**T.C**
**FIRAT ÜNİVERSİTESİ**
**FEN BİLİMLERİ ENSTİTÜSÜ**

# TEMEL KENAR ALGILAMA ALGORİTMALARININ FPGA ÜZERİNDE GERÇEKLENMESİ

**YÜKSEK LİSANS TEZİ**

**Yaser İÇER**
**(141113108)**

**Anabilim Dalı: Elektrik Elektronik Mühendisliği**

**Programı: Elektronik**

**Danışman: Doç. Dr. Mustafa TÜRK**

**Tezin Enstitüye Verildiği Tarih: 25.05.2016**

**HAZİRAN-2016**

**T.C**
**FIRAT ÜNİVERSİTESİ**
**FEN BİLİMLERİ ENSTİTÜSÜ**

# TEMEL KENAR ALGILAMA ALGORİTMALARININ FPGA ÜZERİNDE GERÇEKLENMESİ

**YÜKSEK LİSANS TEZİ**

**Yaser İÇER**
**(141113108)**

**Tezin Enstitüye Verildiği Tarih: 25.05.2016**
**Tezin Savunulduğu Tarih: 10.06.2016**

**Tez Danışmanı :** **Doç. Dr. Mustafa TÜRK (F.Ü)**
**Diğer Jüri Üyeleri :** **Doç. Dr. Arif GÜLTEN (F.Ü)**
**Yrd. Doç. Dr. Mehmet ÜSTÜNDAĞ (B.Ü)**

**HAZİRAN-2016**

# ÖNSÖZ

Kenar algılama, görüntü işlemenin temel konularından biri olup görüntüdeki süreksizlikleri ve keskin tonlama değişikliklerini bulma işlemidir. Kenar algılama işleminin kullanılmasının temel amacı bir görüntüdeki veri miktarının düşürülmesi ve daha ileri görüntü işleme aşamaları için yapısal özelliklerin korunmasıdır. Görüntüler üzerine bu işlemin uygulanması bazı kenar algılama işleçleri ile gerçekleştirilmektedir.

Bu tez çalışmasında temel kenar algılama algoritmalarından Sobel, Prewitt ve Canny algoritmalarının FPGA üzerinde gerçekleştirilmesi ve FPGA kaynaklarının kullanımı incelenmiştir.

Bu çalışma boyunca desteğini ve yardımlarını hiçbir zaman esirgemeyen danışmanım Sayın Doç. Dr. Mustafa TÜRK'e ve çalışma sürecimde manevi desteklerini her zaman yanımda gördüğüm değerli aileme teşekkürü bir borç bilirim.

**Yaser İÇER**
**ELAZIĞ-2016**

## İÇİNDEKİLER

**Sayfa No**

**Sayfa No**

# ÖZET

Kenar algılama, görüntü işleme alanında önemli bir uygulama alanına sahiptir. Günümüzde birçok alanda görüntü işlemeden yararlanıldığı da bir gerçektir. Bu nedenle kenar algılama işleminin sahada uygulanabilirliği de büyük önem arz etmektedir. Bu çalışmada, literatürde temel olarak kullanılan kenar algılama algoritmalarından Sobel, Prewitt ve Canny algoritmalarının FPGA(Alanda Programlanabilir Kapı Dizileri) kullanılarak gerçeklemesi ve incelenmesi sağlanmıştır. FPGA için gerekli olan program dosyası Matlab/Simulink ile bütünleşik çalışabilen Xilinx System Generator DSP blokları ile hazırlanmış olup bilgisayar üzerinde bulunan gri formattaki görüntüler USB yapılandırma portu bağlantısı ile FPGA ya gönderilmiştir. FPGA üzerinde Sobel, Prewitt ve Canny algoritmalarına tabi tutulan görüntüler aynı bağlantı ile bilgisayara taşınarak kenar algılama işlemi gerçekleştirilmiş olur. Aynı görüntü için kenar algılama işlemi Simulink ile de aynı anda gerçekleştirilerek bilgisayar ortamında elde edilen görüntü ile FPGA donanımı ile elde edilen görüntü karşılaştırılmış ve FPGA üzerindeki kaynak kullanımı gözlemlenmiştir.

**Anahtar Kelimeler**: FPGA(Alanda Programlanabilir Kapı Dizileri), Sobel, Prewitt, Canny, Matlab/Simulink, Xilinx System Generator, DSP(Sayısal Sinyal İşleme)

## SUMMARY

### Implementation of Mainly Used Edge Detection Algorithms on FPGA

Edge detection has important applications area in image processing field. Today, it is a fact that the image processing used in many fields. Therefore, the applicability of edge detection process in the field is also has great importance. In this study, mainly used edge detection algorithms in the literature; İe. Sobel, Prewitt and Canny algorithms is provided using the verification and inspection on FPGA (Field Programmable Gate Arrays). Program files required for FPGA is prepared by Xilinx System Generator DSP blocks, which can work integrated with Matlab/Simulink. For this study; gray format images, which is stored on the computer has been sent to FPGA with USB configuration port interface on FPGA. Edge detection process is realized by moving subject images from the computer with the same connection to FPGA and then, Sobel, Prewitt and Canny algorihms are applied to the images on FPGA respectively. Edge detection process for the same images are performed by Simulink and FPGA bord at the same time and then, edge detected images obtained from these two environment are compared and also it has been observed on the FPGA resource usage.



**Keywords:** FPGA (Field Programmable Gate Arrays), Sobel, Prewitt, Canny, Matlab/Simulink and Xilinx System Generator DSP (Digital Signal Processing)

## ŞEKİLLER LİSTESİ

**Sayfa No**

**Sayfa No**

## TABLOLAR LİSTESİ

**Sayfa No**

## KISALTMALAR LİSTESİ

**FPGA** : Alanda Programlanabilir Kapı Dizileri (Field Programmable Gate Array)

**ASIC** : Uygulamaya Özgü Tümleşik Devre (Aplication Specified Integrated Circuit)

**DSP** : Sayısal Sinyal İşleme (Digital Signal Processing)

**SOC** : Çip Üzerinde Sistem (System on Chip)

**FFT** : Hızlı Fourier Dönüşümü (Fast Fourier Transform)

**VHDL** : Çok Hızlı Donanım Tanımlama Dili (Very High Hardware Description Language)

**PROM** : Programlanabilir Sadece Okunur Bellek (Programmable Read Only Memory)

**NMS** : Maksimum Olmayanın Bastırılması (Non-Maximum Suppression)

**ISE** : Tümleşik Yazılım Ortamı (Integrated Software Environment)

**RAM** : Rastgele Erişilen Bellek (Random Access Memory)

**IP** : Akıllı Özellik (Intelligent Property)

**XSG** : Xilinx Sistem Generatörü (Xilinx System Generator)

**RTL** : Kaydedici Transfer Dili (Register Transfer Language)

**MSE** : Ortalama Karesel Hata (Mean Squared Error)

**PSNR** : Tepe Sinyalin Gürültü Sinyaline Oranı (Peak Signal to Noise Ratio)

# 1.GİRİŞ

Günümüzde teknolojinin gelişmesi ile birlikte görüntü işleme tekniklerinden de oldukça fazla yararlanılmaktadır. Yaşantımız ile iç içe bir yapıya bürünen teknolojik cihazların birçoğu bir görüntüyü işleme üzerine yapılandırılmıştır. Bu alanda kullanılan algoritmalar ise teknolojinin hızı ile doğru orantılı bir şekilde çok hızlı bir şekilde gelişmektedir. Güvenlik sistemleri, trafik uyarı sistemleri, mikroskobik ve medikal görüntüleme sistemleri, uzaktan algılama sistemleri gibi pek çok alanda görüntü işleme tekniklerinden yararlanılarak çalışmalar yapılmaktadır.

Görüntü işlemedeki en önemli alanlardan biri kenar algılama işlemidir. Kenar algılama, en önemli görüntü bulma görevlerinden biridir. Kenarlar, görüntü içindeki en önemli bilgidir. İnsan görme sistemi, doğrudan kenarların algılanmasına dayanır [1]. Görüntülerdeki nesnelerin ayırt edilmesi, çıplak gözle fark edilemeyen ayrıntıların gösterilmesi, iki farklı desenin karşılaştırılması gibi durumlarda kenar tespiti algoritmaları oldukça yaygın bir şekilde kullanılmaktadır. Haralick, pikseller üzerinde uyguladığı interpolasyon denklemi üzerinden bulduğu türev değerini kullanarak bulduğu gradyant değeriyle, ikinci türev üzerinde gradyant yönündeki sıfıra geçişleri arayan bir yöntem sunmuştur [2]. Canny, Gaussian maskenin türevinden elde edilen işlecin uygulanmasına dayanan bir yöntem önermiştir [3]. Görüntü işleme alanında kullanılan ilk yöntemler olan Roberts, Prewitt ve Sobel işleçleri de en uygun işleçlere örnektir [4]. Görüntü işleme ve kenar algılama işlemleri hız gerektiren uygulamalar olduğundan bu işlemler için genellikle bilgisayar tabanlı uygulamalar, DSP tabanlı yöntemler ve FPGA' lar kullanılmaktadır.

Görüntü işlemede FPGA kullanılarak yapılan bazı çalışmaları şöyle sıralayabiliriz. Christe, ve arkadaşları MR cihazından alınan beyin görüntülerinde bulunan tümörlerin tespiti için, Xilinx marka bir FPGA geliştirme kartından faydalanmışlardır. FPGA üzerinde uyguladıkları değişik görüntü filtreleri ile tümör yerini belirlemişlerdir. Oluşturdukları sistem Matlab ile bütünleşmiş bir şekilde çalışıp, literatürde bulunan başka sistemlere göre, %50 daha az kaynak kullanımı gerçekleşmiştir [5]. Rodriguez ve arkadaşları ise FPGA üzerinde gerçek zamanlı medyan filtresi tasarlayıp çalıştırarak, üretim merkezlerindeki hataların görsel olarak belirlenmesini sağlayan bir sistem oluşturmuşlardır [6]. Nelson, yüksek lisans tezinde Xilinx ve Altera firmalarının ürettiği farklı FPGA donanımlarını

kullanarak bir takım morfolojik işlemleri ve evrişim işlemini gerçekleştirmiş, aldığı sonuçları MATLAB kullanarak aldığı sonuçlarla karşılaştırmıştır [7]. Nana ve arkadaşları ise DSP ile FPGA donanımını birlikte kullanarak alınan bir görüntünün çözünürlüğünü geliştirerek daha net bir görüntü elde etmişlerdir. Bu işlemde; Xilinx Spartan-3 FPGA donanımını görüntünün ön yapılandırmasında kullanmışlar ve DSP ile çözünürlüğü geliştirmişlerdir [8].

Bu çalışmada FPGA, Alanda Programlanabilir Kapı Dizileri (Field Programmable Gate Array) tabanlı çalışabilen temel kenar algılama işleçlerinden Sobel, Prewitt ve Canny işlecinin gerçekleştirilmesi ve incelenmesi ele alınmıştır. FPGA, herhangi bir sayısal devreyi veya sistemi oluşturmak için elektriksel olarak programlanabilen ve çok sayıda mantık hücresinden meydana gelen silikon teknolojilerdir. FPGA'lar, sabit fonksiyonlu Uygulamaya Özgü Tümleşik Devre (ASIC) teknolojisinden daha üstün birçok avantaj sağlamaktadır. ASIC'ler, tipik olarak istenilen bir devreyi elde etmek için aylarca sürebilen üretim aşamaları ve yüksek miktarda maliyetlere sahiptir. Buna karşın FPGA'lar, bir saniyeden daha kısa bir sürede yapılandırıla bilirlik ve tasarımda herhangi bir hata oluşması durumunda sürekli olarak yeniden programlanabilir bir yapı sunmaktadır. Ayrıca FPGA teknolojisine, birkaç dolardan başlayıp birkaç bin dolara kadar olan düşük maliyetlerle her yerde sahip olma imkânı bulunmaktadır [9].

FPGA'ların sahip olduğu paralel, hızlı işlem yapabilme yeteneği ve sahada birçok kez yeniden programlanabilir olması gibi özellikleri, bu cihazların endüstride ve akademik araştırmalarda geniş bir şekilde kullanılmasına olanak tanımıştır. FPGA'lar, güç kalitesi ile ilgili çalışmalarda [10, 11], gerçek zamanlı görüntü işlemede [12], yapay sinir ağlarının donanımsal olarak gerçekleştirilmesinde [13, 14], Ayrık Dalgacık Dönüşümü ve Ters Ayrık Dalgacık Dönüşümü gibi sinyal işleme algoritmalarının [15, 16] uygulanmasında etkili bir şekilde kullanım alanı bulmuştur.

Bu tez çalışmasında, programlanabilir lojik devre teknolojileri ile FPGA cihazları üzerinde kapsamlı bir araştırma yapılarak kenar algılamada kullanılan temel işleçlerden Sobel, Prewitt ve Canny işlecinin FPGA üzerinde gerçeklemesi ve incelenmesi sağlanmıştır. Çalışma kapsamında Matlab/Simulink ile bütünleşik çalışabilen Xilinx System Generator aracı ile Simulink ortamında bulunan Xlinx DSP blokları ve Simulink blokları kullanılmıştır. Simulink ortamında oluşturulan sistem ile bilgisayar üzerinde

bulunan 256*256 gri formattaki görüntüler alınarak hem simulink ortamında hem de FPGA üzerinde görüntüyü Sobel,Prewitt ve Canny işleçlerine ayrı ayrı tabi tutarak kenar algılaması yapılan görüntüyü bilgisayar üzerinde gözlemlememize imkân sunmaktadır. Hazırlanan sistemde FPGA kartı ve bilgisayar arasındaki veri iletimi USB yapılandırma portu ile sağlanmıştır. FPGA nın çalışması için gereken *.bit uzantılı program dosyası Xilinx System Generator ile oluşturulmuştur. Uygulamada, Xilinx firmasına ait olan Spartan3E XC3S1600E FPGA cihazı kullanılmıştır.

Bu tez çalışmasının ilerleyen kısımları üç bölümden daha oluşmaktadır. Bu bölümlerde incelenen konu kapsamları aşağıda verilmiştir:

Bölüm 2'de, kenar algılama, kenar algılama yöntemleri ve temel kenar algılama işleçleri ile ilgili genel bilgiler verilmiştir. Ayrıca bölüm içerisinde temel kenar algılama işleçleri için simulink ortamında oluşturulmuş modeller ile hazırlanmış görüntü örneklerine de yer verilmiştir.

Bölüm 3'te, FPGA cihazlarının yapısı, programlama teknikleri gibi bilgiler verilmiştir. FPGA da kullanılan VHDL ve Verilog programlama dilleri hakkında genel anlatımlar yapılmıştır. FPGA cihazlarının programlanmasında kullanılan, Entegre yazılım ortamı ve kullanımı hakkında özet bilgiler de bu bölüm içerisinde sunulmuştur.

Bölüm 4'de, temel kenar algılama işleçlerinden Sobel, Prewitt ve Canny işlecinin FPGA üzerinde gerçeklemesi ile ilgili ayrıntılı bilgi ve incelemelere yer verilmiştir. Oluşturulan sistemin FPGA ve bilgisayar tarafındaki ayrıntıları da bu bölüm içerisinde sunulmuştur.

# 2. GÖRÜNTÜDE KENAR ALGILAMA VE İŞLEÇLER

## 2.1. Bir Görüntüde Kenarlar

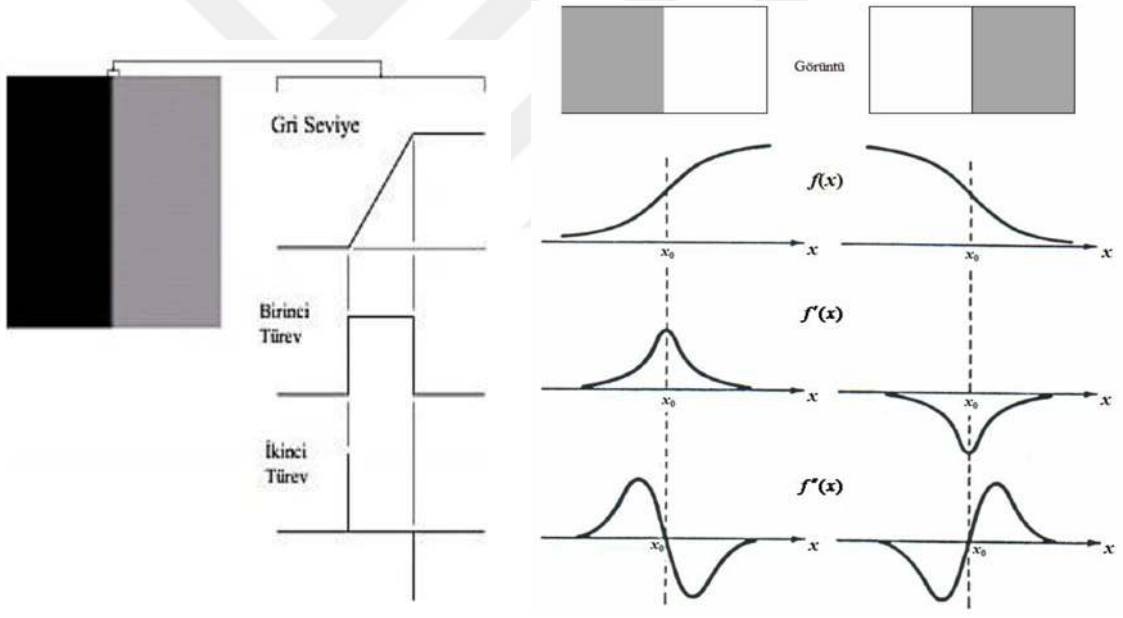
Bir görüntüde kenarlar, düzgün alanlar arasında piksel değerlerindeki ani değişikliklerin olduğu yerlerdir. Kenar algılamanın temel amacı görüntüde üzerinde süreksizliklerin, yani ani değişimlerin yer aldığı bölgeleri ortaya çıkarmaktır [17]. Literatürde kullanılan görüntüde bir kenar tanımı Kitchen ve Rosenfeld [18] tarafından, "her bölgenin kendi içinde homojen fakat bazı yönlerden birbirlerinden farklı olduğu komşu iki bölge arasındaki sınır" olarak yapılmıştır [19]. Yapılan bu tanıma dayanarak kenar algılamadaki amacın temelde aydınlık kenarları algılamak olduğu söylenebilir.

## 2.2. Görüntüde Kenar Belirleme

İnsanlardaki görme sistemi, herhangi bir görüntü üzerinde ilk olarak cismin yer aldığı görüntüdeki kenarları algılama eğilimindedir, bunun nedeni durağan yerler genellikle bilgi içermez. Bir görüntüde, görüntüyü iyileştirme ne derece önemli ise kenarların belirlenmesi de büyük önem taşımaktadır. Genellikle, bu gibi yüksek seviyeli görüntü işlemlerinde ön aşama olarak kenar bulma işleçleri uygulanmaktadır [20]. Bir görüntü içinde optik ve şekle bağlı farklı özellikler kenarların oluşmasına sebep olur. Kenarların belirmesindeki şekil bazlı özellikler; yüzey yönü ya da renk dokusundaki değişimler, nesne sınırları, iki objenin üst üste görünmesinden kaynaklı gözlemlenen renk ve dokulardaki farklılıklarıdır [21]. Kenarları oluşmasını sağlayan optik özellikler de; ışığın direk yansıması, aynı nesnenin bir bölümünden veya diğer nesneler nedeniyle oluşan gölgeler, aynı nesnenin bir bölümünden veya diğer nesnelerden kaynaklanan ara yansımalar, doku veya renk değişimleridir [22]. Genel olarak kenarların belirlenmesinde kullanılan yöntemler, uzun bir süre teorik olarak farklılığa uğramamıştır. Bu yöntemler, temelde birbiriyle komşu pikseller arasındaki değer değişimlerini kullanırlar. Kenar algılama algoritmalarının amacı, belirgin bir şekilde görüntü üzerinde gözüken adım kenarları bulmaktır. Aslında, bir kenar algılama yönteminin görüntüde oluşabilecek her türlü değişimi bulabilmesi gerekmektedir. Kenar çeşitleri üç farklı grupta ele alınabilir: adım kenarlar, hat kenarlar, kesişim kenarlar [22]. Kenar elde etme yöntemleri, görüntü üzerindeki kenarları algılayabilmek için temel

işlemler olan gürültü giderme, türev bulma ve etiketleme işlevlerini yerine getirmeye çalışır. Bu işlemler bütün yöntemlerde aynı olmasına karşın, kullandıkları matematiksel işlemler, filtreler, etiketleme süreçleri, türev operatörleri, amaçları ve hesaplamadaki karmaşık işlemler bakımından kendi aralarında farklılıklar gösterirler.

Görüntüde var olan kenarları algılayabilmek için en verimli yöntemlerden birisi, görüntü üzerinde anlık gri değer değişimlerini tespit etmektir. Bunun için birçok kenar algılama yönteminin kullandığı temel yaklaşımlardan biri, türevin bölgesel olarak hesaplanmasıdır. Görüntünün birinci türevi kenar bölgelerinde bölgesel olarak en yüksek değere ulaşırken, görüntünün ikinci türevi ise kenar bölgelerinde bölgesel olarak sıfır olur. Bunun sonucunda görüntü ile ilgili hesaplanan 1. ve 2. türevler, incelenen görüntü bölgesi için bölgesel maksimum ve sıfır geçiş noktaları ile de kenarlar belirlenmiş olur. Şekil 2.1' de türev alma işleminin kenar bulmadaki etkisini görebiliriz.




**(a)** **(b)**

**Şekil 2.1** (a) Görüntü üzerindeki kenar geçişlerinin grafiksel gösterimi (b) Beyaz ve siyah bant görüntülerde kenar geçişinin türevsel olarak gösterimi

### 2.2.1. Birinci Türev (Gradyant) Tabanlı Yöntemler

Gradyant tek değişkene göre bir fonksiyonun birinci türevi olarak ifade edilmektedir. Bu işlem iki değişkene bağlı olan bir fonksiyonda ise her bir değişkene göre 1. türev alma işlemi uygulanarak hesaplanabilir. Bir görüntü iki değişene bağlı bir fonksiyon olarak ifade edildiği için, bir görüntüdeki kenarlarda görüntü fonksiyonuna uygulanacak gradyantın

büyüklüğüne bakılarak anlaşılabileceği gibi gradyant açısı da kenarların yönü hakkında gerekli bilgiyi bize verecektir. *f(x,y)* gibi bir görüntü fonksiyonuna sahip bir görüntü için denklem 2.1 ve 2.2 bize gradyant büyüklüğünün ve gradyant yönünün nasıl hesaplanacağını göstermektedir.

$$G[f(x,y)] = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} \lim_{\Delta x \to 0} \frac{f(x+\Delta x,y)-f(x,y)}{\Delta x} \\ \lim_{\Delta y \to 0} \frac{f(x,y+\Delta y)-f(x,y)}{\Delta y} \end{bmatrix} \quad (2.1)$$

$$G = \sqrt{G_x^2 + G_y^2} \quad , \quad \theta = \tan^{-1} \frac{G_y}{G_x} \quad (2.2)$$

Görüntüdeki pikseller arası uzaklığın en fazla 1 birim olabileceği göz önüne alınırsa, $\Delta x = \Delta y = 1$ için hesaplanacak gradyant vektörü 2.3'teki gibi olacaktır.

$$\begin{bmatrix} G_x \\ G_y \end{bmatrix} \approx \begin{bmatrix} f[i,j+1] - f[i,j] \\ f[i,j] - f[i+1,j] \end{bmatrix} \quad (2.3)$$

2.3'teki vektörlere karşı oluşan evrişim maskeleri ise 2.4'deki gibi ortaya çıkar.

$$G_x = [-1\ 1] \quad , \quad G_y = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \quad (2.4)$$

Temelde 1. türev tabanlı yöntemlerin kenarları bulma şekline baktığımızda 1. türev sonucunda bulunacak maksimum ve minimum değerlerin belirli bir eşik ile kıyaslanması sonucu kenar bilgisine ulaşılır. Roberts, Sobel, Prewitt ve Canny 1. türev (gradyant) tabanlı yöntemler arasında en çok bilinen işleçlerdir.

#### 2.2.1.1. Roberts Kenar İşleci

Roberts işleci, işleme tabi tutulacak görüntü üzerine 2x2'lik bir maskenin gezdirilmesi ile her noktada gradyant vektörünün yaklaşık olarak hesaplanmasını sağlar. Hesaplanmış olan gradyant vektörünün büyüklüğü, görüntü boyutundaki bir piksel matrisinin üzerine yerleştirilirse kenarları belirlenmiş bir görüntü elde edilebilir.

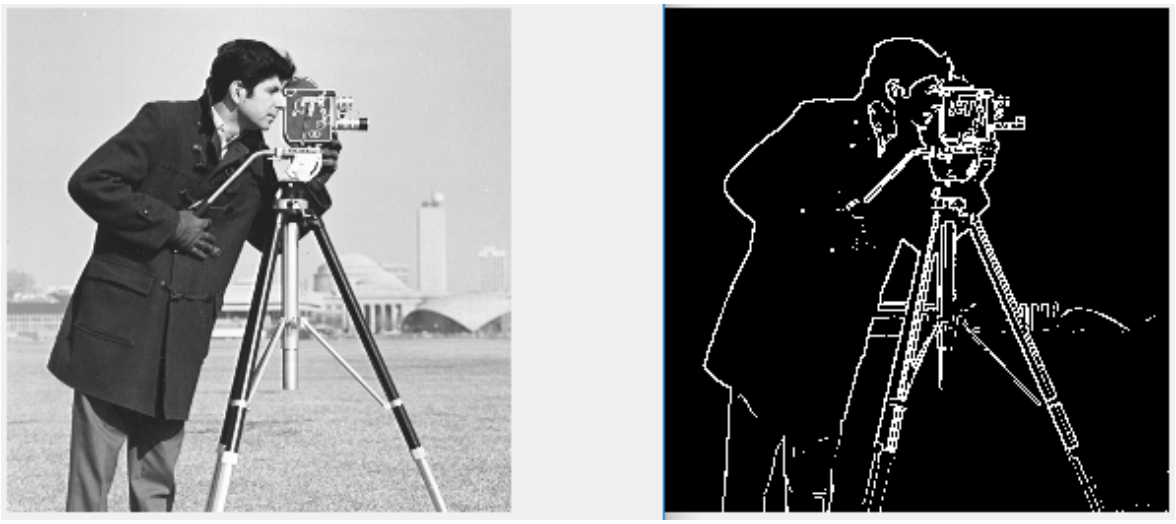
Görüntü üzerinde gradyant işlemi, iki farklı maskenin yatay ve düşey yönde görüntü ile evrişim işlemi sonucu hesaplanır. Birinci türeve ait fark denklemlerinin kullanılması sonucu evrişimde kullanılan iki maske elde edilir. $f(x,y)$ devamlı görüntü fonksiyonu olmak üzere, görüntüye ait yoğunluk fonksiyonunun gradyant vektörü [23], 2.5'teki gibi elde edilir.

$$G|f(x,y)| = \{|f(x,y) - f(x+1,y+1)|^2 + |f(x+1,y) - f(x,y+1)|^2\}^{1/2} \qquad (2.5)$$

Roberts [24] kenar işleci, 2.5'te yer alan terim katsayılarının 2x2'lik bir maskeye yerleştirilmesi ile oluşturulur. Bunun sonucunda yatay ve düşey yönde 2.6'da belirtilen maskeler elde edilir.

$$G_x = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad , \quad G_y = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \qquad (2.6)$$

Roberts işleçleri 2x2 ebadında olması hasebiyle bu işleç için hesaplanan gradyant değerleri yaklaşık değerlerdir. 3x3 yada 5x5 gibi tek sayıda boyutlara sahip kenar işleçlerinden farklı olarak 2x2'lik bir maskeye sahip işleç olmasından ötürü hesaplanmış olan gradyant değeri maskenin merkezi yerine $f(x,y)$ olarak belirtilen noktaya konur. Bu sebeple gradyant ın yaklaşık değeri orijinal görüntü üzerine yerleştirilmiş olur. 2x2 boyutlu bir maskenin yatay ve düşey yönlerde evrişimi ile elde edilen kenarların bulunduğu görüntü gerçekte kenarların bir ölçüde ötelenmiş durumunun yer aldığı görüntüler olacaktır. Öteleme miktarının süreksizliğin yönü ile ilgili olması nedeniyle, bu durumun giderilmesi için sonradan bir işlemin yapılması gerekmektedir. Şekil 2.2'de bir görüntü ve Roberts işleci uygulanmış hali verilmiştir.



