İSTANBUL TEKNİK ÜNİVERSİTESİ
ELEKTRİK-ELEKTRONİK FAKÜLTESİ

AÇIK KAYNAK KODLU İŞLEMCİLERİN KOMUT SETİNİN UYGULAMAYA ÖZEL GENİŞLETİLMESİ

LİSANS BİTİRME TASARIM PROJESİ

Muhammet Hamidullah ERDEM

Ömer Nevzat KÖSEBAY

Emre SEVER

ELEKTRONİK VE HABERLEŞME MÜHENDİSLİĞİ BÖLÜMÜ

OCAK, 2020

İSTANBUL TEKNİK ÜNİVERSİTESİ
ELEKTRİK-ELEKTRONİK FAKÜLTESİ

AÇIK KAYNAK KODLU İŞLEMCİLERİN KOMUT SETİNİN UYGULAMAYA ÖZEL GENİŞLETİLMESİ

LİSANS BİTİRME TASARIM PROJESİ

M. Hamidullah Erdem
(040140081)

Ömer Nevzat Kösebay
(090150105)

Emre Sever
(040140103)

Proje Danışmanı: Doç. Dr. Sıddıka Berna ÖRS YALÇIN

ELEKTRONİK VE HABERLEŞME MÜHENDİSLİĞİ BÖLÜMÜ

OCAK, 2020

İTÜ Elektronik ve Haberleşme Mühendisliği Bölümü'nün ilgili Bitirme Tasarım Projesi yönergesine uygun olarak tamamen kendi çalışmamız sonucu hazırladığımız "Açık Kaynak Kodlu İşlemcilerin Komut Setinin Uygulamaya Özel Genişletilmesi" başlıklı Bitirme Tasarım Projesi'ni sunmaktayız. Bu çalışmayı intihal olmaksızın hazırladığımızı taahhüt eder; intihal olması durumunda bitirme tasarım projesinin başarısız sayılacağını kabul ederiz.

Muhammet Hamidullah Erdem
(040140081)

Emre Sever
(040140103)

Ömer Nevzat Kösebay
(090150105)

Proje Danışmanı : Doç. Dr. Sıddıka Berna ÖRS YALÇIN

ÖNSÖZ

Bitirme tezi boyunca bize yol gösteren, değerli vakitini bize ayıran, yeri geldiğinde bize bilgilerini aktaran, yeri geldiğinde de bizimle birlikte öğrenen, tezimizi ve projemizi başarıyla bitirmemizde emeği çok büyük olan sayın hocamız Doç. Dr. Sıddıka Berna ÖRS YALÇIN'a ve karşılaştığımız sorunları çözmemizde daima güvenebileceğimiz bir isim olarak bizi destekleyen Yüksek mühendis Latif AKÇAY'a, deneyiminlerini sonsuz bir enerji ile bize aktaran sayın Bartu SÜRER'e teşekkür'ü borç biliriz.

Bize kattığı bilgi, deneyim ve vizyon için İTÜ ailesinin her bir ferdine özellikle teşekkür ederiz.

Ocak 2020

M.Hamidullah ERDEM
Emre SEVER
Ömer Nevzat KÖSEBAY

İÇİNDEKİLER

Sayfa

KISALTMALAR

ALU : Aritmetik Mantık Birimi

CPU : Merkezi İşlemci Birimi

DSP : Sayısal İşaret işleme

FPGA : Alanda Programlanabilen Kapı Dizileri

ISA : Komut Seti Mimarisi

OS : İşletim Sistemi

RISC : Azaltılmış Komut Seti Bilgisayarı

RTL : Kaydedici Transfer Seviyesi

SoC : Kırmık Üstü Sistem

VHDL : Çok Yüksek Hızlı Tümleşik Devre Donanım Tanımlama Dili

ASIC : Uygulamaya Özgü Tüm Devre

I/O : Giriş/Çıkış

ÇİZELGE LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

AÇIK KAYNAK KODLU İŞLEMCİLERİN KOMUT SETİNİN UYGULAMAYA ÖZEL GENİŞLETİLMESİ

ÖZET

Günümüzde kullanılan elektronik cihazları pazarda öne çıkaran iki önemli özelliği hızı ve maliyetidir. Elektronik cihazları maliyet açısından en çok zorlayan etken ise işlemcisidir ki bu aynı zamanda cihazın hızı üzerinde büyük etkisi olan bir elemandır. Günümüzde şirketler kendi işlemcisini kendisi üretiyor veya bu konuda deneyimli olan şirketlerin tasarlamış olduğu işlemcileri kullanıyor. İşlemci mimarisinde bir ortak standartın olmaması ise şirketleri farklı farklı amaca yönelik işlemci tasarımına yöneltiyor. İşlemci mimarisinin bu karmaşık ve çoklu yapısı yüzünden bir işlemciyi oluşturmak ve bunu kullanabilmek çok karmaşık bir işlem olarak görülüyor. Son yıllarda bu konuyu bir çözüme kavuşturmak ve ortak standartı olan bir işlemci tasarımı geliştirmek için RISC-V projesi başlatıldı.

FPGA kartları, sahada programlanabilen mantık blokları ve bloklar arası bağlardan oluşan sayısal tümleşik devrelerdir. Geniş programlama alanları ve amaca yönelik kullanımı nedeniyle bu projedeki olmazsa olmazlarımızdan biridir. Bu projede NEXYS 4 DDR FPGA kartta gerçeklediğimiz RISC-V işlemci mimarisi ve daha hızlı çalıştırmayı hedeflediğimiz uygulamamızı test edeceğimiz elemandır.

Açık kaynak kodlu donanım ISA (Instruction Set Architecture)'sı olarak geçen RISC-V ile yapılan çalışmalar günümüzde FPGA programlamada gittikçe yükselen bir trend haline gelmiştir. RISC-V'in basit işlemci mimarisinden biri olan FPGA'da kullanılmak için VHDL'de yazılmış Potato ile birçok uygulama gerçeklenebilmektedir. RISC-V'in en önemli ve ilk göze çarpan özelliği amaca yönelik programlanabilen grupları (ISA uzantıları) bizim projemizde yoğunlaştığımız alandır.

RISC-V üzerinde uygulamaya özel komut seti genişletme işlemi; kabaca uygulamayı yazdıktan sonra, çalışan kodun assembly kodu olarak yaptığı işlemi incelemek ve kullanılmayan komutun tespit edilip onun yerine uygulamamızda daha yararlı ve

işlevsel komutu işlemciye yerleştireyi hedefledik. Bu sayede eklediğimiz komutu kullanarak işlemcide önemli bir performans etkisi oluşturulabilir.

APPLICATION – SPECIFIC EXTENSION OF THE COMMAND SET OF OPEN SOURCE PROCESSORS

SUMMARY

Speed and cost are the two most important features of the electronic devices used in the market today. The most challenging factor in electronic devices is the processor, which is also an element that has a major impact on the speed of the device. Today, companies produce their own processors or use processors designed by companies that are experienced in this field. The lack of a common standard in processor architecture leads companies to design processors for different purposes. Because of this complex and multiple structure of the processor architecture, creating and using a processor is seen as a very complex process. In recent years, the RISC-V project has been initiated to resolve this issue and develop a common standard processor design.

FPGA cards are digital integrated circuits consisting field programmable logic blocks and inter-block connections. It is one of the sine quanon of this project due to its wide programming areas and purposeful use. In this project, the NEXYS 4 DDR FPGA card, the RISC-V processor architecture that we embed in the card and the application that we aim to run faster is the element we will test.

The work with RISC-V, which is called Open Source Hardware ISA (Instruction Set Architecture), has become an increasing trend in FPGA programming today. Many applications can be implemented with POTATO written in VHDL for use in FPGA, which is one of the simple processor architecture of RISC-V. The most important and outstanding feature of RISC-V is its purpose-built programmable groups (ISA extensions) that we focus on in our project.

Application-specific instruction set expansion on RISC-V; After roughly writing the application, we aimed to examine the operation of the running code as assembly code and to identify the unused command and replace it with a more useful and functional command in our application. In this way, a significant performance effect can be created on the processor by using the command added.

1. GİRİŞ

Azaltılmış Komut Seti Bilgisayarı (Reduced Instruction Set Computer-RISC-V), dünya genelindeki kullanıcılarının iş birliğine dayanan işlemci mimarisinde yeni bir çağ açan ücretsiz açık kaynak kodlu komut seti mimarisidir (ISA-Instruction Set Extension). Akademi dünyasında doğan RISC-V ISA, genişletilebilir bir yazılım ve donanım mimarisi sunarak gelecek 50 yıllık bilgisayar ve işlemci tasarım yeniliklerine zeminini hazırlıyor [1]. Bu projenin asıl yoğunlaşılan alan bu yeni ücretsiz ve geliştirilebilen RISC-V işlemcilerin mimarisidir.

Projenin amacı "Açık Kaynak Kodlu İşlemcilerin Komut Setinin Uygulamaya Özel Genişletilmesi" için yapılacak değişiklikleri test etmek amacıyla kenar algılama (edge detection) uygulaması seçildi. Edge detection görüntü filtreden geçirilerek yapılabilir. Filtre algoritması kabaca piksel değerlerine ilgili çekirdek (kernel) ile konvolüsyon işlemi uygulayarak yeni bir çıktı oluşturur. Tez içindeki 4. bölümde daha ayrıntılı olarak tekrar anlatılacaktır. Bu uygulama C yazılım dilinde yazılıp alanda programlanabilir kapı devresi (Field Programmable Gate Array – FPGA) 'de gerçeklenmiş olan "Potato" işlemcisi üzerinde test edilecektir.

Bu projede düşük maliyeti ve kolay programlanabilmesi açısından RISC-V uygulamalarında vazgeçilmez eleman olan FPGA kartı kullanılacaktır. Kullanılacak kart NEXYS 4 DDR kartıdır. RISC-V Potato işlemci algoritması bu karta yüklenecek ve uygulama "edge detection" algoritması bu kartta test edilecektir.

FPGA ile çalışılıyorsa FPGA'i programlayabilmek için gerekli yazılımlar vardır. Xilinx Vivado'da FPGA programlamak için yine Xilinx tarafından geliştirilen Yüksek Hızlı Tümleşik Devre Donanım Tanımlama Dili (VHDL – Very High Speed Integrated Circuit Hardware Description Language) kullanılıyor.

1.1 Projenin Amacı

Daha önce RISC-V ile ilişkili proje olarak görülen Bartu SÜRER'in bitirme projesi "Implementation of A SoC By Using LowRISC Processor on an FPGA for Image Filtering Applications" bu projeye başlanmadan önce tekrarlandı ve gerçekleştirildi [2]. Sürer'in projesinden elde edinilen bilgi ve deneyimleri kullanarak bu projeyi bir

adım daha ileri taşıyıp daha efektif nasıl çalıştırılabileceği üzerine çalışmalar yapıldı. Sürer'in yapmış olduğu uygulamayı FPGA'de gerçeklenen Potato işlemci mimarisinde yapılan değişiklikler ile daha hızlı ve efektif kullanılması hedefleniyor. Bu değişikliklerin nasıl yapıldığı aşağıda ilgili bölümlerde detaylı bir şekilde grafik ve şekiller yardımı ile açıklandı.

