İZMİR BAKIRÇAY ÜNİVERSİTESİ

LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ
ELEKTRİK ELEKTRONİK MÜHENDİSLİĞİ A.B.D.

FPGA ile Şerit Algılama Uygulaması

EE526 İLERİ SAYISAL TASARIM

Yaşar ŞENTÜRK 6017023

Prof.Dr.Mehmet Sağbaş

Mayıs 2023

FPGA ile Şerit Algılama Uygulaması

EE526 İLERİ SAYISAL TASARIM

Yaşar ŞENTÜRK İzmir 2023

6017023

FPGA ile Şerit Algılama Uygulaması

Yaşar ŞENTÜRK
6017023

EE526 İLERİ SAYISAL TASARIM

Elektrik Elektronik Mühendisliği Anabilim Dalı
Prof.Dr. Mehmet Sağbaş

İzmir
İzmir Bakırçay Üniversitesi
Lisansüstü Eğitim Enstitüsü
Mayıs 2023

ÖZET

FPGA ile Şerit Algılama Uygulaması

Yaşar ŞENTÜRK

Elektrik Elektonik Mühendisliği Anabilim Dalı

İzmir Bakırçay Üniversitesi, Lisansüstü Eğitim Enstitüsüi Mayıs 2023

Prof.Dr.Mehmet Sağbaş

Çalışmada otonom sürüş için temel şerit algılama uygulamalarından biri olan Sobel filtre algoritmasının uygulaması ele alınmaktadır. Sobel filtresi, kenar tespiti amacıyla yaygın olarak kullanılmakta olan bir algoritmadır. Görüntü işlemede gradyan hesaplaması yapmaktadır. Sobel filtresinin FPGA uygulaması yapılmış ve performansı değerlendirilmiştir. Çalışmada FPGA üzerinde görüntü işleme amacıyla bir tasarım yapılmış ve Intel Quartus Prime Lite oramında VHDL kodlarıyla derlenmiştir. Tasarım uzak erişim FPGA laboratuvarında gerçek donanım kullanılarak test edilmiştir. Sobel filtresi algoritmasının gerçekleştirilmesiyle Hough dönüşümü ve şerit algılama algoritmalarının temelinin oluşturulması amaçlanmaktadır.Sonuç olarak çalışmada Sobel filtresi kullanarak FPGA ile şerit algılma uygulaması yapılmış, sobel filtresi ile şeritlerin kenarları belirlenmiş ve devrenin tasarımı blok diyagramı ile gösterilip sonuçlar değerlendirilmiştir.

Anahtar Sözcükler: FPGA; Sobel Filtresi; Şerit Algılama; VHDL.

İÇİNDEKİLER

TABLOLAR DİZİNİ

ŞEKİLLER DİZİNİ

SİMGELER VE KISALTMALAR DİZİNİ

FPGA : Field Programmable Gate Array
VHDL : Very High Speed Integrated Circuit Hardware Description Language
HDMI : High Definition Multimedia Interface
RGB : Red-Green-Blue
mA : Mili-Amper

1. GİRİŞ

FPGA' lar görüntü işleme, otonom sürüş gibi alanlarda önemli bir bileşen olarak karşımıza çıkmaktadır. ve bu alanlarda bizlere güçlü çözümler sunmaktadır. Bu çalışmada FPGA ile görüntü işleme konusuna odaklanılmaktadır. Otonom sürüş için şerit algılanması uygulamasının algoritması ve uygulama süreci incelenmektedir.

Şerit tespiti için temel olarak Sobel filtresi ve Hough dönüşümü kullanılmaktadır. Sobel filtresi kenar tespiti için yaygıl olarak kullanılmakta olan bir algoritmadır ve görüntü üzerinde gradient hesaplamasını gerçekleştirmektedir. Hough dönüşümü ise doğru segmentleri tespit etmek amacıyla kullanılmakta olan bir tekniktir. Bu algoritmaların FPGA uygulamasında kullanımına değilinmektedir. Bu çalışmada Sobel filtresi değerlendirilecektir ve şeritlerin kenarları tespit edilecektir.

FPGA kullanımında güç tüketimine dair sorunlarla karşılaşılabilmektedir. FPGA' ler yüksek işlem gücüne rağmen düşük enerji verimliliğine sahip olabilmektedir. Bu nedenle güç tüketimi optimizasyonu gerçekleştirilmesi gerekebilmektedir.

Çalışmada, devre tasarımı gerçekleştirilmiş ve blok diyagramı ile tasarım görselleştirilmiştir. Intel Quartus Prime Lite ortamında VHDL kodları yazılarak derlenmiştir. Ardından uzaktan FPGA laboratuvarı üzerinden sonuçlar değerlendirilmiştir.

Çalışmada, FPGA kullanarak görüntü işleme üzerine çalışılmıştır. Sobel filtresi, Hought dönüşümü ve güç tüketimi konuları ele alınmış ve değerlendirilmiştir.

1.1.Amaç ve Yöntem

Bu çalışmada Cyclone V (5CEBA2F17C6) FPGA, Intel Quartus Prime Lite ortamında oluşturulan .sof dosyası ile "Hochschule Bonn-Rhein-Sieg University of Applied Sciences [1]" 'a ait olan uzak erişim FPGA laboratuvarında programlanmıştır. Uzak laboratuvardaki gerçek donanım kullanılarak kenar algılama algoritması olan Sobel algoritmasının kullanılması ve FPGA değerlerinin yorumlanması amaçlanmaktadır. Sobel algoritmasının gerçekleştirilmesiyle, otonom sürüş için şerit algılama algoritmalarının gerçekleştirilmesi ve uygulanması için temel oluşturulması amaçlanmaktadır.

1.2.Literatür Araştırması

Literatür araştırması esnasında şerit algınlanması ve takibi hakkında yapılan çalışmalan incelenmiştir. Literatürde yapılan bazı çalışmalar aşağıda özetlenmiştir.

Solmaz, Özçevik ve Baysal, şerit tanıma ve takip algoritmaları için otonom sürüş modeli tanıtmaktadır. Çalışmada şerit tanıma ve takip için bir dizi görüntü işleme adımı açıklanmıştır ve gerçek zamanlı simülasyon yapılmıştır. Elde edilen bulgulara göre Canny kenar tespit algortimasının bir şerit tanıma ve takip sistemi için uygun olabileceği görülmüştür. Sobel yöntemi de çalışmada değerlendirilmiştir [2].

Koyuncu, Taşdemir, Alçın, Tuna ve Coşgun, koşe alıglama algoritmaları, sobel algoritması, morfolojik işlem algoritmalarını FPGA üzerinde çalışacak şekilde tasarlanmıştır. Vivado Design Suite, Vivado High Level Synthesis ve Vivado SDK kullanılarak FPGA tabanlı gerçek zamanlı görüntü işleme algoritmaları geliştirilmiştir. Görüntüler HDMI aracılığıyla kameradan alınmış ve FPGA üzerinde işlenebilmesi için VHDL kullanılmıştr. Tasarımda Xilinx Zybo Z7-20 geliştirme kartı kullanılmıştır. FPGA üzerinde elde edilen sonuçlar MATLAB programında elde edilen sonuçlar ile karşılaştırılmıştır [3].