**Şekil 2.2** Orijinal görüntü ve Roberts işleci uygulanmış hali

### 2.2.1.2. Sobel Kenar İşleci

Bir diğer temel kenar algılama işleci Sobel dir. Bu işleç Robert [24] gibi daha kapsamlı kullanılan işleçlere göre yön itibariyle daha bağımsız gradyant hesaplaması ile verimli bir değerlendirme yapabilmeyi sağlamaktadır [25]. Sobel kenar işlecinin oluşturulmasında yatan temel düşünceyi şu şekilde açıklayabiliriz. Bir fonksiyon olarak değerlendirilen sayısal bir görüntüye 3x3 komşulukta ve 4 merkezi yönde oluşturulmuş vektörlerinin gradyant değerlerinin toplamları temeline dayanır. Bu gradyant vektörleri Şekil 2.3'te verilmiştir.

| | | |
|---|---|---|
| *(i-1,j-1)* | *(i-1,j)* | *(i-1,j+1)* |
| *(i,j-1)* | *(i,j)* | *(i,j+1)* |
| *(i+1,j-1)* | *(i+1,j)* | *(i+1,j+1)* |

**Şekil 2.3** Dört temel yönde gradyant vektörleri

| | | |
|---|---|---|
| *a* | *b* | *c* |
| *d* | *e* | *f* |
| *g* | *h* | *i* |

**Şekil 2.4** 3x3 komşulukta yer alan piksellerin yerleşimi

Gradyant değerlerinin vektörel toplamları, gradyant ölçümlerinin yönü üzerinde ortalama değerin bulunmasını sağlar. Eğer yoğunluk fonksiyonu gerçekten düzlemsel ise, o noktanın etrafındaki bütün yakın komşuluklardaki tüm gradyantlar aynı değere sahip olur. Dört gradyant değerinin vektörel toplamı, sekiz yönsel türev vektörünün vektörel toplamıyla aynıdır. Bir nokta ve onun sekiz komşuluğunun yoğunluk değerleri, Şekil 2.4'te verilmiştir. Yoğunluğu verilen notalar *( a,i), (b, h), ( c,g)* ve *(f,d)* şekinde gruplanırsa, $R = \sqrt{2}$ (köşegen noktalarındaki piksellerin merkeze olan uzaklığı) için gradyant değerlerinin vektörel olarak toplamı 2.7'deki gibi elde edilir.

$$G = \frac{(c-g)}{R}.\frac{[1,1]}{R} + \frac{(a-i)}{R}.\frac{[-1,1]}{R} + (b-h).[0,1] + (f-d).[1,0] \tag{2.7}$$

Bunun sonucunda,

$$G = [(c-g-a+i)/2 + f - d, (c-g+a-i)/2 + b - h] \tag{2.8}$$

2.8'deki sonuç vektörü elde edilmiş olur. Elde edilen denklem ½ çarpanından kurtulmak için 2 ile çarpılırsa,

$$G' = 2.G = [(c-g-a+i) + 2.(f-d), (c-g+a-i) + 2.(b-h)] \tag{2.9}$$

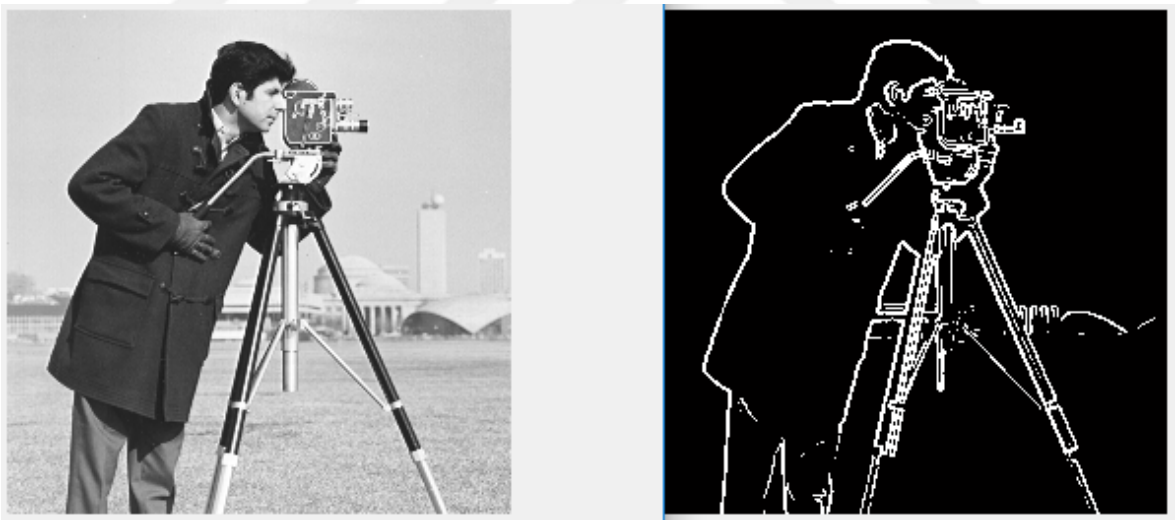
Denklem 2.9'daki vektör bulunmuş olur. Bu vektör Şekil 2.5'te olduğu gibi yatay ve düşey yöndeki ağırlık fonksiyonları kullanılarak yoğunluk ağırlıklarının toplamı olarak gösterilir.

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

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

**Şekil 2.5** Yatay ve düşey yöndeki ağırlık katsayıları

Sobel işlecinde, yatay ve düşey yönde elde edilmiş olan ağırlık katsayı maskeleri görüntü üzerinde sağ üst pikselden başlayarak gezdirilip her piksel değerine tekabül eden maske katsayısı ile çarpılarak bulunan tüm değerler toplanırsa işlecin tepkisi elde edilmiş olur. Yatay ve düşey yöndeki maskelerin aynı piksel üzerindeki tepkilerinin kareleri toplanıp karekök işleminden geçirilirse gerçek gradyant değeri elde edilir. Elde edilen gradyant değerleri orijinal görüntü büyüklüğünde bir matris oluşturacak şekilde bir araya getirilirse kenarlar tespit edilmiş olur. Şekil 2.6'da orijinal görüntü ve Sobel işleci uygulanmış hali gösterilmiştir.



**Şekil 2.6** Orijinal görüntü ve Sobel işleci uygulanmış hali

Türeve dayalı bir kenar algılama işleci olduğundan Sobel işleci gürültüye karşı duyarlı olan bir işleçtir. Bu nedenle görüntü evrişim işlemine tabi tutulmadan önce gürültüden arındırma işlemi gerektirebilir. Gürültüden arındırma işleminde meydana gelebilecek bilgi kayıpları sebebiyle, görüntü üzerindeki bazı kenarların bulunmasında problemler yaşanabilir [22].

### 2.2.1.3. Prewitt Kenar İşleci

Prewitt kenar işleci, görüntü üzerindeki pikseller için her noktada gradyant vektörünü hesaplayan bir işleçtir. Bulunan gradyant vektörünün büyüklüğü kenar iyileştirmesi yapılmış olan görüntüyü verirken gradyant vektörünün yönü ise kenar yönünü vermektedir [26].

Prewitt kenar işleci kullanılarak gradyant değeri, görüntü üzerindeki pikseller ile yatay ve düşey yöndeki iki farklı maskenin evrişim işlemi ile hesaplanır. Bu işlem sırasında kullanılan maskeler ise birinci türeve ait fark denklemleri kullanılarak bulunur. Bir $f(x)$ fonksiyonunun birinci türevi, bu fonksiyonun Taylor serisi açılımı kullanılarak hesaplanır. $f(x)$ fonksiyonunun $x = a$ komşuluğu etrafındaki Taylor serisi açılımı 2.10'da verildiği gibi elde edilir.

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)(x-a)^2}{2!} + \cdots \quad (2.10)$$

Bu eşitlik kullanılarak $f(x + \Delta x)$ in $x$ etrafındaki açılımı 2.11'deki gibi bulunur.

$$f(x + \Delta x) = f(x) + \Delta x f'(x) + \frac{(\Delta x)^2}{2!} f''(x) + \cdots \quad (2.11)$$

2.11'in düzenlenmesi ile 2.12 eşitliği elde edilir.

$$\frac{f(x+\Delta x)-f(x)}{\Delta x} = f'(x) + \frac{\Delta x}{2!} f''(x) + \cdots \quad (2.12)$$

Yukardaki işlemler ile benzer şekilde $f(x - \Delta x)$ in $x$ etrafındaki açılımı 2.13'teki gibi hesaplanır.

$$f(x - \Delta x) = f(x) - \Delta x f'(x) + \frac{(\Delta x)^2}{2!} f''(x) - \cdots \quad (2.13)$$

$f(x - \Delta x)$ ve $f(x + \Delta x)$ açılımları birbirinden çıkarılırsa 2.14 ve 2.15'te verilen denklemler elde edilir.

$$f(x + \Delta x) + f(x - \Delta x) = 2\Delta x f'(x) + \frac{2(\Delta x)^3}{3!} f''(x) + \cdots \quad (2.14)$$

$$\frac{f(x+\Delta x)+f(x-\Delta x)}{2\Delta x} = f'(x) + \frac{2(\Delta x)^2}{3!} f''(x) + \cdots \quad (2.15)$$

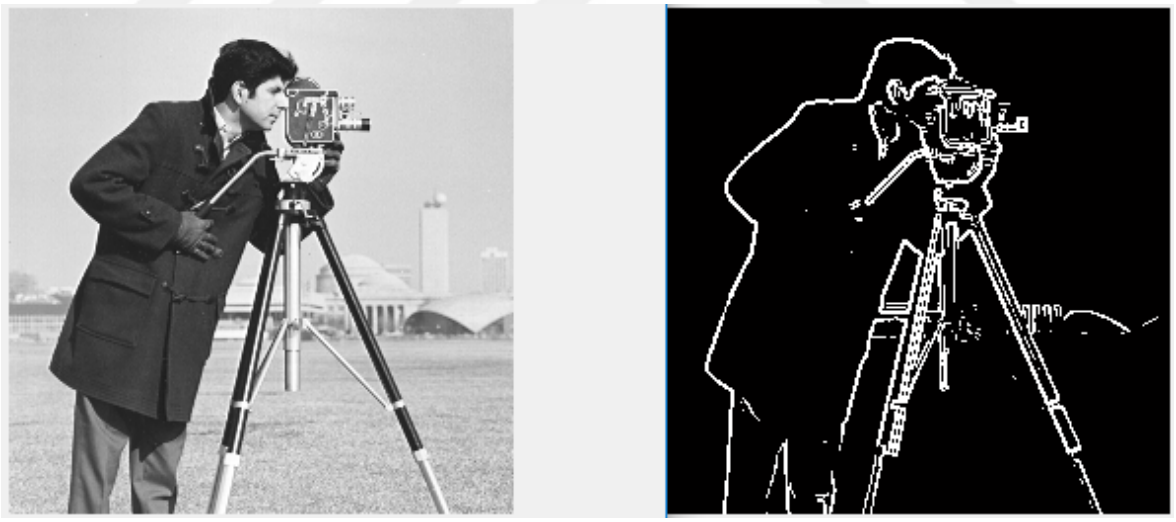
Birinci türev 2.14 ve 2.15 denklemlerinden yararlanılarak nümerik olarak 2.16 ve 2.17'deki gibi elde edilmiş olur [27].

$$\frac{\partial f}{\partial x} = \frac{f(x+\Delta x,y)-f(x,y)}{\Delta x} = \frac{f(x+\Delta x,y)-f(x-\Delta x,y)}{2\Delta x} \tag{2.16}$$

$$\frac{\partial f}{\partial y} = \frac{f(x,y+\Delta y)-f(x,y)}{\Delta y} = \frac{f(x,y+\Delta y)-f(x,y-\Delta y)}{2\Delta y} \tag{2.17}$$

2.16 ve 2.17 denklemleri, sırasıyla x ve y eksenleri yönündeki gradyant büyüklükleridir. Bu ifadelerde pikseller arasındaki fark $\Delta x = \Delta y = 1$ olarak alınıp türev içerisinde yer alan katsayıların 3x3'lük bir evrişim çekirdeği oluşturacak şekilde genişletilmesi ile 2.18'de verilen Prewitt evrişim maskeleri oluşmuş olur [27]. Şekil 2.7'de orijinal görüntü ve Prewitt işleci uygulanmış hali gösterilmiştir.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad , \quad G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{2.18}$$



**Şekil 2.7** Orijinal görüntü ve Prewitt işleci uygulanmış hali

#### 2.2.1.4. Canny Kenar İşleci

Canny [3] optimum bir kenar bulma algoritması üzerinde çalışmıştır. Bu algoritmayı belirlerken de aşağıda belirtilen özellikleri sağlayacak şekilde tasarlamıştır.

- Geliştirilen algoritma kenarların bulunduğu yerleri büyük bir olasılık değeri ile belirlerken, kenar olmayan yerleri küçük bir olasılık değeri ile belirleyebilmeli.
- Belirlenecek kenarlar gerçek kenarlara en yakın olabilecek kenarlar olmalıdır.
- Gerçek bir kenar yalnızca bir kez algılanmalı.

Canny kenar algılama algoritmasının çalışması 5 adımdan oluşmaktadır. Bu adımlar sırası ile aşağıdaki gibidir:

1. Yumuşatma (Smoothing)
2. Gradyant'ların bulunması
3. Yerel maksimum olmayan noktaların bastırılması (NMS)
4. Çift eşikleme
5. Histerezis ile kenar takibi

##### 2.2.1.4.1. Yumuşatma (Smoothing)

Kamera ya da fotoğraf makinesi tarafından elde edilen görüntülerin tümünde bir miktar gürültü bulunması kaçınılmaz bir durumdur. Görüntü üzerindeki bu gürültünün kenar algılama işlemi sırasında hataya sebep olmaması için görüntü üzerinden temizlenmesi gerekmektedir. Bunun için görüntüye Gauss filtresi uygulanmaktadır. Gauss filtresi için kullanılan fonksiyon ve $\sigma$ =1.4 için elde edilen çekirdek maskesi 2.19'da verilmiştir.

$$G_\sigma \equiv \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \quad , \quad G = \frac{1}{159}\cdot\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} \tag{2.19}$$

#### 2.2.1.4.2. Gradyant'ların Bulunması

Canny kenar bulma algoritması kenar bulma işlemini görüntü üzerindeki gri ölçek yoğunluğunun en çok değiştiği yerlere göre gerçekleştirmektedir. Bu işlem için görüntü üzerindeki piksellerde gradyant bulma işleminin yapılması gerekmektedir. Gradyant bulma işlemi Canny algoritmasında yatay ve düşey yönde kullanılan Sobel maskelerinin evrişim işlemi ile sağlanır. Yatay ve düşey yönde kullanılan Sobel maskeleri 2.20'deki gibidir.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad , \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.20}$$

$G_x$ ve $G_y$ sırasıyla x ve y yönündeki gradyant vektörleri olmak üzere, gradyant büyüklüğü ve gradyant yönü 2.21'deki gibi hesaplanır.

$$|G| = \sqrt{G_x^2 + G_y^2} \qquad \theta = \tan^{-1}\left(\frac{|G_y|}{|G_x|}\right)$$

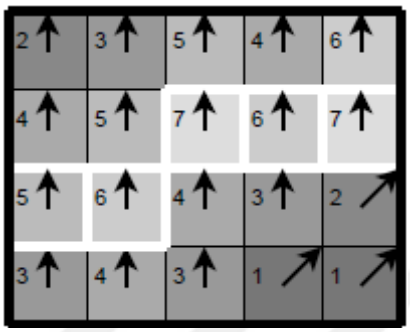
$$|G| = |G_x| + |G_y| \tag{2.21}$$

#### 2.2.1.4.3. Yerel Maksimum Olmayan Noktaların Bastırılması

Yerel maksimum olmayan noktaların bastırılması işlemin yapılmasının temel amacı, gradyant büyüklüğü bulunan görüntüdeki bulanık olan kenarların keskinleştirilmesidir. Bu işlem temel olarak gradyant görüntüsü içerisindeki yerel maksimum olmayan tüm noktaların piksel değerinin sıfır yapılmasıyla gerçekleştirilir. Algoritmanın gerçekleştirilmesi için gradyant görüntüsü üzerinde aşağıdaki işlemler yapılır.

- Bir piksele bağlı olan 8 komşu piksel için hesaplanmış olan gradyant yönü açıları 45 derece ve katlarına yakın olan açılara yuvarlanır.
- Pozitif ve negatif gradyant yönünde bulunan kenar piksellerinin yoğunluğu ile ele alınan pikseldeki kenar piksellerinin yoğunluğu karşılaştırılır (Eğer gradyant yönü 90 derece ise bu piksel 270 derecedeki piksel ile karşılaştırılır).
- İlgilenilen kenar pikselinin yoğunluğu karşılaştırılan diğer bir kenar pikselinin yoğunluğundan büyük ise bu piksel kenar olarak alınır, değilse piksel değeri sıfır yapılarak elenir.

Şekil 2.8'de yerel maksimum olmayan noktaların bastırılmasına bir örnek verilmiştir. Örnekte gradyant görüntüsü üzerindeki piksellerin hem değeri hem de yönü verilmiştir. Aynı yöne ait pikseller arasında yapılan karşılaştırma sonucu gradyant değeri büyük olan pikseller beyaz çerçeve içine alınarak seçilmiş ve diğer pikseller baskılanarak sıfırlanmıştır.




**Şekil 2.8** Yerel maksimum olamayan piksellerin bastırılması örneği

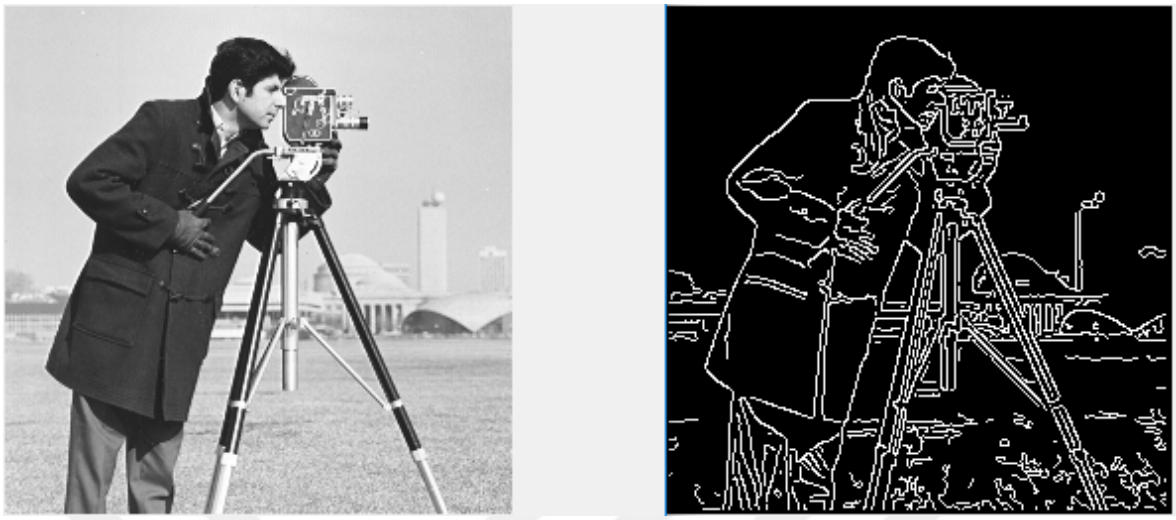
#### 2.2.1.4.4. Çift Eşikleme

Yerel maksimum olmayan noktaların bastırılması sonucu elde edilen piksellerin büyük bölümü gerçek kenar olabileceği gibi bir kısmı da gürültü ve renk çeşitliliğinden dolayı gerçek kenar olmayabilir. Bu aşamadaki çift eşiklemenin amacı kenar olmayan bu piksellerin elenmesini kapsamaktadır. Canny kenar algılama algoritması yüksek ve düşük eşik olmak üzere 2 adet eşik değeri kullanmaktadır. Kenar piksel yoğunluğu yüksek eşik değerinden büyük olan pikseller "güçlü" olarak sınıflandırılırken, kenar piksel yoğunluğu düşük eşik değerinin altında olan pikseller bastırılır. Kenar piksel değeri düşük ve yüksek eşik arasında kalan pikseller kenar olma ihtimaline karşı "zayıf" olarak sınıflandırılır.

#### 2.2.1.4.5. Histerezis ile Kenar Takibi

Çift eşikleme işleminden sonra "güçlü" olarak sınıflandırılan pikseller kesin kenar olarak belirlenir. Bunun dışında kalan ve "zayıf" olarak sınıflandırılan pikseller eğer "güçlü" olarak sınıflandırılan pikseller ile komşu ise alınır değilse bastırılarak kenarlara dahil edilmez.

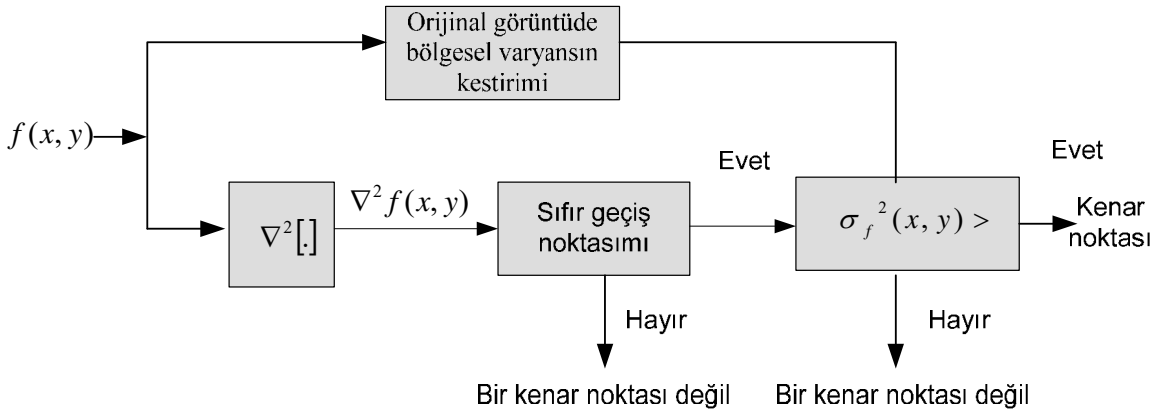
Şekil 2.9'da orijinal görüntü ve Canny işleci uygulanmış hali gösterilmiştir.



**Şekil 2.9** Orijinal görüntü ve Canny işleci uygulanmış hali

### 2.2.2. İkinci Türev (Laplacian) Tabanlı Yöntemler

İkinci türev tabanlı yöntemler, kenarların bulunması için görüntü fonksiyonunun ikinci türevi alınarak sonucun sıfır olduğu kenar noktalarının bulunmasına yönelik bir yöntemdir. Bu yöntemde birinci türev sonucu bulunan maksimum ve minimum noktaları ikinci türev alındığında sıfır olacaktır. Gürültü hassasiyeti bu yöntemde birinci türev yöntemlerine göre daha fazladır. İkinci türev yöntemi ile kenar belirleme Şekil 2.10'da gösterilmektedir. Burada Laplacian uygulanan görüntü fonksiyonunda belirli bir eşiğin üzerindeki pikseller kenar olarak kabul edilir.




**Şekil 2.10** İkinci türev yöntemi için blok diyagramı

Marr-Hildreth (Laplacian of Gaussian) [1] ikinci türev tabanlı yöntemlerden en bilinenidir.