1.2 Literatür Araştırması

Yapılan literatür taraması şu şekildedir:

- (Altameemi, A. A., & Bergmann, N. W. 2016). "Enhancing FPGA softcore processors for digital signal processing applications."

Alteameemi ve Bergmann bu projede, FPGA tabanlı bir softcore işlemciye açık DSP (Digital Signal Processing – Sayısal İşaret İşleme) işlem desteği eklemenin yararlarını inceledi. Deneysel bir DSP işlemciyi, bir Xilinx Virtex-5 XC5VSX50T FPGA yongası üzerinde tasarladılar, uyguladılar, simüle ettiler ve doğruladılar. Başlangıçta işlemcinin DSP uygulamalarının performansını artırmak için yeni donanımlar geliştirdiler. Bu donanım geliştirme teknikleri, uzman bir yürütme birimi, bellek mimarisinde değişiklik yapma, ekstra adres kontrol birimi ekleme, daha etkin adresleme modlarıyla yükseltme ve komut setini sıfır tepegöz döngü desteği ile genişletme gibi yöntemler içeriyor. Bu geliştirmeler, evrişim tabanlı DSP algoritmalarının iç döngüsünde kullanılan komutların sayısını tek bir komutla azalttı. Önerilen bu geliştirme tekniklerini kullanarak, genel performansı FIR Filtre ve Matris Çarpma karşılaştırmalı değerlendirme DSP uygulamalarında ortalama 9 kat (% 800) arttırdılar. Toplam alan yaklaşık 1.41 kat arttı [3].

- (Güngör, Öndeş, Sarı, Uçkun; 2018) "Instruction Set Extension Of Some Processors For Secure Iot Implementations"

Bu tezin ana konusu açık kaynaklı işlemcilerde komut seti genişletmesi (ISE) ile AES ve PRESENT kriptoloji algoritmaları kullanmak ve bu işlemcilerin IoT uygulamaları için güvenliğini artırmaktır. Projenin işlemcilerinden biri ETH Zürih'e ait Pulpino RI5CY'dir. Geliştirici Talip Tolga SARI, bu çekirdeği önce Xilinx'e ait Nexys 4 DDR geliştirme kartında gerçeklemiş ve daha sonra test ederek çalıştığını gözlemlemiştir. Bu aşamadan sonra tezde AES uygulamasının ana konusu, başarılı bir şekilde yürütülen komut seti genişletme yöntemi kullanılarak uygulanmıştır [4].

- (Sürer; 2019) “Implementation of a SOC by Using LowRISC Processor on an FPGA for Image Filtering Applications”

Bartu Sürer, FPGA üzerinde gerçekleştiği RISC-V işlemci mimarisi LowRISC’ı kullanarak görüntü işlemede kullanılan filtrelerden Gauss bulanıklaştırma filtresi, Sobel operasyonu filtresi, Kabartma filtresi, keskinleştirme filtresi, siyah-beyaz filtre ve daha pek çok filtreyi test etmiş ve sonuçlarını paylaşmıştır. Hedefi, bu uygulamaları RISC-V işlemci mimarisi kullanarak gerçeklenebileceğini göstermektir ve bunda başarılı olmuştur [2].

1.3 Projenin Planlanan Yöntemi

FPGA’ya yüklenilen işlemci mimarisi, yazılan uygulama olan edge detection çalıştırıldıktan sonra en çok tekrar eden komut saptandı. Bunu yaptıktan sonra en çok tekrar eden komutu hiç kullanılmayan başka bir komut ile değiştirmek ve bu komutu bir modül olarak tasarlayıp çalışma hızını arttırıp, güç ve alanda tasarruf sağlanması amaçlandı.

2. ÖNBİLGİLER

2.1 Evrensel Asenkron Alıcı Verici

Evrensel Asenkron Alıcı Verici (Universal Asyncronus Receiver Transmitter– UART) iletişiminde iki cihaz birbiriyle doğrudan iletişim kurar. CPU gibi veri gönderen bir kontrol cihazından, paralel olarak veriler seri forma dönüştürülür ve seri halde alıcı cihaza iletilir, daha sonra bu seri veriler tekrar paralel verilere dönüştürülür [5]. İki cihaz arasında veri iletmek için sadece iki kablo yeterlidir bu da UART'ın basitliğinden gelen güzelliğidir. Projede FPGA'da gerçeklenmiş Potato ile haberleşme için "UART" protokolünü kullandık.

2.2 Çok Yüksek Hızlı Tümleşik Devre Donanım Tanımlama Dili

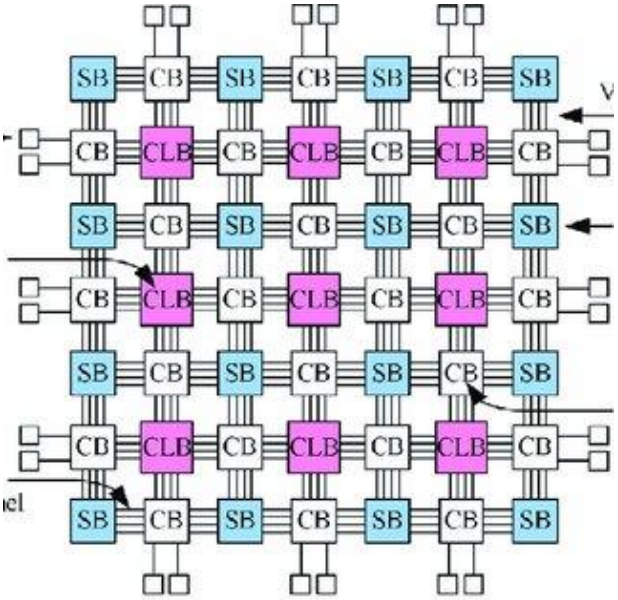
VHDL "VHSIC (Very High-Speed Integrated Circuit) Hardware Description Language" oldukça hızlı tümleşik devre donanım tanımlama dilidir. İlk olarak Amerika savunma bakanlığı tarafından ortaya atılmış ancak sonradan IEEE enstitüsüne devredilmiştir. VHDL dili ilk olarak IEEE 1076 standardı olarak tanımlanmış ve bu Standard 1987 yılında VHDL 87 olarak onaylanmıştır. Bu sürüm daha sonraları birçok düzeltmeye uğramıştır. Bu sürüm 1993 yılında yerini VHDL 93 sürümüne bırakmıştır [6].

VHDL sayısal sistemi birçok seviyede tanımlama ve modelleme işine talip olan üst düzey bir donanım tanımlama dilidir. Bizim de projemizde kullanacağımız dil VHDL'dir.

2.3 Sahada Programlanabilir Kapı Dizileri

Analog devrelerden sayısal devrelere geçildiği günümüzde, sayısal işaret işlemede, sayısal filtreler önemli bir rol oynamaktadır. Birçok cihazda kullanılan sayısal filtreleri gerçeklemek için çeşitli yollar bulunmaktadır. Maliyetinin düşüklüğü, tasarlanan devreyi gerçekleme ve test etme süresinin kısalığı ve tekrar tekrar programlanabilmesinin getirdiği avantajlar dolayısıyla tasarımda FPGA kullanımı tercih edilmiştir [7].

Alanda programlanabilir olarak adlandırılmasının nedeni ise mantık bloklarının ve ara bağlantıların üretim sürecinden sonra da istenen amaca göre tekrar tekrar programlanabilmesidir. FPGA'ların en önemli özelliklerinden biri paralel işlem yapabilmeleridir. Yani uygulamaya ve kapasiteye göre, birbirine paralel birçok işlemi aynı anda yapabilmek mümkündür. FPGA'lar genellikle uygulamaya özel tümleşik devrelere (ASIC – Application Specific Integrated Circuit) göre daha yavaştır ancak ASIC'ler gibi uzun tasarım süreleri ve başlangıç maliyeti gerektirmezler. Buna karşıt daha fazla güç tüketirler [8]. FPGA'lar genellikle ASIC tasarımların fiziksel gerçeklenebilirliğini doğrulamak amacıyla donanım ortamı sağlamak için kullanılır. İlk FPGA'larda CMOS tabanlı yapılandırma için SRAM tabanlı hücreler kullanılıyordu. İlk örnekleri çok basit olsa da temelde var olan mimari günümüzde hala kullanılmaktadır. Kullanıma ilk başlandığı 1980'lerin ortalarında FPGA'lardan, kısıtlı miktarda veri işleme görevlerinden faydalanılmıştır. Bu dönemde FPGA'lar genelde, fonksiyonlar veya cihazlar arasında ara bağlantı olarak kullanılmıştır. 1990'lı yılların başında artan kapasiteleri sayesinde geniş veri işlemleri gerektiren haberleşme ve ağ ortamlarında kullanımı artmıştır. 2000'lerin ilk yıllarında milyonlarca kapı içeren yüksek performanslı FPGA'lar piyasadaki yerini almıştır.



Şekil 2.1: FPGA'nın iç yapısı [9].

FPGA mimarisinin en önemli avantajı paralel işlem kabiliyetidir. Aynı anda birçok paralel işlemi gerçekleştirebilirler ve bu sayede çok hızlı sistemlerin tasarlanmasına olanak sağlarlar. Sinyal ve görüntü işleme uygulamalarında, mikroişlemcilere göre çok daha üstün performans gösterdiği için FPGA'lar tercih edilmektedir. Ayrıca uygulamaya özgü tüm devre (Application Spesific Integrated Circuits – ASIC) tasarımlara göre maliyeti az ve piyasaya çıkma süresi daha kısadır. FPGA'ların önemli avantajlarından bir diğeri de içerisine mikroişlemci gerçeklenebilmenin mümkün olmasıdır. Bütün bir sistemin aynı çip üzerinde yer alması sayesinde bağlantılar arası gecikmeler azaltılır ve FPGA üzerinde yüksek hızlı sistemler tasarlanabilir. Paralel işlem kabiliyeti ve istenildiği zaman tekrar programlanabilmesinin sağladığı önemli avantajlar sebebiyle bu tez çalışmasında FPGA kullanılmıştır [10].

2.3.1 Nexys 4 DDR

Nexys 4 DDR kartı, Xilinx®'in en yeni Artix-7™ FPGA'sını temel alan çevre birimleri açısından eksiksiz sayılabilecek, kullanıma hazır bir dijital devre geliştirme platformudur. Nexys 4 DDR, yüksek kapasiteli FPGA (Xilinx parça numarası XC7A100T-1CSG324C), harici bellekler, USB, Ethernet ve diğer bağlantı noktalarının toplanmasıyla, lojik kapılı devrelerden güçlü gömülü işlemcilere kadar çeşitli tasarımlar gerçeklenebilir. İvmeölçer, sıcaklık sensörü, hafızalar (memory), dijital mikrofon ve birkaç G/Ç (I/O) cihazı dâhil olmak üzere birçok yerleşik çevre birimi, Nexys 4 DDR'nin başka bir bileşene ihtiyaç duymadan çok çeşitli tasarımlar için kullanılmasına izin verir [11]. Projemizde kullanacağımız FPGA kartıdır. Projemizi bu kart üzerinde gerçekleyeceğiz.

Şekil 2.2: Nexys 4 DDR FPGA.