2. ŞERİT ALGILAMANIN TEMELLERİ

Yollarda bulununan şeritlerin algılanmasının temel algortiması olarak karşımıza Sobel Filtresi çıkmaktadır. Aşağıdaki Sobel Filtresi ile ilgili sayısal ifadeler verilmiştir.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{2.1}$$

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{2.2}$$

$$G = \sqrt{G_x^2 + G_y^2} \tag{2.3}$$

Bu sayısal ifadelerde yatayda ve dikeyde işleme gerçekleştirilmesi amacıyla matrisler bulunmaktadır. Denklem 2.1' de yatayda, denklem 2.2' de ise dikeyde işleme amacıyla matrisler gösterilmektedir. Her iki matriste 3x3' lük görüntü bölgesinde çalışmaktadır. Daha sonra bu G_x ve G_y kareleri alarak toplanır ve bu toplam değerinin karekökü alınır. Bu işlemin sonucunda elde edilen G değeri (denklem 2.3) ile görüntüdeki kenarın gücü bulunmaktadır.

Bahsedilen bu sobel operatörleri, görüntü üzerinde her piksel üzerine maske uygulamaktadır. Uygulanan maskeler, etrafındaki piksellerin yoğunluk değerlerini belirli bir ağırlıklandırma düzenine göre hesaplayarak gradyan değerini elde etmektedir. Böylece her bir piksel için gradyan değeri elde edilmektedir.

Görüntüdeki farklı kenarları birleştirerek yoldaki şeritlerin algılanması gerekmektedir. Bu işlem Hough dönüşümü yöntemi ile yapılmaktadır. Hough dönüşümü, farklı kenarların bulunduğu görüntü bölgesi ile kenarların konumları ve açılarına bağlı olarak sıralandığı veya temsil edildiği bir Hough uzayı arasındaki dönüşüm olarak tanımlanmaktadır.

Sobel filtresi, kenar tespiti ve görüntü segmentasyonu gibi birçok görüntü işleme uygulamasında yaygın biçimde kullanılabilmektedir. Kenarları vurgulayarak, nesnelerin sınırlarını ve yapılarını belirlemeye yardımcı olmaktadır.

Sobel Filtresi, FPGA üzerinde VHDL ile uygulandığında işlediği görüntüde bulunan şeritlerin kenarlarını çok anlaşılabilir bir hale getirmektedir. Görüntü, tek kanallı bir görüntüye dönüştürülmekte ve bu görüntü üzerinde sobel filtresi uygulandığında şeritlerin sınırları belirlenmektedir. Bu durumdan sonra şerit algılama ve takibi işlemleri için farklı algoritmalar kullanılabilmektedir. Kısaca Sobel Filteresi şeritlerin sınırlarını belirlemektedir.

3. DÜŞÜK GÜÇ TASARIMI

FPGA' lar yüksek performans ve esneklik sağlayan donanımlar olarak değerlendirilmektedir ancak yüksek güç tüketimi ile karşılaşılabilmektedir. Bu durum maliyet, cihaza güç sağlama ve soğutma gibi durumlarda zorluk olarak karşımıza çıkmaktadır. Optimize edilmiş sistemlerde enerji verimliliği, yüksek performans, güvenilirlik sağlanmaktadır. Bu nedenle düşük güç tasarımı FPGA tabanlı sistemler için büyük önem taşıyabilmektedir.

Düşük güç tasarımını gerçekleştirmek için üreticiler tarafından özelleştirilmiş düşük güç modelleri, başarlı bir tasarım metodoloji, verimli bir FPGA kodlaması yapılması gerekmektedir.

FPGA ile şerit algılama uygulamasında Sobel filtresi ve RGB-Y artimetik işlemleri gerçekleştirilirken optimizasyon gerçekleştirilmek istendiğinde işlenecek görüntünün gereksiz kısmı işlem dışı bırakılarak optimizasyon sağlanabilmektedir. Ayrıca RGB-Y kodlamasında aritmetiği integer olarak gerçekleştirerek sonucu 16' ya bölündüğünde elde edilen sonuç ile 4 bitlik sağa kaydırma elde edilmektedir ve bu şekilde daha az bit kullanarak güç optimizasyonu elde edilmektedir.

Optimizasyon yöntemleri ile FPGA' nın işlem gücü ve kaynakları daha verimli bir şekilde kullanılabilmektedir. Fazla güç tüketen işlemler veya verilerin daha az bit kullanarak temsil edilmesi ile güç tüketimi azaltılır ve sistem performansı optimize edilmektedir. Böylece FPGA tabanlı uygulamalarda düşük güç tüketimiyle birlikte performans elde edilebilmektedir.

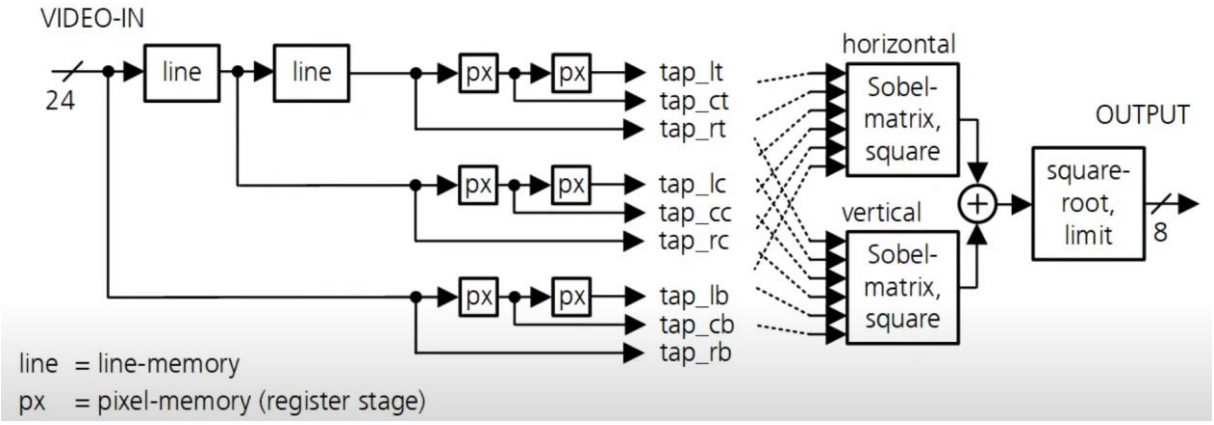
4. TASARIM

4.1. Konsept

Yolda bulunan şeritleri algılamak amacıyla işlenecek görüntüdeki 3x3 alanda bir bölge alınmaktadır. Bu 3x3 alan bir matris olarak düşünülmektedir ve Sobel filtresi için kullanılmaktadır. Burada X ve Y koordinatları için ayrı filtre matrisleri bulunmaktadır. Bu ayrı filtre matrislerinin sonuçlarının karelerinin toplamı alınır ve karekök değeri elde edilirse kenarların göstergesi olan G değeri elde edilmektedir. Yüksek elde edilen G değeri kenar olduğunun göstergesi, düşük G değeri ise kenar olmadığının göstergesi olarak tanımlanmaktadır. Elde edilen G değerleri 2' ye bölünüp 255' ten çıkarıldığında tespit edilen kenarlar siyah olarak gösterilmektedir ve geri kalan kısımlar beyaz olarak gösterilmektedir. Bu şekilde şeritlerin kenarları Sobel filtresi ile tespit edilebilmektedir.