#### 2.2.2.1. Marr-Hildreth (LoG) Kenar İşleci

Marr ve Hildreth 1980 yılında geliştirdikleri filtrede Gauss filtresinden geçirilen görüntü fonksiyonuna Laplas filtresi uygulayarak görüntü üzerindeki kenarların bulunmasını sağlamışlardır. Gauss filtresi ile görüntü üzerindeki yüksek frekansa sahip gürültü temizlenirken, Laplas filtresi ile de görüntüye ikinci türev işlemi uygulanarak kenar bulma işlemi gerçekleştirilmiş olur. Oluşturulan bu işleç Laplacian of Gaussian (LoG) olarak isimlendirilirken bu işlece ait olan matematiksel işlemler 2.22 ve 2.23'te verilmiştir.

$$h(x,y) = \nabla^2[g(x,y) * f(x,y)] = [\nabla^2 g(x,y)] * f(x,y) \tag{2.22}$$

$$\nabla^2 g(x,y) = \left(\frac{x^2+y^2-2\sigma^2}{\sigma^4}\right).e^{\frac{(x^2+y^2)}{2\sigma^2}} \tag{2.23}$$

# 3. ALANDA PROGRAMLANABİLİR KAPI DİZİLERİ VE PROGRAMLAMA TEKNİKLERİ

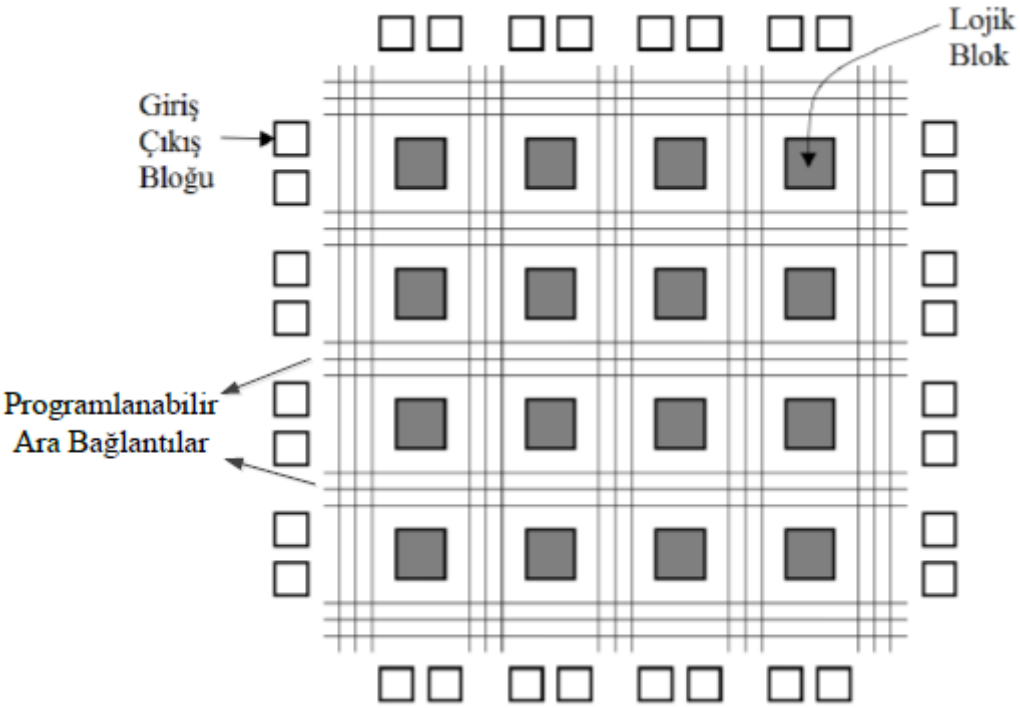
FPGA, üretiminden sonra istenen fonksiyona göre donanım yapısı kullanıcı tarafından değiştirilebilen bütünleşmiş devreler olarak tanımlanabilir. FPGA'lar, mantıksal işlemleri yerine getirmek için binlerce ve hatta milyonlarca transistor ün bağlantısından oluşurlar. FGPA bir tür entegredir fakat onu diğer entegrelerden ayıran özelliği, donanım yapısının yani iç konfigürasyonunun istenildiği gibi değiştirilebilmesidir. Donanımı programlanabilir olmayan standart entegrelerde, transistörler arasındaki bağlantılar sabittir ve değişmezler. FPGA, içindeki transistörleri birbirinden ayrı ve serbest olarak üretilmiş ham bir entegre olarak düşünülebilir. Tasarımcı tarafından belirlenen fonksiyona göre FPGA içindeki transistorlar birbirlerine bağlanır ve bu sayede istenilen fonksiyon gerçekleştirilir. Yani teorik olarak transistör kapasitesi dâhilinde akla gelen herhangi bir entegrenin yaptığı iş FPGA ile yapılabilir.

FPGA'ların en önemli özelliklerinden biri de, paralel işlem yapabilme yani aynı anda birden fazla işlemi yapabilme yeteneğidir. Sıradan entegreler ya hiç paralel işlem yapamazlar ya da çok sınırlı yapabilirler. FPGA'de ise uygulamaya ve kapasiteye göre, birbirine paralel onlarca belki binlerce işlem aynı anda yapılabilir. Bu da paralel işlem gerektiren uygulamalarda FPGA'leri eşsiz kılmaktadır. Örneğin, gerçek zamanlı yüksek çözünürlüklü bir video görüntüsü üzerinde filtreleme işlemi yaparken; peş peşe sıralanan resimlerin ilk karesi giriş portlarından alınır, filtrelenir ve çıkış portlarından gönderilir. Sonra ikinci resim için de aynı işlemler gerçek zamanlı olarak tekrarlanır. Standart entegreler kullanılırsa, bu üç işlemi (alma, filtreleme, gönderme) sırayla yapıp bitirdikten sonra gelen ikinci resim alınmaya başlanır. Eğer bu işlemler yeterince hızlı yapılamazsa sıradaki resim kaçırılabilir. FPGA'de ise bu işlemler paralel olarak devam eder. Örneğin, ilk resim alınıp filtreleme işlemi yapılırken ikinci resim alınmaya başlanır. İlk resim gönderirken ikinci resim filtrelenir ve üçüncü resim alınmaya başlanır. Buna ilaveten, filtreleme işlemi genelde yoğun çarpım gerektiren bir işlemdir. Standart bir işlemci ile bu çarpma işlemleri, sırayla yapılmak zorundadır. Oysa ki FPGA ile bu işlemler paralel olarak, yani çok hızlı bir şekilde yapılmaktadır.

FPGA'lar özetle, paralel işlem yeteneğine sahip ve içyapısını istediğimiz işlev ve uygulamaya göre değiştirilebilen, donanımı program ile düzenlenebilen entegrelerdir. Karmaşık sayısal filtreler, hata tespiti ve iyileştirmesi gibi işlemler için basit toplama ve çıkarma fonksiyonlarını yerine getirirler. Hava araçları, otomobiller, radar sistemleri, savunma sanayi ve bilgisayarlar FPGA'ların kullanıldığı sadece birkaç alandır.

### 3.1. FPGA'nın İç Yapısı

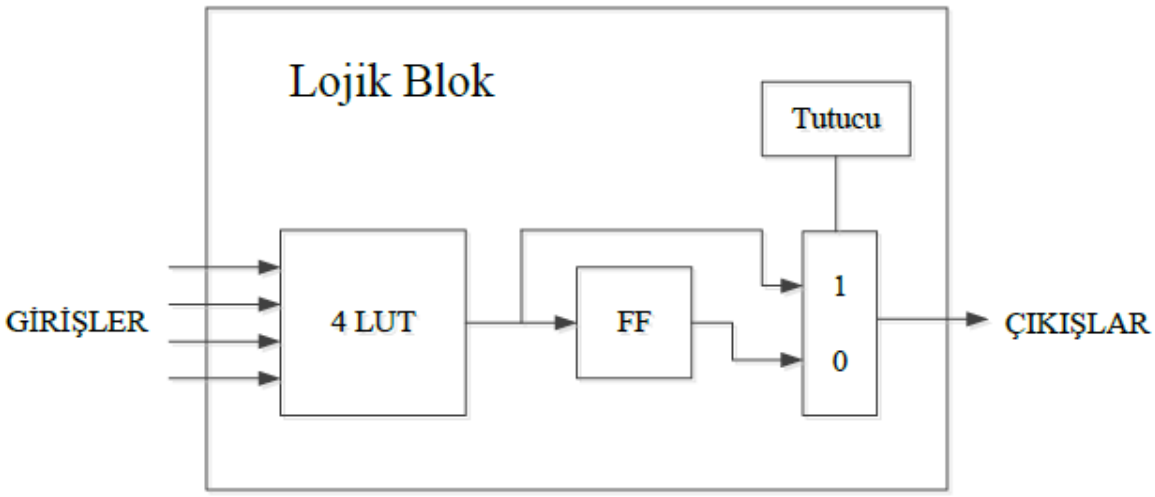
FPGA temelde giriş ve çıkış blokları ara bağlantılar ve mantık hücrelerinden oluşur. Şekil 3.1'de iki boyutlu mantıksal hücre dizilerini barından temel bir FPGA yapısı gösterilmiştir.




**Şekil 3.1** Mantıksal hücrelerden oluşmuş temel FPGA yapısı

#### 3.1.1. FPGA Mantık Hücresi

Mantık hücreleri FPGA'nın ana yapısını oluşturmaktadır. Bir mantık hücresini, 1 adet (LUT) Lookup Table, 1 adet Flip-Flop ve bir adet 2x1 Mux meydana getirir. Şekil 3.2'de FPGA daki mantık hücresi yapısı gösterilmiştir.




**Şekil 3.2** FPGA'daki mantık hücresi

LUT yapıları bir mantık işlemini gerçekleştiren küçük belleklerdir. N girişe sahip bir LUT, 2N girişe sahip bir belleği ifade eder. Birçok mantık hücrelerinin bir araya gelmesiyle ileri seviye ve yapı olarak büyük programlar meydana getirilir.

Programlanabilen anahtarlar ve şekil olarak matris tarzında veri yolları mantık hücrelerinin ara bağlantılarını meydana getirir. FPGA üzerinde oluşturulan program, mantık hücrelerinin birbirleri arasındaki bağlantıları ve programlanabilir anahtarların nasıl konumlanacağını belirler.

### 3.1.2. FPGA Bağlantı Pinleri

FPGA'yı oluşturan bağlantı pinleri temelde iki kısma ayrılır. Bunlar;

- Ayrılmış pinler
- Kullanıcı pinleri

**a) Ayrılmış Pinler:**

FPGA üzerinde bulunan pinlerin %30'a yakını FPGA'nın kendi işlevleri için ayrılmıştır. Bu pinler, FPGA üzerindeki işlevlerine göre 3 guruba ayrılır.

- Güç Pinleri: Güç bağlantısı için FPGA'ya enerji sağlayan pinlerdir.
- Konfigürasyon Pinleri: Hazırlanan programın FPGA'ya yüklenmesi için gerek duyulan pinlerdir.
- Saat Pinleri: FPGA'nın saat sinyali için kullanılan pinlerdir.

**b) Kullanıcı Pinleri**:

Kullanıcılar tarafından konfigüre edilebilen giriş/çıkış pinleridir. Bu pinler giriş, çıkış ve giriş/çıkış olarak üç kategoriye ayrılır. FPGA da her bir giriş/çıkış pini bir giriş/çıkış hücresine bağlanır. VCCIO ile giriş/çıkış hücrelerinin enerjisi sağlanır. Daha önce üretilen FPGA'ların VCCIO pin sayısı birden fazla olmasına karşın, tüm pinler aynı gerilimi kullanırdı. Son çıkan FPGA'larda ise giriş çıkışlar farklı gruplandırılıp farklı gerilim değerleriyle beslenebilir. Bu sayede bir kısım giriş/çıkış pini 2.5 V ile beslenebilirken diğer bir kısmı ise 3.3 V ile beslenebilmektedir.

### 3.1.3. Clock ve Global Line

FPGA'nın düzgün çalışabilmesi için içindeki mantıksal yapıların senkron çalışması gerekir. Bu da FPGA'nın clock sinyali tabanlı olmasından ve içindeki flip-flop'ların bu sinyal ile durum değiştirmesinden kaynaklanmaktadır. Senkron olarak çalışan tasarımlarda bütün flip-flop'ların clock sinyali ile tetiklenmesine ihtiyaç vardır. Aksi durumda, FPGA üzerinde zamana bağlı ve elektriksel problemler meydana gelir. Bu probleme karşı FPGA üreticileri tarafından "Global Line" olarak isimlendirilen bir bağlantı oluşturulmuştur. Oluşturulan bu bağlantı ile FPGA içerisinde yer alan clock bağlantı pinlerinden alınan sinyaller, tüm flip-flop'ların aynı anda clock sinyali ile tetiklenmesine olanak sağlamaktadır.

### 3.1.4. FPGA RAM Blokları

FPGA'ların birçoğunda bellek üniteleri olarak RAM yapıları kullanılmaktadır. Bu yapılar FPGA'daki mantıksal yapıların işleyişi sırasında gereksinim duyulan geçici depolama işlevini yerine getirmek için kullanılır. Tekli ya da çoklu erişim bu yapılarda desteklenen bir özelliktir. Birden fazla uygulamanın RAM üzerinde okuma/yazma yapabilmesi çoklu erişim ile sağlanmaktadır. Çoklu erişim ile farklı clock sinyalleri ile çalışan bloklar arasında veri transferi gerçekleştirilebilmektedir. Buna örnek olarak; 30 MHz clock sinyali ile çalışan bir veri toplama ünitesinden, 60 MHz ile çalışan bir veri

işleme ünitesine veri aktarımı verilebilir. 30 MHz ile çalışan ünite veriyi RAM'e kaydederken 60 MHz ile çalışan ünite veriyi RAM'den alarak işleyebilir.

FPGA ihtiyaç duyulan büyük RAM miktarları için, içerisinde bulunan blok RAM'ler kullanılırken, düşük veri kullanımı durumunda ise mantık hücreleri içerisine dağıtılmış daha küçük RAM yapıları kullanılmaktadır. FPGA üreticileri bazında bakıldığında, Xilinx dağıtılmış RAM için bazı mantık hücrelerini kullanırken, Altera ise blok RAM bloklarını farklı boyutlarda paylaştırmaktadır.

## 3.2. FPGA'ların Programlanması

FPGA cihazları donanımı yazılım ile programlanabilen cihazlardır. FPGA cihazlarının kullanılır hale getirilmesi için programlanması gerekir. Ürün bazında bakıldığında, Xilinx firması düzenleme ara yüzü olarak ISE, Altera firması ise Quartus II programı kullanılır.

FPGA'nın programlanması iki yöntem ile yapılabilir:

- Grafiksel tasarım
- HDL

Grafiksel olarak tasarımda, düzenleyici program içerisinde yer alan mantık kapıları ve diğer program araçları kullanılır.

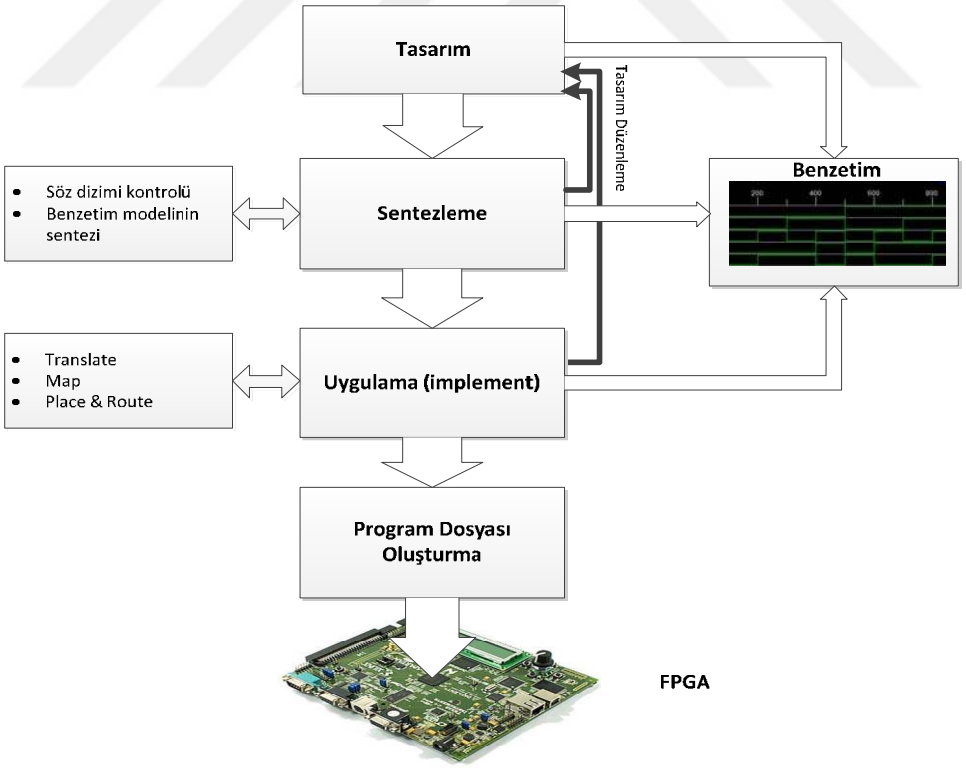
HDL kullanılarak yapılan tasarımda ise, tasarım için HDL dillerinden biri kullanılır. Bir donanımın yazılımsal olarak tanımlanmasını sağlayan dil HDL olarak adlandırılır. HDL için FPGA'larda kullanılan en yaygın programlar VHDL ve Verilog'dur.

### 3.2.1. Akıllı Özellik (IP)

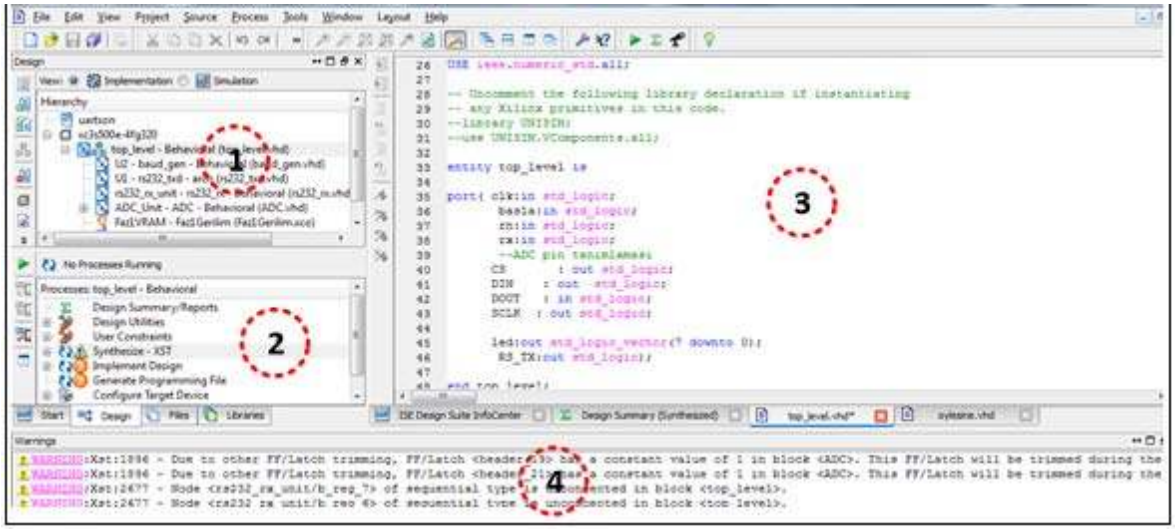
Bu özellik farklı FPGA gurupları için performans ve hız itibariyle iyileştirilmiş hazır işlevleri temsil etmektedir. Temel işlevler ücretsiz olarak üretici tarafından sağlanırken, üst seviye işlevlerde genelde ücret talep edilmektedir. IP özelliğini üretici firma sağlayabildiği gibi şahıslar ya da üçüncü parti firmalar da sağlayabilmektedir.

### 3.2.2. ISE Entegre Yazılım Ortamı

Bu tez çalışmasında gerçekleştirilen uygulamalarda, Xilinx firmasının üretmiş olduğu Spartan 3E FPGA cihazı kullanıldığından dolayı bu bölümde, ISE yazılımının sahip olduğu özellikler ve kullanım şekli ile ilgili bilgiler sunulmuştur. Şekil 3.3'te, FPGA programlamasında ISE yazılım aracı kullanarak tasarım sürecinden programlama sürecine kadar olan aşamalar gösterilmiştir. Tüm bu araçlar, tasarım dosyalarını ve işlemlerini organize eden tek bir yönleyici aracı ile başlatılabilmektedir. ISE yazılımı içerisinde, tüm bu işlemleri ve dosyaları barındıran yönleyici aracı "Project Navigator" olarak adlandırılmaktadır. Bu araca erişebilmek için ISE program kurulduktan sonra, masa üstünde yer alan ilgili ikona tıklanarak ya da başlat menüsünden "Başlat → Tüm Programlar → Xilinx ISE Design Suite 14.5 → Project Navigator" yolu ile erişilebilir. Şekil 3.4'de, proje yönleyici aracına ait ekran görüntüsü verilmiştir.




**Şekil 3.3** ISE ile FPGA programlama basamakları




**Şekil 3.4** ISE tasarım yönleyici ekranı

Tasarım yönleyici ekran görüntüsü üzerindeki işaretli bölümler şu şekilde sıralanabilir:

***1-) Kaynak penceresi*:** Tasarıma ait tüm kaynak dosyalarının yer aldığı penceredir. Bu pencere, içerisindeki herhangi bir dosyaya çift tıklandığında dosya içeriğini görüntülemektedir.

***2-) İşlem penceresi:*** Bu pencere içerisinde, tasarıma ait işlemler yer almaktadır. Bu işlemlerden herhangi birisine çift tıklandığında, ilgili işlem yürütülmektedir. Bu işlemler sırası ile sentezleme, uygulama, program dosyası oluşturma ve cihaz programlama şeklindedir.

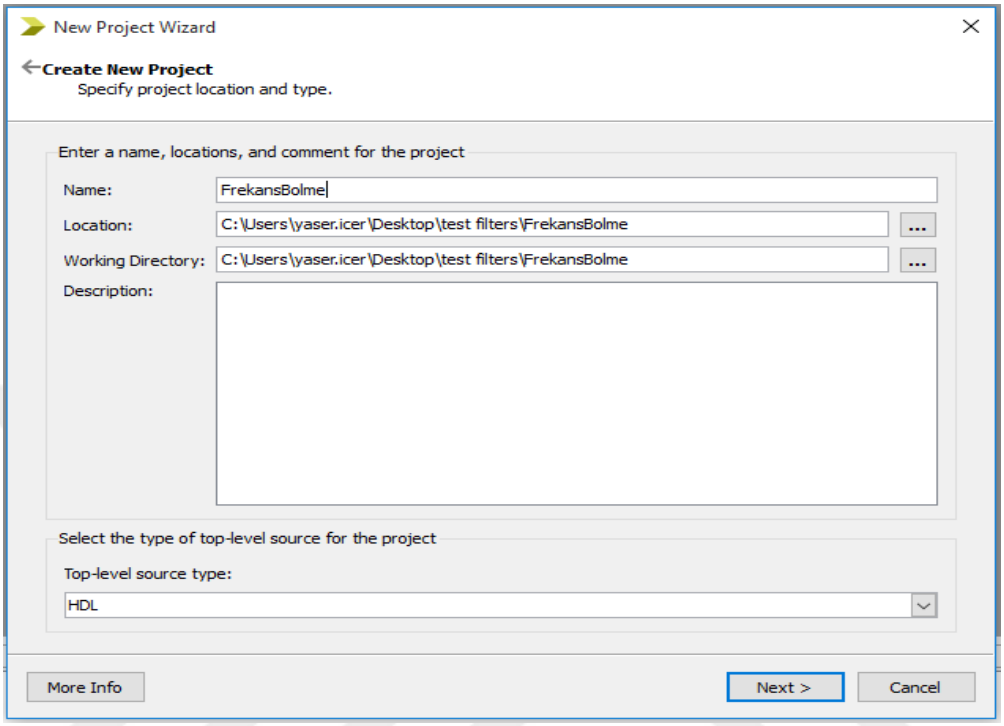
***3-) Kod penceresi:*** Seçilen herhangi bir HDL dosyasına ait HDL kodlarını göstermektedir. Ayrıca, bu pencere üzerinde tasarıma ait donanımsal bilgiler ve diğer gerekli sayfalarda görüntülenmektedir.

***4-) Konsol penceresi:*** Konsol paneli, işlemlerin durumları hakkında bilgiler sunmaktadır. Yürütülen işlemler sırasında oluşan tüm hata ve uyarı mesajları bu panel üzerinde görüntülenmektedir.

### 3.2.2.1. ISE ile Tasarım Projesi Oluşturmak

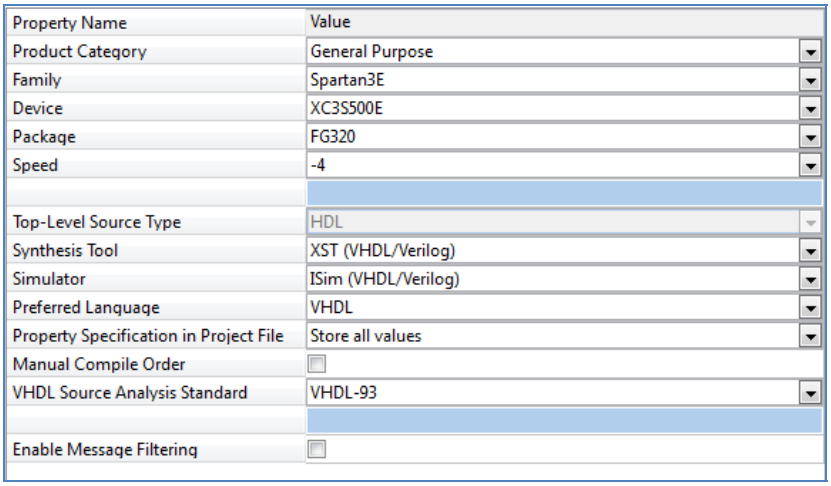
Yönleyici aracı başlatıldıktan hemen sonra, yeni bir proje dosyası oluşturmak ya da var olan bir proje dosyası üzerinde çalışmak mümkündür. Proje yönleyici ekranı ilk açıldığında daha önceden üzerinde çalışılan proje dosyaları kullanıcılara sunulmaktadır. Bir tasarım

oluşturulacağı zaman yapılması gereken ayarlamalar bulunmaktadır. Yeni bir proje oluşturmak için File→NewProject komutu kullanılır. Bu komuttan sonra Şekil 3.5'deki gibi bir pencere ekranı açılmaktadır.



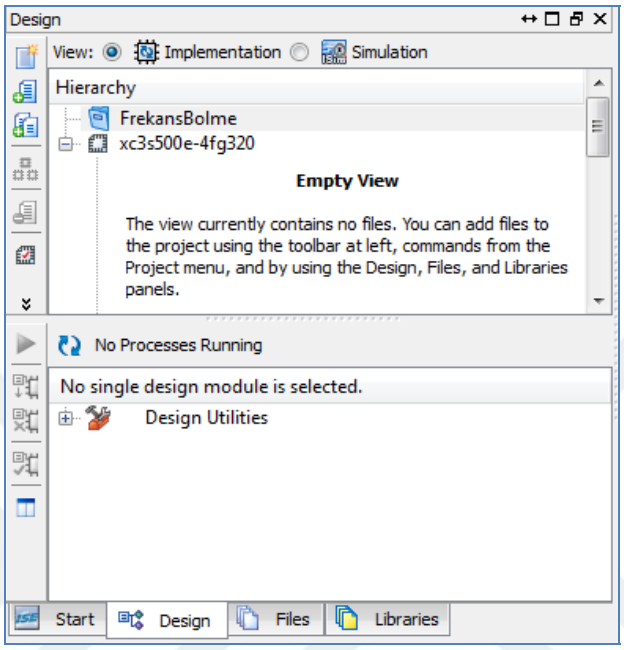

**Şekil 3.5** Yeni proje oluşturma ekranı

Bu pencereden projenin adı ve projenin oluşturulacağı konum belirtildikten sonra, tasarımın oluşturulacağı yöntem belirtilmelidir. Burada, istenirse şematik tasarım ya da HDL kod tasarımı seçilebilir. Bir sonraki adımda, Şekil 3.6'da gösterilen, proje ile ilgili ayarlamaların yapılacağı pencere ekranı açılır.




**Şekil 3.6** Proje ile ilgili ayarlamaların yapılacağı ekran

Bu pencerede, kullanılan FPGA cihazının hangi aileden olduğu (Spartan, Virtex gibi) aygıtın adı ve paket kodu belirtildikten sonra kullanılacak sentezleme aracı, benzetim aracı ve programlama dili tanımlanır. Tüm bu işlemlerden sonra, yönleyici aracının üzerinde oluşturulan tasarım Şekil 3.7'de gösterildiği gibi verilir.




**Şekil 3.7** Oluşturulan projenin Project Navigator deki yerleşimi

Şekil 3.7'de görüldüğü gibi şimdiye kadar gerçekleştirilen adımlarda, herhangi bir kaynak kod veya dosya oluşturulmadı. Sadece yeni bir tasarım dosyası oluşturularak, bu tasarımın uygulanacağı FPGA cihaz tanımlamaları ve kullanılacak HDL dili ile ilgili çeşitli ayarlamalar yapıldı. Tüm bu ayarlamalardan sonra, artık tasarım içerisinde yapılacak işlemler paralelinde program dosyaları ve diğer gerekli kaynaklar tanımlanabilir.