2.4 Xilinx Vivado Design Suite

Vivado Design Suit Xilinx firmasının geliştirdiği ve ilk olarak 2012 yılında tanıttığı bir arayüz yazılımıdır. Vivado'dan önce Xilinx'in tasarımcılara sunduğu ISE programı mevcuttu. Vivado'nun yayınlanmasıyla birlikte Xilinx, ISE'nin geliştirmesini durdurmuştur. Vivado, ISE'ye göre daha kullanışlı bir arayüze sahiptir ve birçok özelliği tek bir çatı altında toplaması sayesinde çok daha kullanışlı bir

ortam sağlar. Vivado'da Xilinx'in yeni nesil FPGA'ları programlanabilirken, eski nesil FPGA'ları desteklememektedir. Vivado platformu blok tasarım yapmayı, tasarlanan blokların paketlenmesini ve yazılım donanım olarak ortak tasarım yapılmasını oldukça pratik hale getirmiştir [12].

2.5 Softcore İşlemciler

Yarı iletkenlerin ve elektroniklerin genel olarak gelişimi ilerlerken fiyatları yavaş yavaş azaldı. İkisini tek bir pakette birleştirilen FPGA'lar ve mikroişlemciler karıştırılarak daha fazla esneklik elde edildi. FPGA'lar ASIC'lardan yavaş olsalar da esneklik açısından çeşitli avantajlar sunarlar. Bununla birlikte, FPGA kullanırken I/O bağlantıları açısından problem yaşarız. En basitinden, iletişim için gerekli modülleri tasarıma tek tek eklememiz gerekir çünkü işlemciler ve işletim sistemleri gibi sürücüleri yoktur. Sürücüleri kendi tasarımınıza göre eklemeniz gerekir [13]. Bu problemi aşmak için hibrit bir mimari oluşturulmuştur. Bunu basitçe açıklamak gerekirse, işlemci ve FPGA'nın beraber olduğu bir tasarımdır. Bu yapı iki şekilde oluşturulabilir;

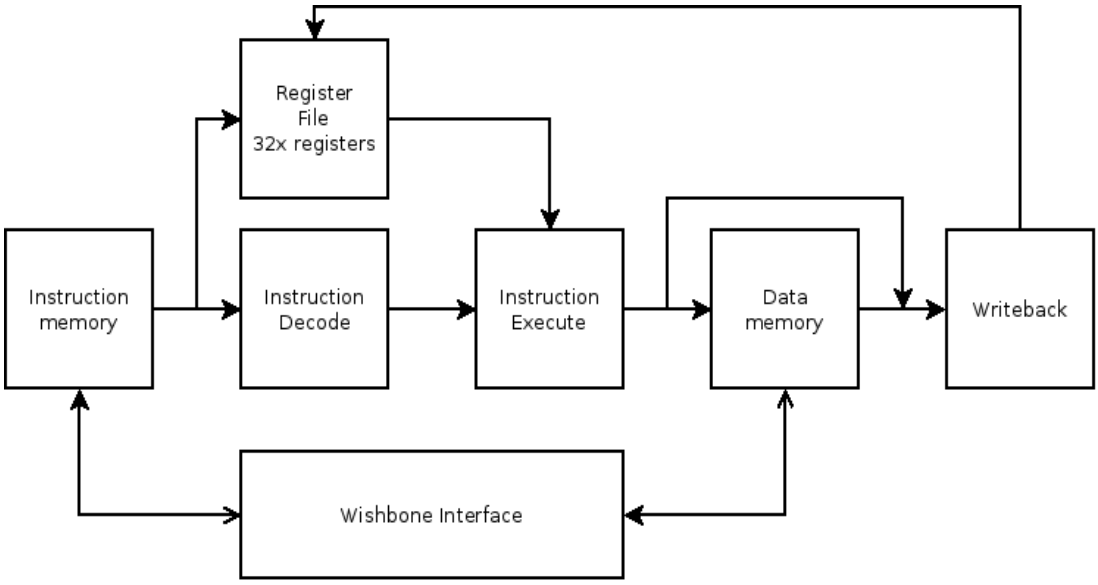
Birincisi, FPGA üzerinde özel bir blok oluşturarak işlemciyi hard olarak yani fiziksel olarak oraya yerleştirmek. İkincisi ise, çekirdeğinin FPGA'da yazılımsal olarak gerçeklenmesi ile elde edilir [13]. Her ne kadar hardcore işlemciler birçok açıdan softcore işlemcilerden iyi olsalar da (Çizelge 2.1'de görüldüğü üzere) softcore işlemciler esnekliğin gerekli olduğu yerlerde vazgeçilmez hale geliyorlar.

	Soft-core	Hard-core
Power consumption	✘	✓
Speed	✘	✓
Resource utilization	✘	✓
Flexibility for design	✓	✘

Çizelge 2.1: Softcore ve Hardcore işlemcilerin özelliklerinin karşılaştırılması [13].

2.5.1 Potato işlemcisi

Potato RISC-V RV32E komut setini kapsayan VHDL dili kullanılarak yazılmış softcore bir işlemcidir. İşlemciye ek olarak Wishbone arayüzü ve UART, GPIO, Timer, ROM ve RAM modülleri bulunmaktadır [14].



Şekil 2.3: Potato beş aşamalı pipeline[14].

2.6 Dijital Görüntü İşleme

Dijital görüntü işleme, görüntülerin bilgisayarlar yardımı ile değiştirilerek daha yararlı bilgiler elde etmesi ile oluşur. Görüntüler matematiksel olarak iki boyutlu diziler olarak kabul edilebilir ve bunlar uygun filtreler yardımıyla işlevsel hale getirilebilirler. Son yirmi yılda kullanımı katlanarak artmaktadır. Uygulama alanları jeoloji'den uzaktan algılamaya, tıptan eğlence sektörüne kadar değişir. Modern bilgi toplumunun temel taşlarından biri olan multimedya sistemleri büyük ölçüde görüntü işlemeye dayanmaktadır [15]. Kenar algılama, görüntünün kenarların bulunması işlemidir. Görüntüdeki kenarların algılanması görüntü özelliklerini anlama yolunda çok önemli bir adımdır. Kenarlar, anlamlı özelliklerden oluşur ve önemli bilgiler içerir. Görüntü boyutunu önemli ölçüde azaltır ve daha az ilgili olduğu düşünülebilecek bilgileri filtreleyerek, önemli yapısal bilgileri koruyarak geriye anlam oranı daha yüksek bir görüntü bırakır [16]. Bu uygulama, çizgi takip sistemleri ve özellikle otonom araçlarda çokça kullanılır. Bu projede ise Edge Detection,

Laplacian operatörü kullanılarak yapıldı. Uygulama detayları ve temel algoritma Bölüm 4.1'de anlatılmıştır.

Şekil 2.4 Sobel filtre kullanılmış bazı edge detection örnekleri [17].

3. POTATO İŞLEMCİSİNİN FPGA'DA GERÇEKLENMESİ

3.1 Çalışma Ortamının kurulması

Başlangıç olarak tüm çalışmalar "Ubuntu 16.04" sürümünde gerçekleştirildi. Buna ek olarak Potato işlemcisini FPGA'da gerçekleyebilmek için Vivado 2018.2 sürümüne ihtiyacımız var. Sürüm önemli çünkü Vivado daha eski veya daha yeni olan sürümler için IP (Intellectual Property) çekirdeklerinde değişiklikler yapabiliyor ve bu da tasarımda karışıklıklara yol açabiliyor.

3.1.1 Sürücülerin yüklenmesi

Vivado'nun FPGA'yı tanıyabilmesi için "Cable drivers" (FPGA sürücüsü) kurulumuna ihtiyaç vardır. Bu sürücüler yüklü değilse, Vivado FPGA'yı programlama işleminde tanımıyor. Bu sürücülere Digilent'in web sitesinden ulaşılabilir. Bunlar sürücüler için önerilen programlar;

Adept 2.16.1 Runtime, X64 DEB

Adept 2.2.1 Utilities, X64 DEB

Bu sürücüler Digilent'in web sitesinden indirilebilir [18]. İndirdikten sonra kurulum işlemi basit bir şekilde gerçekleştirilebilir.

Bu sürücülerin kurulumundan sonra (JTAG - Joint Test Action Group) sürücüsünün de yüklenmesi gerekir. Yeni bir terminal açılıp aşağıdaki kodlar girilir;

```
$cd

$/opt/Xilinx/Vivado/2018.1/data/xicom/cable_drivers/lin64/install_script/install

_drivers

$ sudo sh install_digilent.sh
```

Bu yüklemeler gerçekleştirildikten sonra, sonraki aşamaya geçilebilir.

3.1.1 RISC-V GNU Toolchain'in yüklenmesi

İşlemciyi programlamak için uygun bir derleyici toolchain'ine ihtiyaç vardır. Potato için gerekli olan sürüm "rv32i" sürümüdür. Bu komut seti kümesi, RISCV mimarili işlemcilerin en temel halidir. RISCV GNU toolchain'in reposundan kaynağı indirmek için kökte açılan terminale sıradaki komutlar girilebilir [19].

```
$git clone --recursive https://github.com/riscv/riscv-gnu-toolchain
```

Dosyanın boyutu biraz büyük olduğu için internet hızına bağlı olarak biraz uzun sürebilir. Buna alternatif olarak aşağıdaki komutlar da kullanılabilir.

```
$ git clone https://github.com/riscv/riscv-gnu-toolchain

$ cd riscv-gnu-toolchain

$ git submodule update --init –recursive
```

Toolchain'i kurabilmek için birkaç standart paketin önceden kurulması gerekiyor. Bunun için aşağıdaki komut yeterli olacaktır.

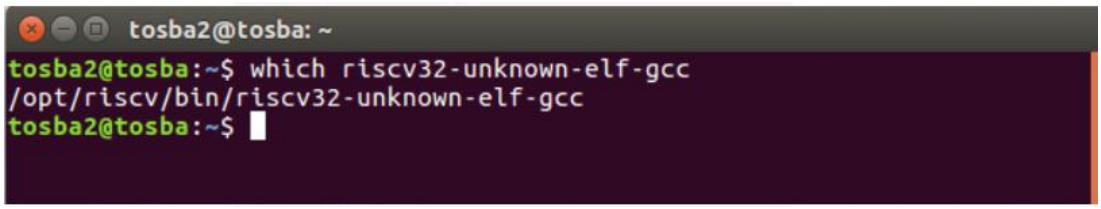
```
$ sudo apt-get install autoconf automake autotools-dev curl libmpc-dev libmpfr-dev
libgmp-dev gawk build-essential bison flex texinfo gperf libtool patchutils bc zlib1g-dev
libexpat-dev
```

Çapraz derleyiciyi kurmak için bir dizin seçilir. Bu projede /Opt/riscv seçilmiştir, şimdi PATH'e /opt/riscv/bin yolu eklenir. Ardından, aşağıdaki komut çalıştırılır:

```
$./configure --prefix=/opt/riscv-toolchain --with-abi=ilp32 --with-arch=rv32i

$ make
```

Bu kısım oldukça uzun sürebilir. İşlem tamamladıktan sonra aşağıdaki koda Şekil 3.1'deki geri bildirim alınıyorsa işlem tamamlanmış demektir.

```
$ which risc32-unknown-elf-gcc
```



Şekil 3.1: Derleyicinin tepki vermesi.

