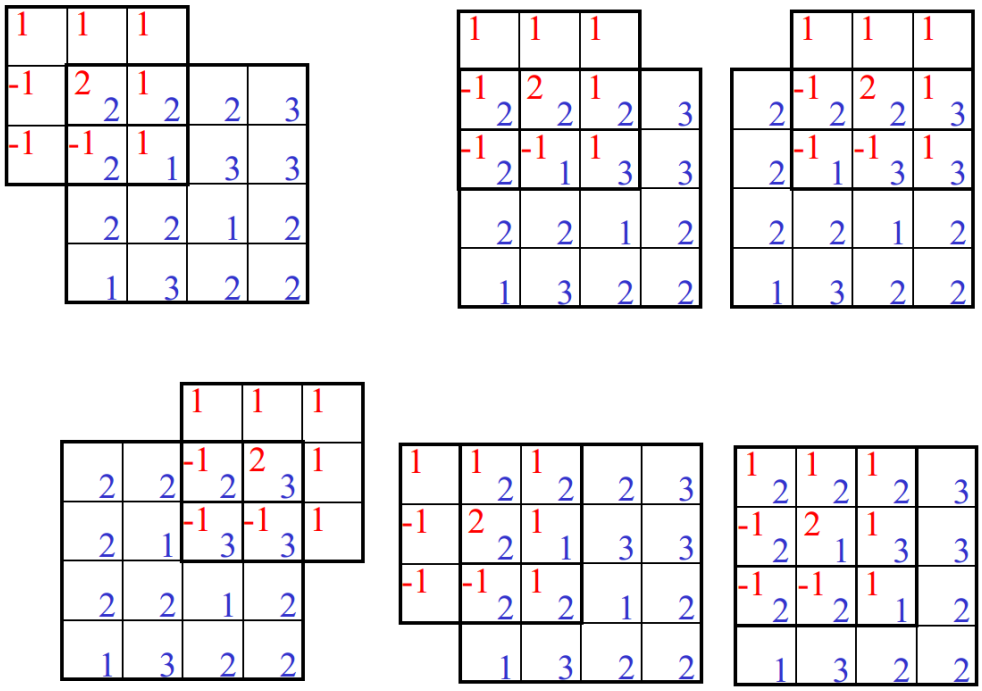
$$I_y = \frac{dI(x,y)}{dy} = \frac{I(x,y+1) - I(x,y-1)}{2} \quad \textbf{(2.12)}$$

Denklem **(2.11)** ve **(2.12)** numerik türevlerdir. Resim içerisindeki pikseller için bu işlemi sistematik bir biçimde yapabilmek için konvolüsyon işlemine ihtiyaç vardır. Konvolüsyon işlemi sırasında **(2.11)** ve **(2.12)** denklemlerinde bulunan 2 değerindeki payda kaldırılacaktır, yalnızca çıkarma işlemine yer verilecektir. Şekil 2.6'te verilen 2 adet matristen f filtreyi h ise resmi temsil etmektedir:



Şekil 2.6: f filtresi ve h resmi.

Konvolüsyon işlemi görsel olarak f filtresini h resmi üzerinde gezdirme manasına gelir. f filtresi üzerinde rotasyon yapıldıktan sonra elde edilen yeni matris ile h üzerinde Şekil 2.7'te görüldüğü gibi konvolüsyon uygulanır:



Şekil 2.7: Konvolüsyon işlemi.

Bu işlem uygulanırken kırmızı renkli sayılar örtüştükleri mavi renkli sayı ile çarpılır ve bu çarpım sonucu elde edilen sayıların hepsi toplanır. Sonuç f filtresinin ortasında yer alan h resmindeki pikselin yeni değeri olur. Sobel ve prewitt operatorleri işte bu f filtresinin yerine geçen operatorlerdir. **(2.11)** ve **(2.12)** denklemleri kullanılarak elde edilen sobel ve prewitt filtreleri Şekil 2.8 ve Şekil 2.9'de verilmiştir (Collins & State, 2007b):

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

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

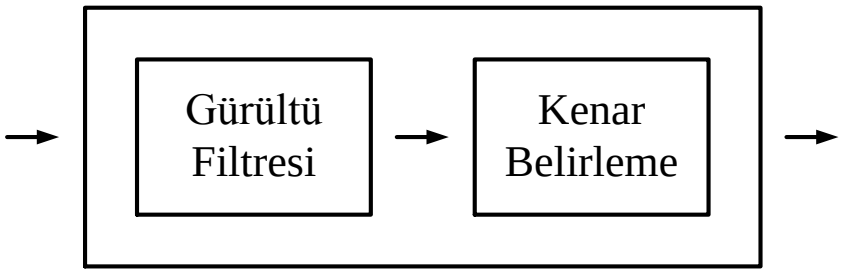
Şekil 2.8: Sobel filtreleri.

-1	0	1
-1	0	1
-1	0	1

-1	-1	-1
0	0	0
1	1	1

Şekil 2.9: Prewitt filtreleri.

Şekillerdeki sol taraftaki filtreler dikey doğrultudaki kenarları belirlerken yatay doğrultuda gürültünün azaltılmasını sağlar, sağ taraftaki filtreler ise yatay doğrultudaki kenarları belirlerken dikey doğrultuda gürültünün azaltılmasını sağlar. Bu filtrelerin içerisinde az da olsa bulunan gürültü temizleme özelliği dışında resimlere bu filtreler uygulanmadan önce Şekil 2.10'de görüldüğü üzere Gauss filtresi uygulanarak orijinal resimdeki gürültünün azaltılması sağlanır:

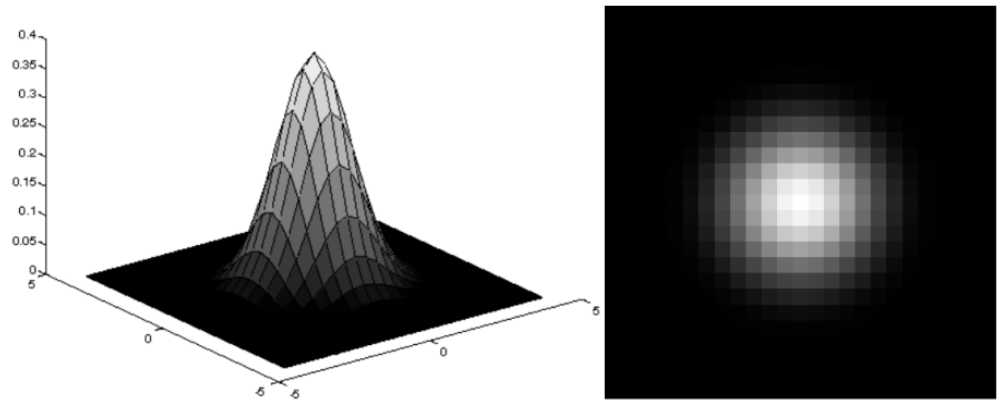


Şekil 2.10: Filtrelerin uygulanma sırası.