## 3.3. FPGA Programlama Yöntemleri

Sayısal işlemli sistemler, sayısal bilgileri işlemek için hazırlanmış hızlı donanımlar ve bu donanımlara işlevsellik kazandıracak yazılımlar ile gerçekleştirilir. Sayısal işlemli sistem oluşturmada, donanım ve yazılım tabanlı yöntemler kullanılmaktadır [28].

### 3.3.1. Donanım Tabanlı Programlama

Bu yöntemde, sayısal bilgiyi işlemek için kullanılan özel tüm devreler kullanılır. Bu devreler belirli bir işlevi yerine getirmek için üretildiğinden, bu işlevi hızlı ve etkili bir şekilde yerine getirirler. Fakat yapabildikleri kısıtlıdır ve belirli bir uygulama için üretildiklerinden esneklikleri oldukça azdır. Yeni problemler için, yeni ASIC yapıları tasarlanmalıdır. Bu maliyet ve zaman kaybına neden olmaktadır. Yazılım tabanlı yöntem hızlı, farklı yapılar, algoritmalar, aktivasyon fonksiyonları için esnek yapısı sebebiyle tercih edilir. FPGA-VHDL/Verilog yardımı ile bu esneklik, hızlı tasarım ve yeniden kullanılabilirlik sağlanmıştır. Ayrıca bu yöntemdeki paralellik, yazılımda sağlanamamaktadır [28].

### 3.3.2. Yazılım Tabanlı Programlama

Bu yöntemde, tasarım oldukça esnek bir yapıya sahiptir. Mikroişlemcinin çalıştırdığı komutlar değiştirilerek, bir donanım değişikliğine gereksinim duymadan yeni fonksiyonlar eklenebilir. Mikroişlemciler üzerinde çalışan uygulamalar; tek bir işlemcinin kaynaklarını tüketirken, yavaş fakat esnek yazılımlar ile çalışırlar. Bu tip kullanımlar, ASIC devre elemanlarına göre daha işlevseldir. Fakat basit bir uygulama için komutların bellekten okunup, yorumlanıp gerçekleştirilmesi, sistemin performansını ve hızını düşürür. Yeniden düzenlenebilen işlem sistemleri olarak da adlandırılan bu sistemler, esnek ve genel amaçlı yapıları sayesinde yeni bir üretim aşamasına ihtiyaç duymadan, kullanıcı ihtiyaçlarına, sistem özelliklerine ve değişik protokollere kısa sürede cevap verebilirler. Bu özellikleri sayesinde yeniden düzenlenebilen işlem sistemleri, donanım tabanlı sayısal işlem sistemlerine göre daha hızlı sayısal tasarım ortamı oluşturarak, bu iki sistem arasındaki boşluğu doldururlar [28]. Yeniden düzenlenebilir sayısal işlem sistemlerinin ihtiyaç duyduğu esnek donanımlar FPGA ile mümkündür. Özellikle, SRAM tabanlı FPGA'lar tekrar düzenlene bilirlik yetenekleri ve yüksek performansları sayesinde, genel amaçlı sayısal donanımların tasarlanmasında önemli bir yer tutmaktadır.

Yakın bir tarihten beri FPGA'lar kadar etkili olan başka bir kavram ise HDL'dir Donanım tanımlama dillerinin kullanılması ile çip üzerinde sistem (SoC) teknolojisini oluşturmuştur. SoC teknolojisinde çoğunlukla, donanım tanımlama dili kullanılmaktadır.

Sistemin tanımlama aşamasından sonra, derleme ve davranışsal benzetim adımları gerçekleştirilir. Sistemden beklenen cevap elde edildiğinde, zamansal benzetim aşamasına gelinir. Bütün birimler sentezleme ve yerleştirme işlemi sonunda tekrar programlanabilir olan devreye aktarılır. FPGA ve VHDL/Verilog beraber kullanılarak, donanımın hızı ve paralelliği ile yazılımın esnekliği birleştirilmek istenmiştir.

#### 3.3.2.1. VHDL Programlama Dili

VHDL, VHSIC Hardware Description Language sözcüklerinden oluşan bir kısaltmadır. VHSIC ise Very High Speed Integrated Circuit sözcüklerini içeren bir kısaltmadır.

Bu tasarım dili, aşağıda belirtilen maddeler halinde tanımlanabilir:

- Davranış
- Yapı
- Zamanlama

VHDL, C ya da C++ gibi bir programlama dili değildir. VHDL, sayısal sistemlerdeki donanımların oluşturulmasında kullanılır.

#### 3.3.2.2. Verilog Programlama Dili

Verilog bir donanım tanımlama dili olup C programlama diline söz dizimi yönüyle benzemektedir. Verilog ile tanımlanan tasarımlar belirli modüllerin kendi arasında bir hiyerarşiye bağlı olarak bir araya getirilmesi ile oluşturulmaktadır. Kullanılan modüller bazı giriş/çıkış portları ya da çift yönlü portlar ile tanımlanır. Bir modül içinde bulundurduğu kablo ve yazmaçlar ile portlar arasındaki ilişkiyi kullanarak gerçeklemek istenen programın davranışsal olarak gerçeklemesi sağlanabilir [29].

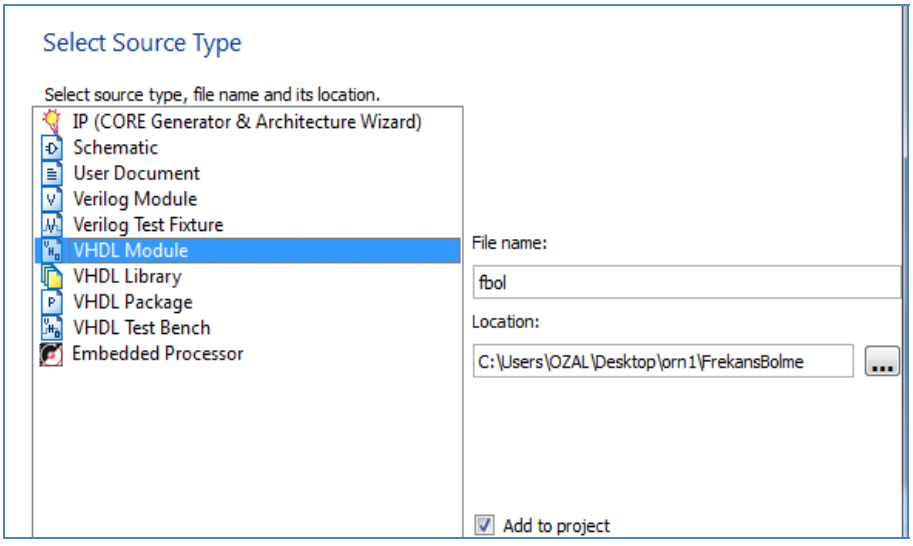
Modül içerisinde yer alan ardışık ifadeler begin/end blokları arasına yerleştirilerek program sürdürülür. Oluşturulan modüllerin sentezlenebilir olması durumunda hazırlanan program FPGA ya yüklenebilir hale gelmiş olur [29].

### 3.4. VHDL veya Verilog Modül Oluşturma

Modül oluşturma işlemi bir önceki kısımda anlatılan adımlardan sonra tasarımları belirlenen HDL dillerinden herhangi biri ile oluşturulabilir. Tasarım içerisine VHDL ya da Verilog dosyası oluşturmak için bu kısımda basit olarak Şekil 3.8'de verilen bir frekans bölücü örneğini kullanılmıştır.

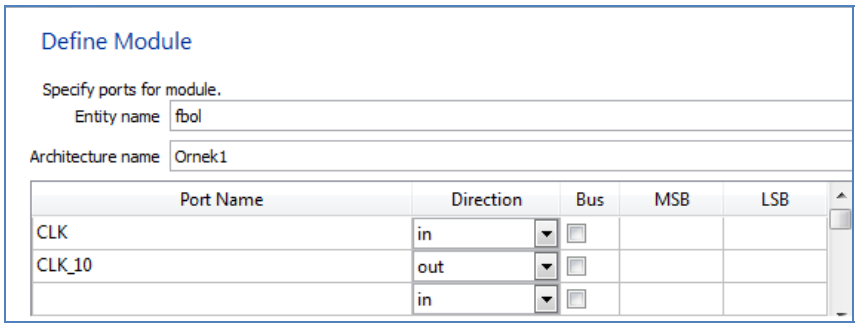

**Şekil 3.8** Frekans bölücü çalışma diyagramı

Frekans bölücü, girişine uygulanan sinyali belirlenen bir sayıya bölerek bu sinyali çıkışa aktarır. Gerçekleştirilecek olan bu örnekte, clk sinyali 10'a bölünerek çıkışa aktarılacaktır. VHDL ya da Verilog dosyasını oluşturmak için yönleyici ekranında daha önceden oluşturulan tasarım dizini üzerine sağ tıklanarak "Add→New Source" komutu verilmelidir. Bu komuttan sonra Şekil 3.9'daki gibi bir pencere ekranı ile karşılaşılır.



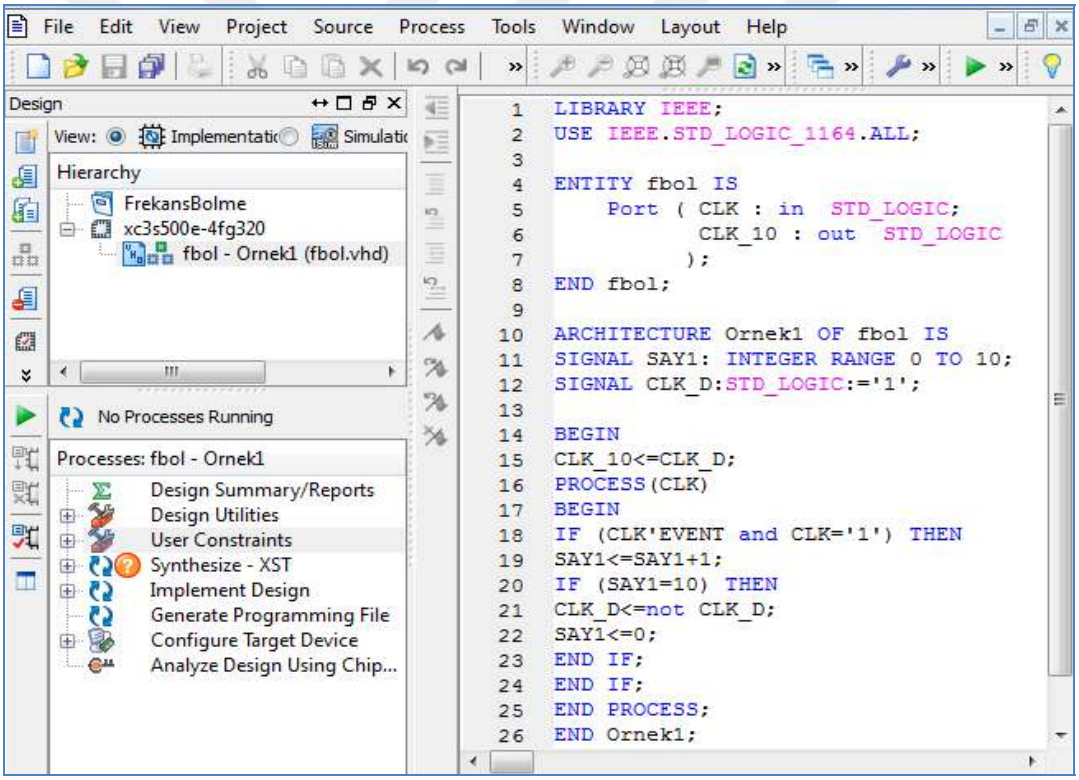

**Şekil 3.9** VHDL veya Verilog dosya oluşturma ekranı

Tasarım içerisine, birçok farklı kaynak tipi eklenebilmektedir. Bu ekrandan, VHDL ya da Verilog modül seçeneği işaretlenerek dosyanın adı girilmelidir. Bir sonraki ekran, Şekil 3.10'da görüldüğü gibi, bu modüle ait giriş çıkış pinlerinin ayarlandığı kısımdır.




**Şekil 3.10** VHDL ya da Verilog modülü için port tanımlama ekranı

Tüm bu adımlardan sonra, basit olarak bir VHDL veya Verilog tasarım projesi oluşturulabilir. Sentezleme ve diğer işlemlerden sonra tasarlanan devre, cihaz üzerinde uygulanmaya hazır hale getirilerek gerekli program dosyaları oluşturulur. Şekil 3.11'de, basit olarak oluşturulmuş bir tasarım proje örneği gösterilmiştir.




**Şekil 3.11** Örnek bir tasarım projesi

Hazırlanan tasarımların ne kadar kaynak kullandığını bilmek, tasarım açısından çok büyük önem taşımaktadır. Çünkü programın ihtiyaç duyduğu kaynaklar paralelinde cihaz seçimi yapılmalıdır. Hazırlanan bir tasarımın, sentezleme ve uygulama işlemlerinden sonra harcadığı kaynak oranlarına Şekil 3.12'de gösterildiği gibi "Design Summary" sekmesinden erişilebilmektedir.

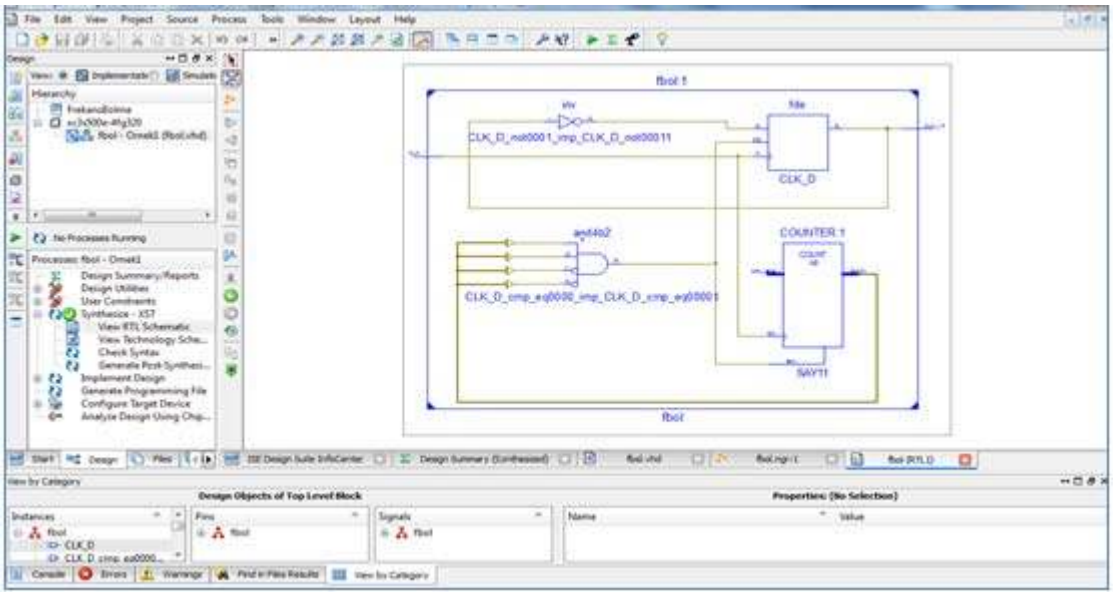
| Device Utilization Summary (estimated values) | | | [-] |
|---|---|---|---|
| **Logic Utilization** | **Used** | **Available** | **Utilization** |
| Number of Slices | 3 | 4656 | 0% |
| Number of Slice Flip Flops | 5 | 9312 | 0% |
| Number of 4 input LUTs | 6 | 9312 | 0% |
| Number of bonded IOBs | 2 | 232 | 0% |
| Number of GCLKs | 1 | 24 | 4% |

**Şekil 3.12** Hazırlanan tasarımın özeti

### 3.4.1. RTL Şeması

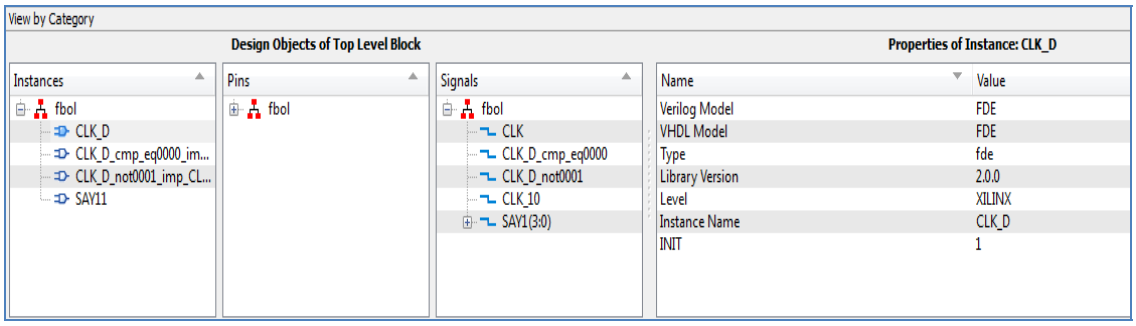
VHDL ya da Verilog kodları ile oluşturulan yazılımların her bir parçası aslında arka planda mantıksal devrelere karşılık gelmektedir. FPGA cihazı ele alınacak olursa tasarımcılar, bu cihaz içerisindeki milyonlarca programlanabilir hücreyi farklı birleşimler ile bir araya getirerek isteğe yönelik davranış sergileyen bir sistem oluşturmaktadır. ISE yazılım aracı içerisinde yer alan RTL şematik gösterim özelliği, yazılan VHDL ya da Verilog kodlarına karşılık gelen mantıksal devreleri şekilsel olarak tasarımcılara sunmaktadır.

Hazırlanan tasarım sentezlendikten sonra ISE yazılımı içerisinde Synthesize→ View RTL Schematic yolu ile devrenin şeması Şekil 3.13'de gösterildiği gibi görüntülenebilmektedir.



**Şekil 3.13** RTL şematik gösterim ekranı

RTL şematik gösterim ekranı üzerinde devrede yer alan bağlantıların ve elemanların özellikleri Şekil 3.14'te gösterildiği gibi izlenebilmektedir.



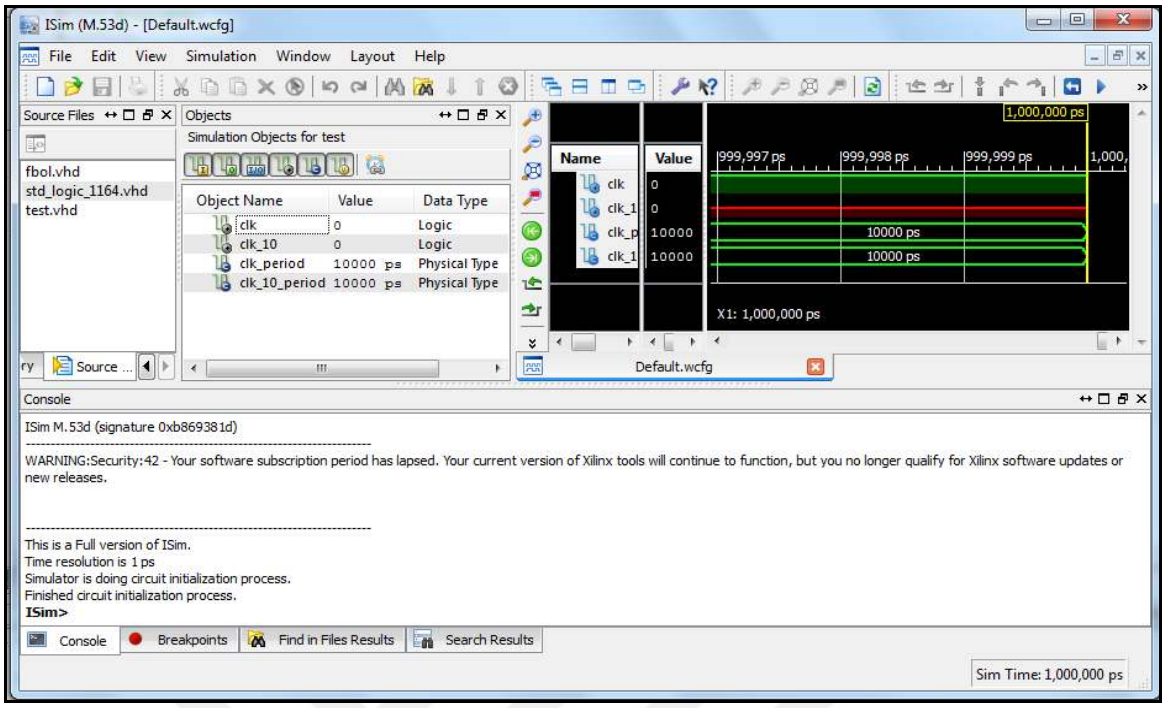

**Şekil 3.14** RTL şematik ekranın kategorik gösterimi

## 3.5. ISE ile Simülasyon İşlemi

Simülasyon işlemi tasarımcıya, devreyi donanımsal olarak uygulamadan önce tüm girişler için oluşabilecek birleşimlere göre devre çıkışlarının gözlemlenmesini sağlar. Geniş çaplı uygulamalarda simülasyon işlemi, hata tespitinde donanımsal prototip oluşturma ve test etme işleminden çok daha ekonomiktir. Simülatör çıkışlarında herhangi bir hata tespit edildiğinde, devre kolayca düzeltilerek tekrar simule edilebilir.

Simulatör, devreyi tanımlayan kaynak kod ve simülasyon süresince mantıksal hücrelerin girişini oluşturan tetikleyici girişler olmak üzere iki çeşit girişe gereksinim duyar. Simülatör içerisindeki fonksiyonlar, 10 *ps*'ler mertebesinde çok küçük parçalara ayrılmaktadır. Her bir zaman diliminde simülatör, değişen sinyalleri tespit ederek bu sinyalleri devrenin HDL kaynak kodu üzerinden işletir. Simülasyon aşaması, tasarım ve genel kurallar kadar önemli bir aşamadır. Simülasyona ayrılan zamanın, tasarıma ayrılan zamandan fazla olması gerekir. Simülasyon aşamasında kullanılan araçlar genel olarak, metin düzenleyiciler ve bir simülatörden oluşmaktadır [28].

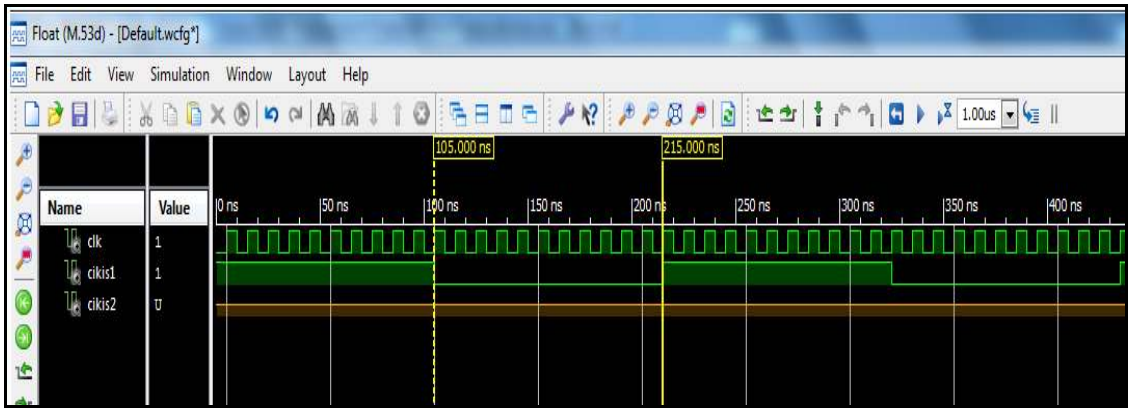
Xilinx firmasına ait ISIM aracı VHDL, Verilog gibi donanım tanımlama dilleri üzerinde davranışsal ve zamana bağlı simülasyonlar gerçekleştirmektedir. Şekil 3.15'te ISIM aracının ekran görüntüsü gösterilmiştir.




**Şekil 3.15** Xilinx ISE Simulator (ISim) aracının ekran görüntüsü

ISIM grafiksel ara yüzü içerisinde panelleri içeren; ana pencere, çalışma penceresi ve araçlar bulunmaktadır. Ana pencere içerisinde; simüle edilebilir tasarım kısımları, dalga ekranına sinyal ekleme ve görüntüleme, simülasyon için komut penceresini kullanma gibi işlemler gerçekleştirilebilmektedir.

Bölüm 3.4'te anlatılan frekans bölme örneğine ait simülasyon işlemi sonucu Şekil 3.16'da verilmiştir.



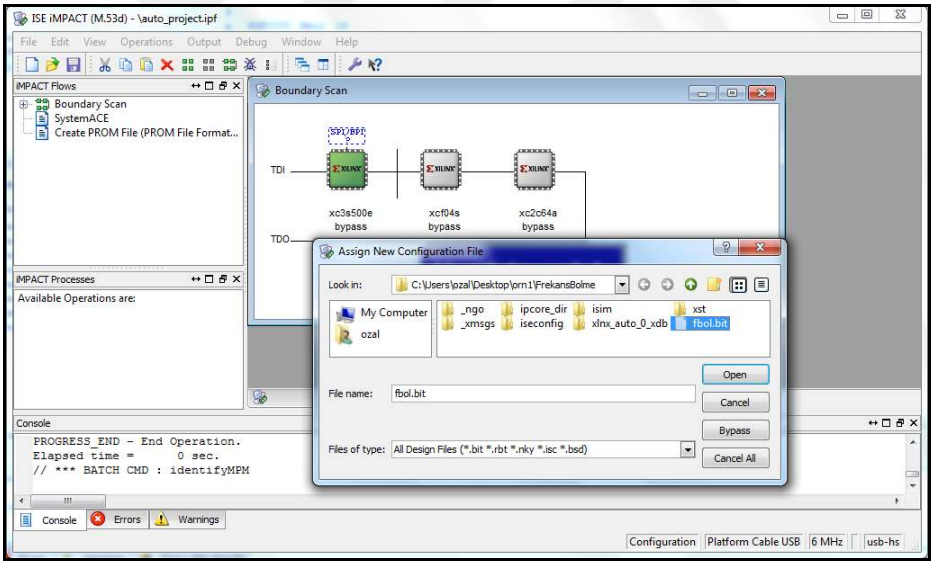

**Şekil 3.16** Frekans bölme örneğinin simülasyon sonucu

Simülasyon ekranı üzerinde, oluşturulan tasarıma ait hem zaman olarak hem de davranış olarak bilgiler sunulmaktadır. Frekans bölme işleminde clk'nın 10 darbesinden

sonra durum değiştiren başka bir clk sinyali üretildiği benzetim ekranı üzerinde açıkça görülmektedir.