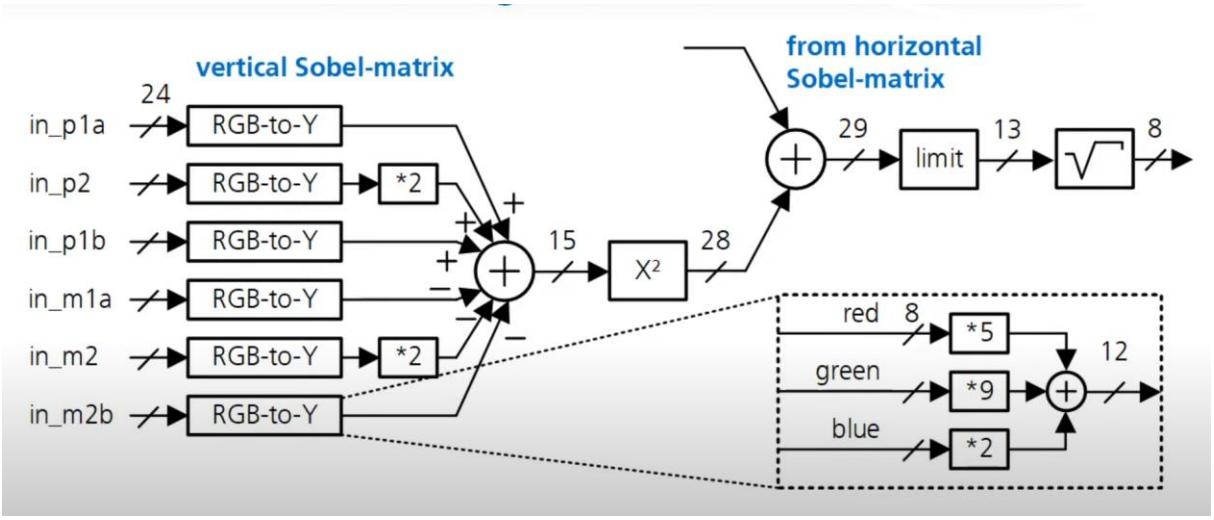
4.2. Mimari

Şerit algılamasının mimarisini şu şekilde açıklayabiliriz. Tasarımın ilk aşamasında görüntü hatlarını depolayabilmek amacıyla hat hafızaları oluşturulmaktadır ve bu şekilde görüntünün üst, orta ve alt pikselleri elde edilmektedir. Ardından sol, orta ve sağ pikselleri elde edebilmek amacıyla piksel hafızaları oluşturmaktadır. Piksel hafızalarının sahip olunan üç sıra için üç kere eklenmesi gerekmektedir. Bu şekilde 3x3' lük matris elde edilir ve 9 piksel oluşturulmaktadır. Bu matris üzerinde filtreleme amacıyla aritmetik işlem yapılmaktadır. Yatay filtreleme amacıyla bir adet aritmetik birim bulunmaktadır. Ayrıca dikey filtreleme amacıyla da başka bir aritmetik birim bulunmaktadır. Her iki birim de altı girişe sahiptir ve 3x3'lük matristeki ilgili piksellerle birlikte birbirine bağlıdır. Bu iki aritmetik birimdeki işlemlerin sonuçları birbirlerine eklenir ve karekökü alınarak çıkış değeri elde edilmektedir. Aşağıdaki şekilde şerit algılama mimarisinin blok diyagramı verilmiştir.



Şekil 4.1. Şerit Algılama Mimarisi Blok Diyagramı

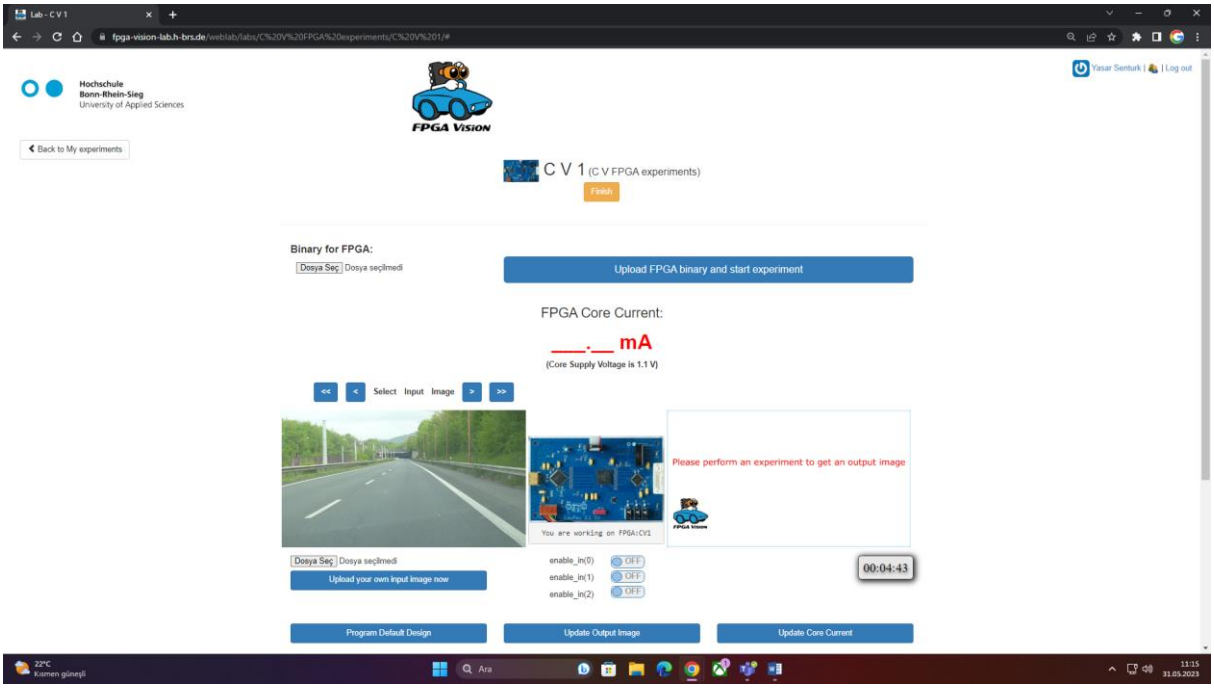
Aritmetik birim, filtre matrisinin altı değeri için altı girişe sahip olmaktadır. Üç adet pozitif ve üç adet negatif giriş bulunmaktadır. Bu altı adet giriş ilk olarak RGB piksellerinde Y parlaklık değerine çevrilmektedir. İki ile çarpamk için iki adet durum bulunmaktadır. Ardından bu altı değer için bir toplayıcı eklenmektedir. Toplayıcıya üç adet pozitif işaretli üç adet de negatif işaretli değer gelip toplanmaktadır. Elde edilen değer filtre matrisinin sonucudur ve bu değerin karesi alınmaktadır. Diğer filtre matrisinden başka bir toplayıcı eklenir. Bu diğer filtre matrsinin yapısı da aynıdır. Bu iki filtre matrisinin çıkışlarının toplamının karekökü alınır ve çıkışın sınırını belirler. Limit pikseli elde edilen değerin 2' ye bölünüp 255' ten çıkarılmasıyla elde edilmektedir. Aşağıdaki şekilde Sobel filtresi aritmetik biriminin blok diyagramı verilmiştir [4].