3.2 İşlemcinin Gerçeklenmesi

3.2.1 Potato processor'un git'deki reposundan indirilmesi

İlk olarak işletim sisteminde güncellenmesi gereken paketleri güncelleştirmek için Ubuntu'da kökte açılmış bir terminale aşağıdaki komutlar girilir:

```
$ sudo apt-get update
```

```
$ sudo apt-get upgrade
```

Ardından terminal üzerinden repo'dan gerekli kaynak kodlarını indirebilmek için "git" [20] kurulur. Bu işlem için gerekli terminal kodu aşağıdadır.

```
$ sudo apt-get install git
```

Şimdi Potato'nun kaynak kodlarını repo'dan [14] indirebiliriz. Bunun için terminale

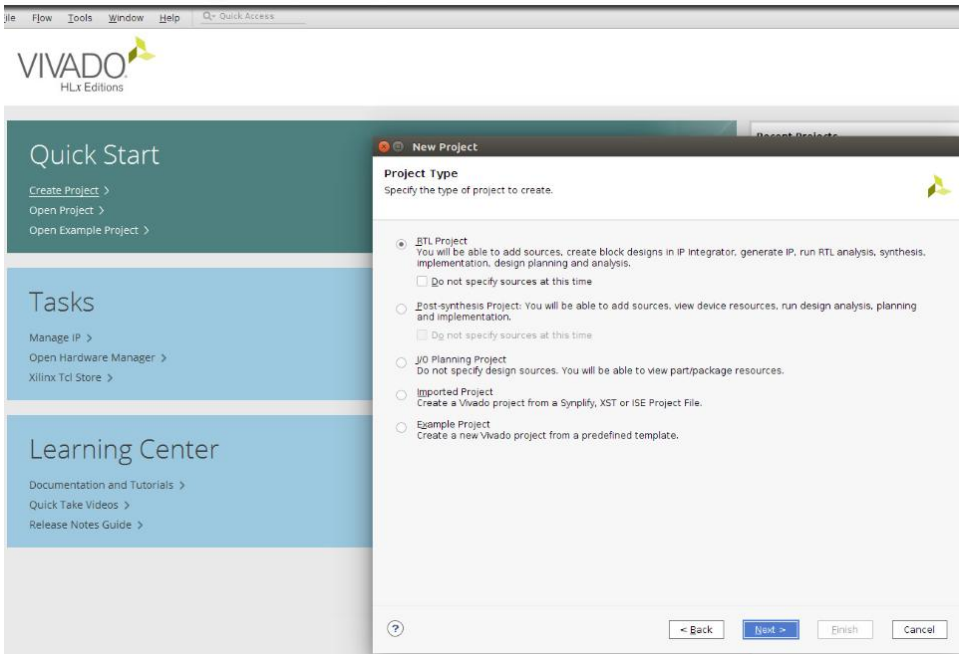
```
$git clone https://github.com/skordal/potato.git
```

kodu girilir. Artık Potato işlemcisinin kaynak kodlarına sahibiz. Bundan sonraki adımda, işlemcinin Vivado'da proje haline getirilmesini inceleyeceğiz.

3.2.2 Potato işlemcisinin Vivado'da proje haline getirilmesi

3.2.2.1 Vivado'da proje oluşturulması

Vivado'da yeni proje oluşturulur. Proje tipi ekranından "RTL Project" seçilir ve "Do not specify source at this time" kutucuğu işaretli ise işaret kaldırılır.

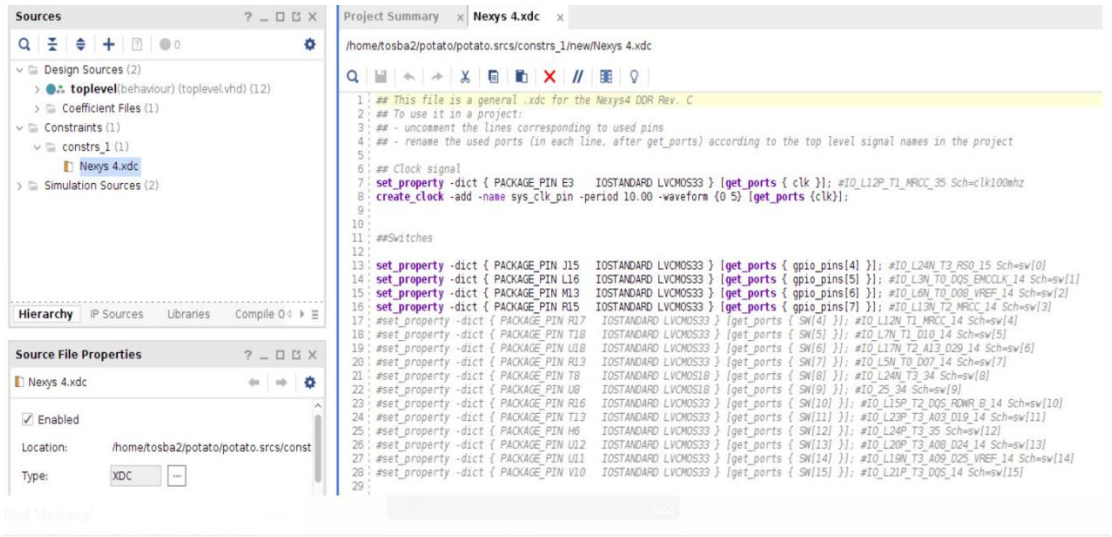


Şekil 3.2: Vivado project type ekranı.

3.2.2.2 Kaynak dosyaların projeye eklenmesi

Add sources ekranına daha önce bölüm 3.2.1 de indirdiğimiz dosyaların içinden src/, soc/ ve example/ dizinlerindeki dosyaların VHDL kodları design source olarak eklenir (tb_toplevel.vhd hariç). Projede example/ dizininde Xilinx Arty kartına göre

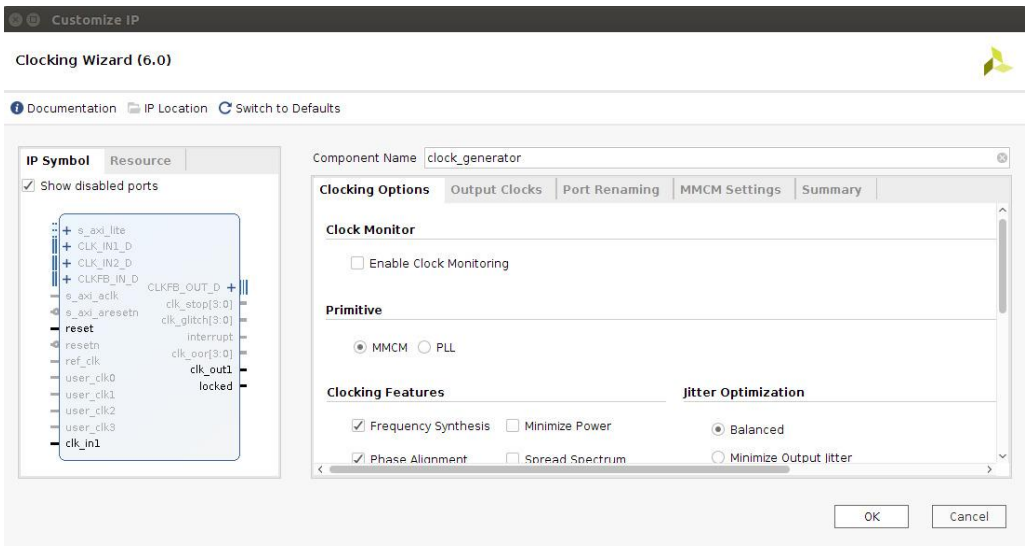
yazılmış bir constraint dosyası bulunmaktadır. Bu çalışmada Nexys 4 DDR kartı kullanıldığından bu dosyanın düzenlenmesi gerekir. Nexys 4 DDR için master constraint dosyası projeye eklenir. Bu dosyalara ek olarak projede eksik olan saat üreteci ve AEE ROM, IP block generator üzerinden tasarlanır.



Şekil 3.3: Nexys 4 DDR'ye göre hazırlanmış constraint dosyası.

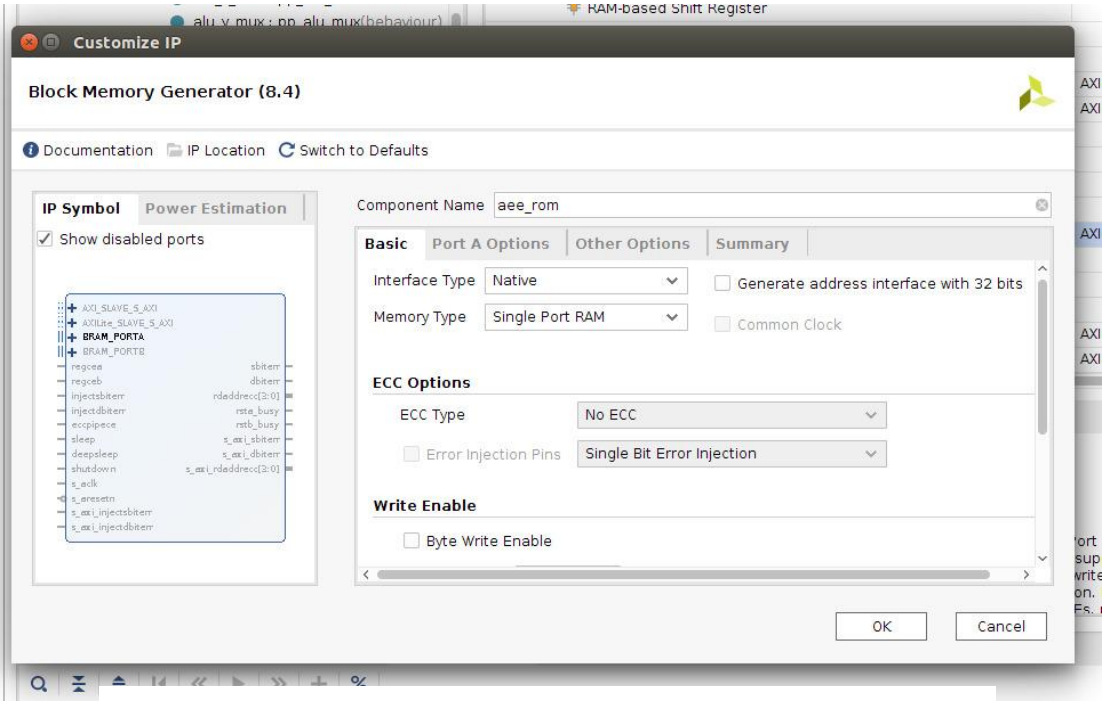
Saat üreteci:

Projeye bir adet Clocking Wizard IP'si eklenir. İsmi clock_generator olarak belirlenir. Clocking Options sekmesinde Frequency Synthesis ve Safe Clock Startup kutucukları işaretlenir. Giriş saatinin ismi clk yapılır. Çıkış saatleri system_clk ve timer_clk olmak üzere iki adettir. system_clk 50MHz, timer_clk 10MHz ayarlanır. Output Clocks sekmesinde "Enable Optional Inputs/Outputs" bölümünde reset ve locked sinyalleri işaretlenir. Reset sinyali active low olarak ayarlanır.



Şekil 3.4 : Vivado IP kataloğundan oluşturulan saat üreteci (clock generator).

Uygulama çalıştırma ortamı (AEE-Application Execution Environment):



Şekil 3.5: Block memory IP'si.