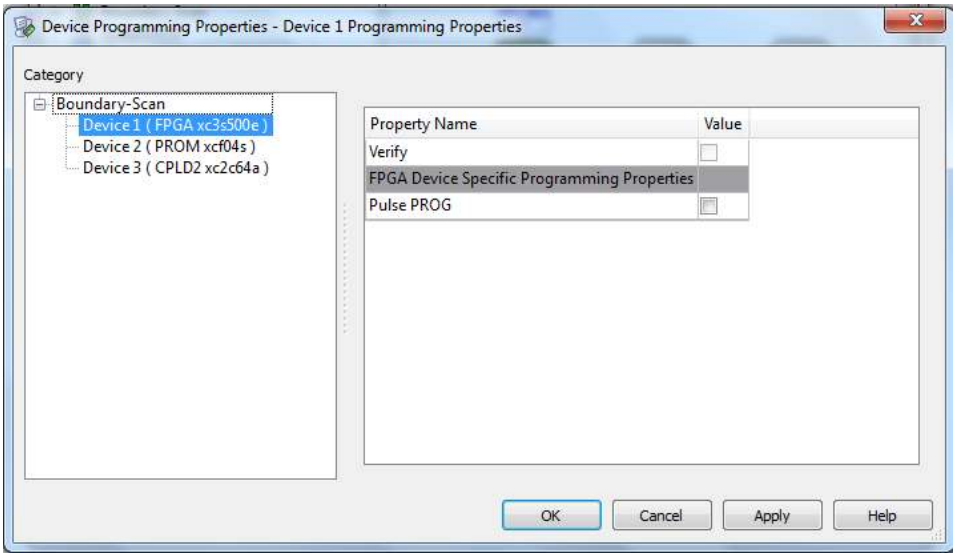
### 3.6. Oluşturulan Yazılımın FPGA'ya Yüklenmesi

ISE ortamında oluşturulan tasarımların, FPGA cihazı üzerinde yürütülmesi için yazılımın cihaza yüklenmesi gerekmektedir. Xilinx firmasına ait FPGA cihazlarına yazılım yüklemek için iMPACT adlı araç kullanılmaktadır. İMPACT, bir program dosya üretim ve cihaz programlama aracıdır. Bu araç çeşitli paralel kablolar, USB ve JTAG gibi protokollerle tasarımların FPGA cihazına yüklenmesini sağlar. iMPACT aracı, bit dosyaları, System ACE dosyaları, PROM dosyaları ve SVF/XSVF dosyaları oluşturabilmektedir. Şekil 3.17'de, iMPACT aracına ait ekran görüntüsü gösterilmektedir.



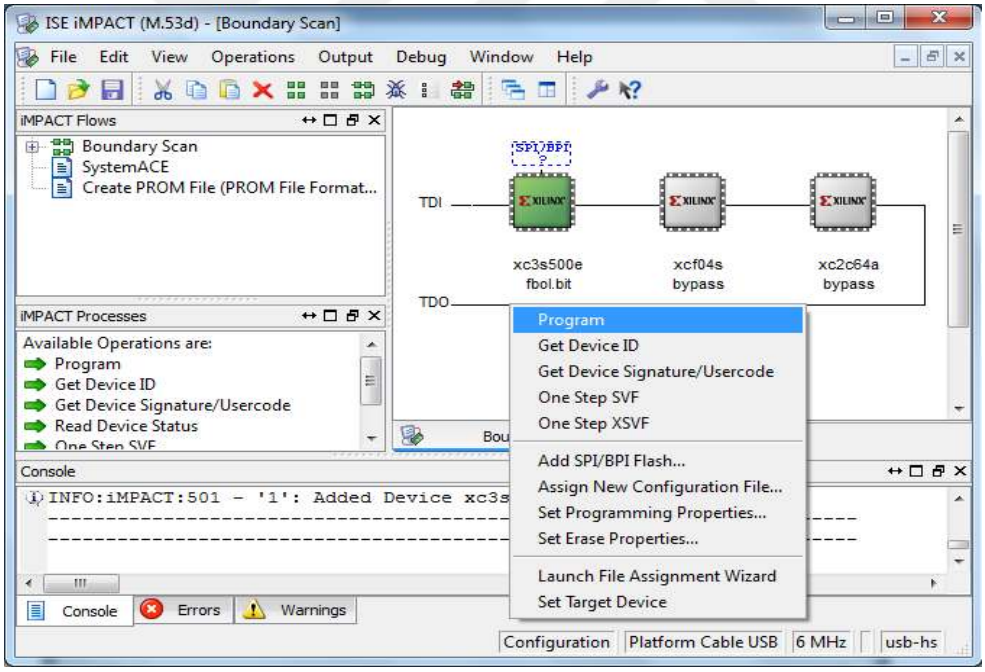

**Şekil 3.17** iMPACT aracının ekran görüntüsü

ISE içerisinde, VHDL veya Verilog ile oluşturulan tasarımlar sentezlenip yürütüldükten sonra "Program Dosyası Oluştur" seçeneği ile tasarımların FPGA üzerinde çalışabileceği kodlar oluşturulur. Bu dosya uzantıları "bit" olarak kaydedilmektedir. iMPACT ile bu program dosyası yukarıdaki gibi seçilir ve Şekil 3.18'deki aygıt programlama özelliği ekranı açılır.




**Şekil 3.18** FPGA programlama özellik ekranı

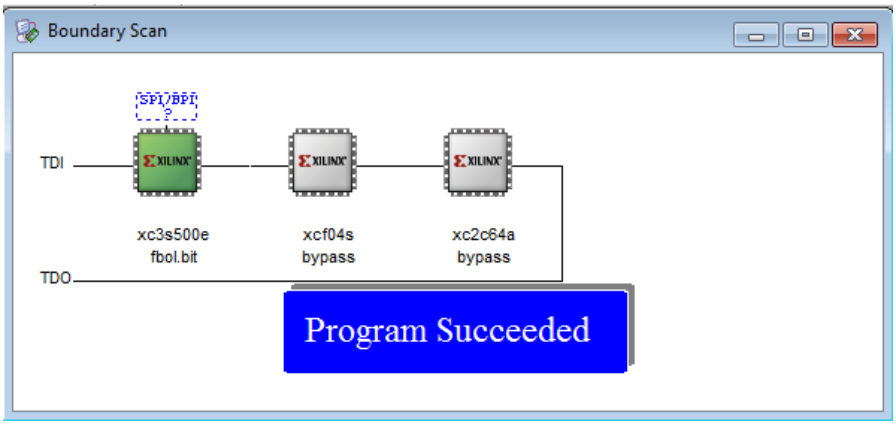
Özellik ekranından herhangi bir değişiklik yapılmadan tamam komutu ile bir sonraki ekran olan program kısmına geçilir.




**Şekil 3.19** FPGA programlama ekranı

Şekil 3.19'daki programlama ekranında, tasarım dosyasının yürütüleceği cihaz sağ tıklanarak Program komutu verilir. Program komutu ile daha önceden seçilen tasarım dosyası FPGA cihazına yüklenir.

Yükleme işleminin başarı ile tamamlandığı uyarısı Şekil 3.20'deki gibi gösterilerek programlama sonlanır.



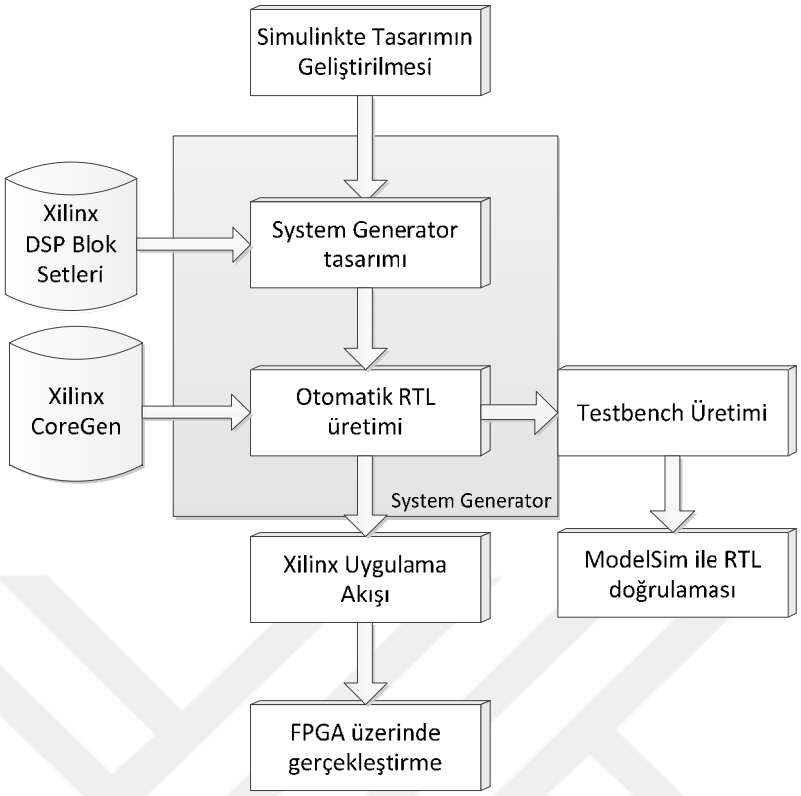

**Şekil 3.20** Programın başarı ile yüklendiğini gösteren ekran

## 3.7. XSG (Xilinx System Generator)

System Generator (XSG), Xilinx tarafından üretilen bir DSP tasarım aracıdır. Bu yazılım aracı, MATLAB'ın model tabanlı tasarım ortamı olan Simulink'in FPGA'lar için kullanılmasına olanak tanımaktadır. SysGen kullanırken tasarımlar, simulink ortamında DSP fonksiyonlarını içerisinde barındıran Xilinx özel tasarım blokları kullanılarak gerçekleştirilmektedir. FPGA program dosyası oluşturmada gerekli olan sentezleme ve yerleştirme/yönlendirme adımları otomatik olarak bu ortamda yürütülmektedir.

### 3.7.1. SysGen ile Tasarım İşlemleri

SysGen, simulink'in model tabanlı tasarım yöntemine göre çalışmaktadır. Genellikle, çalıştırılabilir bir fonksiyon standart simulink blokları kullanılarak oluşturulmaktadır. SysGen'in bu özelliği fonksiyon donanım detayı kullanmaksızın kayan noktalı sayı sistemi kullanılarak tasarlanabilmektedir. SysGen, simulink içerisinde Xilinx DSP bloklarını kullanmaktadır ve DSP blokları için gerekli olan bağlantıların uygunluklarını yüksek derecede gerçekleştirecek çekirdek fonksiyonlarını otomatik olarak çalıştırmaktadır. Şekil 3.21'de, SysGen tasarımının akış diyagramı gösterilmiştir.

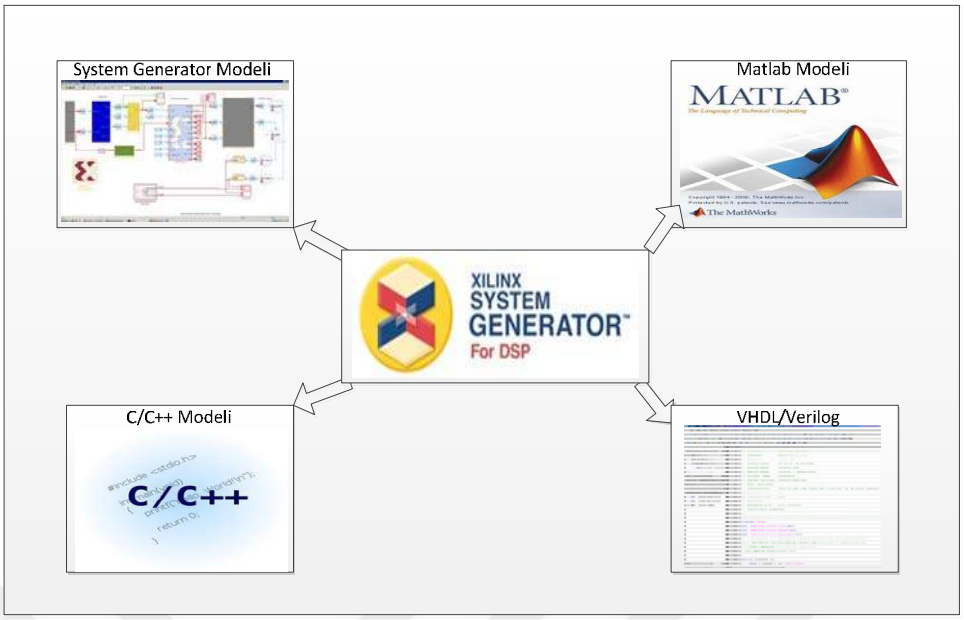



**Şekil 3.21** SysGen tasarımının akış diyagramı

Fonksiyonel ve temel akış diyagramı oluşturulduktan sonra SysGen Xilinx cihazlarına yazılımı gömme işlemlerini otomatik olarak gerçekleştirmektedir. SysGen, FPGA programlaması için veri akışının gerekli olan tüm alt uygulama araçlarını etkili bir şekilde yürütebilmektedir. SysGen ile seçime bağlı test işlemleri için ModelSim veya Xilinx ISE Simulator ile kullanılmak üzere, gerekli test vektörleri simulink ortamından elde edilebilir.

### 3.7.2. XSG ile Sistem Bütünleştirme

SysGen, DSP FPGA'larının tasarımında RTL, Simulink, MATLAB ve C/C++ gibi DSP sistem bileşenlerinin tek bir benzetim ve uygulama ortamında bir araya gelmesi için bir sistem bütünleştirme platformu sağlar. SysGen sistem bütünleştirme platform yapısı Şekil 3.22'de gösterilmiştir.




**Şekil 3.22** SysGen sistem bütünleştirme platformunun yapısı

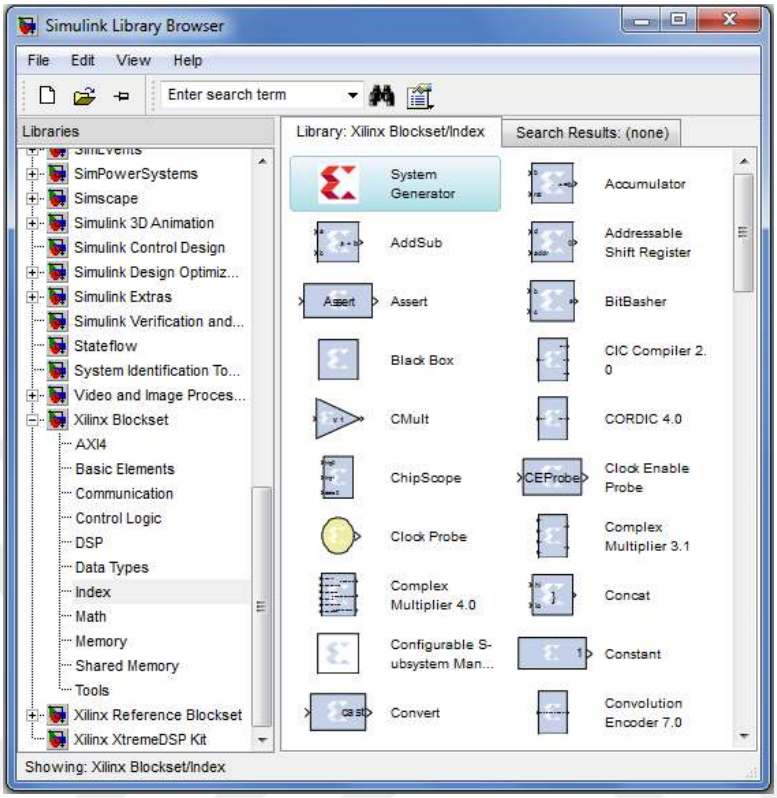
SysGen, RTL'nin Simulink'e aktarılması ve ModelSim veya Xilinx ISE Simulator uygulamalarından herhangi birisi ile eş-benzetim yapılmasına olanak sağlayan bir kara kutu bloğu destekler. SysGen ayrıca C/C++ programlarını çalıştıran dahili bir MicroBlaze gömülü işlemcisini desteklemektedir.

### 3.7.3. XSG DSP Blokları

Xilinx DSP bloklarına, Matlab aracı ile çalıştırılan Simulink kütüphanesi aracılığı ile erişilmektedir. Arama işlemlerinin kolayca yapılabilmesi için bloklar alt kategorilere ayrılmıştır. "Index" isimli alt kategori tüm blok ve sıklıkla kullanılan blokların kısa yollarını tutarak, kolay kullanım imkânı sunmaktadır. Bir DSP sistem tasarlamak için SysGen içerisinde 90'nın üzerinde DSP yapı blokları bulunmaktadır. Şekil 3.23'te, Matlab Simulink içeresinde yer alan SysGen DSP bloklarının yer aldığı pencere görüntüsü verilmiştir.

SysGen standart Simulink modelleri ile çalışmaktadır. "Gateway In" ve "Gateway Out" olarak tanımlanan iki blok, simulink benzetim modeli içerisinden FPGA cihazının giriş ve çıkışlarını tanımlamada kullanılmaktadır. "Gateway In" bloğu, kayan noktalı-sayı girişlerini sabit-noktalı sayı tipine dönüştürmektedir. Blok üzerine çift tıklanarak, gelen özellik penceresinde sabit noktalı sayılara ait taşma ve yuvarlama durumu ile ilgili

düzenlemeler yapılabilmektedir. “Gateway Out” bloğu FPGA çıkışlarını çift-hassasiyetli sayı tipine dönüştürmektedir.




**Şekil 3.23** XSG DSP blok yapılarının yer aldığı ekran

### 3.7.4. XSG ile DSP Tasarımları Oluşturma

Geçit blokları ile FPGA’nın sınırları belirlendikten sonra, Xilinx DSP blokları kullanılarak DSP tasarımları gerçekleştirilebilmektedir. Standart Simulink blokları içinde Gateway In/Gateway Out blokları kullanımı desteklenmez. SysGen içerisinde DSP tasarımı için FFT, filtre blokları, hafıza birimleri, aritmetik ve lojik bloklar gibi zengin yapılar bulunmaktadır.

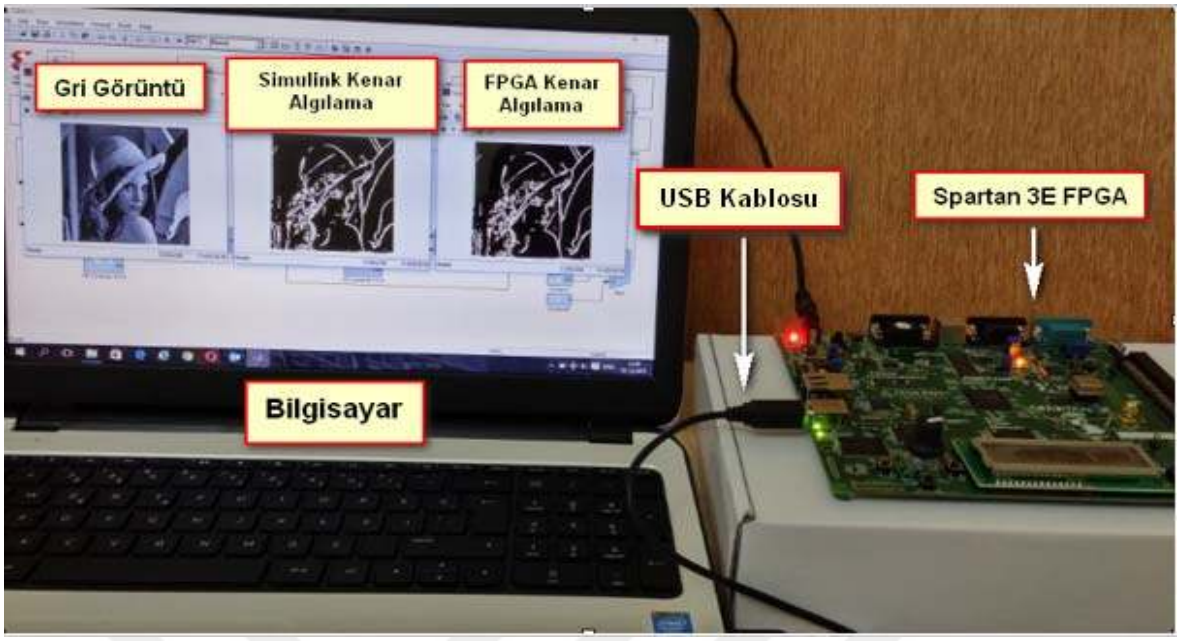
# 4. KENAR ALGILAMA İŞLEÇLERİNİN FPGA ÜZERİNDE GERÇEKLEMESİ

Bu tez çalışmasında, bilgisayar üzerinde bulunan gri formattaki görüntülerin Sobel, Prewitt ve Canny kenar işleçleri ayrı ayrı kullanılarak kenar algılama işleminin FPGA üzerinde gerçeklemesi incelenmiştir. Çalışma için oluşturulan sistemde, bilgisayar üzerinde bulunan gri formattaki görüntü Matlab/Simulink üzerinde bulunan görüntü işleme blokları kullanılarak görüntü ön işleme işlemi yapıldıktan sonra görüntüye ait piksel bilgileri 8 bit formatında seri olarak FPGA kartına gönderilir. FPGA kartı üzerinde Sobel, Prewitt ve Canny kenar algılama algoritmalarına ayrı ayrı tabi tutulan görüntü yine Simulink blokları kullanılarak görüntü son işleme işleminden sonra kenar bulma işlemi gerçekleşmiş görüntü olarak elde edilmiş olur. Aynı anda giriş görüntüsü hem Simulink üzerinden hem de FPGA kartı üzerinden Sobel, Prewitt ve Canny işlecine ayrı ayrı tabi tutularak kenar algılama işlemi gerçekleştirilebilir. Oluşturulan kenar algılama sistemleri için blok diyagram Şekil 4.1'de verilmiştir.


**Şekil 4.1** Kenar algılama işlemi için oluşturulan sistem diyagramı

Sistem diyagramında yer alan XSG ile kenar algılama bölümünde gerçekleştirilen kenar algılama algoritması FPGA kartı üzerinden gerçekleştirilmektedir. FPGA kartı ile bilgisayar arasındaki veri alışverişi FPGA üzerinde bulunan USB yapılandırma portu ile sağlanmaktadır.

Kenar algılama işlemi için kullanılan orijinal görüntü, Simulink üzerinden kenar algılaması yapılan görüntü ve FPGA kartı ile kenar algılama işlemi yapılan görüntü Matlab ortamında bilgisayar üzerinden gözlemlenebilmektedir. Şekil 4.2'de FPGA kartı üzerinde kenar algılama işlemi için oluşturulmuş olan sistemin görüntüsü verilmiştir.

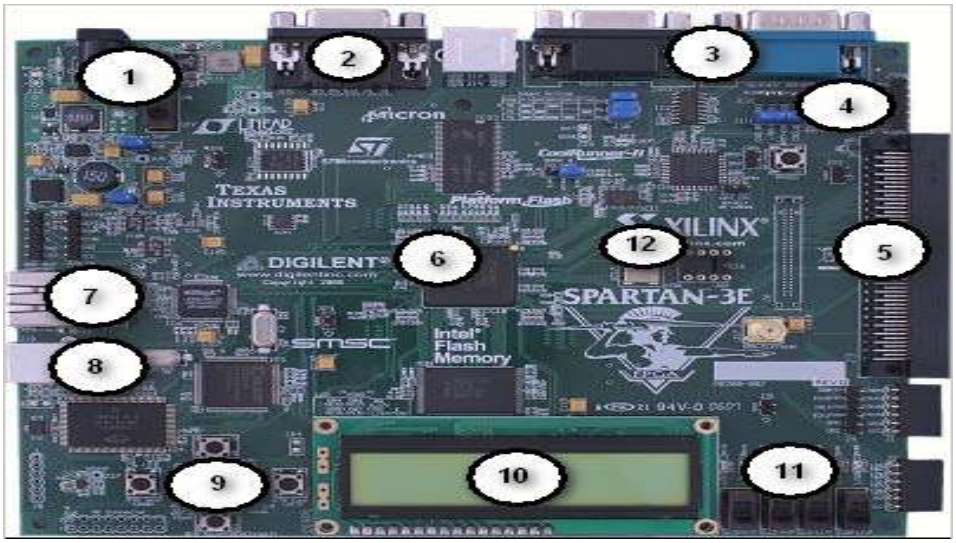



**Şekil 4.2** Kenar algılama işlemi için oluşturulan sistem

Oluşturulan sistem bilgisayar üzerinde bulunan Matlab/Simulink programı ve kenar algılama işleminde kullanılan FPGA kartından oluşmaktadır.

## 4.1. FPGA Geliştirme Kartı

Yapılan çalışmada, Xilinx üreticisi tarafından imal edilen Spartan ailesine ait olan Spartan-3E XC3S1600E modeli FPGA geliştirme kartı kullanılmıştır. Şekil 4.3'te Spartan-3E XC3S1600E kartının üsten görüntüsü verilmiştir.




**Şekil 4.3** Spartan-3E XC3S1600E FPGA kartı

Şekil 4.3'te verilen FPGA kartı üzerindeki numaralı bölgelerin hangi işlevi yerine getirdiği aşağıda verilmiştir [30];

**1-)** AC adaptör ve güç anahtarı: FPGA cihazına elektriksel güç sağlayan kısımlardır.

**2-)** VGA: Harici ekran monitörü için kullanılan VGA bağlantı birimi.

**3-)** 2 adet RS232 seri port (DCE-DTE): FPGA cihazının diğer cihazlar ile haberleşmesini sağlayan seri veri portudur.

**4-)** JTAG yapılandırma portu: FPGA cihazının programlanması için kullanılan kısımdır.

**5-)** Genişletme pinleri (XGI): Faklı uygulamalarda kullanılmak üzere cihaz üzerinde yer alan bağlantı pinleri.

**6-)** Spartan-3E 1600 FPGA: Kart üzerinde bulunan FPGA birimi.

**7-)** 10/100 Ethernet: FPGA'nın Ethernet bağlantısını sağlayan bağlantı noktası.

**8-)** USB yapılandırma portu: USB yapılandırma portu, FPGA cihazının programlanması için kullanılan kısımdır.

**9-)** Butonlar: Genel amaçlı kullanılabilecek 5 butondur. Sağ, sol, güney, kuzey ve orta olarak kart üzerine yerleştirilmiştir.

**10-)** 16 karakter x 2-satır LCD: Metin bilgilerini görüntülemek için kart üzerinde bulunan ekranı.

**11-)** Anahtarlar: FPGA üzerinde bulunan 4 adet genel amaçlı anahtar.

**12-)** Clock: 50 MHz dahili sistem saati

FPGA kartları kullanıcılar tarafından VHDL, Verilog ya da görsel tasarım araçları kullanılarak çok fonksiyonlu işlevleri yerine getirmek amacıyla programlanabilir. Kullanılan FPGA kartını sahip olduğu diğer bazı teknik özellikler Tablo 4.1'de verilmiştir [30].