Şekil 4.2. Sobel Filtresi Aritmetik Birimi Blok Diyagramı

5. DENEY

Gerçekleştirilen deneyde Cyclone V (5CEBA2F17C6) FPGA, Intel Quartus Prime Lite ortamında oluşturulan .sof dosyası ile “Hochschule Bonn-Rhein-Sieg University of Applied Sciences” ‘a ait olan uzak erişim FPGA laboratuvarında programlanmıştır. Uzak laboratuvardaki gerçek donanım kullanılarak kenar algılama algoritması olan Sobel algoritmasının kullanılması ve FPGA değerlerinin yorumlanması amaçlanmaktadır. Aşağıdaki şekilde uzak erişim FPGA laboratuvarının arayüzü görülmektedir.

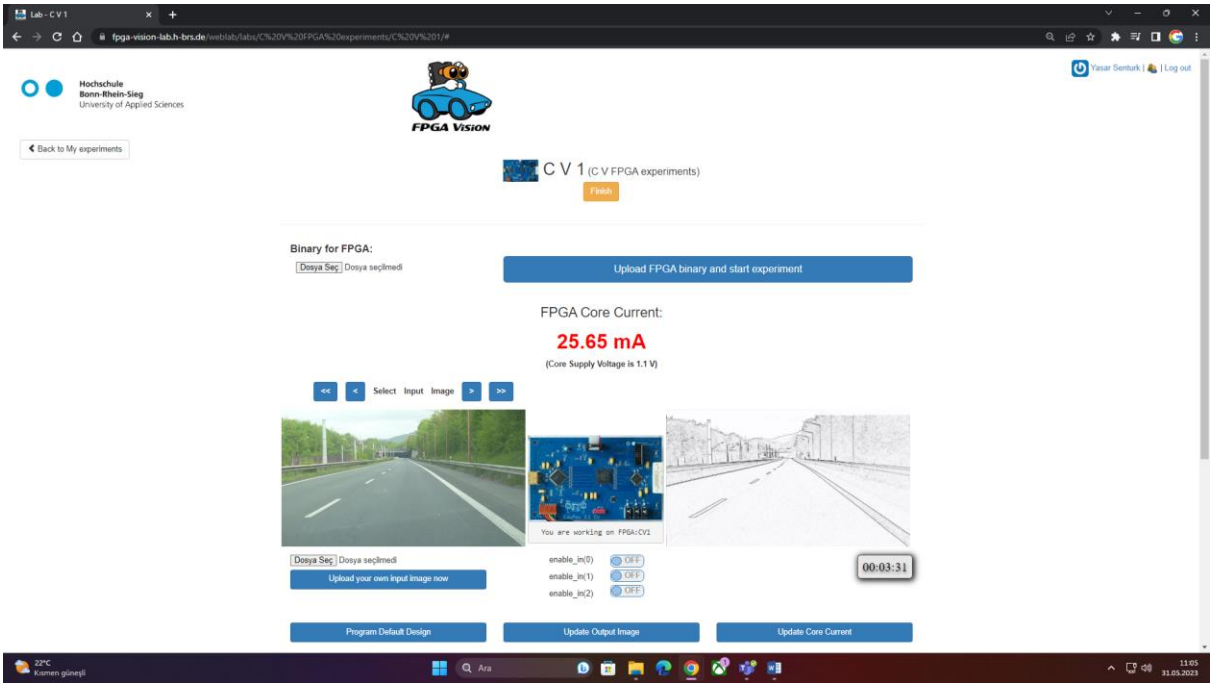


Şekil 3.1. Uzak Erişim FPGA Laboratuvarı Arayüzü

Bu laboratuvar arayüzünde derlenen .sof binary dosyası yüklenip uzaktaki gerçek donanıma aktarılmaktadır. Böylece işlenmek istenen yol görselinin şeritleri algılanmaktadır ve FPGA’ nın sayısal metrikleri gözlemlenebilmektedir.

5.1. Şerit Algılama Uygulaması ve Sonuçları

İlk yapılan uygulama optimize edilmeden gerçekleştirilen şerit algılama uygulamasıdır. Bu uygulamanın ilgili görseli aşağıdaki gibidir.



Şekil 4.2. Şerit Algılama Uygulaması ve Sonuçları

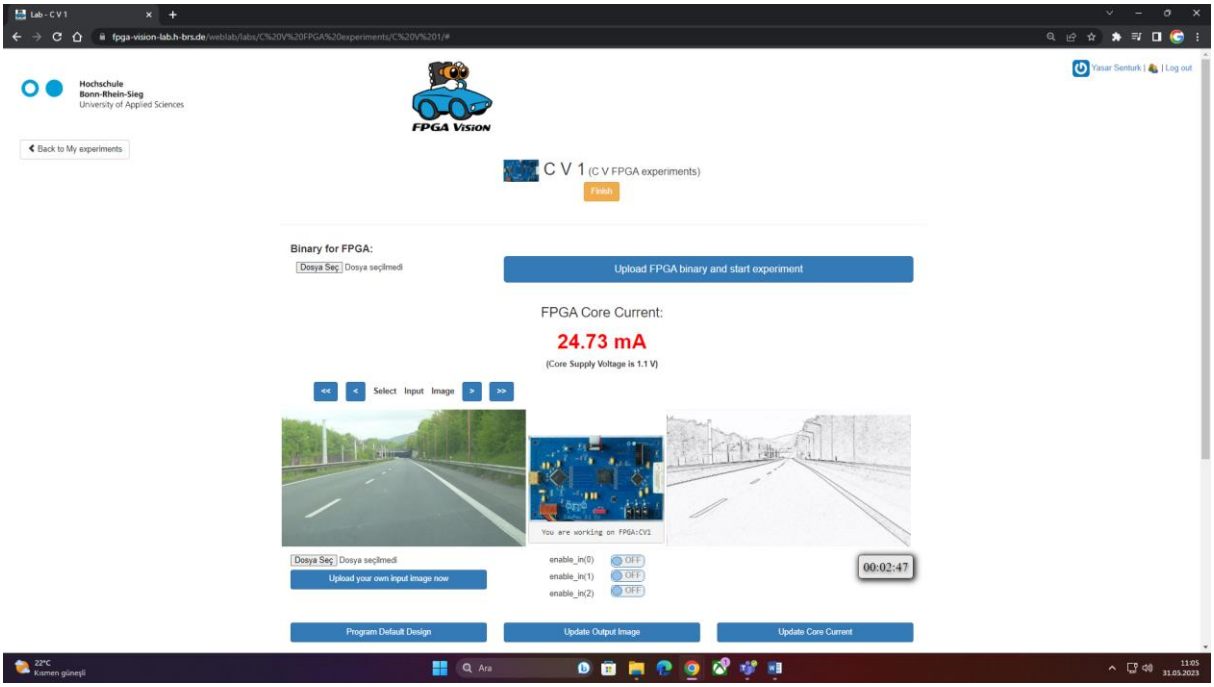
Yukarıdaki uygulamada sol taraftaki görselde işlenmesi istenen orijinal görüntü yer almaktayken, sağ tarafta ise FPGA tarafından işlenmiş görüntü görülmektedir. FPGA, donanımsal bir platform olarak şerit algılama işlemini gerçekleştirmek amacıyla özel olarak programlanmıştır. Sağ taraftaki FPGA tarafından işlenen görüntüde şeritlerin sınırları oldukça net bir şekilde belirlenebilmektedir.