Projeye "Single-Port ROM" tipinde bir Block Memory IP'si (Şekil 3.5) eklenir. İsmi aee_rom olarak belirlenir. "Port A Options" sekmesinde width: 32, depth: 4096 seçilir. Enable pin seçeneği "Always enabled" olarak ayarlanır. Bu ROM içerisine FPGA ilk kez programlandığında çalışacak olan kod yüklenmelidir.

Bunun için yazılan kodun .coe uzantılı bir dosyasına ihtiyaç duyulmaktadır. potato-master/software/bootloader dizininde hazır bulunan Makefile, bu dosyayı üretmektedir. Terminal üzerinden bu dizine giderek

```
$ make all
```

komutu çalıştırılarak .coe uzantılı dosya üretilir. Eklenen ROM'un ayarlarında "Other Options" sekmesinde "Load Init File" işaretlenir ve oluşturulan "bootloader.coe" dosyası seçilir. Projede hazır gelen bootloader kodu, başlangıçta UART üzerinden bir açılış mesajı yollar ve bir dosya bekler. Bootloader programı, gönderilen dosyayı ana hafızaya yerleştirir ve çalışmaya oradan devam eder. Yani işlemcide çalışacak herhangi bir C kodu derlenip, .bin dosyası olarak ayarlanıp UART üzerinden gönderildiğinde işlemci o programı çalıştırır. Bu aşamadan sonra artık proje sentezlenerek FPGA programlanabilir.

3.3 Minicom

Minicom, USB üzerinden alınan veriyi görebilmek ve veri gönderebilmek için kullanılan bir Serial COM uygulamasıdır [21]. Bu uygulamayı kurmak için terminale

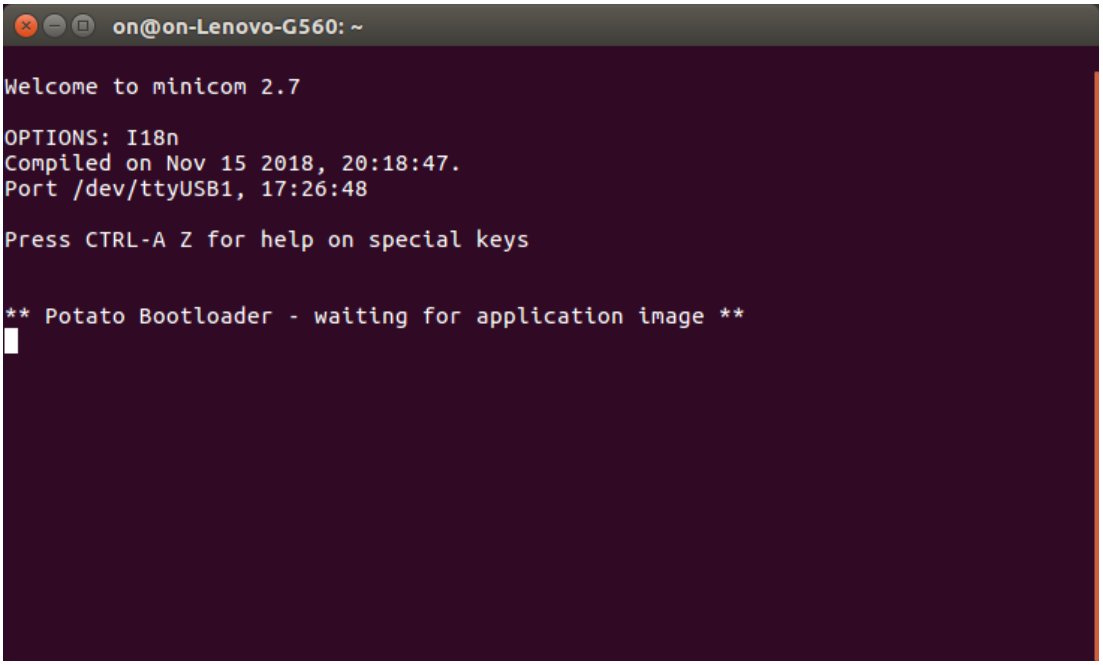
```
$ sudo apt-get install minicom
```

komutu yazılırsa, yükleme yapılacaktır.

3.4 İşlemcinin Çalıştığını Test Etmek İçin Gerçekleştirilen Uygulamalar

3.4.1 Basit hello word uygulaması

Öncelikle Minicom açılır ve uygun portun dinlenmesi ayarlanır, bu projede **/dev/ttyUSB1**. FPGA'da Potato gerçeklendikten sonra, Potato atılacak uygulamayı beklerken Minicom Şekil 3.6'daki gibi gözükür.



Şekil 3.6 : Potato gelecek uygulamayı bekliyor.

Önce Potato tasarımcısı tarafından örnek olarak hazırlanmış "Hello World" uygulamasını derleyip Potato'ya gönderiyoruz. Bu işlem için daha önce 3.2.1 bölümünde bahsi geçen repo'dan indirilmiş olan dosyaların içinde Software/Hello dizininde olan bir terminale sırayla aşağıdaki komutlar girilir.

```
$ make all
```

```
$ cat hello.bin /dev/zero | head -c128k | pv -s 128k -L 14400 > /dev/ttyUSB1
```

Bu komutlardan ilki dosyanın içinde olan makefile'yi çalıştırarak programın Potato için derlenmesini sağlar. Ardından **cat** komutu ile 128 kB boyutundaki program seriport üzerinden Potato'ya aktarılır ve burada çalıştırılır. Şekil 3.7'de gösterilmiş olan uygulamada "uart.h" kütüphanesinde bulanan "uart_tx_string" fonksiyonu kullanılarak string, uart üzerinden bilgisayara bağlı olduğu porta gönderilir.

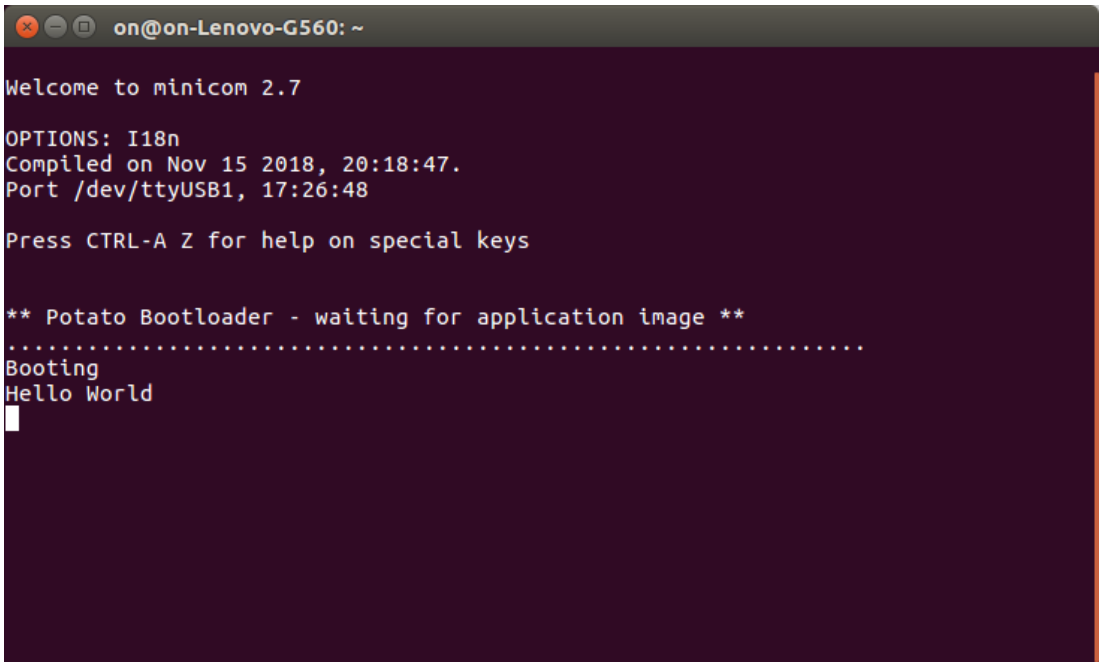
```c
#include <stdint.h>

#include "platform.h"
#include "uart.h"

static struct uart uart0;

void exception_handler(uint32_t cause, void * epc, void * regbase)
{···
}

int main(void)
{
    uart_initialize(&uart0, (volatile void *) PLATFORM_UART0_BASE);
    uart_set_divisor(&uart0, uart_baud2divisor(115200, PLATFORM_SYSCLK_FREQ));
    uart_tx_string(&uart0, "Hello world\n\r");

    return 0;
}
```

Şekil 3.7: Hello world uygulaması

```
on@on-Lenovo-G560: ~/potato/software/hello
on@on-Lenovo-G560:~/potato/software/hello$ make all
riscv32-unknown-elf-gcc -c -o main.o -march=rv32i -Wall -Os -fomit-frame-pointer
 -ffreestanding -fno-builtin -I../.. -I../../libsoc -std=gnu99 -Wall -Werror=imp
licit-function-declaration -ffunction-sections -fdata-sections main.c
riscv32-unknown-elf-gcc -c -o start.o -march=rv32i -Wall -Os -fomit-frame-pointe
r -ffreestanding -fno-builtin -I../.. -I../../libsoc -std=gnu99 -Wall -Werror=im
plicit-function-declaration -ffunction-sections -fdata-sections ../start.S
riscv32-unknown-elf-gcc -o hello.elf -march=rv32i -nostartfiles -L../libsoc -Wl,
-m,elf32lriscv --specs=nosys.specs -Wl,--no-relax -Wl,--gc-sections -Wl,-T../pot
ato.ld -Wl,--Map,hello.map main.o start.o
riscv32-unknown-elf-size hello.elf
   text    data     bss     dec     hex filename
    436       0       4     440     1b8 hello.elf
riscv32-unknown-elf-objcopy -j .text -j .data -j .rodata -O binary hello.elf hel
lo.bin
echo "memory_initialization_radix=16;" > hello.coe
echo "memory_initialization_vector=" >> hello.coe
hexdump -v -e '1/4 "%08x\n"' hello.bin >> hello.coe
echo ";" >> hello.coe
on@on-Lenovo-G560:~/potato/software/hello$ cat hello.bin /dev/zero | head -c128k
 | pv -s 128k -L 14400 > /dev/ttyUSB1
28,1KiB 0:00:02 [14,6KiB/s] [======>                          ] 21% ETA 0:00:07
```

Şekil 3.8: Potato'ya gönderilen Hello world programı.

Program çalışınca Minicom'un ekranında işlemciden gönderilmiş olan çıktı görülebilir.