Gauss denklemi **(2.13)** denkleminde verilmiştir:

$$G_{\sigma} \equiv \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \qquad \textbf{(2.13)}$$

Bu denklemin resim üzerinde uygulanan filtreye dönüştürülmüş hali Şekil 2.11'da verilmiştir (Collins & State, 2007b):



Şekil 2.11: Gauss filtresi.

Gauss filtresinin σ değeri ve filtre matrisinin boyutları değiştirilerek Gauss filtresinin etkisi değiştirilebilir. Filtre matrisinin boyutları Şekil 2.11'in sağ tarafında verildiği gibi büyük de olabilir ya da 3x3 ve 5x5 gibi küçük boyutlarda da olabilir. Değişik σ değerlerinin etkisi Şekil 2.12'da görülebilir (Collins & State, 2007b):

Şekil 2.12: Değişik σ değerlerinin Gauss filtresi üzerindeki etkisi.

Sobel ve prewitt operatorlerinden sonra LoG dönüşümü için tekrar Taylor serisinden faydalanılır. Bu kez ikinci derece türev denklemleri kullanılır. Daha önce birbirinden çıkarılan denklem **(2.4)** ve **(2.5)** bu kez toplanır ve denklem **(2.14)** elde edilir:

$$f(x+h)+f(x-h)=2f(x)+h^2 f''(x)+O(h^4) \qquad \textbf{(2.14)}$$

Elde edilen denklem düzenlenir ve denklem **(2.15)** elde edilir:

$$\frac{f(x+h)-2f(x)+f(x-h)}{h^2}=f''(x)+O(h^4) \qquad \textbf{(2.15)}$$

(2.15) denkleminin sol tarafından [1 -2 1] katsayıları elde edilir. Böylece ikinci derece türev operatorleri denklem **(2.16)** ve **(2.17)**'deki gibi elde edilmiş olur (Collins, 2007):

$$I_{xx} = \frac{d^2 I(x,y)}{dx^2} = \begin{bmatrix} 1 & -2 & 1 \end{bmatrix} * I \qquad \textbf{(2.16)}$$

$$I_{yy} = \frac{d^2 I(x,y)}{dy^2} = \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} * I \qquad \textbf{(2.17)}$$

$I_{xx} + I_{yy}$ işlemi yapılarak denklem **(2.18)**'deki gibi LoG filtresi elde edilir:

$$I_{xx} + I_{yy} = \left(\begin{bmatrix} 1 & -2 & 1 \end{bmatrix} + \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} \right) * I = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} * I \qquad \textbf{(2.18)}$$

LoG resimdeki kenarları ve oval bölgeleri bulan bir filtredir. Şekil 2.13'deki gibi bir çıktı üretir:

Şekil 2.13: LoG filtre etkisi.

Canny algoritması aşağıdaki adımları içeren kenar belirleme algoritmalarından bir tanesidir:

- Gürültü azaltma
- Kenarı baskınlaştırma
- Kenar yerini tespit etme

John Canny optimal bir kenar belirleme algoritması yapmak için bu üç adımı formülize ederek kendi 3 prensibini oluşturdu. Bunlar iyi tespit, iyi yer saptama ve az sayıda yanlış pozitif olarak isimlendirilebilir. İyi tespit prensibi filtrenin kenar

üzerindeki cevabının gürültüden daha güçlü ve etkin olması anlamına gelmektedir. İyi yer saptama ise filtre cevabının kenara en yakın kısımda maksimum değeri vermesidir. Üçüncü prensip ise kenarın makul civarında yalnızca bir maksimum olması durumunu ifade eder. Canny algoritması verilen bu üç kriteri mümkün olan en iyi hale getiren lineer ve sürekli bir filtredir. Canny algoritmasının adımları verilmiştir (Canny, 1986):

1. Resmin dikey (x) ve yatay (y) türevleri Gauss filtresi de uygulanarak bulunur **(2.19)**:

$$I_x = G_\sigma^{\,x} * I \,,\; I_y = G_\sigma^{\,y} * I \qquad \textbf{(2.19)}$$

2. Her pikseldeki gradyantın büyüklüğü hesaplanır **(2.20)**:

$$M(x,y) = \left|\nabla I\right| = \sqrt{I_x^2 + I_y^2} \qquad \textbf{(2.20)}$$

3. Gradyantın yönünde hesaplanan büyüklüğün yerel maksimum noktası olmayan pikseller elenir.

4. Histerezis eşiklemesi yapılır:

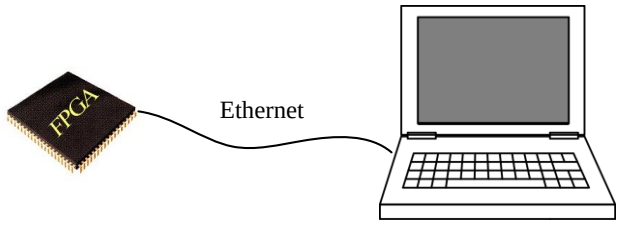
   - T_h yüksek eşikten büyük değerde olan tüm pikseller alınır ve kenar kabul edilir.
   - T_l düşük eşikten büyük değerde olan pikseller ancak kenar kabul edilen komşusu var ise alınır ve kenara dahil edilir.

3. ANALİZ VE MODELLEME

Analiz bölümünde problemin analizi yapılacak ve çözüm yolları açıklanacaktır ve modelleme bölümünde ise çözüme yönelik izlenen yol sistematik olarak anlatılacaktır.

3.1. Analiz

Problemin gerçek dünyadaki (problem domeni) bileşenleri bir bilgisayardan ve FPGA'den oluşmaktadır. Bilgisayarda basit bir arayüz ile FPGA'a resim gönderilmesi ve gelen resmin ekranda gösterilmesi sağlanır. FPGA ise kendisine gelen resmi işleyip geri bilgisayara gönderir. Sistem bileşenleri Şekil 3.1'de gösterilmiştir:



Şekil 3.1: Sistemin donanım beleşenleri.

Bilgisayar ile FPGA bir ethernet kablosu vasıtasıyla haberleşir. FPGA için gerekli besleme gerilimi ise prizden sağlanır.