FPGA' nın besleme gerilimi 1.1 Volt olarak ölçülmüştür. Ayrıca çekirdek akımı da 25.65 mA olarak ölçülmüştür. Bu bilgiler FPGA' nın güç tüketimi hakkında bilgi sağlamaktadır. Bir sonraki uygulamada güç optimizasyonu gerçekleştirilecektir.

Sonuç olarak sağ taraftarki FPGA tarafından işleen görüntüde şeritlerin sınırlarının iyi bir şekilde belirlenip algılandığı görülmektedir. FPGA'nın uygulamada kullanılan VHDL kodları verilmiştir (EK-1).

5.2. Optimize Edilmiş Şerit Algılama Uygulaması ve Sonuçları

Yapılan uygulama düşük güç optimizasyonu yapılarak gerçekleştirilen şerit algılama uygulamasıdır. Bu uygulamanın ilgili görseli aşağıdaki gibidir.



Şekil 5.3. Güç Optimizasyonu Gerçekleştirilmiş Şerit Algılama Uygulaması ve Sonuçları

Yukarıdaki uygulamada sol taraftaki görselde işlenmesi istenen orijinal görüntü yer almaktayken, sağ tarafta ise FPGA tarafından işlenmiş görüntü görülmektedir. FPGA, donanımsal bir platform olarak şerit algılama işlemini gerçekleştirmek amacıyla özel olarak programlanmıştır. Sağ taraftaki FPGA tarafından işlenen görüntüde şeritlerin sınırları oldukça net bir şekilde belirlenebilmektedir

FPGA ile şerit algılama uygulamasında Sobel filtresi ve RGB-Y artimetik işlemleri gerçekleştirilirken düşük güç optimizasyonu gerçekleştirilmek istendiğinde RGB-Y kodlamasında aritmetiği integer olarak belirterek sonucu 16’ ya bölündüğünde elde edilen sonuç ile 4 bitlik sağa kaydırma elde edilmektedir ve bu şekilde daha az bit kullanarak güç optimizasyonu elde edilmektedir. FPGA’ nın VHDL kodları ilk uygulama ile aynıdır ancak ilgili optimizasyonun yapıldığı kod bloku ektedir (EK-2).

FPGA’ nın besleme gerilimi 1.1 Volt olarak ölçülmüştür. Ayrıca çekirdek akımı da 24.73 mA olarak ölçülmüştür. Bu bilgiler FPGA’ nın güç tüketimi hakkında bilgi sağlamaktadır..

Sonuç olarak sağ taraftarki FPGA tarafından işlenen görüntüde şeritlerin sınırlarının iyi bir şekilde belirlenip algılandığı görülmektedir

5.3. Elde Edilen Sonuçların Karşılaştırılması

Aşağıdaki tabloda akım değerleri karşılaştırmalı olarak verilmiş ve analiz edilmiştir.

Tablo 5.1. Uygulamalar Sonucunda Elde Edilen Sonuçların Karşılaştırılması

	Çekirdek Akımı(mA)	Besleme Gerilimi(V)
Normal Uygulama	25.65	1.1
Güç Optimizasyonu Uygulaması	24.73	1.1

Tablodan da görüleceği üzere VHDL kodlarında güç odaklı güncelleme yapılmadan önce akım değeri 25.65 mA' dir. VHDL kodlarında gerçekleştirilen optimizasyonun ardından yaklaşık %3.5867 değerinde bir kazanç sağlanarak 24.73 mA değeri elde edilmiştir.. Bu sayede güçten de kazanç sağlanmaktadır.

6. SONUÇLAR VE ÖNERİLER

Çalışmada, FPGA kullanarak şerit algılama uygulaması ve güç optimizasyonu ele alınmıştır. Sonuçlar, FPGA tabanlı şerit algılama algoritmasının başarılı bir şekilde çalıştığını ve şeritlerin sınırlarının net bir şekilde belirlenebildiğini göstermiştir. Güç tüketimi optimizasyonu için RGB-Y aritmetik işlemlerinde düşük güç kullanımı sağlanmış ve FPGA' nın çekirdek akımında yaklaşık %3.5867 oranında bir azalma elde edilmiştir.

İleriye dönük şerit algılama uygulamasında güç tüketimi optimizasyonu alanında daha kapsamlı çalışmalar yapılabilir. Farklı güç yönetimi teknikleri ya da algoritmalar kullanılarak FPGA' nın verimliliği arttırılabilir. FPGA tabanlı şerit algılama algoritmasının gerçek zamanlı olarak kullanıldığı otonom sürüş sistemlerine entegre edilebilmesi için ileri çalışmalar yapılabilir.

Sonuç olarak, çalışmada FPGA ile görüntü işleme ve şerit algılama uygulaması ele alınmış ve güç optimizasyonu ile ilgili adımlar atılmıştır. Elde edilen sonuçlar FPGA' nın güçlü bir çözüm sağladığını ve şerit algılama uygulamasında başarılı bir şekilde kullanılabildiğini göstermektedir. Yapılabilecekler arasında güç tüketiminin daha başarılı optimize edilmesi ve şerit algılama algoritmasının gerçek zamanlı otonom sürüş sistemlerine entegre edilmesi için çalışılması yer almaktadır.

KAYNAKÇA

[1] https-1: https://fpga-vision-lab.h-brs.de/weblab/login (Erişim Tarihi 18.05.2023)

[2] Solmaz, Ö., Özçevik, Y., Baysal, E., Ökten, M., & Panpallı, A. (2021). Gerçek Zamanlı Simülasyonda Şerit Takibi için Otonom Araç Tasarımı. International Symposium on Automotive Science and Technology, Ankara, Türkiye, 8-10 Eylül 2021.

[3] Koyuncu, I., Taşdemir, M.F., Alçın, M., Tuna, M., & Coşgun, E. (2022). FPGA Üzerinde Görüntü İşleme Algoritmalarının Gerçek Zamanlı Gerçekleştirilmesi. BAUN Fen Bil. Enst. Dergisi, 24(1), 125-137. Doi:10.25092/baunfbed.892032

[4] Schwandt,A. & Winzker,M. Bonn-Rhein-Sieg University, FPGA Vision Open Online Course, Lane Detection – Implementation of Fixed-Point Algorithm.