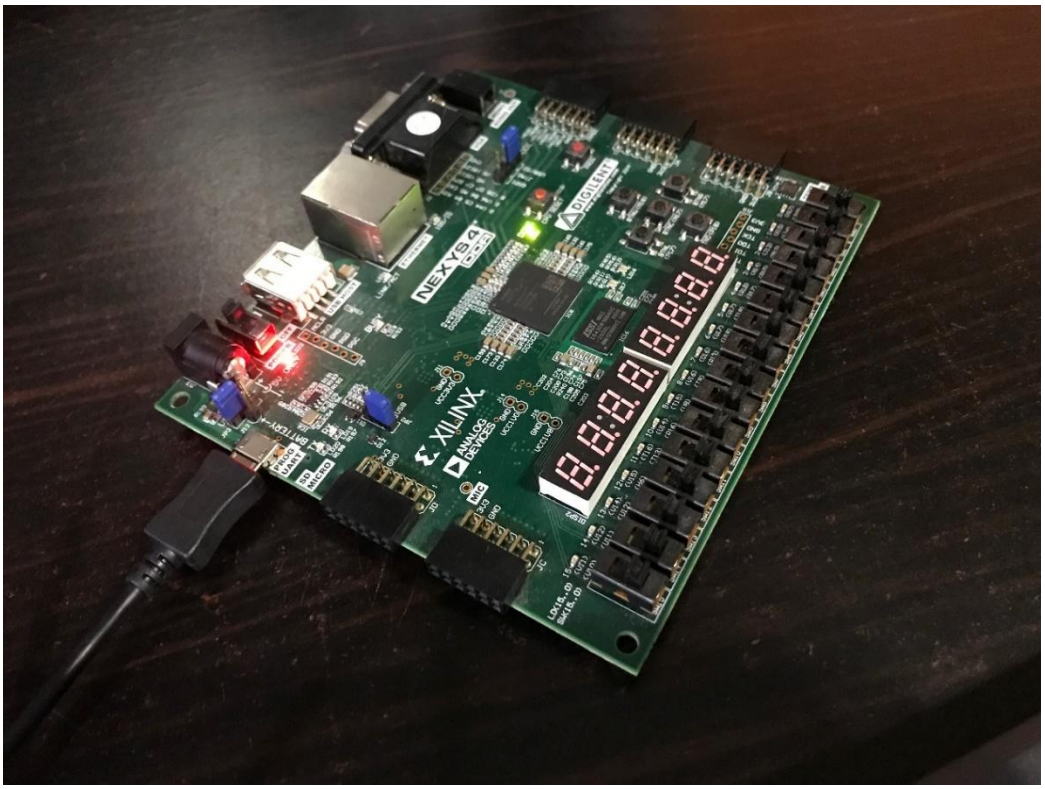
Şekil 3.9: Hello world programının Minicom üzerinden okunması

3.4.2 Led yakma uygulaması

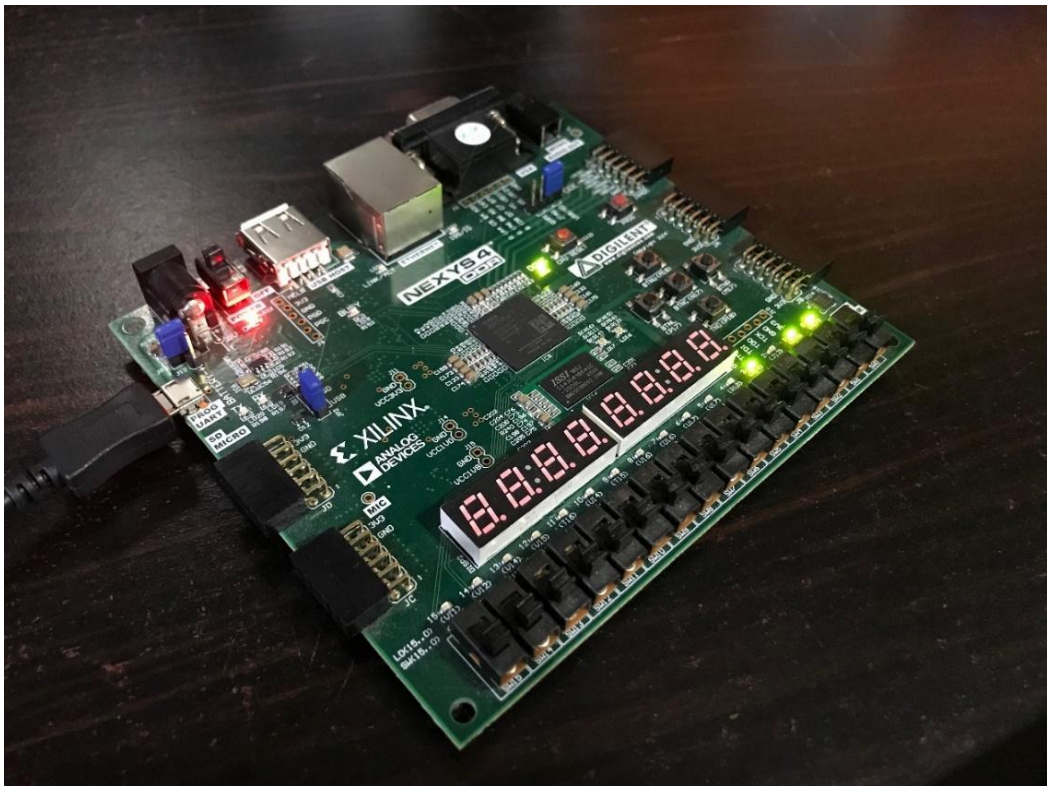
```
5   #include <stdint.h>
6
7   #include "platform.h"
8   #include "uart.h"
9   #include "gpio.h" // potato/libsoc'da, ledler gpioda tanımlanmış
10
11  struct gpio gpio0;
12
13  void exception_handler(uint32_t cause, void * epc, void * regbase)
14  {
15      // Not used in this application
16  }
17
18  int main(void)
19  {
20      gpio_initialize(&gpio0, (volatile void *) PLATFORM_GPIO_BASE);
21      gpio_set_direction(&gpio0, 0xF00);
22      gpio_set_output(&gpio0, 0xB00);
23
24      return 0;
```

Şekil 3.10: Led yakma uygulaması.

Programı Potato'ya gönderdiğimiz zaman daha önce constraint dosyasında tanımladığımız ledler yanar.



Şekil 3.11: Potato programı beklerken.



Şekil 3.12: Led yakma programı çıktısı.

3.4.3 Basit aritmetik işlemler

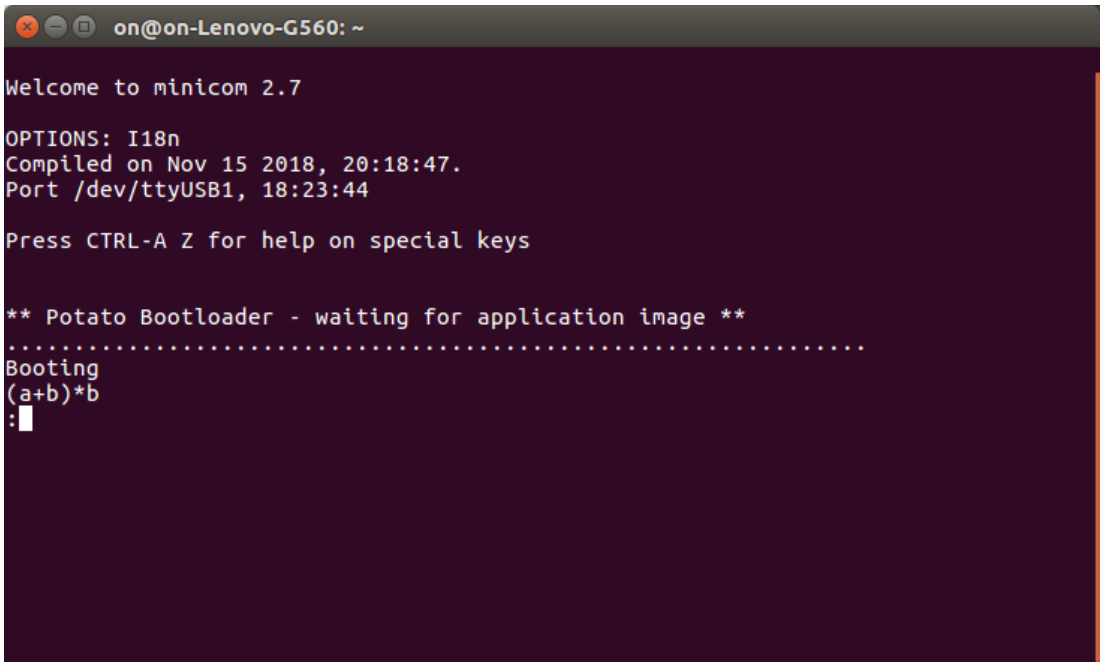
```
int main(void)
{
    uint8_t a, b, c, d, e;

    a=2;
    b=3;

    c=a+b;



    c=c+0x30; //1'den 9'a kadar sayıları terminalde göstermek için.
    d=d+0x30; // Çünkü minicom sayıları ascii deki karşılıklarına çevirerek
    e=e+0x30; // O şekilde ekranda gösteriyor.

    uart_initialize(&uart0, (volatile void *) PLATFORM_UART0_BASE);
    uart_set_divisor(&uart0, uart_baud2divisor(115200, PLATFORM_SYSCLK_FREQ));

    uart_tx_string(&uart0, "a+b=");
    uart_tx(&uart0,c);
```

Şekil 3.13: İşlemciyi test etmek için kullandığımız basit aritmetik uygulama.

```
on@on-Lenovo-G560: ~

Welcome to minicom 2.7

OPTIONS: I18n
Compiled on Nov 15 2018, 20:18:47.
Port /dev/ttyUSB1, 18:20:40

Press CTRL-A Z for help on special keys


** Potato Bootloader - waiting for application image **
............................................................
Booting
a+b=
5
```

Şekil 3.14: Yukarıdaki programın çıktısı.

Sayıların ekranda ASCII değerleri olarak Şekil 3.14'de gösterildiği gibi çıkmasından dolayı, büyük sayıların kolay okunabilmesini sağlamak için integer değerlerinin string'lere çevrilmesi için bir fonksiyon yazıldı.



Şekil 3.15: İşlem sonucunun terminalde ASCII değeri

":" karakterinin ASCII karşılığı 10 sayısıdır. Yani, biz Minicom'a işlemci üzerinden 10 değerini gönderdiğimiz zaman 10 sayısının ASCII'daki karakter karşılığı yazdırılıyor. Bu problemi aşmak için integer'leri string'e çeviren bir fonksiyon (Şekil 3.16) yazılmıştır.

```
149  char *uint_to_str(uint8_t sayi)
150  {
151
152      ret[0]='\0';
153      ret[1]='\0';
154      ret[2]='\0';
155      ret[3]='\0';
156      if(sayi < 10){
157          ret[0]=sayi+48;
158          ret[1]='\n';
159      }
160      else if (sayi<100){
161          ret[0]=sayi/10+48;
162          ret[1]=sayi%10+48;
163          ret[2]='\n';
164      }
165      else{
166          ret[0]=sayi/100+48;
167          ret[1]=(sayi/10)%10+48;
168          ret[2]=sayi%10+48;
169          ret[3]='\n';
170      }
171      return ret;
172
173  }
174
```

Şekil 3.16: Integer'den string'e çeviren fonksiyon

4. POTATO KULLANARAK KENAR ALGILAMANIN GERÇEKLENMESİ

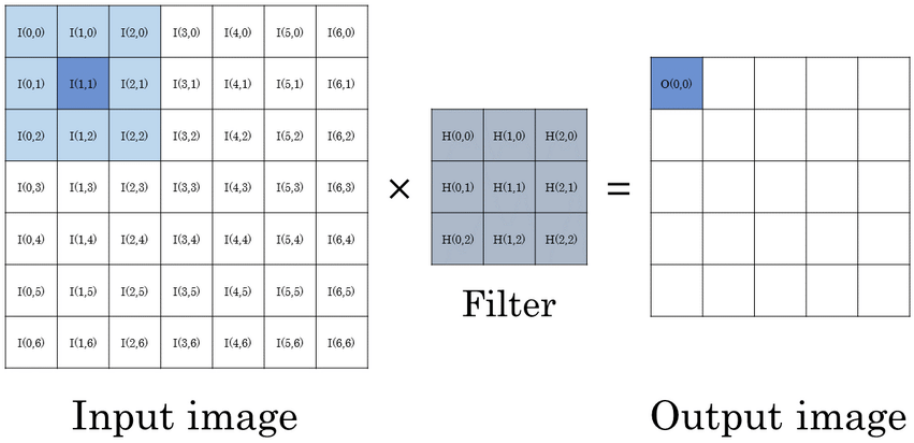
"Edge detection" veya "Kenar algılama" bizim bu projede test amaçlı kullanacağımız uygulamamızdır. Edge detection uygulaması bir görüntü işleme uygulamasıdır. Bu uygulamanın yaptığı iş aslında bir görüntünün sınırlarını kullandığı filtre yardımı ile belirleme ve çizme işlemidir. Görüntüyü matris biçiminde algılayan bilgisayar her bir piksele aldığı renge göre bir sayı değeri atar. Renk geçişlerinin yani sayı değişimlerinin ani olduğu yerleri kenar olarak algılar ve çizimde bu noktalara yoğunlaşır. Temel olarak çalışma prensibi bu şekildedir. Kullandığı filtrelere göre isim alabilir. Bu filtrelere:

- Prewitt Filtresi
- Sobel Filtresi
- Roberts Filtresi
- Laplacian Filtresi

  gibi filtreler örnek verilebilirler. Bu projede Laplacian filtreye yoğunlaşılmıştır.