3.2. Modelleme

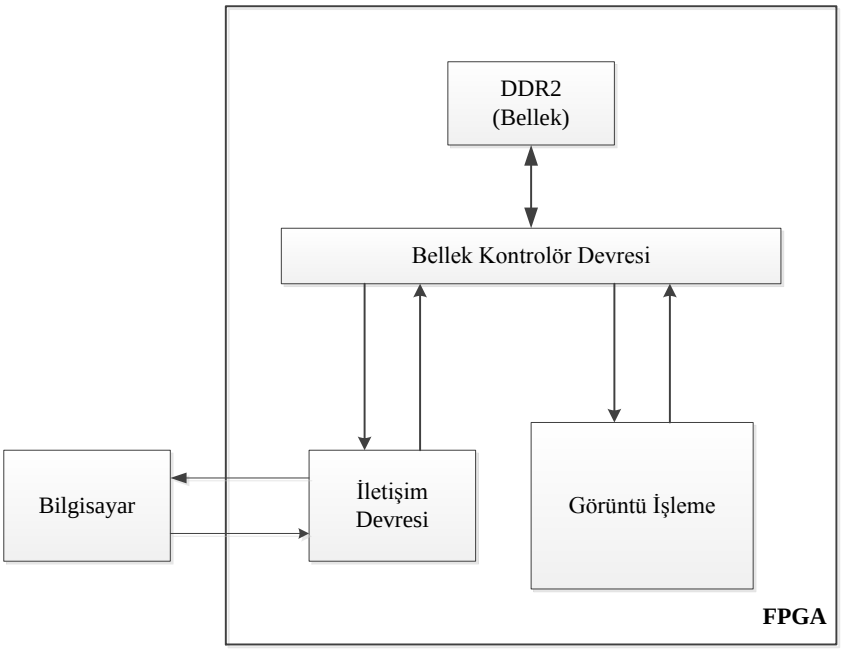
FPGA programlama için seçilen programlama dili C++'a benzer olması nedeniyle Verilog'tur. FPGA üzerinde bir donanım tasarım dili kullanılarak yapılan programlamada iki önemli programlama yöntemi bulunmaktadır. Bunlar yapısal programlama ve davranışsal programlamadır. Yapısal programlama FPGA programlama alanında düşük seviyeli programlama olarak kabul görür. Genelde mühendislerin bir sayısal devre tasarımını elle yaparken kullandıkları bu yöntem yapısal yöntemdir. Bu programlama yöntemi mühendisin tasarımının başka bir seçenek bulunmadan değişmeden FPGA'e aktarılmasını sağlar. Bu nedenle bu

yöntem görece küçük devrelerin tasarımında bu devrelerin efektif bir şekilde çalışmasını sağlamak için kullanılır. Davranışsal programlama daha yüksek seviye bir programlama olarak kabul görür. Davranışsal programlama daha esnektir ve mühendisin kapı kapı devre tasarımından ziyade sistemin davranışsal noktaları ile tasarım yapmasına olanak tanır. Bu yöntem daha büyük sistemlerin tasarlanmasını kolaylaştırır. Davranışsal programlamanın dezavantajı tasarımların FPGA üretici firmanın sağladığı yazılım geliştirme ortamının performansına bağlı olmasıdır. Programlama işlemi kod yazma ile yapıldığından tasarımcı sistemi tasarladıktan sonra yalnızca yazılım geliştirme ortamının sağladığı olanaklar dahilinde iyileştirme yapabilir. Bu durum her ne kadar tasarımcının asıl devrenin tasarımını yapma konusunda uzmanlaşmak yerine yazılım geliştirme ortamını kullanma konusunda yetkin hale gelmesini sağlasa da sayısal sistemlerin hızlı bir şekilde geliştirilmesinde bu amaç için üretilmiş yazılım geliştirme ortamları büyük rol oynar ve ayrıca mühendislere çok daha büyük sistemleri daha az bir çaba ile tasarlama imkanı tanır. Projede kullanılan yöntem genelde davranışsal yöntem olsa da bazı durumlarda tasarımın basitleştirilmesi için yapısal yöntem de kullanılmıştır.

Teorik bilgi edinildikten sonra bu bilgilerden yola çıkarak problem küçük parçalara bölünebilir. Sobel, prewitt, LoG, Gauss filtreleri ve Canny algoritmasındaki histerezis işlemleri konvolüsyon işlemi yardımıyla yapılabilir. Algoritmaların implementasyonu yanı sıra bilgisayar ve FPGA haberleşmesi de önemlidir. Bunların dışında işlemler yapılırken gerekli olan bellek kontrolü de proje kapsamında yer almaktadır.

Modelleme için projede kullanılacak sistemin özellikleri de göz önünde bulundurulmalıdır. Projede kullanılacak olan FPGA kartı Spartan-6 SP601'dir. İçerisinde bulunan 128 MB DDR2 bellek yapılacak projede kullanılacak hafıza miktarı için yeterlidir. Eğer yeterli olmasaydı FPGA üzerinde sağlanan bağlantı yollarıyla ihtiyaca yönelik ek bellek takılabilirdi.

Sistemin FPGA kısmındaki modeli Şekil 3.2'de verilmiştir. Şekil 3.2'de görüldüğü üzere bilgisayar ile FPGA arasındaki veri alışverişi FPGA'de tasarlanan bir modül sayesinde gerçekleşmektedir. Veri iletişim modülünden sonra bellek kontrolör devresi yardımıyla belleğe aktarılır. Daha sonra görüntü işleme işlemleri yapılır.



Şekil 3.2: Sistemin FPGA üzerindeki modeli.

Sistemin bilgisayar üzerinde çalışan kısmında ise basit bir arayüz programı mevcuttur. Bu arayüz programı sayesinde resim dosyaları FPGA'e aktarılır ve geri okunur. Bu program C# dili kullanılarak yazılmıştır.

4. TASARIM, GERÇEKLEME VE TEST

Tasarım bölümünde teorik bilgi, analiz ve modelleme bölümlerinde yapılan açıklamalardan da faydalanarak algoritmaların nasıl tasarlandığı anlatılacaktır. Gerçekleme ve test kısmında ise sistemin nasıl oluşturulduğu ve test edildiği hakkında açıklamalar yapılacaktır.