EKLER

EK-1. Uygulamada Kullanılan VHDL Kodları

```
-- lane.vhd
--
-- top level


library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity lane is
  port (clk       : in  std_logic;                    -- input clock 74.25 MHz, video 720p
        reset_n   : in  std_logic;                    -- reset (invoked during configuration)
        enable_in : in  std_logic_vector(2 downto 0);   -- three slide switches
        -- video in
        vs_in     : in  std_logic;                    -- vertical sync
        hs_in     : in  std_logic;                    -- horizontal sync
        de_in     : in  std_logic;                    -- data enable is '1' for valid pixel
        r_in      : in  std_logic_vector(7 downto 0);   -- red component of pixel
        g_in      : in  std_logic_vector(7 downto 0);   -- green component of pixel
        b_in      : in  std_logic_vector(7 downto 0);   -- blue component of pixel
        -- video out
        vs_out    : out std_logic;                     -- corresponding to video-in
        hs_out    : out std_logic;
        de_out    : out std_logic;
        r_out     : out std_logic_vector(7 downto 0);
        g_out     : out std_logic_vector(7 downto 0);
        b_out     : out std_logic_vector(7 downto 0);
        --
        clk_o     : out std_logic;                    -- output clock (do not modify)
        led       : out std_logic_vector(2 downto 0));  -- not supported by remote lab
end lane;

architecture behave of lane is

  -- input/output FFs
  signal reset           : std_logic;
  signal enable          : std_logic_vector(2 downto 0);
  signal rgb_in, rgb_out  : std_logic_vector(23 downto 0);
  signal vs_0, hs_0, de_0 : std_logic;
  signal vs_1, hs_1, de_1 : std_logic;

begin

  process
  begin
    wait until rising_edge(clk);

    -- input FFs for control
    reset  <= not reset_n;
    enable <= enable_in;
    -- input FFs for video signal
```

```
  vs_0   <= vs_in;
  hs_0   <= hs_in;
  de_0   <= de_in;
  rgb_in <= r_in & g_in & b_in;
 end process;

 -- signal processing
 sobel : entity work.lane_sobel
  port map (clk      => clk,
            reset    => reset,
            de_in    => de_0,
            data_in  => rgb_in,
            data_out => rgb_out);

 -- delay control signals to match pipeline stages of signal processing
 control : entity work.lane_sync
  generic map (delay => 7)
  port map (clk    => clk,
            reset  => reset,
            vs_in  => vs_0,
            hs_in  => hs_0,
            de_in  => de_0,
            vs_out => vs_1,
            hs_out => hs_1,
            de_out => de_1);

 process
 begin
  wait until rising_edge(clk);

  -- output FFs for video signal
  vs_out <= vs_1;
  hs_out <= hs_1;
  de_out <= de_1;
  if (de_1 = '1') then
    -- active video
    r_out <= rgb_out(23 downto 16);
    g_out <= rgb_out(15 downto 8);
    b_out <= rgb_out(7 downto 0);

  else
    -- blanking, set output to black
    r_out <= "00000000";
    g_out <= "00000000";
    b_out <= "00000000";

  end if;
 end process;

 -- do not modify
 clk_o <= clk;
 led   <= "000";

end behave;
```

```
-- lane_g_matrix.vhd
--
-- arithmetic for 3x3 matrix of Sobel filter

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity lane_g_matrix is
  port ( clk       : in  std_logic;
         reset     : in  std_logic;
         in_p1a    : in  std_logic_vector(23 downto 0);
         in_p2     : in  std_logic_vector(23 downto 0);
         in_p1b    : in  std_logic_vector(23 downto 0);
         in_m1a    : in  std_logic_vector(23 downto 0);
         in_m2     : in  std_logic_vector(23 downto 0);
         in_m1b    : in  std_logic_vector(23 downto 0);
         data_out  : out integer range 0 to 268435456);
end lane_g_matrix;

architecture behave of lane_g_matrix is
signal   lum_p1a      : integer range  0 to  4095;
signal   lum_p2       : integer range  0 to  4095;
signal   lum_p1b      : integer range  0 to  4095;
signal   lum_m1a      : integer range  0 to  4095;
signal   lum_m2       : integer range  0 to  4095;
signal   lum_m1b      : integer range  0 to  4095;
signal   sum          : integer range -16383 to 16383;

function rgb2y (vec : std_logic_vector(23 downto 0)) return integer is
   variable result : integer range  0 to  4095;
begin
   -- convert RGB to luminance: Y (5*R + 9*G + 2*B)
   result := 5*to_integer(unsigned(vec(23 downto 16)))
          + 9*to_integer(unsigned(vec(15 downto  8)))
          + 2*to_integer(unsigned(vec( 7 downto  0)));
return result;
end function;

begin

process
begin
  wait until rising_edge(clk);

   -- convert RGB to Y with VHDL-function
                          -- pixel with factor
   lum_p1a   <= rgb2y(in_p1a);   --  plus 1
   lum_p2    <= rgb2y(in_p2);    --  plus 2
   lum_p1b   <= rgb2y(in_p1b);   --  plus 1
   lum_m1a   <= rgb2y(in_m1a);   -- minus 1
   lum_m2    <= rgb2y(in_m2);    -- minus 2
   lum_m1b   <= rgb2y(in_m1b);   -- minus 1

   -- add values according to sobel matrix
```

```
    --        |-1  0  1|      | 1  2  1|
    --        |-2  0  2|  or  | 0  0  0|
    --        |-1  0  1|      |-1 -2 -1|
    sum   <=  lum_p1a + 2*lum_p2 + lum_p1b - lum_m1a - 2*lum_m2 - lum_m1b;

    -- square of sum
    data_out <= sum*sum;

end process;

end behave;
```

```
-- lane_linemem.vhd
--
-- line memory with 1280 pixel delay


library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity lane_linemem is
  port (clk      : in  std_logic;
        reset    : in  std_logic;
        write_en : in  std_logic;
        data_in  : in  std_logic_vector(23 downto 0);
        data_out : out std_logic_vector(23 downto 0));
end lane_linemem;

architecture behave of lane_linemem is

  type ram_array is array (0 to 1279) of std_logic_vector(23 downto 0);
  signal ram : ram_array;

begin

  process
    variable wr_address : integer range 0 to 1279;
    variable rd_address : integer range 0 to 1279;
  begin
    wait until rising_edge(clk);

    if (write_en = '1') then
      data_out        <= ram(rd_address);
      ram(wr_address) <= data_in;
    end if;

    if (reset = '1') then
      wr_address := 0;
      rd_address := 1;
    elsif (write_en = '1') then
      wr_address := rd_address;
      if (rd_address = 1279) then
        rd_address := 0;
      else
        rd_address := rd_address + 1;
      end if;
    end if;
  end process;

end behave;
```

```
-- lane_sobel.vhd
--
-- lane detection algorithm
-- storage of 3x3 image region and calculation with Sobel filter

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity lane_sobel is
  port (clk      : in  std_logic;
        reset    : in  std_logic;
        de_in    : in  std_logic;
        data_in  : in  std_logic_vector(23 downto 0);
        data_out : out std_logic_vector(23 downto 0));
end lane_sobel;

architecture behave of lane_sobel is


  signal tap_lt, tap_ct, tap_rt,
         tap_lc, tap_cc, tap_rc,
         tap_lb, tap_cb, tap_rb : std_logic_vector(23 downto 0);
           -- 3x3 image region
           --     Y->          (left)    (center)    (right)
           --  X      (top)    tap_lt     tap_ct     tap_rt
           --  |   (center)    tap_lc     tap_cc     tap_rc
           --  v   (bottom)    tap_lb     tap_cb     tap_rb
  signal g_x_2, g_y_2          : integer range 0 to 268435456;
  signal g_sum_2               : integer range 0 to 262143;

  signal g2_limit   : std_logic_vector(12 downto 0);
  signal lum_new    : std_logic_vector(7 downto 0);


begin

  -- current input pixel is right-bottom (rb)
  tap_rb <= data_in;

  -- two line memories: output is right-center (rc) and right-top (rt)
  mem_0 : entity work.lane_linemem
    port map (clk      => clk,
              reset    => reset,
              write_en => de_in,
              data_in  => tap_rb,
              data_out => tap_rc);
  mem_1 : entity work.lane_linemem
    port map (clk      => clk,
              reset    => reset,
              write_en => de_in,
              data_in  => tap_rc,
              data_out => tap_rt);

  process
```

```
  begin
    wait until rising_edge(clk);
    -- delay each line by two clock cycles:
    --    previous value of right pixel is now center pixel
    --    previous value of center pixel is now left pixel
    tap_ct <= tap_rt;
    tap_lt <= tap_ct;
    tap_cc <= tap_rc;
    tap_lc <= tap_cc;
    tap_cb <= tap_rb;
    tap_lb <= tap_cb;
  end process;

-- horizontal and vertical sobel matrix and square of G
  g_x : entity work.lane_g_matrix
    port map (clk      => clk,
              reset    => reset,
              in_p1a   => tap_rt,
              in_p2    => tap_rc,
              in_p1b   => tap_rb,
              in_m1a   => tap_lt,
              in_m2    => tap_lc,
              in_m1b   => tap_lb,
              data_out => g_x_2);

  g_y : entity work.lane_g_matrix
    port map (clk      => clk,
              reset    => reset,
              in_p1a   => tap_lt,
              in_p2    => tap_ct,
              in_p1b   => tap_rt,
              in_m1a   => tap_lb,
              in_m2    => tap_cb,
              in_m1b   => tap_rb,
              data_out => g_y_2);

  process

  begin
    wait until rising_edge(clk);
    -- adding the values of horizontal and vertical sobel matrix
    g_sum_2 <= (g_x_2 + g_y_2)/8192;

    -- limiting and invoking ROM for square-root
    if (g_sum_2 > 8191) then
      g2_limit <= (others => '1');
    else