**Tablo 4.1.** Spartan-3E XC3S1600E FPGA Kartının sahip olduğu teknik özellikler

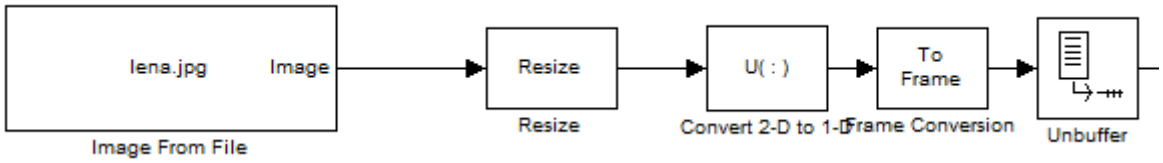
| |
|---|
| Xilinx Spartan-3E XC3S1600E FPGA |
| 2 adet 4 Mbit Xilinx Platform Flash |
| Xilinx System ACE CompactFlash |
| 64 MB, 512 Mbit DDR SDRAM Bellek |
| 250 kullanıcı I/O pin |
| 1.600.000 mantıksal kapı |
| Programlanabilir sistem saat üreteci |
| Harici saat sinyali |
| Genel Amaçlı anahtarlar(4), LED(8), butonlar |
| Genişletme pinleri |
| SPI flash (16 Mbit) |
| RS-232 seri port |
| 2 satır, 16 karakter LCD ekran |
| PS/2 fare veya klavye portu |
| VGA ekran bağlantı portu |
| 10/100 Ethernet PHY yuvası |
| 8-Kb IIC EEPROM |
| FPGA/CPLD hata ayıklama/programlama ara yüzü USB portu |
| Intel P30 StrataFlash (16 MB-128Mbit) |
| JTAG yapılandırma portu |

## 4.2. Kenar Algılama İşleminin FPGA Tasarımı

Kenar algılama işleminin FPGA üzerinde gerçekleştirilebilmesi için kullanılacak kenar algılama algoritmasına göre FPGA'nın programlanması gerekmektedir. Bu çalışmada FPGA'nın çalışması için gereken program dosyası Matlab/Simulink ile entegre çalışabilen Xilinx System Generator DSP blokları kullanılarak gerçekleştirilecektir. Xlinx DSP blokları Simulink üzerinde FPGA ara yüzünü temsil eden Gateway In/Gateway Out blokları arasında kullanılabilen bloklardır. Gateway In bloğu kendisine gönderilen veriyi seri ve sayısal veri formatında istediği için bilgisayar üzerinden alınan gri görüntü öncelikle görüntü ön işleme işleminden geçirilmesi gerekmektedir. Gateway Out bloğu ise FPGA dan gelen veriyi seri ve sayısal formatta verdiğinden kenar algılaması yapılmış görüntünün izlenebilmesi için görüntü son işleme işleminin yapılması gerekmektedir. Son olarak ise kenar algılama işlemi yapılmış görüntünün hem Simulink üzerindeki sonucunu (yazılımsal) hem de FPGA kartı üzerindeki sonucunu (donanımsal) aynı anda görmek için Xilinx System Generator (XSG) kullanılarak donanımsal ve yazılımsal birlikte benzetim işlemi gerçekleştirilmiştir. Tüm bu işlemler ayrıntılı olarak ilerleyen bölümlerde anlatılacaktır.

### 4.2.1. Görüntü Ön İşleme

Bu işlem esnasında bilgisayar üzerinde bulunan gri formattaki görüntü FPGA kartı için giriş bloğu olan Gateway In bloğunun veri giriş formatı olan seri ve sayısal veri formatına dönüştürülür. Görüntü ön işleme aşamasında kullanılan Simulink blokları Şekil 4.4'te verilmiştir.



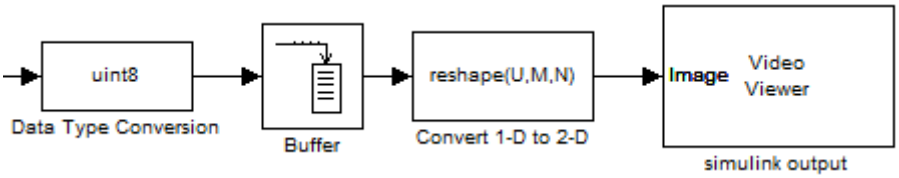

**Şekil 4.4** Görüntü ön işleme blok yapısı

Görüntü ön işlemede kullanılan blokların işlevi aşağıdaki gibidir:

- ***Image From File***: Bilgisayar üzerinde bulunan görüntüyü M*N formatında bir görüntü matrisine çevirir.
- ***Resize***: M*N boyutundaki görüntü matrisini istenen boyuta çevirir. (Bu çalışmada 256*256 alınmıştır.)
- ***Convert 2-D to 1-D***: 2 boyutlu olan görüntü piksel matrisini 1 boyutlu bir matrise çevirir.
- ***Frame Conversion***: 1 boyutlu görüntü matrisi kare-kare seri veriye dönüştürülür.
- ***Unbuffer***: Gelen seri veri yüksek örnekleme oranı ile sayısal örneklere çevrilir.

### 4.2.2. Görüntü Son İşleme

FPGA kartı için çıkış bloğu olan Gateway Out bloğunun çıkış verisi seri ve sayısal formatta olduğu için bu verinin görüntü olarak izlenebilmesi için sayısal ve seri formatta olan görüntü verisi 2 boyutlu görüntü formatına çevrilir. Görüntü son işleme aşamasında kullanılan Simulink blokları Şekil 4.5'te verilmiştir.



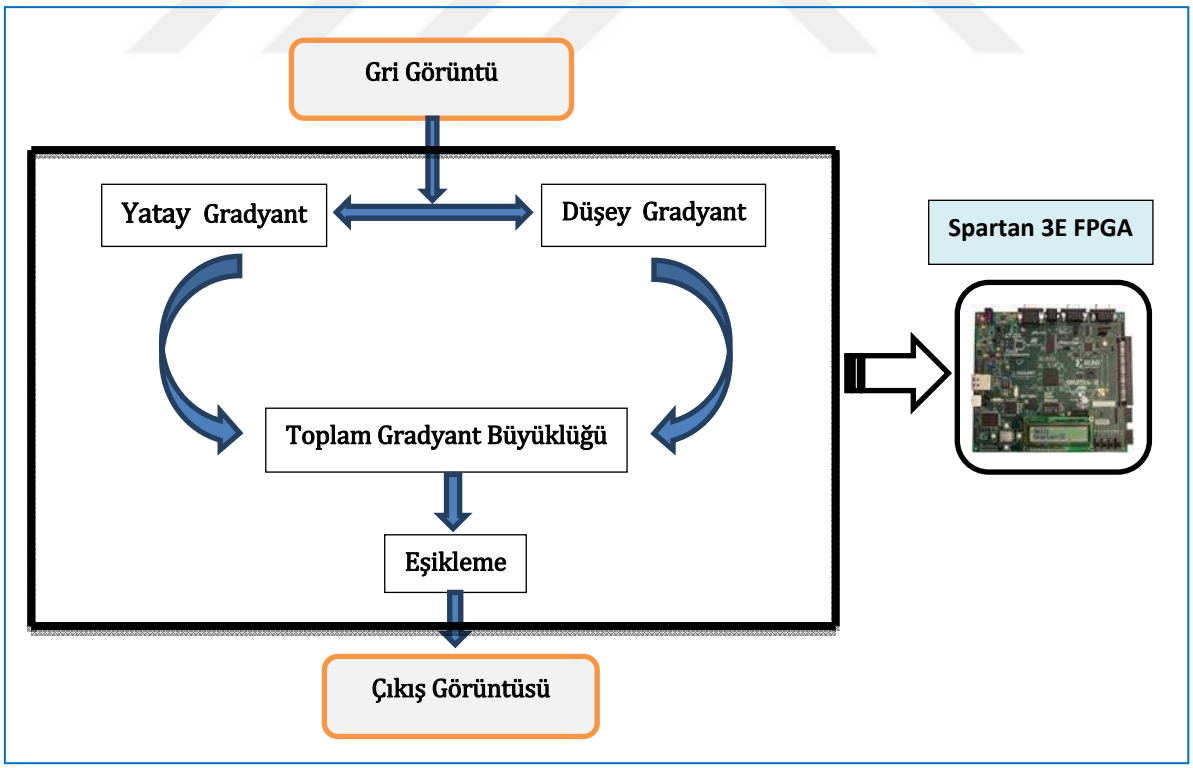

**Şekil 4.5** Görüntü son işleme blok yapısı

Görüntü son işlemede kullanılan blokların işlevi aşağıdaki gibidir:

- ***Data Type Conversion***: Sayısal veri 8 bit işaretsiz tamsayı olarak dönüştürülür.
- ***Buffer***: Sayısal veriyi düşük örnekleme oranı ile 1 boyutlu kare seri veriye çevirir.
- ***Convert 1-D to 2-D***: 1 boyutlu olan görüntü verisini 2 boyutlu görüntü matrisine çevirir. (Matris boyutu 256*256 alınmıştır.)
- ***Image Viewer***: 2 boyutlu olan görüntü matrisini görüntü olarak gösterir.

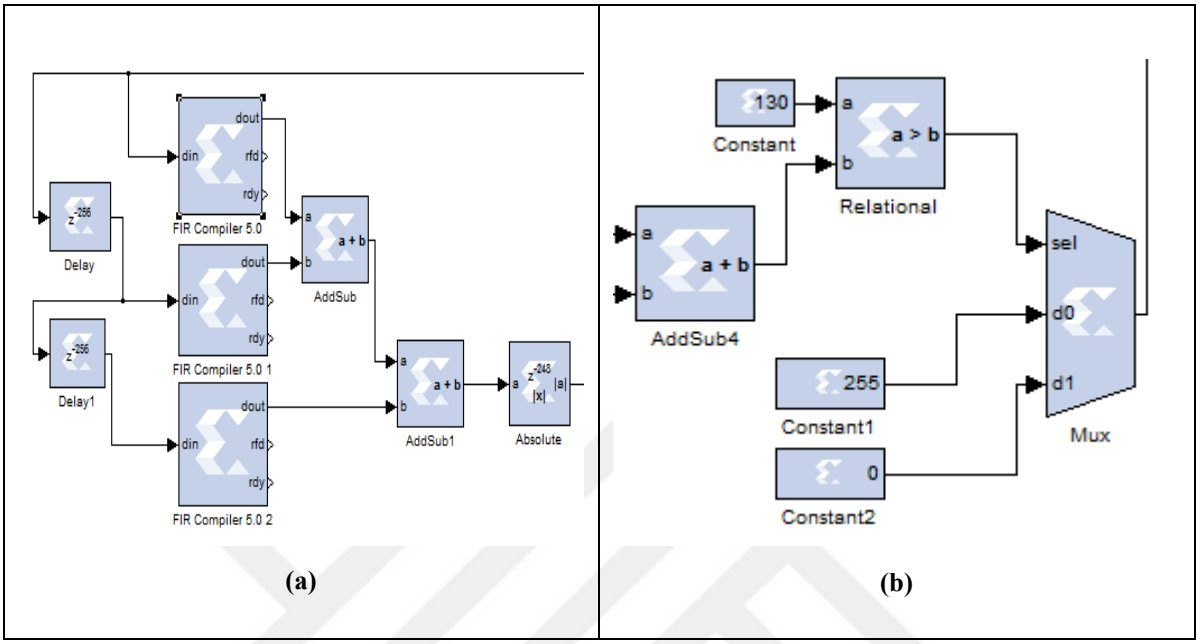
### 4.2.3. XSG ile Sobel İşlecinin Gerçeklemesi

Sobel kenar algılama işlecinin FPGA kartı üzerinde gerçekleme işlemi XSG (Xlinx System Generator) DSP blokları kullanılarak gerçekleştirilmiştir. XSG DSP blokları bölüm 4.2.1 ve 4.2.2'de anlatılan Simulink blokları arasında kullanılarak bilgisayar üzerinde bulunan gri formattaki görüntü Sobel işlecine tabi tutulup kenarları algılanmış görüntü elde edilmiştir. Şekil 4.6 XSG kullanarak FPGA kartı üzerinde Sobel işlecinin oluşturulmasına ait akış diyagramını göstermektedir.




**Şekil 4.6** Sobel işleci akış diyagramı

Sobel işleci için yatay ve düşey yönde gradyant büyüklüklerinin hesaplanıp toplam gradyant büyüklüğünün elde edilmesi Şekil 4.7 (a)'da, eşikleme işleminin yapılması ise Şekil 4.7 (b)'de verilmiştir.




**Şekil 4.7** (a) Yatay ve düşey yönde gradyant büyüklüğü hesaplama, (b) XSG DSP blokları ile eşikleme

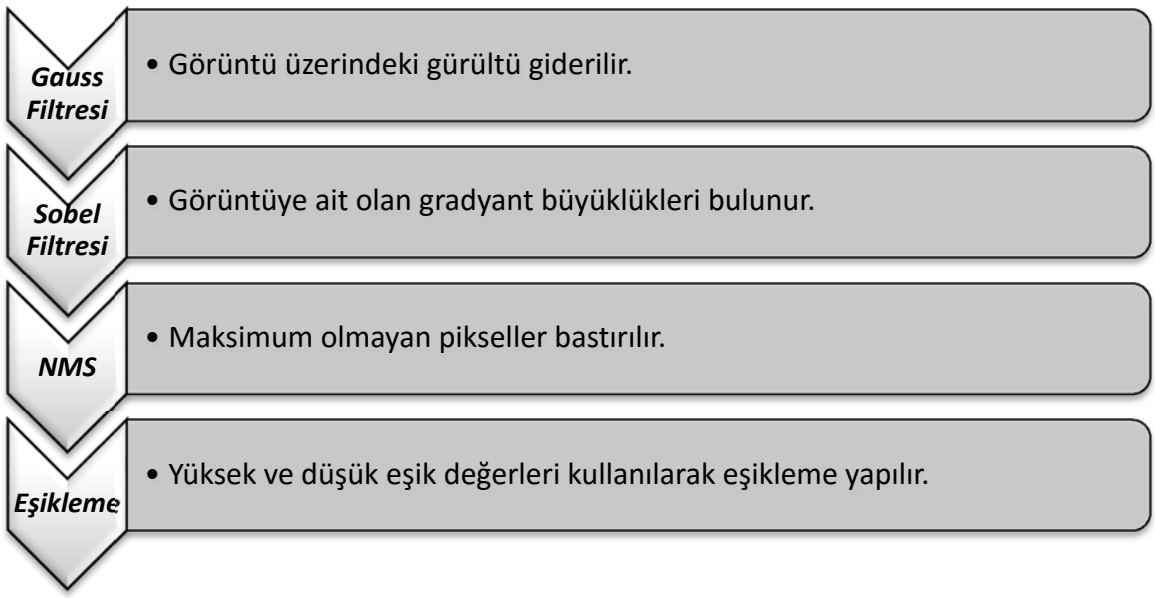
Gateway In bloğuna gelen sayısal, kayan nokta sistematiği formatındaki görüntü verisi 8 bit sabit nokta sistematiğine dönüştürülüp yatay ve düşey yönde gradyant büyüklüğünün hesaplanması için Şekil 4.7 (a)'da bulunan bloklara gönderilir. Gelen veri burada bulunan "FIR Compiler" blokları ile Sobel yatay ve düşey yön için bölüm 2.2.1.2'de gösterilen maskelerdeki katsayı vektörleri ile evirişime sokularak yatay ve düşey yöndeki gradyant büyüklüğü hesaplanır. "AddSub" ve "Absolute" blokları ile sayısal veriler toplanıp büyüklük hesaplaması yapılır. Boloklar içerisinde yer alan "$z^{-256}$ " ifadesi sayısal veriye uygulanan gecikme işlemini göstermektedir. Yatay ve düşey yönde hesaplanan gradyant büyüklükleri toplandıktan sonra görüntü verisi eşikleme işlemi için Şekil 4.7 (b)'deki bloklara gönderilir. Eşiklemede kullanılan "Constant" bloğu içerisinde yer alan değer 8 bitlik veri için girilen eşiği göstermektedir. "Constant1" ve "Constant2" bloklarında bulunan 0 ve 255 değeri ise 8 bitlik piksel değerleri için siyah ve beyaz renklerini ifade etmektedir. Eşikleme işleminden sonra kenar algılama işlemi yapılmış görüntü verisi Gateway Out bloğuna gönderilir. Bu işlem sonrasında görüntü verisi bilgisayar üzerinde görüntüleme işlemi için bölüm 4.2.2'de anlatılan görüntü son işleme bloklarına gönderilir.

### 4.2.4. XSG ile Prewitt İşlecinin Gerçeklemesi

Prewitt kenar algılama işlecinin FPGA üzerinde gerçekleme işlemi Sobel işlecinin FPGA kartı üzerinde gerçekleme işlemi ile neredeyse birebir benzerlik göstermektedir. İki işlece ait model arasındaki tek fark ise bölüm 2.2.1.3'te anlatıldığı üzere yatay ve düşey yönde uygulanan evirişim maskelerinin Prewitt işleci için farklı olmasıdır. Bu nedenle Şekil 4.7 (a)'da gösterilen yatay ve düşey yön için gradyant hesaplama blokları içerisinde yer alan "FIR Compiler" blokları için tanımlanmış olan katsayı vektörleri Prewitt işlecine ait evirişim maskelerinde yer alan katsayılar ile güncellenmiştir. Yapılan bu işlem dışındaki diğer basamaklar Sobel işleci ile aynı olarak gerçekleştirilmiştir.

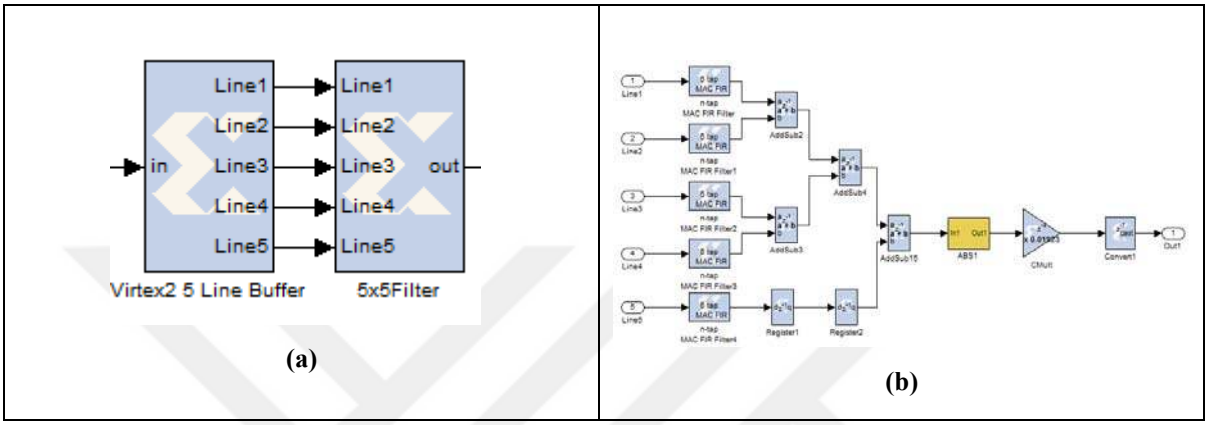
### 4.2.5. XSG ile Canny İşlecinin Gerçeklemesi

Canny kenar algılama işlecinin FPGA kartı üzerinde gerçekleme işlemi XSG (Xlinx System Generator) DSP blokları kullanılarak gerçekleştirilmiştir. XSG DSP blokları bölüm 4.2.1 ve 4.2.2'de anlatılan Simulink blokları arasında kullanılarak bilgisayar üzerinde bulunan gri formattaki görüntü Canny işlecine tabi tutulup kenarları algılanmış görüntü elde edilmiştir. Şekil 4.8 XSG kullanarak FPGA kartı üzerinde Canny işlecinin oluşturulmasına ait akış diyagramını göstermektedir.



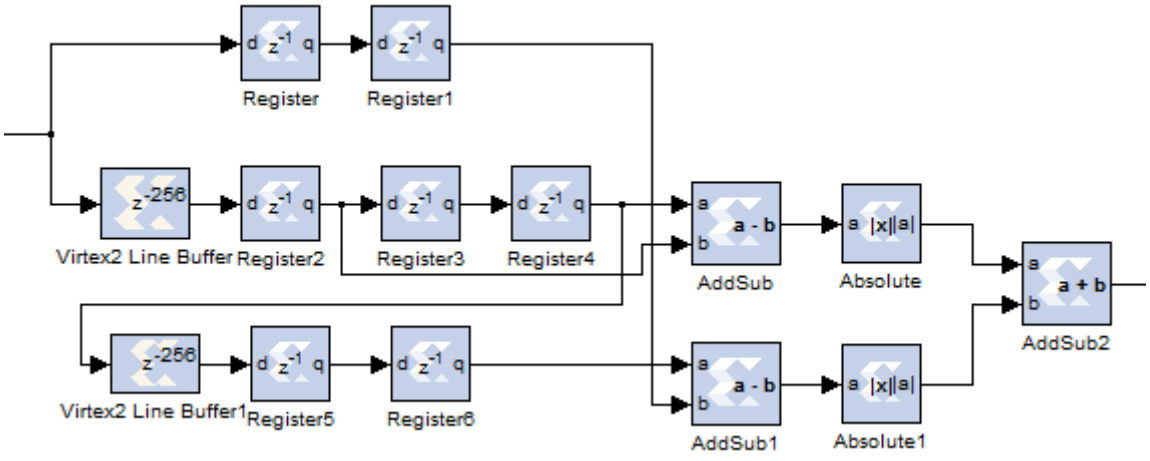

**Şekil 4.8** Canny işleci akış diyagramı

Canny kenar işleci için seri sayısal görüntü verisi öncelikle üzerindeki gürültünün giderilmesi için Şekil 4.9 (a)'da verilen Gauss Filtresi için kullanılan bloklara uygulanır. Bu bloklar içerisinde yer alan "Virtex2 5 Line Buffer" bloğu görüntüye ait ilk 5 satırlık piksel verisini tutmaktadır. Şekil 4.9 (b)'de iç yapısı verilen "5x5 Filter" bloğu ise 5x5'lik maske katsayıların görüntü verisi ile evirişimi sonucu Gauss Filtresini oluşturmaktadır.



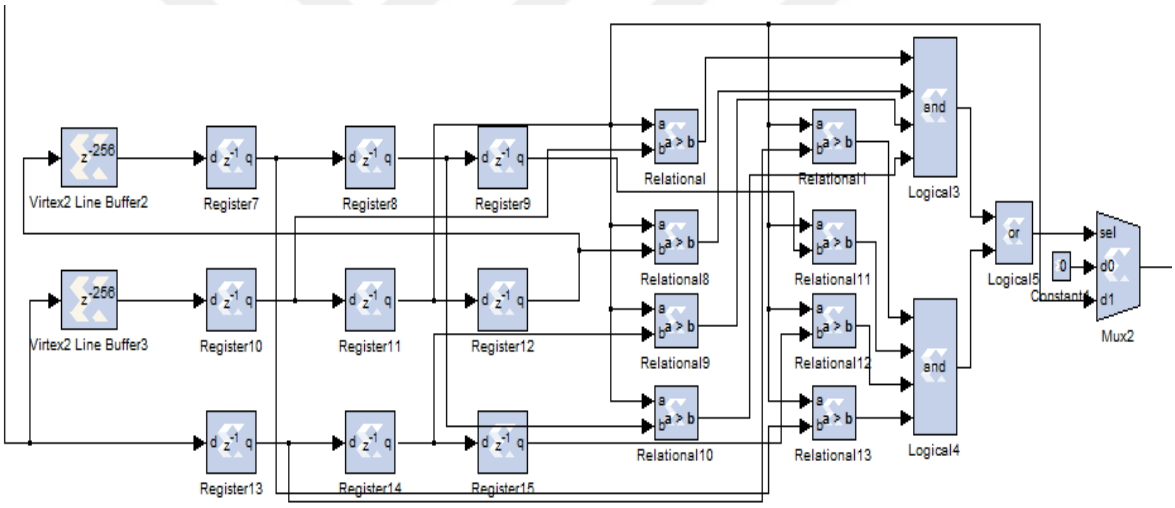

**Şekil 4.9** (a) XSG Gauss Filtresi, (b) "5x5 Filter" bloğu iç yapısı

Gürültüden arındırılmış olan görüntü verisi kenar çıkarımı işleminin gerçekleştirilmesi için Şekil 4.10'da verilen Sobel Filtresi olarak kullanılmak üzere oluşturulmuş olan bloklara gönderilir. Burada bulunan "Virtex2 Line Buffer" blokları ile 3 satırlık görüntü piksel verisi tutulurken "Register" blokları ile 3x3'lük görüntü verisi alınır. "AddSub" blokları ile de yatay ve düşey yöndeki fark bulunarak elde edilen verinin mutlak değeri alınıp toplanarak görüntüye ait gradyant büyüklükleri bulunmuş olur.




**Şekil 4.10** XSG Sobel Filtresi

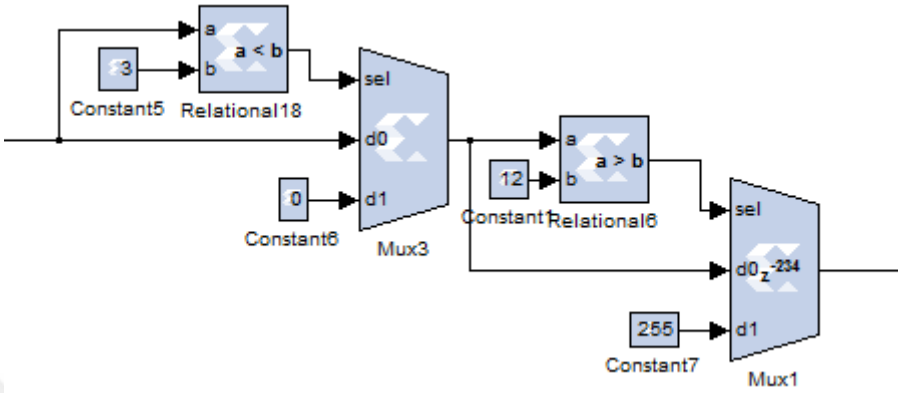
Gradyant büyüklükleri hesaplanan görüntü sinyali NMS (maksimum olmayan piksellerin bastırılması) işlevinin gerçekleştirilmesi amacıyla Şekil 4.11'de gösterilen XSG DSP bloklarına gönderilir. Burada bulunan "Virtex2 Line Buffer" blokları ile 3 satırlık görüntü piksel verisi tutulurken "Register" blokları ile 3x3'lük görüntü verisi alınır. "Relational" blokları kullanılarak bir piksele ait görüntü verisi yatay ve düşey yöndeki komşu pikseller ile karşılaştırılır. Aynı işlem çaprazda bulunan piksellere de uygulanarak "Logical" blokları kullanılarak kontrol işlemi gerçekleştirilir. Sonrasında ele alınan pikselin değeri yatay ve düşey piksel değerinden büyükse ya da çaprazda bulunan piksel değerlerinden büyükse bu piksel korunur değilse piksel değeri sıfır yapılır (Bu işlem "Mux" bloğu ile gerçekleştirilir). Sonuç olarak alınan pikselin 0º, 45º, 90º, 135º deki komşu piksel değerleri ve bu değerleri 180º ye tamamlayan diğer komşu piksel değerleri ile kıyaslama işlemi yapılarak kenarlar inceltilmiş olur.