4.1. Tasarım

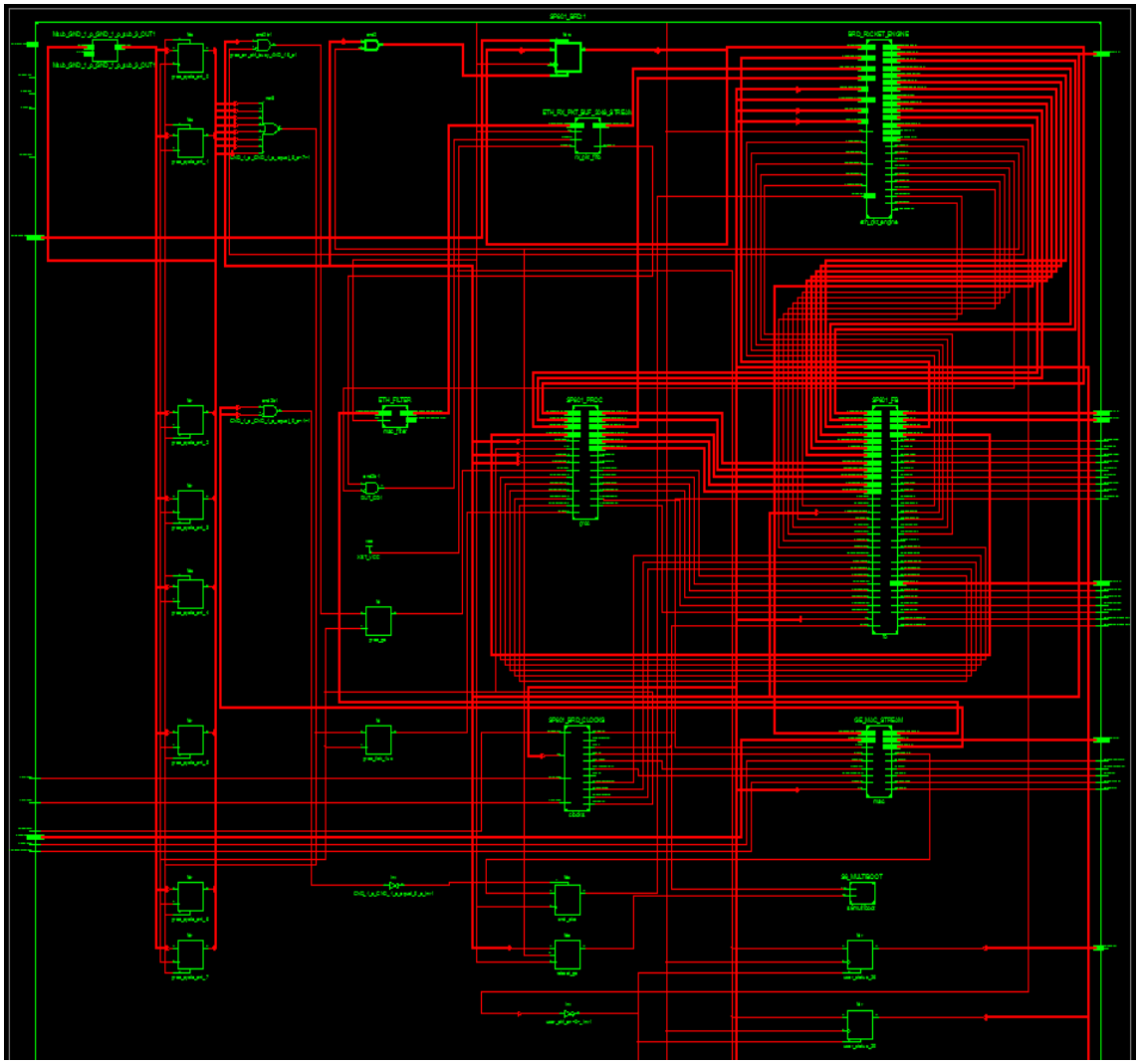
Projede gerçeklenen algoritmalarda da belirtildiği üzere sistemin ana bölümü için bir konvolüsyon modülü kurulmalıdır. Bu modül sayesinde konvolüsyon işlemi hızlı ve doğru bir biçimde gerçekleştirilebilir olmalıdır.

Sistemin tasarımı Şekil 3.2'deki tasarımdır. Bilgisayar bölümünde C# dili ile yazılmış arayüz programı bulunur. İletişim devresi ile bellek yönetimi ile ilgili bölümler kontrolör devreleridir. Görüntü işleme algoritmalarının çalıştığı bölüm görüntü işleme bölümü olarak verilmiştir. İletişim devresi ethernet kontrolör devresidir ve iletişim ethernet bağlantısı aracılığıyla olmaktadır. Bellek ve ethernet kontrolör devreleri programlanırken FPGA üreticisinin örnek olması amacıyla sunduğu sürücü kodlarından yararlanılmıştır. Görüntü işleme bölümünde kullanılan yöntem iş hattı yapısı ile çalışan klasik bir lineer FIR filtresidir. Yapılan işlem kısaca n x n boyutlarındaki bir filtre matrisinin A x B boyutlarındaki resim üzerinde gezdirilirken yapılan çarpma ve toplama işlemleridir. Oluşacak çıktıdaki bir piksel değeri n x n kadar çarpma işlemi yapılması ve yine bu kadar işlemden oluşan n x n adet değerin toplanmasıyla elde edilir. Dolayısıyla A x B boyutlarındaki bir resim için kabaca $A \times B \times n^2$ kadar çarpma işleminin yapılması gerekir. Çarpılan sayılar 8 bitlik olsa da çok sayıda olması bakımından işlem yükü getirmektedir. Dolayısıyla bir piksel için n x n adet işlemi yaparken bu işlemler iş hattı mantığı ile oluşturulmuş paralel bir yapı ile eş zamanlı hesaplanır. Sonuç olarak n adet satır için yapılan işlem bir satırın yapılmasındaki işlem süresi kadar zaman alır. Dolayısıyla sistemin karmaşıklığı $\Theta(A \times B \times n^2)$'den $\Theta(A \times B \times n)$'e düşürülmüş olur. n sayısı 3, 5 veya 7 gibi kullanılan kare filtre matrisinin boyutudur dolayısıyla küçük bir sayıdır fakat bu

sayının karesinin alınması işlem yükünü arttırır çünkü A ve B sayıları resmin boyutlarıdır ve n'den oldukça büyük olan değerlerdir. RGB'den griye çevirme işlemi kaydırma kullanılarak yapılan çarpma işlemi ve sonrasında toplama işlemi ile yapılır ve işlemin karmaşıklığı Θ(A x B) kadardır. Konvolüsyon işlemi projede yapılan filtre algoritmalarının ortak noktasıdır ve bir kez tasarlanması halinde değişik filtreler için de kullanılabilmektedir. Canny algoritması kısmında bulunan histerezis eşikleme işlemi de yine aynı şekilde matrissel hale getirilerek yapılabilir. Bunun için ufak bir karşılaştırma mantığı kurulmuştur. Böylelikle bu işlem de konvolüsyon işlemindeki gibi filtre matrisini gezdirerek gerçekleştirilebilmiştir.