      g2_limit <= std_logic_vector(to_unsigned(g_sum_2, 13));
    end if;
  end process;

  square_root : entity work.lane_g_root_IP  -- 255 minus square-root of 8*g_sum_2
    port map (clock   => clk,
              address => g2_limit,
              q       => lum_new);
```

```
  -- set new luminance for red, green, blue
  data_out <= lum_new & lum_new & lum_new;

end behave;
```

```
-- lane_sync.vhd
--
-- delay video sync signals


library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;

entity lane_sync is
  generic (delay : integer := 7);
  port (clk    : in  std_logic;
        reset  : in  std_logic;
        vs_in  : in  std_logic;
        hs_in  : in  std_logic;
        de_in  : in  std_logic;
        vs_out : out std_logic;
        hs_out : out std_logic;
        de_out : out std_logic);
end lane_sync;

architecture behave of lane_sync is


  type delay_array is array (1 to delay) of std_logic;
  signal vs_delay : delay_array;
  signal hs_delay : delay_array;
  signal de_delay : delay_array;

begin

  process
  begin
    wait until rising_edge(clk);

     -- first value of array is current input
     vs_delay(1) <= vs_in;
     hs_delay(1) <= hs_in;
     de_delay(1) <= de_in;

    -- delay according to generic
    for i in 2 to delay loop
      vs_delay(i) <= vs_delay(i-1);
      hs_delay(i) <= hs_delay(i-1);
      de_delay(i) <= de_delay(i-1);
    end loop;

  end process;

  -- last value of array is output
  vs_out <= vs_delay(delay);
  hs_out <= hs_delay(delay);
  de_out <= de_delay(delay);

end behave;
```

```
-- sim_lane.vhd
--
-- testbench for lane detection
-- reading and writing images in text-file format

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use std.textio.all;
use ieee.std_logic_textio.all;

entity sim_lane is
end sim_lane;

architecture sim of sim_lane is

-- define constants for simulation
  constant stimuli_filename  : string  := "street_0_stimuli.txt";
  constant expected_filename : string  := "street_0_expected.txt";
  constant response_filename : string  := "street_0_response.txt";
  constant x_size            : integer := 1280;  -- horizontal size of image
  constant y_size            : integer := 720;   -- vertical size of image
  constant x_blank           : integer := 100;   -- horizontal blanking
  constant trail             : integer := 1000;  -- clock cycles after active image

-- signals of testbench
  signal clk       : std_logic := '0';
  signal reset_n   : std_logic;
  signal enable_in : std_logic_vector(2 downto 0);
  signal vs_in     : std_logic;
  signal hs_in     : std_logic;
  signal de_in     : std_logic;
  signal r_in      : std_logic_vector(7 downto 0);
  signal g_in      : std_logic_vector(7 downto 0);
  signal b_in      : std_logic_vector(7 downto 0);
  signal vs_out    : std_logic;
  signal hs_out    : std_logic;
  signal de_out    : std_logic;
  signal r_out     : std_logic_vector(7 downto 0);
  signal g_out     : std_logic_vector(7 downto 0);
  signal b_out     : std_logic_vector(7 downto 0);
  signal clk_o     : std_logic;
  signal led       : std_logic_vector(2 downto 0);
  signal x, y      : integer;
  signal end_tb    : integer   := 0;
  signal mismatch  : integer   := 0;

begin

-- clock generation
  clk <= not clk after 5 ns;

-- instantiation of design-under-verification
  duv : entity work.lane
    port map (clk       => clk,
              reset_n   => reset_n,
```

```
            enable_in => enable_in,
            vs_in     => vs_in,
            hs_in     => hs_in,
            de_in     => de_in,
            r_in      => r_in,
            g_in      => g_in,
            b_in      => b_in,
            vs_out    => vs_out,
            hs_out    => hs_out,
            de_out    => de_out,
            r_out     => r_out,
            g_out     => g_out,
            b_out     => b_out,
            clk_o     => clk_o,
            led       => led);

-- main process for stimuli
  stimuli_process : process

-- variables for stimuli file
    file stimuli_file       : text;
    variable l              : line;
    variable stimuli_status : file_open_status;
    variable r, g, b        : std_logic_vector(7 downto 0);

  begin

-- init
    reset_n   <= '0', '1' after 50 ns;  -- reset for 10 clock cycles
    enable_in <= "111";
    vs_in     <= '0';
    hs_in     <= '0';
    de_in     <= '0';
    r_in      <= "00000000";
    g_in      <= "00000000";
    b_in      <= "00000000";

-- wait for reset
    wait for 100 ns;

    file_open(stimuli_status, stimuli_file, stimuli_filename, read_mode);
    readline(stimuli_file, l);          -- read first line with comments

-- loop for one frame
    for y in 0 to y_size-1 loop
      if (y = 0) then
        vs_in <= '1';
      else
        vs_in <= '0';
      end if;

      hs_in <= '1';
      for x in 0 to x_blank-1 loop
        wait until falling_edge(clk);
      end loop;  -- x, blanking
```

```
      hs_in <= '0';
      de_in <= '1';
      for x in 0 to x_size-1 loop
        readline(stimuli_file, l);      -- read one line
        hread(l, r);
        hread(l, g);
        hread(l, b);
        r_in <= r;
        g_in <= g;
        b_in <= b;

        wait until falling_edge(clk);
      end loop;  -- x, active line
      de_in <= '0';
      r_in  <= "00000000";
      g_in  <= "00000000";
      b_in  <= "00000000";

    end loop;  -- y

-- simulation for trailing clock cycles
    for i in 0 to trail-1 loop
      wait until falling_edge(clk);
    end loop;

    end_tb <= 1;                    -- signal to close response_file
    file_close(stimuli_file);
    wait for 20 ns;

-- stop simulation
    if (mismatch = 0) then
      assert false
        report "Simulation completed, EVERYTHING OK"
        severity failure;
    else
      assert false
        report "Simulation completed, " & integer'image(mismatch) & " MISMATCHES XXXX XXXX
            XXXX XXXX XXXX XXXX"
        severity failure;
    end if;

  end process;

-- second process to handle DUT output
  response_process : process
    variable x_pos, y_pos     : integer := 0;
-- variables for writing simulated response
    file response_file        : text;
    variable l                : line;
    variable response_status  : file_open_status;
    variable r, g, b          : std_logic_vector(7 downto 0);
-- variables for expected response file
    file expected_file        : text;
    variable l_ex             : line;
```

```
    variable expected_status  : file_open_status;
    variable r_ex, g_ex, b_ex : std_logic_vector(7 downto 0);

  begin

-- open file for output
    file_open(response_status, response_file, response_filename, write_mode);
    write (l, string'("# Output from edupow-testbench"));
    writeline(response_file, l);        -- write first line with comments
-- open file for expected data
    file_open(expected_status, expected_file, expected_filename, read_mode);
    readline(expected_file, l_ex);      -- read first line with comments

    wait until (hs_out = '1');

    while (end_tb /= 1) loop
      wait until falling_edge(clk);
      if (de_out = '1') then

        -- get response and write to response file
        if (x_pos >= 2 and y_pos >= 2) then
          -- valid data
          r := r_out;
          g := g_out;
          b := b_out;
        else
          -- left or top border, set to white
          r := "11111111";
          g := "11111111";
          b := "11111111";
        end if;
        hwrite(l, r);
        write (l, string'(" "));
        hwrite(l, g);
        write (l, string'(" "));
        hwrite(l, b);
        writeline(response_file, l);

        -- read expected response and compare to simulation response
        readline(expected_file, l_ex);  -- read one line
        hread(l_ex, r_ex);
        hread(l_ex, g_ex);
        hread(l_ex, b_ex);
        if (x_pos >= 2 and y_pos >= 2) then
          -- check only if valid data
          if (r /= r_ex) or (g /= g_ex) or (b /= b_ex) then
            mismatch <= mismatch + 1;
            assert false
              report "MISMATCH in simulation at position x=" & integer'image(x_pos) & " y=" &
             integer'image(y_pos)
              severity note;
          end if;
        end if;

        x_pos := x_pos + 1;
```

```
        if x_pos = x_size then
          y_pos := y_pos + 1;
          x_pos := 0;
        end if;

      end if;
    end loop;

    file_close(response_file);
    file_close(expected_file);
    wait until (end_tb = 2);

  end process;

end sim;
```

```
# edupow_default.sdc
#
# Timing constraints for EduPow-Board with 74.25 MHz 720p-signal


# Clock constraints
create_clock -name input_clk -period 13.47ns [get_ports {clk}]
create_generated_clock -name output_clk -source [get_ports {clk}] -master_clock input_clk -add
        [get_ports {clk_o}]

# Define IO-paths
set_input_delay -clock input_clk -max 0.1 [get_ports {reset_n *_in*}]
set_output_delay -clock output_clk -max 0.1 [get_ports {*_out* led*}]

set_input_delay -add_delay -clock input_clk -fall -min 0.05 [get_ports {reset_n *_in*}]
set_output_delay -add_delay -clock output_clk -fall -min 0.05 [get_ports {*_out* led*}]

set_input_delay -add_delay -clock input_clk -rise -min 0.05 [get_ports {reset_n *_in*}]
set_output_delay -add_delay -clock output_clk -rise -min 0.05 [get_ports {*_out* led*}]

# Automatically calculate clock uncertainty to jitter and other effects.
derive_clock_uncertainty
```