**Şekil 4.11** XSG DSP blokları ile NMS işlemi

NMS işlemi ile kenar inceltmesi yapılan görüntü sinyali eşikleme işlemi yapılmak üzere Şekil 4.12'de belirtilen çift eşikleme işlem bloklarına gönderilir. Burada "Constant" bloklarına piksel değerinin ne olacağını ya da hangi değer ile kıyaslama yapılacağını belirlemek için 8 bitlik sabit değerler atanır. "Relational" bloklarrı kıyaslama işlemini gerçekleştirmek için kullanılırken "Mux" blokları ise kıyaslama işlemi sonrasında ilgili piksel değerinin değişmeden mi geçeceğini, piksel değerinin 0 mı olacağını ya da 8 bit sayı sisteminde 255 (ikilik sistemde 1 olup beyazı temsil eder) mi olacağının belirlenmesi için kullanılmıştır. İlk kıyaslama işleminde düşük eşik değerinin altında olan piksel değerleri

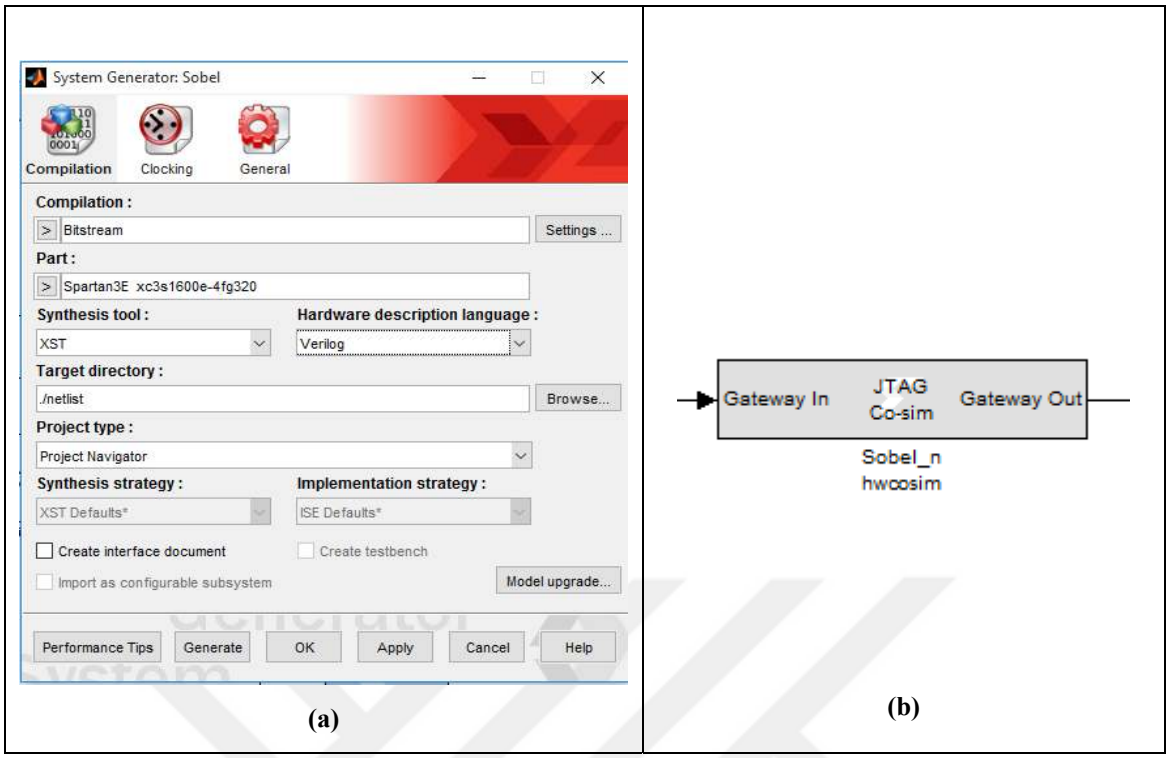
sıfırlanırken diğer piksel değerleri aynen korunmaktadır. İkinci kıyaslama işleminde ise yüksek eşik değerinin üzerindeki piksel değerleri beyaz (8 bit sayı sisteminde 255) yapılırken diğer piksel değerlerinde değişiklik yapılmamıştır. Böylece kenar olan pikseller belirlenmiş olmaktadır.




**Şekil 4.12** XSG DSP blokları ile çift eşikleme işlemi

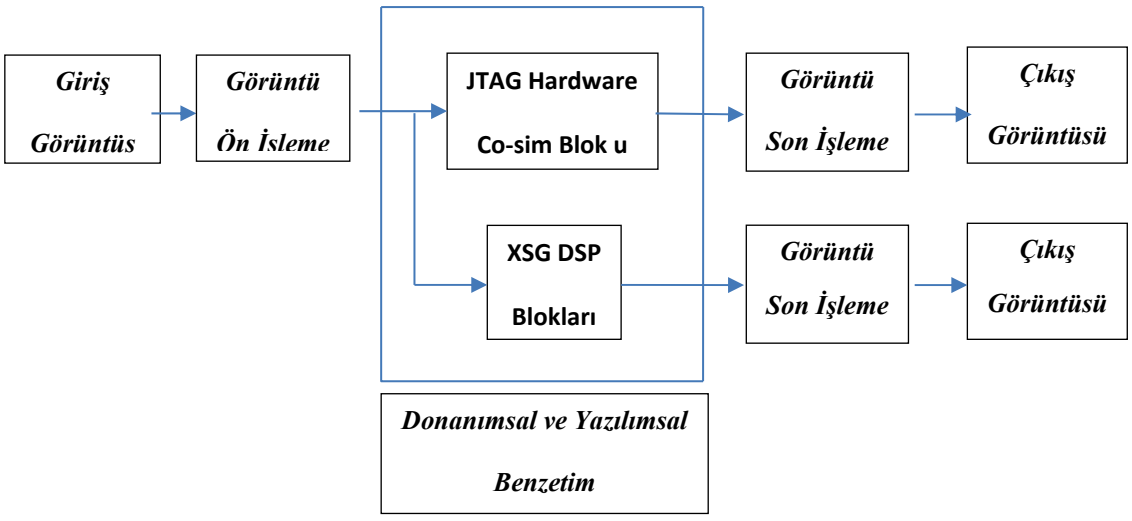
### 4.2.6. XSG ile Donanımsal ve Yazılımsal Benzetim

Sobel, Prewitt ve Canny kenar algılama işleçleri için oluşturduğumuz tasarımların FPGA kartı ile gerçeklemesi (donanımsal) ve Simulink ile gerçeklemesi (yazılımsal) işlemleri için "System Generator" bloğu bilgi penceresi üzerinde yer alan "Compilation" bölümünden öncelikle "Bitstream" seçeneği seçilir. Sonrasında aynı pencere üzerinde yer alan "Part" bölümünden kullanılacak FPGA kartının modeli seçilmelidir (Bu çalışmada Spartan-3E XC3S1600E-4FG320 kullanıldığı için bu seçenek seçilmiştir.). "Hardware description language" sekmesinden kullanılacak HDL program dili seçilebilir. "Generate" butonu ile hazırlanan tasarım için FPGA kartını programlamada gerekli olan *.bit uzantılı program dosyası ve HDL modül oluşturulmuş olur. Program dosyası oluşturulduktan sonra FPGA kartını Simulink üzerinde temsil etmek için bu defa "System Generator" bloğu bilgi penceresi üzerinde yer alan "Compilation" bölümünden "Hardware Co-Simulation" seçeneği içerisinde kullanılan FPGA kartı modeli seçilerek "Generate" butonuna basılır. Bu işlem sonrasında FPGA kartını temsil eden "JTAG Co-sim" bloğu System Generator tarafından oluşturulur [31]. Şekil 4.13 (a) System Generator bilgi penceresini, Şekil 4.13 (b) ise oluşturulan "JTAG Co-sim" bloğunu göstermektedir.




**Şekil 4.13** (a) System Generator bilgi penceresi, (b) "JTAG Co-sim" bloğu

"JTAG Co-sim" bloğunun oluşturulan sistem modeli üzerindeki yeri Şekil 4.14'te verilmiştir.

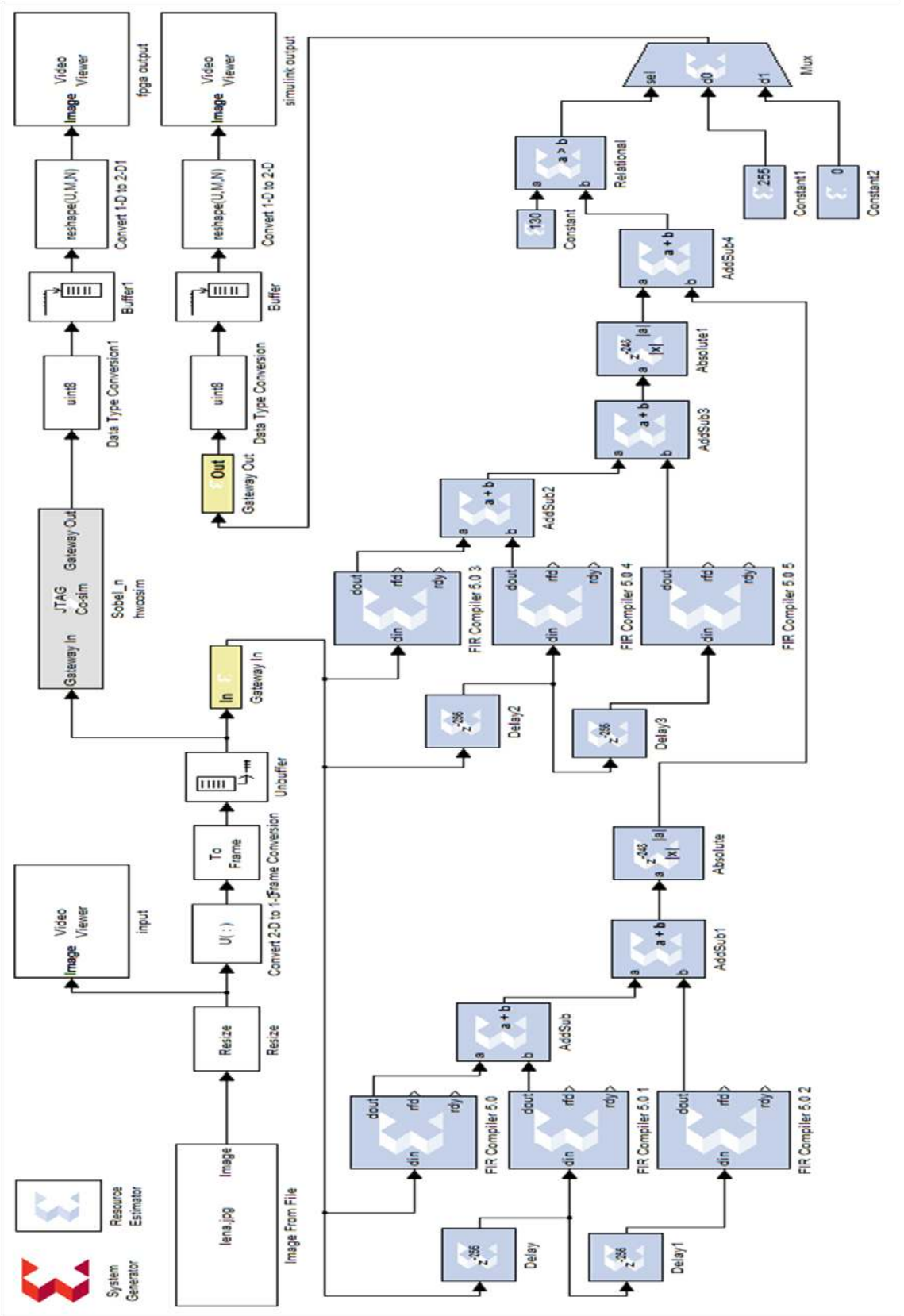



**Şekil 4.14** JTAG Co-sim bloğu eklenmiş Simulink model diyagramı

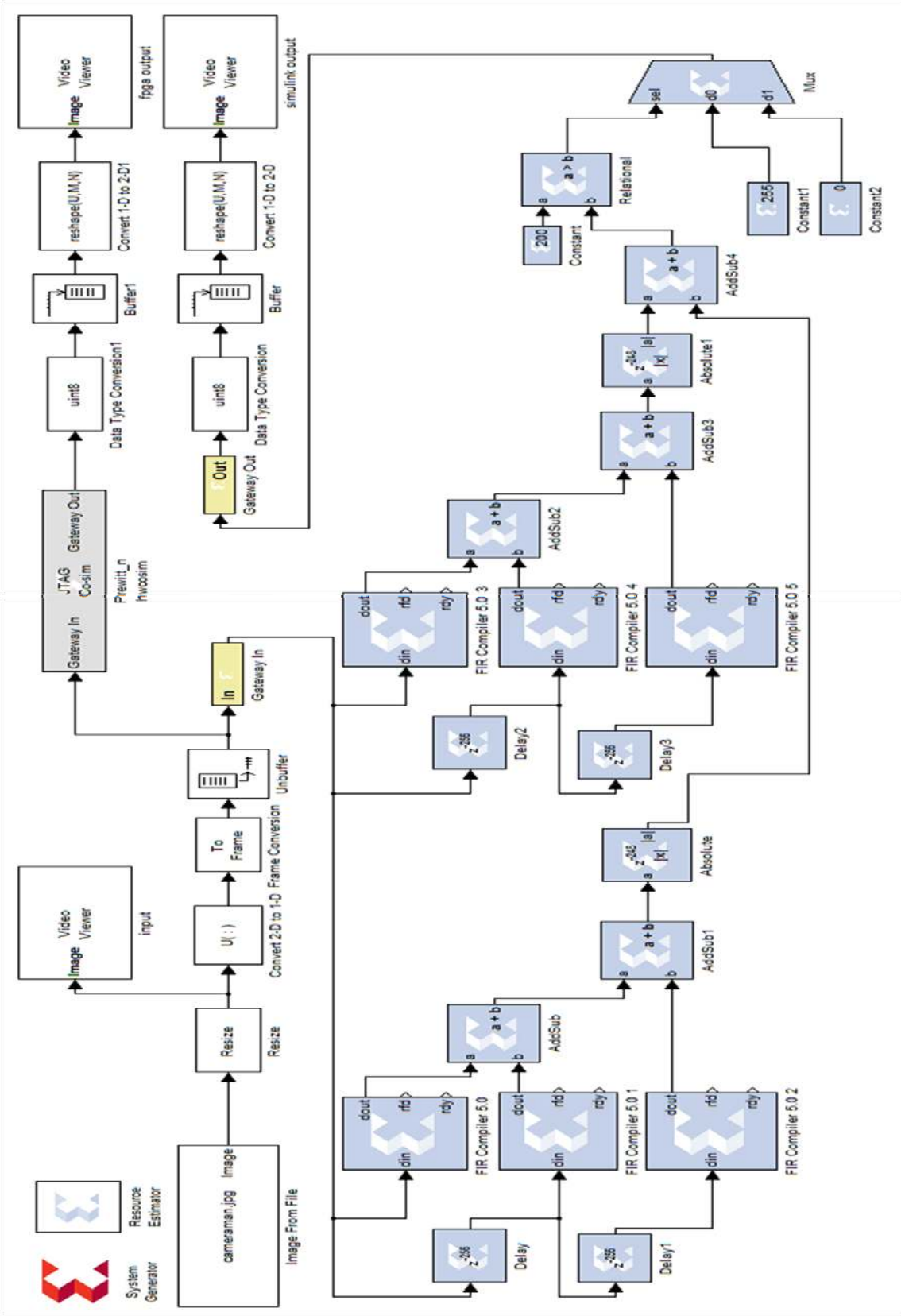
“JTAG Co-sim” bloğu kullanılarak oluşturulan Şekil 4.15 - 4.17’de gösterilen Simulink tabanlı Sobel, Prewitt ve Canny kenar algılama modelleri çalıştırıldığında *.bit uzantılı program dosyası USB yapılandırma portu üzerinden FPGA kartına her bir model için ayrı ayrı gönderilir. FPGA’nın programı çalıştırması sonucu bilgisayar ortamında bulunan gri formattaki giriş görüntüsü, görüntü ön işleme bloklarından geçerek hem XSG DSP bloklarından hem de FPGA’yı temsil eden JTAG Co-sim bloğundan geçer. Simulink ve FPGA kartı üzerinde kenar algılama işlemi yapılmış olan görüntü, görüntü son işleme bloklarından geçerek bilgisayar üzerinden görüntülenebilir hale getirilir.

Bilgisayar üzerinde bulunan görüntünün FPGA kartına gönderilmesi de USB yapılandırma portu üzerinden seri ve sayısal veri olarak gerçekleştirilir.

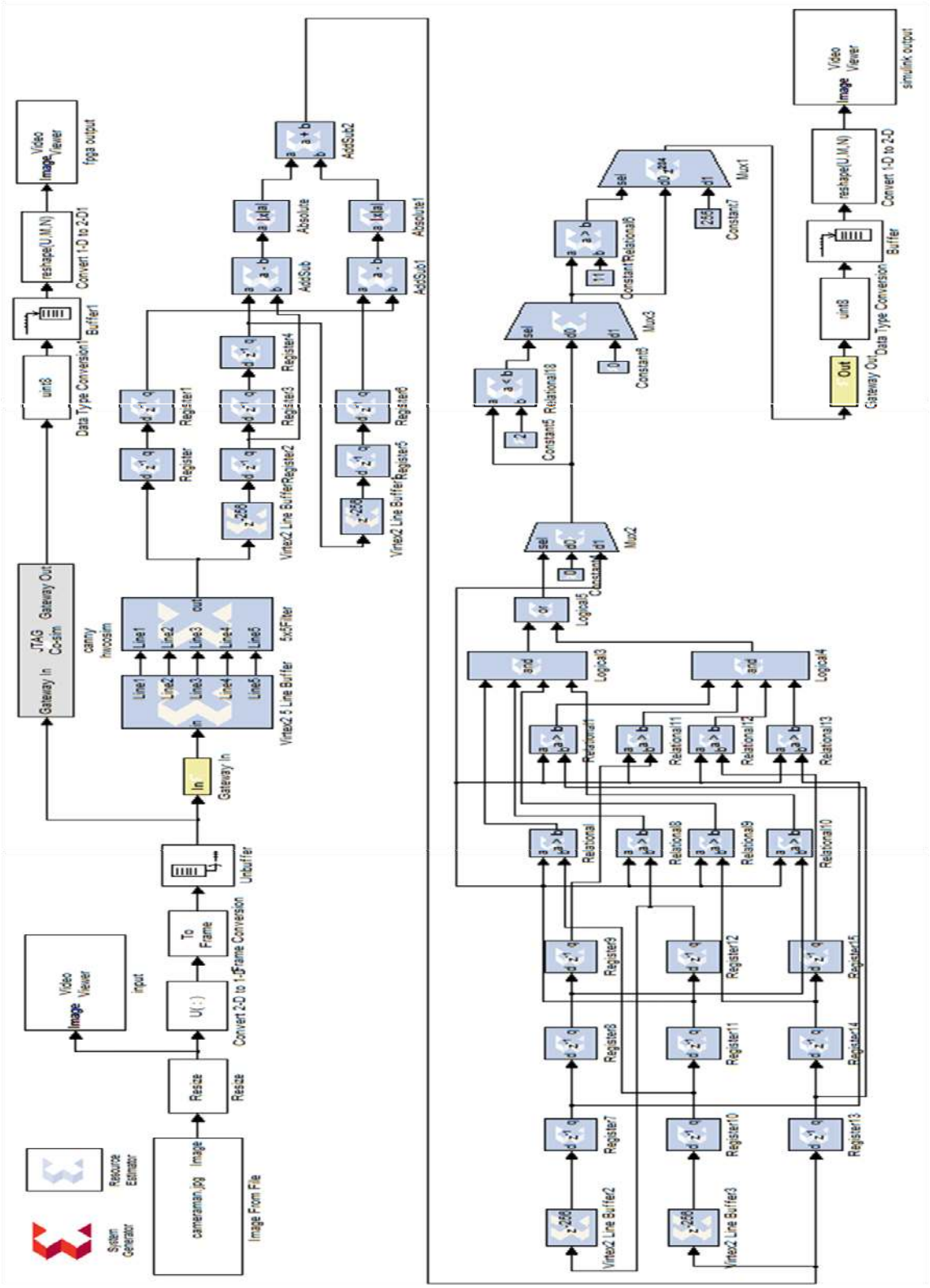
Hazırlanan model üzerinden çıkış görüntü verisinin tamamının elde edilebilmesi için model çalışma süresi Simulink üzerinden $T=W^2+2*W+30$ olarak hesaplanıp girilmiştir [32]. Bu eşitlikte T zamanı; W ise kare matris olan giriş görüntüsünün piksel genişliği sayısını belirtmektedir. Bu çalışmada kullanılan görüntüler 256*256 boyutunda kullanıldığından W=256 alınmıştır.




**Şekil 4.15** Sobel işleci için oluşturulan Simulink ve XSG DSP modeli

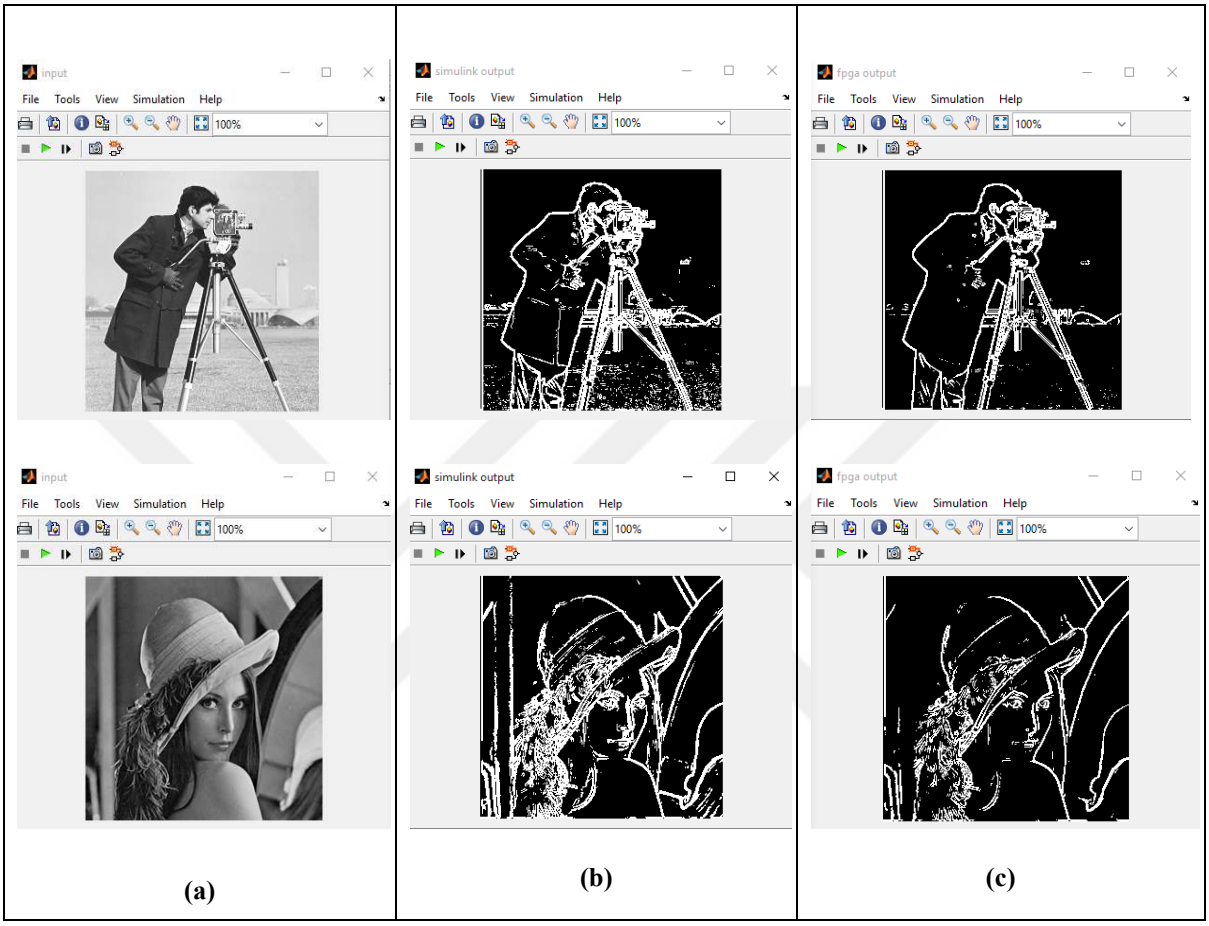



**Şekil 4.16** Prewitt işleci için oluşturulan Simulink ve XSG DSP modeli



**Şekil 4.17** Canny işleci için oluşturulan Simulink ve XSG DSP modeli

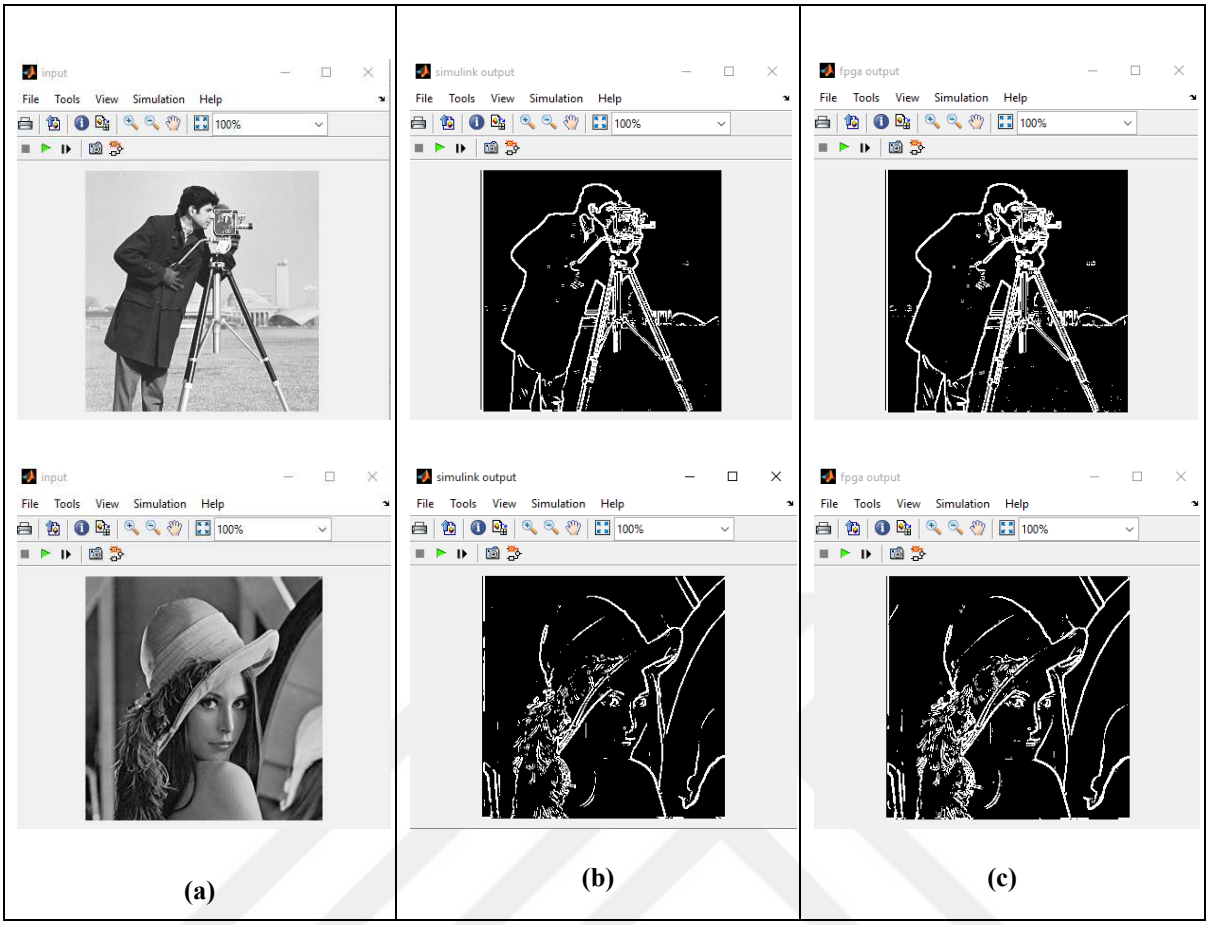
FPGA ile Sobel, Prewitt ve Canny kenar işleçlerinin gerçeklemesi için kullanılan gri formattaki giriş görüntüleri, Simulink ortamında elde edilen ve FPGA kartı kullanılarak elde edilen kenar algılaması yapılmış görüntüler Şekil 4.18 - 4.20'de verilmiştir.