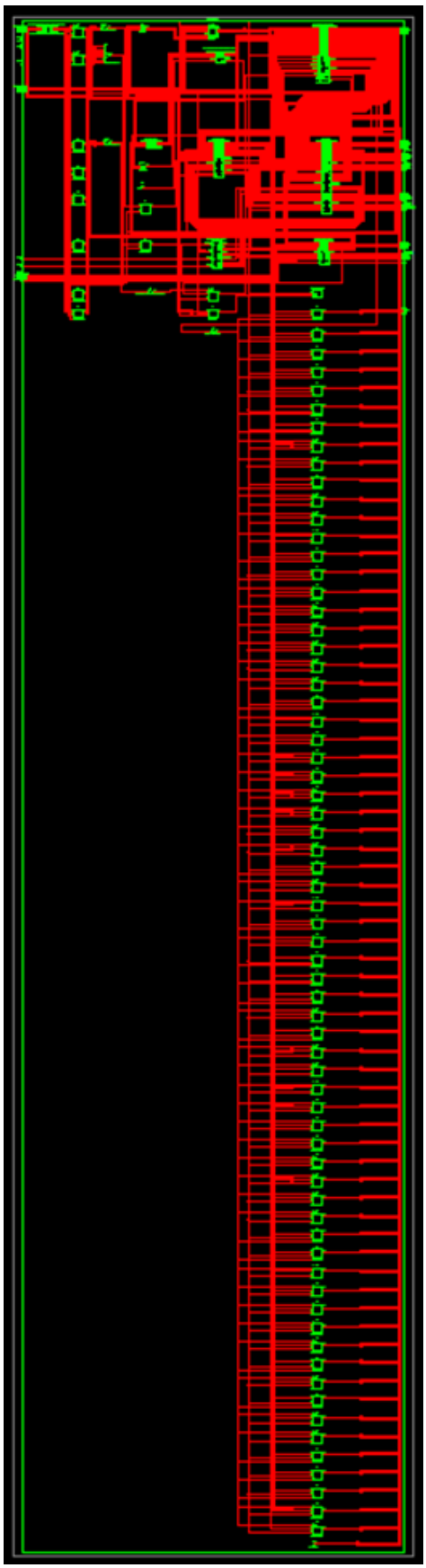
4.2. Gerçekleme ve Test

Sistemin gerçeklenmesi tasarım bölümünde açıklandığı şekliyle yapılmıştır. Elde edilen RTL şematiğinin üst kısmı Şekil 4.1'de verilmiştir:



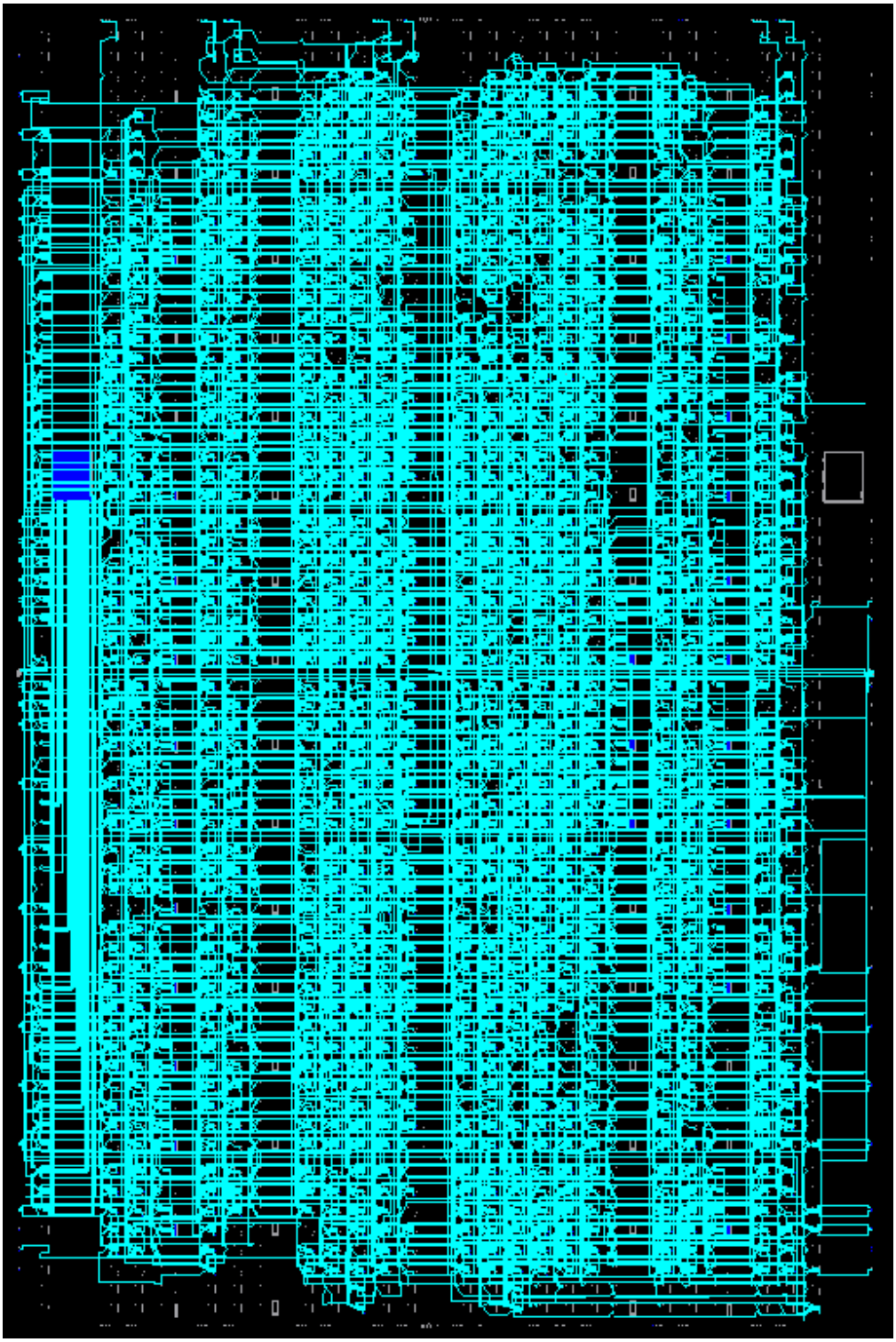
Şekil 4.1: Sistemin RTL şematiği.

Bütün RTL şematiği Şekil 4.2’de verilmiştir:



Şekil 4.2: Bütün RTL şematiği.

Yerleştirme ve rotalama şematiği Şekil 4.3'te verilmiştir:



Şekil 4.3: Yerleştirme ve rotalama.

Şekil 4.3'te görüldüğü üzere üretilen devre FPGA üzerinde alansal olarak yayılmıştır. Bunun nedeni yapılan iyileştirmenin gecikmenin en az olacak şekilde gerçekleştirilmesidir.

Entegre geliştirme ortamındaki gerçekleme sonuçları Şekil 4.4'te verilmiştir:

Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	3,614	18,224	19%
Number of Slice LUTs	4,583	9,112	50%
Number of occupied Slices	1,548	2,278	67%
Nummber of MUXCYs used	3,052	4,556	66%
Number of LUT Flip Flop pairs used	5,156		
Number with an unused Flip Flop	1,997	5,156	38%
Number with an unused LUT	573	5,156	11%
Number of fully used LUT-FF pairs	2,586	5,156	50%
Number of unique control sets	138		
Number of slice register sites lost to control set restrictions	290	18,224	1%
Number of bonded IOBs	90	232	38%
Number of RAMB16BWERs	22	32	68%
Number of RAMB8BWERs	0	64	0%
Number of BUFIO2/BUFIO2_2CLKs	3	32	9%
Number of BUFIO2FB/BUFIO2FB_2CLKs	0	32	0%
Number of DCM/DCM_CLKGENs	0	4	0%
Number of STARTUPs	0	1	0%
Number of SUSPEND_SYNCs	0	1	0%

Şekil 4.4: Tasarım sonuçları.