4.1 Laplacian Filtre

Edge detection yöntemi görüntü işleme sisteminin önemli bir parçası olmuştur. Görüntüde tasvir edilen nesnelerin veya dokuların sınırlarını bulmak için kullanılan işlemdir. Bu sınırların konumlarını bilmek, görüntü geliştirme, tanıma, geri yükleme ve sıkıştırma sürecinde kritik öneme sahiptir. Görüntünün kenarları, insan görüş algısı için değerli bilgiler sağlayan görüntünün en önemli özellikleri arasında olarak kabul edilir. Gerçek görüntülerde edge detection yöntemini geliştirmek için önerilen farklı yöntemler vardır [22]. Edge detection yönteminde görüntüler büyüdükçe veriler de çok büyüdüğü için, görüntü işleme hızı zor bir problem haline gelir. Laplacian operatörü kenar tespiti için kullanılır.



Şekil 4.1: Konvolüsyon işlemi [23].

Laplacian operatörü ile yapılan kenar algılama işlemi basitçe bir matris konvolüsyon işlemidir.

4.2 Filtre İçin Yazılan C Kodu

```
15  static inline void my_uart_tx_string(struct uart * module, const char * string);
16  int main()
17  {
18
19      uart_initialize(&uart0, (volatile void *) PLATFORM_UART0_BASE);
20      uart_set_divisor(&uart0, uart_baud2divisor(115200, PLATFORM_SYSCLK_FREQ));
21
22      int kernel[3][3] =
23      {
24          {0, -1, 0},
25          {-1,  4, -1},
26          {0, -1, 0}
27      };
28      int x=0, y=0, j=0, i = 0, pixel = 0;
29      uint8_t pixel0 = 0;
30
```

Şekil 4.2: C kodu kısmı.

```
41      int img[64][64] ={
42          {134,132,132,127,134,148,145,99,62,71,75,75,75,82,91,95,98,99,98,102,102,102,101,101,103,103,103,100,10
43          {131,130,128,129,138,149,137,95,58,66,73,71,72,82,89,94,96,98,99,98,100,101,101,99,99,98,100,98,99,99,1
44          {130,131,129,132,143,143,134,96,59,66,75,71,70,80,88,93,96,97,100,98,99,99,99,97,101,98,98,99,98,103
45          {130,132,134,139,144,136,134,95,56,66,74,71,70,79,88,92,96,96,96,97,97,97,94,97,101,92,90,93,94,93,97,9
46          {134,134,141,146,140,134,134,94,55,66,72,71,70,78,87,90,93,95,93,94,95,97,95,97,95,91,87,87,86,86,92,98
47          {136,139,142,132,133,137,133,93,53,64,71,70,70,79,87,90,91,92,92,94,93,94,95,92,93,94,98,108,116,110,98
48          {135,144,128,109,131,139,136,94,52,63,70,70,71,79,86,90,92,93,95,94,91,95,95,99,116,123,134,155,169,168
49          {142,142,98,93,136,143,137,93,48,59,66,68,71,77,84,88,89,91,92,93,99,109,97,107,115,121,134,145,158,164
50          {150,123,61,85,140,144,135,90,46,57,67,66,69,75,81,83,88,89,89,97,89,93,94,103,106,111,119,136,155,166,
51          {141,89,45,88,139,145,133,89,45,57,68,68,68,76,81,83,86,86,95,89,76,85,94,99,105,107,114,126,150,167,17
52          {113,59,50,90,138,146,134,88,46,58,65,69,69,75,81,85,87,88,87,76,82,90,88,94,104,107,111,121,145,170,17
53          {74,52,54,89,134,145,135,87,47,61,65,68,69,74,82,85,86,90,77,78,87,83,87,98,108,114,114,119,139,154,163
54          {55,58,54,87,132,144,133,90,48,59,66,66,68,75,82,83,90,83,79,81,83,86,97,106,111,115,110,110,128,159,18
55          {58,61,56,87,132,141,133,90,48,57,67,68,69,73,80,84,106,71,76,80,84,92,104,111,111,108,94,117,162,184,1
56          {59,58,52,86,132,141,136,90,45,54,62,65,69,74,75,97,120,65,77,78,87,97,104,106,101,90,122,166,172,169,1
57          {58,61,57,83,133,145,135,88,42,53,66,65,67,72,70,128,115,64,77,81,92,94,97,97,88,130,158,154,160,167,17
58          {64,66,62,84,132,147,140,91,46,58,67,69,67,75,74,142,103,63,74,83,88,91,94,90,124,143,144,155,165,168,1
59          {67,65,63,87,132,149,146,95,47,59,68,67,68,72,74,151,95,67,70,77,86,88,91,118,124,124,143,156,155,159,1
60          {67,65,64,89,136,153,148,96,46,58,66,63,65,68,78,169,106,73,72,77,84,90,115,117,112,125,136,149,148,149
61          {65,64,62,88,137,153,146,96,47,59,66,65,65,65,80,173,124,92,68,74,86,107,112,104,121,121,123,137,146,15
62          {63,61,61,89,138,151,146,98,47,59,65,64,65,61,84,171,134,105,72,77,104,103,99,114,112,118,122,132,138,1
63          {65,64,64,91,137,152,148,100,45,58,64,63,64,57,88,185,148,122,88,91,100,97,107,105,104,115,119,109,101,
64          {67,66,67,94,140,154,149,99,44,56,65,65,65,58,78,191,160,109,94,83,89,106,102,98,104,108,105,96,73,61,7
65          {71,66,66,94,139,152,148,101,45,56,64,65,65,62,61,181,169,103,100,80,100,98,96,102,106,98,90,72,51,42,7
66          {73,68,66,92,139,151,147,103,45,56,63,65,64,66,55,142,188,132,95,94,96,90,95,98,92,77,52,38,23,51,51,60
```

Şekil 4.3: C kodu görsel matrisi.

```
109     for (y = 0; y < 64; y++)
110     {
111         for (x = 0; x < 64; x++)
112         {
113             pixel = 0;
114             // Apply convolution
115             for (j = 0; j < 3; j++)
116             {
117                 for (i = 0; i < 3; i++)
118                 {
119                     pixel += img[x+i][y+j] * kernel[i][j];
120                 }
121             }
122 
123 
124             if(pixel>255)
125             {
126                 pixel=255;
127             }
128 
129             if(pixel<=0)
130             {
131                 pixel=0;
132             }
133 
134             pixel0 = pixel;
135 
136 
137             while(!uart_tx_ready(&uart0)){};
138             my_uart_tx_string(&uart0,uint_to_str(pixel0));
139 
140         }
141     }
142     while(!uart_tx_fifo_empty(&uart0)){};
143 
144     return 0;
145 }
```

Şekil 4.4: C kodu konvolüsyon işlemi.

Şekil 4.5: Program girdisi ve çıktısı.

Şekil 4.2’de algoritma için gerekli olan Laplacian operatörü ve konvolüsyon için gerekli sayılar tanımlanmıştır.

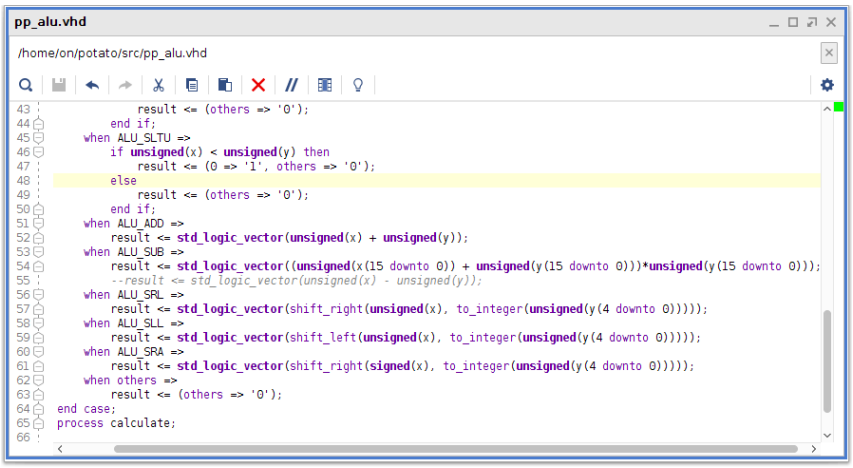
Şekil 4.5’de görülen görüntü, Şekil 4.3’de görüleceği şekilde piksel piksel iki boyutlu dizi şeklinde tanımlanmıştır.

Şekil 4.4’de konvolüsyon işleminin C programlama dilinde yazılmış halini görüyoruz. Algoritma her bir çıkan piksel değerini UART üzerinden bilgisayara gönderiyor ve buradan Minicom üzerinden pikseller tek tek bir dosyaya yazılıyor. Dosyaya yazılan piksel değerleri daha sonra Python dilinde yazılmış bir kod ile işlenerek Şekil 4.5’deki çıktı elde ediliyor.

5. UYGULAMAYA ÖZEL KOMUTUN İŞLEMCİYE EKLENMESİ

5.1 Aritmetik Mantık Biriminin Değiştirilmesi

Çarpma işleminin işlemciye eklenebilmesi için uygulamada kullanılmayan bir komut bulunması gerekmektedir. Bu proje için, filtre kodunun assembly koduna bakılarak kullanılmadığı tespit edilen çıkarma komutu seçilmiştir. Çıkarma komutu, aritmetik mantık biriminde değişiklik yapılarak Şekil 5.1'de görülebileceği gibi çarpma komutuna çevrilmiştir. Bu noktadan sonra işlemci, çıkarma olarak gelen komut yerine çarpma işlemi yapacaktır. Burada atlanmaması gereken nokta iki otuz iki bitlik sayıyı çarparsak sonuç altmış dört bit olacaktır ve işlemcimiz otuz iki bitlik bir işlemci olduğu için bu durum sorun çıkaracaktır. Bunu engellemek için çarpılan



Şekil 5.1: Değiştirilmiş aritmetik mantık birimi.

sayılar on altı bitlik olacak şekilde seçilmiştir. Uygulamada altmış dört bit büyüklüğünde sayılar kullanılmayacağı için de bu durum sorun teşkil etmemektedir.