**Şekil 4.18** Sobel işleci uygulanmış görüntüler (a) Gri formattaki orijinal görüntü (b) Simulink ortamında kenar algılama (c) FPGA kartında kenar algılama

**Tablo 4.2** Sobel işlecinin FPGA üzerindeki kaynak kullanımı

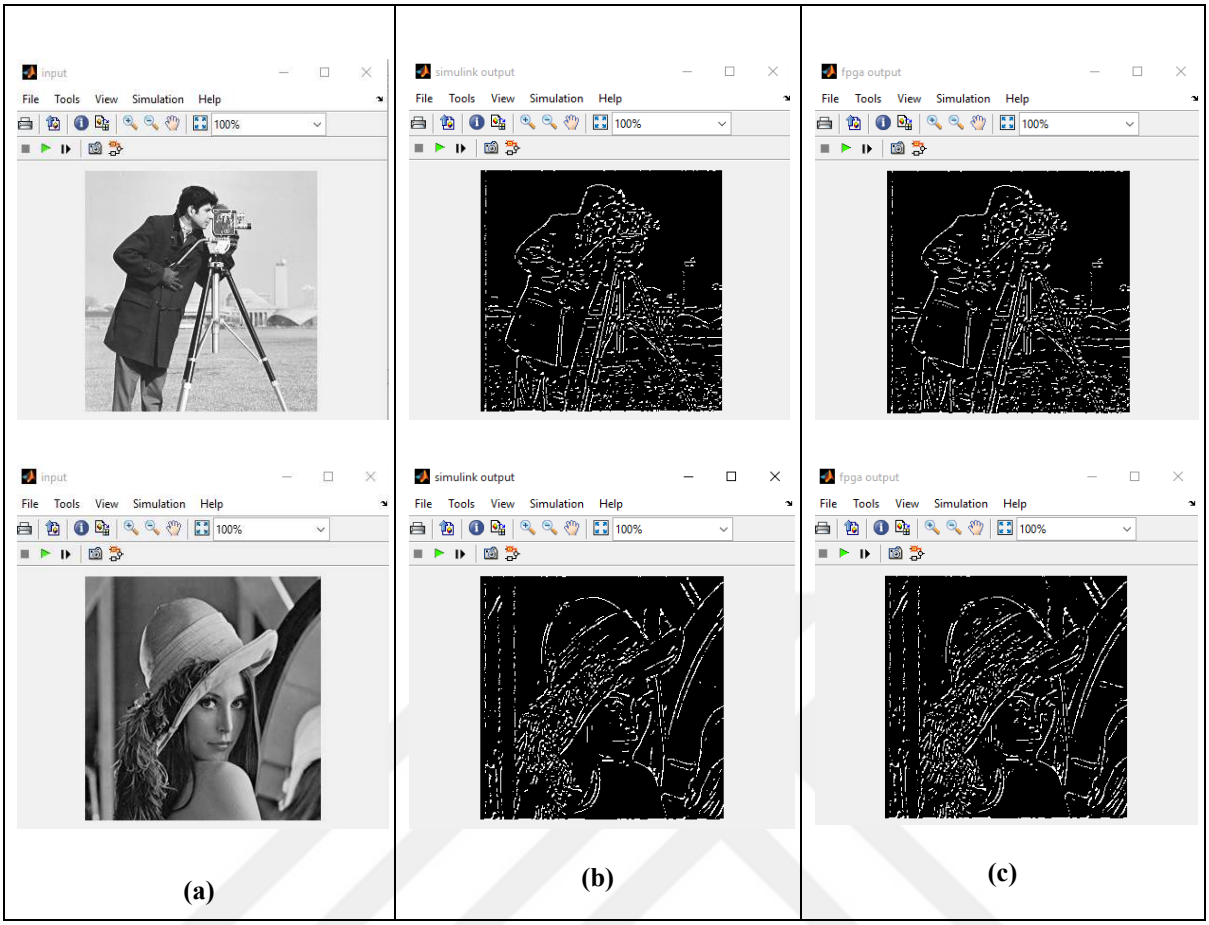
| **FPGA Kaynak Kullanımı** | | | |
|---|---|---|---|
| Mantık Devreleri | Kullanılan | Mevcut | Kullanım |
| Number of Slice Flip Flops | 1,404 | 29,504 | 4% |
| Number of 4 input LUTs | 2,201 | 29,504 | 7% |
| Number of occupied Slices | 1,401 | 14,752 | 9% |
| Total Number of 4 input LUTs | 2,369 | 29,504 | 8% |
| Number of bonded IOBs | 17 | 250 | 6% |
| Number of BUFGMUXs | 1 | 24 | 4% |
| Number of MULT18X18SIOs | 18 | 36 | 50% |




**Şekil 4.19** Prewitt işleci uygulanmış görüntüler (a) Gri formattaki orijinal görüntü (b) Simulink ortamında kenar algılama (c) FPGA kartında kenar algılama

**Tablo 4.3** Prewitt işlecinin FPGA üzerindeki kaynak kullanımı

| FPGA Kaynak Kullanımı | | | |
|---|---|---|---|
| Mantık Devreleri | Kullanılan | Mevcut | Kullanım |
| Number of Slice Flip Flops | 1,404 | 29,504 | 4% |
| Number of 4 input LUTs | 2,200 | 29,504 | 7% |
| Number of occupied Slices | 1,496 | 14,752 | 10% |
| Total Number of 4 input LUTs | 2,371 | 29,504 | 8% |
| Number of bonded IOBs | 17 | 250 | 6% |
| Number of BUFGMUXs | 1 | 24 | 4% |
| Number of MULT18X18SIOs | 18 | 36 | 50% |




**Şekil 4.20** Canny işleci uygulanmış görüntüler (a) Gri formattaki orijinal görüntü (b) Simulink ortamında kenar algılama (c) FPGA kartında kenar algılama

**Tablo 4.4** Canny işlecinin FPGA üzerindeki kaynak kullanımı

| **FPGA Kaynak Kullanımı** | | | |
|---|---|---|---|
| Mantık Devreleri | Kullanılan | Mevcut | Kullanım |
| Number of Slice Flip Flops | 916 | 29,504 | 3% |
| Number of 4 input LUTs | 603 | 29,504 | 2% |
| Number of occupied Slices | 654 | 14,752 | 14% |
| Total Number of 4 input LUTs | 688 | 29,504 | 2% |
| Number of bonded IOBs | 17 | 250 | 6% |
| Number of RAMB16s | 13 | 20 | 65% |
| Number of BUFGMUXs | 1 | 24 | 4% |
| Number of MULT18X18SIOs | 5 | 36 | 14% |

FPGA üzerinde gerçekleştirilen Sobel, Prewitt ve Canny işleçleri kullanılarak kenar algılama işleminin Spartan-3E 1600E FPGA kartı üzerinde kullanmış olduğu kaynak

tüketimine ilişkin veriler Tablo 4.2 - 4.4'te verilmiş olup gerçekleştirilen tasarımların FPGA üzerinde az bir kaynak tükettiğini söyleyebiliriz.

### 4.3. Elde Edilen Görüntülerin Performans İncelemesi

Simulink ortamında kenar algılama işlemi uygulanmış görüntüler ile FPGA üzerinde kenar algılaması yapılmış görüntülerin kendi aralarında ve algoritma tipine göre kenar çıkarım performansını değerlendirmek için hata oranları ve gürültü seviyeleri hakkında bilgi sahibi olmamız gerekiyor. Bu çalışmada bunun için gerekli olan bilgiler görüntüler için kullanılabilen istatistiksel karşılaştırma yöntemlerinden MSE (Mean Squared Error-Ortalama Kare Hata Seviyesi) ve PSNR (Peak Signal to Noise Ratio-Tepe Sinyalin Gürültü Sinyaline Oranı) yöntemleri kullanılarak incelenecektir.

MSE yönteminde orijinal giriş görüntüsü ile kenar algılama işlemi gerçekleştirilmiş çıkış görüntüsünün pikselleri arasındaki ortalama kare farkı ölçülür. Bu farkın büyük olması durumunda orijinal görüntü ile işlenmiş görüntü arasındaki farkın büyük olduğu anlaşılır [33]. Fakat eğer işlenmiş görüntü kenar algılaması yapılmış görüntü ise MSE değerinin büyük olması kenar yakalama performansının daha iyi olduğunu gösterir. Orijinal görüntü ve kenar algılaması yapılmış görüntü arasındaki MSE değeri 4.1'deki eşitlik ile hesaplanır.

$$MSE = \frac{\sum_{M,N}[I_1(m,n) - I_2(m,n)]^2}{M * N} \tag{4.1}$$

4.1'deki eşitlikte $I_1$ orijinal görüntüdeki piksel değerini, $I_2$ kenar algılaması yapılmış görüntüdeki piksel değerini, M ve N değerleri ise kullanılan görüntünün yatay ve düşey piksel boyutunu belirtmektedir.

PSNR değeri ise bir sinyalin maksimum taşıyabileceği sinyal gücünün bu sinyale etki eden bozucu sinyale oranı olarak değerlendirilmektedir. Bu değer genellikle desibel (dB) olarak ifade edilmektedir. Görüntüler için yüksek PSNR değeri genellikle sıkıştırılmış görüntülerde yüksek kaliteyi ifade etse de kenar algılaması yapılmış görüntülerde düşük PSNR değeri detay çıkarım konusunda daha iyi bir sonucu ifade etmektedir [34].

MSE değeri kullanılarak PSNR değeri 4.2'deki gibi hesaplanır. Eşitlik 4.2'de kullanılan R değeri giriş görüntüsü datasında kullanılan maksimum çeşitliliği ifade etmekte olup bu çalışmada kullanılan giriş görüntü verileri 8-bit işaretsiz tam sayı formatında olduğundan R=255 alınmıştır.

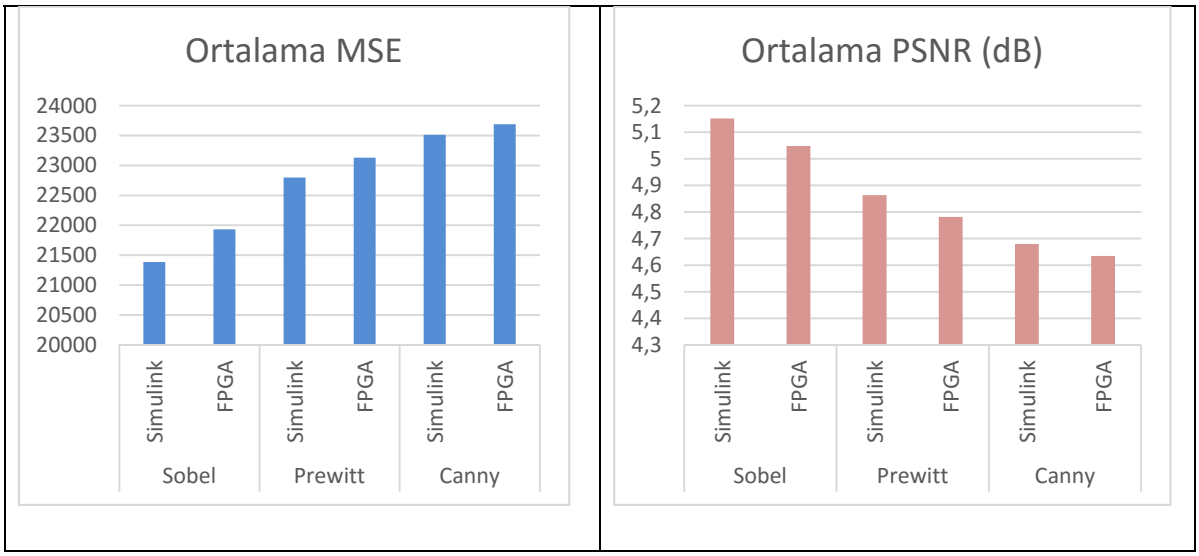
$$PSNR = 10\log_{10}\left(\frac{R^2}{MSE}\right) \tag{4.2}$$

Simulink ve FPGA kartı üzerinde kenar algılaması yapılan görüntüler için hesaplanan MSE ve PSNR değerleri Tablo 4.5'te verilmiştir.

**Tablo 4.5** Görüntüler için hesaplanan MSE ve PSNR değerleri

| | | **Cameraman** | | **Lena** | |
|---|---|---|---|---|---|
| | | MSE | PSNR (dB) | MSE | PSNR (dB) |
| **Sobel** | Simulink | 29327.8 | 3.458 | 13442.5 | 6.846 |
| | FPGA | 30149.5 | 3.338 | 13714.5 | 6.759 |
| **Prewitt** | Simulink | 31137.2 | 3.198 | 14460.4 | 6.529 |
| | FPGA | 31345.8 | 3.169 | 14916.9 | 6.394 |
| **Canny** | Simulink | 31447.1 | 3.155 | 15580.4 | 6.205 |
| | FPGA | 31497.8 | 3.148 | 15884.7 | 6.121 |

Cameraman ve Lena görüntüleri için hesaplanan ortalama MSE ve ortalama PSNR değerlerine ait grafiksel gösterim Şekil 4.21'de verilmiştir.




**Şekil 4.21** Hesaplanan ortalama MSE ve ortalama PSNR ait grafikler

Tablo 4.5'teki veriler incelendiğinde FPGA kartı üzerinde kenar algılaması yapılan görüntülerin MSE değerleri Simulink üzerinde kenar algılaması yapılan görüntülerin MSE değerlerinden daha büyük olduğu ve bunu destekleyici şekilde PSNR değerlerinin ise daha küçük olması nedeniyle FPGA üzerindeki kenar algılama işleminin Simulink'e göre daha iyi sonuç verdiği söylenebilir.

Şekil 4.21'deki "Cameraman" ve "Lena" görüntülerine ait ortalama MSE ve PSNR grafiklerde benzer şekilde incelenirse en yüksek MSE ve en düşük PSNR değerlerinin Canny algoritmasına ait olduğu, en düşük MSE ve en yüksek PSNR değerlerinin de Sobel algoritmasına ait olduğu görülmektedir. Bu nedenlerden dolayı en iyi kenar çıkarımının Canny algoritması ile gerçekleştiği söylenebilir.

# 5. SONUÇLAR

FPGA'lar alanda defalarca programlanabilmeleri ve paralel işlem yapabilme özelliğine sahip olmaları sayesinde, günümüzde birçok alanda (sinyal işleme uygulamaları, görüntü işleme gibi) yaygın olarak kullanılabilmektedir. Bu tez çalışmasında, ilk olarak görüntüde kenar algılama, kenar algılamada kullanılan yöntemler ve bu yöntemler içerisinde yer alan temel kenar algılama işleçleri hakkında temel bilgilere yer verilmiştir. FPGA'ların programlamasında kullanılan donanım programlama dilleri ve programlama tasarım araçlarının özellikleri üzerinde durulmuş olup bunların nasıl kullanılacağı hakkında anlatımlara yer verilmiştir. Son olarak, Sobel, Prewitt ve Canny kenar algılama algoritmalarının FPGA üzerinde gerçeklemesi için bir sistem oluşturulmuş ve bu sistem ile ilgili ayrıntılı bilgilere yer verilmiştir.

Kenar algılama işlemi için oluşturulan sistemde gri formattaki görüntüler bilgisayar ortamından alınıp FPGA kartı ve Matlab/Simulink üzerinden kenar algılama işlemi gerçekleştirilmiştir. Kullanılan görüntülerin her bir piksel değeri 8 bit sayısal veri olarak kenar algılama işlemine tabi tutulmuş olup oluşturulan sistemde FPGA ile bilgisayar arasındaki bağlantı FPGA üzerinde bulunan USB yapılandırma portu ile gerçekleştirilmiştir.

Simulink üzerinden ve FPGA kartı üzerinden aynı anda kenar algılama işlemi gerçekleştirilmiş olan görüntüler karşılaştırıldığında, her üç kenar algılama algoritması için elde edilen Simulink çıkış görüntüleri ile FPGA çıkış görüntülerin birbiriyle tutarlı olduğu gözlemlenmiştir. MSE ve PSNR değerleri göz önüne alındığında ise FPGA kartı ile kenar algılaması yapılmış görüntülerin kenar çıkarımı konusunda Simulink ile kenar algılaması yapılmış görüntülerden biraz daha iyi sonuç verdiği gözlemlenmiştir. FPGA ve Simulink ile gerçekleştirilen kenar algılama algoritmaları kenar çıkarımı yönünden incelendiğinde ise; MSE ve PSNR değerlerine göre Prewitt işlecinin Sobel işlecine göre kenar çıkarımında biraz daha iyi olduğu gözlemlenirken ayrıntılı kenar çıkarımı konusunda en iyi işlecin Canny işleci olduğu gözlemlenmiştir. FPGA üzerinde gerçekleştirilen tasarımların FPGA üzerindeki kaynak tüketimi ele alındığında; Sobel ve Prewit işleçleri için oluşturulan donanımsal tasarımlarda tek farkın 3x3'lük filtre maskelerindeki katsayı farklılığı olmasından ötürü aynı kaynak tüketimine sahip olduğu ve FPGA kaynaklarının her iki işleç için de ortalama %10 civarında kullanıldığı görülmüştür. Canny işlecinde ise

kullanılan mantıksal blok sayısının diğer iki işlece göre yüksek olması ve “Register” hafıza bloklarının fazla kullanılması sonucu FPGA kaynaklarının ortalama %14 civarında kullanıldığı görülmüştür. Bunun sonucunda daha karmaşık sistemlerin de FPGA üzerinde kaynak sıkıntısı olmadan gerçekleştirilebileceği sonucuna varılabilir.

Kenar algılama işleminin FPGA üzerinde gerçekleştirilmesinin avantajları; alanda programlanabilir olması, esnek yapısı, paralel işlem kabiliyeti ve tasarım için ekonomik oluşu olarak belirtilebilir.

Bu tez çalışmasında Sobel, Prewitt ve Canny kenar algılama algoritmalarının FPGA üzerinde gerçekleştirilmesinin başarılı sonuç verdiği gözlemlenmiştir.

## KAYNAKLAR


[1] **Marr, D., Hildreth E., 1980"** Theory of edge detection", Proc. R. Soc. Lond. A, Math. Phys. Sci., B 207, syf. 187–217.

[2] **Haralick, R.M., 1984,"** Digital step edges from zero crossing of second directional derivatives", IEEE Trans. Pattern Anal. Mach. Intell syf. 58–68.

[3] **Canny, J F., 1986,** "A Computational Approach to Edge Detection", IEEE Transactions on Pattern Analysis and Machine Intelligence, syf. 679-698.

[4] **Bovik A., 2010,** "Handbook of Image anf Video Processing",Academic Press, 1384 syf.

[5] **Christe, S.A.,2011,** Vignesh, M., Kandaswamy, A., "*An efficient FPGA implementation of MRI image filtering tumour characterization using Xilinx system generator*", International Journal of VLSI Desing & Communication Systems, 2 (4): 95-109.

[6] **Vega-Rodríguez., M.A., Sánchez-Pérez, J.M., Gómez-Pulido, J.A., 2002** "An FPGAbased implementation for median filter meeting the real-time requirements of automated visual inspection systems", *10th Mediterranean Conference on Control and Automation*, Lisbon-Portugal,.

[7] **Nelson, A.E., 2000** "Implementation of image processing algorithms on FPGA hardware", Master of Science Thesis, *Faculty of the Graduate School of Vanderbilt University,* Nashville, TN-USA.

[8] **Nana, L., Weixing, Z., Shiying, M., Wen, H., 2011** "Super resolution video reconstruction in DSP+FPGA based on lifting wavelet", *The Tenth International Conference on Electronic Measurement & Instruments*, China, 101-104 (2011).

[9] **Kuon, I., Tessier, R., ve Rose, J.,** (2007), "*FPGA Architecture*: *Survey and Challenges*", Foundations and Trends in Electronic Design Automation: Vol. 2, No 2, S.135-253.

[10] **Landi, C., Laracca, M., Ferrigno L., (2008),** "FPGA-based Measurement Instrument for Power Quality Monitoring according to IEC Standards", International Instrumentation and Measurement Technology Conference 2008. Victoria, Canada. 12-15 maggio 2008. (pp. 906-911).

[11] **Choong, F., Reaz, M. B. I.,** (2005), "Implementation of Power Quality Disturbance Classifier in FPGA Employing Wavelet Transform,ANN and Fuzzy Logic", SETIT, 27-31 Mart, Tunus.

[12] **Torres-Huitzil, C., Arias-Estrada, M.,** (2004), "Real-time image processing with a compact FPGA-based systolic architecture ", Real-Time Imaging, Vol:10-3, S:177:187.

[13] **Omondi, A. R., Rajapakse, J. C., (2006**), "FPGA Implementations of Neural Networks" , Springer, ISBN: 0387284850.

[14] **Yılmaz, N., (2008),** "Alan Programlamalı Kapı Dizileri (FPGA) Üzerinde Bir YSA'nın Tasarlanması ve Donanım Olarak Gerçekleştirilmesi" Yüksek Lisans Tezi, Fen Bilimleri Enstitüsü, Konya.

[15] **Çavuşlu, M. A, Karakuzu, C., Şahin, S., (2006)**, "Neural Network Hardware Implementation Using FPGA", ISEECE 3rd International Symposium on Electrical, Electronic and Computer Engineering Symposium Proceedings, Nicosia, TRNC, S. 287-290.

[16] **Mahmoud, M. I., Dessouky, M. I. M., Deyab, S. and Elfouly, F. H., (2007),** "Comparison between haar and daubechies wavelet transformions on FPGA technology, " Proceedings Of World Academy Of Science, Engineering and Technology, Vol. 20, ISSN 1307-6884, pp. 68–72.

[17] **Pavlidis T., (1992),** *Why Progress in Machine Visian is so Low,* Pattern Recognition Letters 13, 221-225.

[18] **Kitchen, L. J. ve Rosenfeld, A., (1981)** *Edge Evaluation Using Local Edge Coherence,* IEEE Trans. System Man and Cybemetic.

[19] **Loh, A. W. K., Robey, M. C. ve West, G. A. W., (2001),** *Analysis of the Interaction Between Edge and Line Finding Techniques,* The Journal of the Pattern Recognition Society, 34, 1127-1146.

[20] **Davies, E. R. , (1997)** *Machine Vision,* Academic Press.

[21] **Aarnink, R. G., Rosette, J. M. C. H., Feitz, Wouter F. J., Debruyne, F. M. ve Wijkstra, H., (1997)** *A Preprocessing Algorithm for Edge Detection with Multiple Scales of Resolution,* Europan Journal of Ultrasound, 5, 113-126.

[22] **Ziou, D. ve Tabbone, S., (1997)** *Edge Detection Techniques - An Overview,* Technical Report, No. 195, Dept. Math & Informatique, Universit de Sherbrooke.

[23] **Gonzalez, R. ve Wintz, P.,(1987)** *Digital Image Processing,* Addison-Wesley.

[24] **Roberts, L. G., (1965)** *Machine Perception of Three-Dimensional Solids,* in optical and Electro-Optical Information Processing ( J. Tippett, ed.), 159-197, MIT Pres.

[25] **Sobel, I., (1990)** *An Isotropic 3x3 Gradient Operator,* Machine Vision for ThreeDimensional Scenes, Freeman, H., Academic Pres, NY, 376-379.

[26] **Prewitt, J., (1970)** *Object Enhancemet And Extraction,* Picture Processing and Psychopictorics (B. Lipkin ve A. Rosenfeld, editör), NY, Academic Pres.

[27] **Mccane, B., (2001)** *Edge Detection,* Cosc 453: Computer Vision Notes, Department of Computer Science, University of Otoga.

[28] **Maeyer, U., (2001),** "Digital Signal Processing with Field Programmable Gate Arrays", Springer, Germany.

[29] **Pedroni, V., (2004)** "Circuit Design with FPGA", MIT Press, Massachusetts.

[30] **Xilinx, (2007) "**Spartan-3E 1600E Edition User Guide", downloadable from;http:// www. Xilinx.com

[31] **Xilinx, (2010)** "System Generator User's Guide", downloadable from;http:// www. Xilinx.com

[32] **Ganley, T., (2010),** "FPGA Co-Simulation of Gaussian Filter Algorithm", 8.

[33] **P. Vidya, S. Veni and K.A. Narayanankutty, (2009)** "*Performance Analysis of Edge Detection Methods on Hexagonal Sampling Grid*", International Journal of Electronic Engineering Research, Volume 1, pp. 313–328.

[34] **Peter Kellman and Elliot R. McVeigh, (2005)** "*Image Reconstruction in SNR Units: A General Method for SNR Measurement*", Magn Reson Med. Author manuscript, Magn Reson Med. Volume 54(6) pp. 1439–1447.

# ÖZGEÇMİŞ

Yaser İÇER, 02.12.1987 yılında Diyarbakır/Bağlar ’da doğdu. İlk ve ortaöğrenimini Diyarbakır’da tamamladı. Gaziantep Üniversitesi Mühendislik Fakültesi Elektrik-Elektronik Mühendisliği bölümünden 2010 yılında mezun oldu. 2014 yılında Fırat Üniversitesi Elektrik-Elektronik Mühendisliği Elektronik Anabilim Dalında yüksek lisans öğrenimine başladı. Şubat 2015’ten beri Dicle Üniversitesi Diyarbakır Teknik Bilimler Meslek Yüksekokulu’nda Elektronik ve Otomasyon bölümünde Öğretim Görevlisi olarak çalışmaktadır.

**İletişim Bilgileri**:

Dicle Üniversitesi Diyarbakır Teknik Bilimler MYO
Elektronik ve Otomasyon Bölümü / DİYARBAKIR

E-mail: yaser.icer@gmail.com