Sistemin test edilmesi ile ilgili olarak basit toplama ve çarpma işlemleri için elde edilen sonuçlar incelenerek birim testler yapılmıştır. İstenmeyen sonuçlar görüldüğünde ilgili bölüm tekrar incelenmiştir. Sistemin test edilmesini kolaylaştırması açısından ağırlık iletişim kontrolör devresine verilmiştir. Böylece FPGA'den çıktı olarak alınan resim incelenerek hataların daha kolay bulunması amaçlanmıştır.

5. DENEYSEL SONUÇLAR

Deneysel sonuçlar bölümünde elde edilen sonuçlar şekiller halinde verilecektir. Bu şekiller üzerinde yorumlar yapıldıktan sonra aynı işlemler Matlab'da yapıldıktan sonra elde edilen resimler verilerek FPGA çıktıları ile Matlab çıktıları karşılaştırılacaktır.

5.1. Elde Edilen Sonuçlar

Bu bölümde elde edilen sonuçlar kullanılan algoritmalara göre verilecektir. Kullanılan orijinal resim görüntü işleme algoritmalarında standartlaşmış resim olan Şekil 5.1'de görülen kameraman resmidir:

Şekil 5.1: Kameraman resmi.

Sobel ve prewitt operatorleri ilk gösterilen algoritmalardır. Bu iki algoritmada kenar belirleme doğrultusu dikey veya yatay doğrultuda olabilir. Bu nedenle bu iki algoritmada kullanılan filtre de verilecektir. Kullanılan sobel operatorü ve bu operatorün kameraman resmi üzerindeki etkisi Şekil 5.2 ve Şekil 5.3'te verilmiştir:

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

Şekil 5.2: Yatay eksen için sobel operatorü.

Şekil 5.2'de verilen yatay eksen için olan sobel operatorü yatay eksen üzerinde gidildiğinde belirginleşen dikey doğrultuda uzanan kenarları belirtir, yatay doğrultuda ise hafif bir yumuşatma etkisi yapar. Bu operator kameraman resmi üzerinde uygulandığında Şekil 5.3'te görülen resim elde edilir:

Şekil 5.3: Yatay eksen sobel operatorünün kameraman resmi üzerindeki etkisi.

Şekil 5.3'te görüldüğü üzere kameraman resmindeki dikey doğrultuda uzanan kenarlar belirginleşir ve yatay doğrultuda oluşan yumuşatma etkisi ile gürültüde azaltma görülür. Şekil 5.4'te görülen sobel operatorü ise dikey eksen için olan sobel operatorüdür:

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

Şekil 5.4: Dikey eksen için sobel operatorü.

Şekil 5.5'te dikey eksen için olan sobel operatorü kameraman resmine uygulanmıştır:

Şekil 5.5: Dikey eksen sobel operatorünün kameraman resmi üzerindeki etkisi.

Şekil 5.5'te görüldüğü üzere dikey eksen için olan sobel operatorü resme uygulandığında resimde yatay doğrultuda uzanan kenarlar belirginleşir ve dikey doğrultuda yumuşatma olduğu için gürültüde azalma gözlenir.

Sobel operatöründen sonra gelen prewitt operatorü için de uygulanacak işlemler oldukça benzerdir. Prewitt operatorlerinde tek fark 0 olmayan satır veya sütunların her zaman 1 değerinde olmalarıdır. Dolayısıyla sobel operatorüne kıyasla yumuşatma etkisi az da olsa farklılık gösterir. Yatay eksen için olan prewitt operatorü Şekil 5.6'da gösterilmiştir:

-1	0	1
-1	0	1
-1	0	1

Şekil 5.6: Yatay eksen için prewitt operatorü.

Prewitt operatorü kameraman resmi üzerinde uygulanır ve elde edilen resim olan Şekil 5.7'de görüldüğü gibi dikey doğrultaki kenarlar belirginleşmiş ve yatay doğrultuda bir yumuşatma işlemi olmuştur:

Şekil 5.7: Yatay eksen prewitt operatorünün kameraman resmi üzerindeki etkisi.

Dikey eksen için olan prewitt operatorü Şekil 5.8'de gösterilmiştir:

-1	-1	-1
0	0	0
1	1	1

Şekil 5.8: Dikey eksen için prewitt operatorü.

Bu operator kameraman resmi üzerinde uygulandığında Şekil 5.9'daki resim elde edilmektedir:

Şekil 5.9: Dikey eksen prewitt operatorünün kameraman resmi üzerindeki etkisi.

Şekil 5.9'da görüldüğü üzere dikey eksen için olan prewitt operatorü resme uygulandığında resimde yatay doğrultuda uzanan kenarlar belirginleşir ve dikey doğrultuda yumuşatma olduğu için gürültüde azalma gözlenir. Prewitt operatorünün sobel operatoründen farkı bu şekilde belirgin olarak gözlenebilmektedir.

Sobel ve prewitt operatorlerinden sonra sırada LoG dönüşümü vardır. LoG dönüşümü de gerçekleme sırasında sobel ve prewitt operatorleri gibi bir filtre matrisi yardımıyla gerçekleşir. Bu matris ile resim arasında konvolüsyon işlemi yapılır. Bu işlemde kullanılan matris Şekil 5.10'da verilmiştir.

0	1	0
1	-4	1
0	1	0

Şekil 5.10: LoG operatorü.

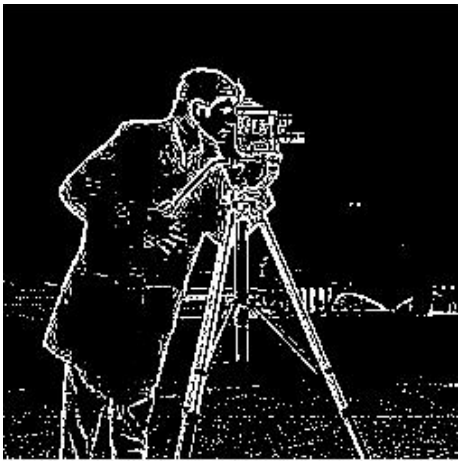
LoG dönüşümü de kenar bulmak için kullanılabilir. Şekil 5.11'de LoG dönüşümünün yapıldığı resim verilmiştir. Çıkan sonuç Şekil 5.10'da kullanılan matrise göre yorumlanabilir. Sobel ve prewitt operatorlerindeki yumuşatma etkisi burada çok da görülmez. Kenar belirleme yatay veya dikey doğrultularda olmak üzere ayrı ayrı verilmez. Kenarlar nokta nokta belirlenmiş gibi görünür. Elde edilen resim sobel ve prewitt operatorleri yardımıyla elde edilen resimlere kıyasla işlemde herhangi bir yumuşatma yapılmamasından dolayı daha gürültülüdür.

Şekil 5.11: LoG dönüşümünün kameraman resmi üzerindeki etkisi.