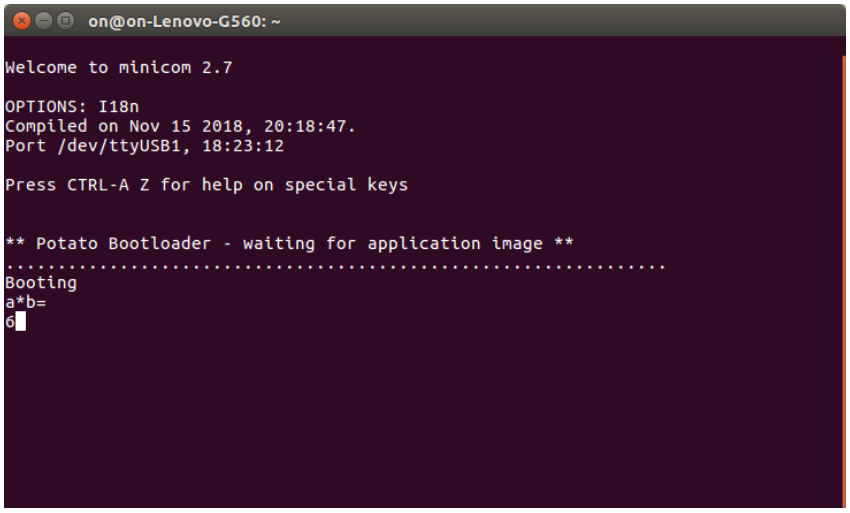
5.2 Komut Eklenmiş İşlemcinin Basit İşlemlerde Denenmesi

İşlemcinin yeni halinin denenebilmesi için öncelikle basit çarpma işlemleri yapıldı. Şekil 5.2'de görülebileceği gibi öncelikle volatile tip tanımlayıcısının kullanılması gereklidir, aksi takdirde a ve b değişkenleri başka yerlerde kullanılmadığı için derleyici kodu derlerken c değişkeninin sonucunu 1 olarak belirleyip c değişkenine bu sonucu yükler.

```
volatile uint8_t a, b, c;

a=3;
b=2;
c=a-b;
```

Şekil 5.2: Basit çarpma işleminin yeni işlemciye göre yazılması.

Çıktı Şekil 5.3'de görüleceği üzere doğru sonuç vermiştir. Çıkarma komutu değiştirilerek çarpma komutu yapar hale gelmiştir.



Şekil 5.3: Çarpma işleminin çıktısı.

5.3 İşlemcide Filtre Kodunun gerçeklenmesi

Filtre kodu işlemcinin yeni halinde test edilirken derleyicinin iç yapısı tam olarak bilinmediğinden dolayı derleyici bu adımı çıkarma, yani yeni çarpma komutunu değil optimize ettiği halini kullanıyor. Bu problemi aşmak için satır içi makine dili (Inline Assembly) kullanılmaya çalışıldı. Ancak, GCC ile çalışan Inline Assembly ile yazılmış olan kod, bilgisayar üzerinde çalışırken, kodun yeni işlemciye göre yazılmış hali RISCV GNU derleyicisinde derlenmeye çalışılırken hata vermiştir. Süremizin kısıtlı olması sebebiyle bu aşama sonuca ulaştırılamamıştır.

5. GERÇEKÇİ KISITLAR VE SONUÇ

5.1 Tasarım Kısıtları

Edge Detection uygulamasını gerçeklerken görüntü dosyasını işlemciye yükleme, işlemcinin bu dosyayı tanıması ve üzerinde işlem yapması proje için ekstra bir yük oluşturuyordu. Bu problemi çözmek için görüntü dosyasını işlemciye yüklemek yerine, dosya matris haline getirilip işlemciye atılacak C kodu içinde tanımlandı. Bu sayede görüntü dosyasını işlemciye aktarmak ile zaman kaybedilmedi.

5.2 Maliyet

RISC-V Potato işlemcisinin açık kaynak kodlu olması, Vivado programının ve Ubuntu işletim sisteminin ücretsiz olması maliyeti büyük ölçüde düşüren etkenlerdir. Projenin maliyetini sadece FPGA kartı oluşturmuştur.

5.3 Sonuç

FPGA ve RISC-V üzerinde daha önce yapılan çalışmalar, FPGA üzerinde RISC-V işlemcilerinin kurulabildiğini ve bu işlemciler ile başarılı bir şekilde uygulamaların gerçekleştirilebildiğini ispatlamıştı. Bu çalışmada ise RISC-V Potato işlemcisinin komut setini uygulamaya özel olarak başarılı bir şekilde değiştirilmesi sağlandı.

5.4 Geleceğe Yönelik Öneriler

RISC-V işlemcilerinin kullanımının artması ve bu işlemcilerinin komut setlerinin uygulamaya özel genişletilebilmesi sayesinde yakın bir gelecekte uygulamaya özel üretilmiş ve hızlı çalışabilen işlemciler çok daha düşük maliyetlerle tasarlanabileceklerdir. Bu elektronik aletlerin ulaşılabilmesi ve geliştirilebilmesi açısından önemli bir etken olabilir.

KAYNAKLAR

[1] R.-V. Foundation, "RISC-V Cores and SoC," RISC-V, 05 2017. [Online]. Available: https://riscv.org/risc-v-cores/#. [Accessed 28 12 2018].

[2] Sürer, B. (2019) Implementation of a SOC by Using LowRISC Processor on an FPGA For Image Filtering Application, İstanbul Teknik Üniversitesi, Elektrik Elektronik Fakültesi, İstanbul.

[3] **Altameemi**, A. A., & Bergmann, N. W. (2016, December). Enhancing FPGA softcore processors for digital signal processing applications. In 2016 Sixth International Symposium on Embedded Computing and System Design (ISED) (pp. 294-298). IEEE.

[4] **Güngör C. B., Öndeş Y., Sarı T. T. and Uçkun B.** , "Instruction Set Extension of Some Processors for Secure IoT Implementation," Istanbul, 2018.

[5] **Osborne, A.** (1980), *An Introduction to Microcomputers Volume 1: Basic Concepts*, Osborne-McGraw Hill Berkeley California USA, pp. 116–126

[6] **Wikipedia** ,"VHDL"**,** https://en.wikipedia.org/wiki/VHDL>, erişim tarihi 03.01.2020.

[7] **J. B. Evans**, "Efficient FIR filter architectures suitable for FPGA implementation," in *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 41, no. 7, pp. 490-493, July 1994. doi: 10.1109/82.298385

[8] **Xilinx,** "What is FPGA"**,** https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html, erişim tarihi 03.01.2020.

[9] Streaming Architecture for Large-Scale Quantized Neural Networks on an FPGA-Based Dataflow Platform - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Image-convolution-with-an-input-image-of-size-7-7-and-a-filter-kernel-of-size-3-3_fig1_318849314 erişim tarihi 08.09.2020.

[10] **Amara, A., Amiel, F., & Ea, T.** (2006). *FPGA vs. ASIC for low power applications. Microelectronics Journal, 37(8), 669–677.* doi:10.1016/j.mejo.2005.11.003

[11] **Digilent, Nexys 4,** https://reference.digilentinc.com/reference/programmable-logic/nexys-4-ddr/start erişim tarihi 06.09.2020.

[12] **Xilinx,** 2017. Vivado Design Suite User Guide.

[13] **Kanhiroth, V., & Vivek, J.** (2017). Embedded processors on FPGA: Hard-core vs Soft-core.

[14] **Github Potato,** https://github.com/skordal/potato, erişim tarihi 03.01.2020.

[15] **Da Silva, E. A. B., & Mendonça, G. V.** (2005). *Digital Image Processing. The Electrical Engineering Handbook, 891–910.* doi:10.1016/b978-012170960-0/50064-5

[16] **Vincent, O. R., & Folorunso, O.** (2009, June). A descriptive algorithm for sobel image edge detection. In *Proceedings of Informing Science & IT Education Conference (InSITE)* (Vol. 40, pp. 97-107). California: Informing Science Institute.

[17] **Gupta, S., & Mazumdar, S. G.** (2013). Sobel edge detection algorithm. *International journal of computer science and management Research*, *2*(2), 1578-1583.

[18] **Digilent,** https://store.digilentinc.com/digilent-adept-2-download-only/, erişim tarihi 08.01.2020.

[19] **Github, "**RISCV Tools**" ,**https://github.com/riscv/riscv-gnu-toolchain, erişim tarihi 06.01.2020.

[20] **"Linux"** , https://git-scm.com/download/linux, erişim tarihi 08.01.2020.

[21] **Ubuntu,** "Minicom"**,** https://help.ubuntu.com/community/Minicom, erişim tarihi 07.01.2020.

[22] **Gonzalez, C.I., Melin, P., Castro, J.R. et al. Soft Comput** (2016) 20: 773. https://doi.org/10.1007/s00500-014-1541-0

[23] Streaming Architecture for Large-Scale Quantized Neural Networks on an FPGA-Based Dataflow Platform - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Image-convolution-with-an-input-image-of-size-7-7-and-a-filter-kernel-of-size-3-3_fig1_318849314 erişim 08.01.2020

ÖZGEÇMİŞLER

Ad Soyad : Muhammet Hamidullah ERDEM
Doğum yeri ve tarihi : Sivas, 23.04.1996
E- mail : erdemmuha@itu.edu.tr
Muhammet Hamidullah ERDEM ilkokulu Konya Envar İlkokulu'nda, liseyi Konya Envar Lisesi'nde okumuştur. Kendisi şu an İstanbul Teknik Üniversitesi'nde Elektronik ve Haberleşme Mühendisliği bölümü son sınıf öğrencisidir. Stajlarını İTÜ bünyesindeki İTÜ ARC'de ve Kocaeli, Gebze'de bulunan TÜBİTAK BİLGEM'de tamamlamıştır.

Ad Soyad : Ömer Nevzat KÖSEBAY
Doğum yeri ve tarihi : Ankara, 08.03.1996
E- mail : kosebay15@itu.edu.tr
Ömer Nevzat KÖSEBAY ortaokulu İstek Barış Ortaokulu'nda, liseyi İstek Semiha Şakir Anadolu Lisesi'nde okumuştur. Kendisi şu an İstanbul Teknik Üniversitesi'nde Fizik Mühendisliği – Elektronik ve Haberleşme Mühendisliği Çift Anadal Programı son sınıf öğrencisidir. Stajlarını İstanbul Tuzla'da bulunan Kale Arge ve Kocaeli, Gebze'de bulunan ANKASYS şirketlerinde tamamlamıştır.

Ad Soyad : Emre SEVER
Doğum yeri ve tarihi : Konya, 05.02.1996
E- mail : severem@itu.edu.tr
Emre SEVER ilkokulu Çumra Şehit Koçak İlköğretim Okulu'nda tamamladıktan sonra liseyi Konya Selçuklu Anadolu Lisesi'nde tamamlamıştır. Kendisi şu an İstanbul Teknik Üniversitesi'nde Elektronik ve Haberleşme Mühendisliği bölümü son sınıf öğrencisidir.