EK-2. Düşük Güç Uygulamasında Optimize Edilen lane_g_matrix.vhd

```
-- lane_g_matrix.vhd
--
-- arithmetic for 3x3 matrix of Sobel filter


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity lane_g_matrix is
  port ( clk       : in  std_logic;
         reset     : in  std_logic;
         in_p1a    : in  std_logic_vector(23 downto 0);
         in_p2     : in  std_logic_vector(23 downto 0);
         in_p1b    : in  std_logic_vector(23 downto 0);
         in_m1a    : in  std_logic_vector(23 downto 0);
         in_m2     : in  std_logic_vector(23 downto 0);
         in_m1b    : in  std_logic_vector(23 downto 0);
         data_out  : out integer range 0 to 268435456);
end lane_g_matrix;

architecture behave of lane_g_matrix is
signal   lum_p1a      : integer range  0 to  4095;
signal   lum_p2       : integer range  0 to  4095;
signal   lum_p1b      : integer range  0 to  4095;
signal   lum_m1a      : integer range  0 to  4095;
signal   lum_m2       : integer range  0 to  4095;
signal   lum_m1b      : integer range  0 to  4095;
signal   sum          : integer range -16383 to 16383;

function rgb2y (vec : std_logic_vector(23 downto 0)) return integer is
   variable result : integer range  0 to  4095;
begin
   -- convert RGB to luminance: Y (5*R + 9*G + 2*B)
   result := (5*to_integer(unsigned(vec(23 downto 16)))
         + 9*to_integer(unsigned(vec(15 downto  8)))
         + 2*to_integer(unsigned(vec( 7 downto  0)))) / 16;
return result;
end function;

begin

process
begin
  wait until rising_edge(clk);

   -- convert RGB to Y with VHDL-function
                        -- pixel with factor
   lum_p1a   <= rgb2y(in_p1a);   --  plus 1
   lum_p2    <= rgb2y(in_p2);    --  plus 2
   lum_p1b   <= rgb2y(in_p1b);   --  plus 1
   lum_m1a   <= rgb2y(in_m1a);   -- minus 1
   lum_m2    <= rgb2y(in_m2);    -- minus 2
```

```
    lum_m1b   <= rgb2y(in_m1b);   -- minus 1

    -- add values according to sobel matrix
    --         |-1  0  1|      | 1  2  1|
    --         |-2  0  2|  or  | 0  0  0|
    --         |-1  0  1|      |-1 -2 -1|
    sum   <=  lum_p1a + 2*lum_p2 + lum_p1b - lum_m1a - 2*lum_m2 - lum_m1b;

    -- square of sum
    data_out <= sum*sum;

end process;

end behave;
```