Şekil 5.12'de Gaussian filtresinin kameraman resmi üzerindeki etkisi görülmektedir. Gauss filtresi genel olarak resimdeki gürültünün azaltılmasında kullanılır. Resim görünümüne etkisi resmi bulanıklaştırmasıdır. Verilen örnekte kullanılan Gauss filtre matrisinin boyutu 5x5 olarak alınmıştır. Matris boyutu ve kullanılan σ değeri arttırılırsa resimdeki bulanıklık artar.

Şekil 5.12: Gauss filtresinin kameraman resmi üzerindeki etkisi.

Canny algoritmasının kameraman resmi üzerindeki etkisi Şekil 5.13'te verilmiştir. Kullanılan histerezis sınırlarına göre kenar çevresindeki noktalar da kenar olarak kabul edilebilmektedir. Elde edilen sonuç çeşitli denemeler sonucunda uygun olduğu belirlenen histeresiz sınır değerlerine göre bulunmuştur. Bu değerler değiştirilerek Canny algoritmasının etkisi değiştirilebilir. Şekil 5.13'te görüldüğü üzere kenarların dışında ara bölümlerde de bazı noktalar görülmektedir. Bunun nedeni seçilen histeresiz sınır değerleridir ve resimde en uygun değerler bulunmaya çalışılmıştır.



Şekil 5.13: Canny algoritmasının kameraman resmi üzerindeki etkisi.

Bir sonraki bölümde bu bölümde verilen resimler Matlab programından alınan çıktılarla karşılaştırılacaktır.

5.2. Matlab Sonuçlarıyla Karşılaştırma

Sobel ve prewitt operatorleri ile LoG dönüşümü Matlab programında gerçeklenirken Matlab'daki hazır fonksiyonlar kullanılmamıştır. Bunun yerine algoritmaların açıklamalarına uygun olarak elle yazılan fonksiyonlar kullanılmıştır. Bunların başında konvolüsyon algoritması gelir. Bu algoritma Ek A'da verilmiştir.

Matlab'da yatay sobel operatorünün kameraman resmi üzerindeki etkisi Şekil 5.14'te verilmiştir:

Şekil 5.14: Matlab yatay eksen sobel operatorünün kameraman üzerindeki etkisi.

Matlab kullanılarak elde edilen resim FPGA'den elde edilen Şekil 5.3'teki resim ile karşılaştırıldığında bu işlemin doğru olduğu sonucuna ulaşılır. Şekil 5.15'te Matlab kullanılarak yapılan dikey eksen için sobel operatorünün kameraman resmi üzerindeki etkisi görülmektedir:

Şekil 5.15: Matlab dikey eksen sobel operatorünün kameraman üzerindeki etkisi.

Matlab kullanılarak yapılan dikey eksen için sobel operatorü gerçeklemesi FPGA kullanılarak yapılan Şekil 5.5'deki resim ile örtüşmektedir.

Şekil 5.16: Matlab yatay eksen prewitt operatorünün kameraman üzerindeki etkisi.

Şekil 5.16'da yatay eksen için olan prewitt operatorünün Matlab programı yardımıyla edinilen sonucu yer almaktadır. Elde edilen şekil FPGA ile elde edilen şekille aynıdır. Dolayısıyla işlem doğru yapılmıştır.

Şekil 5.17: Matlab dikey eksen prewitt operatorünün kameraman üzerindeki etkisi.

Şekil 5.17'de bu kez dikey eksen için olan prewitt operatorünün Matlab çıktısı verilmiştir. Matlab çıktısı FPGA'den elde edilen resim ile benzerdir. FPGA'deki algoritma doğru çalışmaktadır. Şekil 5.18'de LoG operatorünün etkisi görülmektedir. FPGA ile elde edilen sonuçla aynı olmaktadır.

Şekil 5.18: Matlab LoG dönüşümünün kameraman resmi üzerindeki etkisi.

Şekil 5.19'da ise Gauss filtresi uygulanmıştır. Matlab'da kullanılan Gauss matrisi de FPGA'de kullanılan Gauss matrisi ile aynı boyutta olup 5x5 boyutuna sahiptir. Elde edilen sonuç FPGA'den elde edilen sonuçla uyuşmaktadır.

Şekil 5.19: Matlab Gauss filtresinin kameraman resmi üzerindeki etkisi.

Kullanılan Gauss filtresinin σ değeri bulanıklaştırmayı sağlayacak şekilde yapılan çeşitli denemelerin ardından belirlenmiştir. Bu değer arttırıldığında bulanıklaşma artar, azaltıldığında bulanıklaşma azalır.

Şekil 5.20'de Matlab kullanılarak elde edilen Canny algoritmasının resim üzerindeki etkisi gösterilmiştir. Matlab Canny algoritmasında Matlab içerisinde bulunan standart algoritma fonksiyonu kullanılmıştır bu nedenle elde edilen sonuç biraz farklıdır. Bunun nedeni Matlab içersinde yer alan algoritmada görüntüyü iyileştirmek adına ek işlemlerin yapılmasıdır.

Şekil 5.20: Matlab Canny algoritmasının kameraman resmi üzerindeki etkisi.

Şekil 5.20'de Matlab ile elde edilen resim FPGA ile elde edilen resimle karşılaştırıldığında daha az gürültüye sahip olduğu görülür. Matlab'da hazır olarak kullanılan bu fonksiyon FPGA'de elde edilen sonuçla kıyaslandığında daha iyi çalışmaktadır.

6. SONUÇ VE ÖNERİLER

Projede ele alınan sistemin performans, fiyat ve çevre faktörlerini ele alarak yorumlamak gerekirse verilen sistem FPGA üzerinde gerçeklendiği için performansı aynı iş için özel tasarlanmış ASIC devrelere göre düşük, fiyatı yüksek ve çevreden gelen etkilere daha açık olacaktır. Bu proje konum itibariyle ASIC devresi tasarlanmadan önce tasarlanan FPGA prototip devresidir. Eğer ki bir işlem FPGA üzerinde yapılabilir ise ihtiyaca bağlı olarak bu devrenin ASIC versiyonu da yapılabilir. FPGA devresinin önce yapılmasının nedeni tasarlanan sistemdeki hataların seri ASIC üretimine geçmeden önce daha kolay ayıklanması ve devrenin istenilen sonucu verecek hale getirilmesidir. Böylece seri üretimde tespit edilebilecek bir hatadan kaynaklanan maliyet artışından ve zaman kaybından korunulacaktır.

6.1. Çalışmanın Uygulama Alanı

Çalışmanın uygulama alanı görüntü işleme gerektiren herhangi bir alan olabilir. İşlevsellik açısından daha ileri olan gelişmiş görüntü işleme algoritmalarının bölünerek küçük parçalar halinde tasarlanması yapılabilecek büyük bir projedeki modülerliği arttırır dolayısıyla sistemin her bir parçası daha rahat bir şekilde kontrol edilebilir, bakımı yapılabilir ve gerektiğinde değiştirilebilir.

Tasarlanan her bir küçük birimdeki yapılan işleme göre değişik güvenlik katmanları oluşturulabilir ve büyük sistemin daha güvenli hale gelmesi sağlanabilir. Ayrıca küçük birimler bulundukları konuma göre maliyeti daha ucuz tutacak şekilde tasarlanabilir. Sistemin küçük parçalara ayrılmasındaki diğer bir amaç ise sistemin hızlı çalışmasını sağlamaktır. Örneğin bir trafik kontrol sisteminde her bir aracın plakası yardımıyla konumu belirlenebilir. Bu büyük sistem plaka okuyan ve araçların her birinin bir harita üzerinde gösterilmesini sağlayan iki alt bölüme ayrılabilir. Plaka okuma bölümü trafikte değişik noktalara konumlandırılmış kameralar vasıtasıyla yapılabilir. Tüm araçların bir harita üzerinde gösterilmesi ise merkezde tüm verileri toplayan bir sistem aracılığıyla gerçekleştirilebilir. Bu sistemdeki plaka tanıma işlemi her bir kamerada bulunan küçük bir devre yardımıyla gerçekleştirilebilir böylece merkeze giden tek veri plakaları içeren bir karakter dizisi olur ve merkezdeki işlem

yoğunluğu ile sistemdeki veri trafiği oldukça azaltılmış ve dolayısıyla sistem hızlandırılmış olur. Elbetteki buradaki koşul kameradaki plaka tanımlama yapan o küçük devrenin maliyetinin kameraya oranla az olması koşuludur. Ayrıca sistem bu şekilde tasarlandığında merkezde tüm veriler ile yapılan işlemler arttırılabilir ve çeşitlendirilebilir. Bu projede tasarlanan yapı plaka tanılama yapmasa dahi bu işlemin gerektirdiği temel işlevleri içerisinde barındırır. Üzerine yapılacak geliştirmeler ile tanılama yapacak hale getirilebilir. Elbetteki FPGA üzerinde yapılan bu tasarım çalıştığından emin olunduğunda seri üretim ve düşük maliyet için ASIC'e çevrildikten sonra tanılama sistemi gerçeklenmelidir.

Diğer uygulama alanlarından birkaçı ise askeri amaçlı kullanım veya uzay araştırmaları alanı olabilir. Askeri amaçlı kullanımlarda hızlı veri transferinin önemli olması nedeniyle sadece amaca yönelik görüntü işleme işlemlerinin gerçek zamanda yapılıp elde edilen verilerin merkezdeki bir birime gönderilmesi daha kısa zamanda yapılmış olur ve veri akış trafiği azaltılır. Uzay çalışmalarında ise gönderilecek veri daha az olacağı için veri transferi yapılması sırasındaki veri kaybı azaltılır.

6.2. Öneriler ve Gelecek

Çalışmanın uygulama alanı bölümünde bahsedildiği üzere değişik türde uygulamalar için başlangıç niteleğinde olan bu proje geliştirilip çeşitli amaçlara yönelik hale getirilebilir. Proje içerisine dahil edilen işlevsellikten daha fazlası sistem geliştirilerek gerçeklenebilir. FPGA üzerinde bir görüntü işleme yazılım kütüphanesi oluşturulabilir.

Gelecekte insan hayatında kullanım alanının yaygınlaşacağı düşünülen insansı robotların tasarımlarında tüm verinin tek bir merkezde toplanıp işlenmesi ve uzuvlara hareketlerinin oradan bildirilmesi uzuvlara tek tek karar alma yapılarının konumlandırılmasından daha yavaş bir şekilde sistemi çalıştırmaktadır. Dolayısıyla her bir ekleme konumlandırılan hareket mekanizmalarının tek başına karar verebilmesi sistemin gerçek zamanda çalışması için daha uygun olmaktadır. Bu nedenle hareketlerin kararlaştırılması yapılan kısımlarda görüntü işleme sistemlerinin yerleştirilmesi insansı robotun daha hızlı çalışmasını sağlayacaktır.

KAYNAKLAR

Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *8*(6), 679–698. http://doi.org/10.1109/TPAMI.1986.4767851

Collins, R. (2007). Lecture 11 : LoG and DoG Filters Today ' s Topics. Retrieved from http://www.cse.psu.edu/~rtc12/CSE486/

Collins, R., & State, P. (2007a). Lecture 2 : Intensity Surfaces and Gradients Visualizing Images. Retrieved from http://www.cse.psu.edu/~rtc12/CSE486/

Collins, R., & State, P. (2007b). Lecture 4 : Smoothing Summary about Convolution Computing a linear operator in neighborhoods centered at each. Retrieved from http://www.cse.psu.edu/~rtc12/CSE486/

Greisen, P., Heinzle, S., Gross, M., & Burg, A. P. (2011). An FPGA-based processing pipeline for high-definition stereo video. *EURASIP Journal on Image and Video Processing*. http://doi.org/10.1186/1687-5281-2011-18

Maurya, S., & Gupta, I. (2007). Fpga Based Hardware Implementation of Advanced Encryption, *3*(6), 2135–2137.

Nelson, A. E. (2000). Implementation of image processing algorithms on FPGA hardware. *Annals of Physics*, *54*(Cs 684), 258. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.98.2105&rep=rep1&type=pdf

Wei, Z., Lee, D.-J., & Nelson, B. E. (2007). FPGA-based Real-time Optical Flow Algorithm Design and Implementation. *Journal of Multimedia*. http://doi.org/10.4304/jmm.2.5.38-45

Xilinx. (2003). XC2064/XC2018 Logic Cell Array. Retrieved from http://www.datasheetarchive.com/XC2064-datasheet.html

Xilinx, I. (2009). Spartan-6 FPGA SP601 Evaluation Kit FAQ. Retrieved from http://www.xilinx.com/publications/prod_mktg/sp601_product_brief.pdf

EKLER

EK A: Matlab'da yazılan konvolüsyon fonksiyonu

```
function B = convolve(Img,filter)

   [r,c] = size(Img); % r satır, c sütun
   [m,n] = size(filter); % m satır, n sütun
   h = rot90(filter, 2); % rotasyon

   Rep = zeros(r + 2, c + 2);

   for x = 2 : r+1
      for y = 2 : c+1
         Rep(x,y) = Img(x-1, y-1);
      end
   end

   B = zeros(r, c);

   for x = 1 : r
      for y = 1 : c

         for i = 1 : m
            for j = 1 : n
               B(x, y) = B(x, y) + Rep(x + i-1, y + j-1) * h(i, j);
            end
         end

      end
   end
end
```

ÖZGEÇMİŞ

Ad Soyad **:** Mevlüt Mert ÇİL

Doğum Yeri ve Tarihi **:** Şişli / İstanbul, 27.09.1989

E-Posta **:** mevlutmert.cil@gmail.com

ÖĞRENİM DURUMU:

- **Lisans** **:** 2012, İstanbul Teknik Üniversitesi,
  Elektrik Elektronik Fakültesi,
  Elektronik Mühendisliği Bölümü