MARMARA ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

FPGA ÜZERİNDE DÜŞÜK GÜÇ TÜKETİMİ ODAKLI SİSTEM GERÇEKLEMESİ VE PERFORMANS ANALİZİ

HAKAN GÜRBÜZ

YÜKSEK LİSANS TEZİ

Elektrik-Elektronik Mühendisliği Anabilim Dalı

Elektrik-Elektronik Mühendisliği Programı

DANIŞMAN

Prof. Dr. Hayriye KORKMAZ

İSTANBUL, 2024

MARMARA ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

FPGA ÜZERİNDE DÜŞÜK GÜÇ TÜKETİMİ ODAKLI SİSTEM GERÇEKLEMESİ VE PERFORMANS ANALİZİ

HAKAN GÜRBÜZ

YÜKSEK LİSANS TEZİ

Elektrik-Elektronik Mühendisliği Anabilim Dalı

Elektrik-Elektronik Mühendisliği Programı

DANIŞMAN

Prof. Dr. Hayriye KORKMAZ

İSTANBUL, 2024

ÖNSÖZ

Hem lisans dönemimde hem de yüksek lisans eğitimim boyunca vermiş olduğu değerli katkılarından dolayı danışman hocam Prof. Dr. Hayriye Korkmaz'a teşekkürü borç bilirim.

Tüm zorluklarda desteğini sürekli yanında hissettiğim sevgili eşime, beni yetiştirip bu günlere gelmemi sağlayan, desteğini hiçbir zaman esirgemeyen sevgili aileme ve tez boyunca teknik konularda yardımcı olan tüm dostlarıma teşekkürlerimi sunarım.

Ocak 2024 **Hakan GÜRBÜZ**

İÇİNDEKİLER

ÖNSÖZ ..... **i**

İÇİNDEKİLER ..... **ii**

ÖZET ..... **iv**

ABSTRACT ..... **vi**

SEMBOLLER ..... **viii**

KISALTMALAR ..... **ix**

ŞEKİL LİSTESİ ..... **x**

TABLO LİSTESİ ..... **xiii**

1. GİRİŞ **1**

2. TEZ KAPSAMINDA KULLANILAN TEKNOLOJİLER **2**

2.1. Soft CPU Kullanılarak Yapılan Benzer Çalışmalar 10

3. SİSTEM MİMARİSİ **12**

3.1. IP Tasarım 17

3.2. Microblaze Yapılandırma Seçenekleri 22

3.2.1. Minimum Alan 24

3.2.2. Maksimum Performans 26

3.2.3. Uygulamaya Özel Donanım Yapılandırması 27

3.3. Konvolüsyon İşlemini Yapan Donanımının Gerçeklenmesi (IP Tasarımı) ... 32

3.4. Konvolüsyon IP Tasarımının Doğrulanması 34

3.5. Bitstream Dosya Oluşturma 36

3.6. Donanımın SDK'ya Aktarılması 38

4. UYGULAMANIN GERÇEKLENMESİ **39**

5. GÜÇ ÖLÇÜMLERİ VE KARŞILAŞTIRMALAR **45**

5.1 FreeRTOS ve Bare-metal için İşlemci Kullanım Yüzdeleri..... 52

5.2. Konvolüsyon İşleminin FreeRTOS ile Gerçekleştirilmesi..... 56

6. SONUÇ 57

KAYNAKLAR 60

ÖZGEÇMİŞ .....

ÖZET

FPGA ÜZERİNDE DÜŞÜK GÜÇ TÜKETİMİ ODAKLI SİSTEM GERÇEKLEMESİ VE PERFORMANS ANALİZİ

Donanım hızlandırıcıları, özellikle yüksek hesaplama gücü gerektiren uygulamalarda işlemcilere göre daha iyi bir performans sergilemektedir. Diğer taraftan işlemciler üzerinde farklı işletim sistemlerinin çalışmasına imkân tanıdıklarından, sistemlere esneklik ve modülerlik kazandırmaktadır. Donanım hızlandırıcılar ile işlemcilerin görevleri paylaşmaları durumunda ise (donanım/yazılım ortak tasarımları, hardware/software co-desing) kaynak kullanımı ve güç tüketimi optimizasyonu gibi avantajlar elde edilebilmektedir. Günümüzde üretim yapan fabrikalardan, evlerde kullanılan cihazlara; trafikte seyreden araçlarda kullanılan donanımlardan çevre izleme-gözetlemede kullanılan aygıtlara kadar kadar pek çok alanda, nesnelerin algılayıcılar, yazılımlar ve ağ bağlantısı ile donatılarak veri toplama, işleme, karar verme ve bu bilgileri paylaşma yeteneğine sahip olması Internet of Things (IoT) kavramı ile mümkün hale gelmiştir. İnternete bağlanan nesnelerin sayısı, internet kullanımının hızla yaygınlaşması ve kolay erişilebilir olmasıyla birlikte her geçen gün artmaktadır. Bu birbirinden uzakta ve çoğunlukla da şebekeden beslenmeyen dağıtık nesneler için enerji, işlemci ve hafıza kapasitesi gibi kısıtlı kaynaklar bulunmaktadır ve bu nedenle IoT'yi de içine alan birçok uygulamada bu kısıtların göz önünde bulundurulması gerekmektedir. Düşük güç tüketimi göz önünde bulundurularak tasarlanan sistemler, cihazlarının ve sistemlerinin çevre üzerindeki olumsuz etkilerini azaltarak, daha sürdürülebilir ve çevre dostu bir IoT ekosistemi oluşturmaya da yardımcı olacaktır. Böyle bir sistem tasarımı için açık kaynak projelerden faydalanmakta çeşitli avantajlar sağlayacaktır. Çünkü günümüzde işlemci üreten firmalar ürünlerini yüksek lisanslama maliyetleriyle satmaktadırlar. Ancak açık kaynak işlemciler, şeffaf ve geliştirilmeye açık oldukları gibi maliyet açısından da avantaj sunmaktadır. Düşük güç tüketimi gibi ihtiyaçlar göz önünde bulundurularak, açık kaynak işlemciler üzerinde donanım tasarımı mümkündür. Ayrıca bu donanımlar için desteği bulunan birçok açık kaynak işletim sistemi de bulunmaktadır. Bu projelerden faydalanarak amaca uygun donanım ve yazılım birlikte kullanımı ile bütün bir sistem

tasarımı gerçekleştirilebilir. Bu tez kapsamında düşük güç tüketimi sağlayan bir görüntü işleme uygulaması senaryosuna yönelik donanım/yazılım ortak tasarımı, (hardware/software co-desing) sunulmuştur. Konvolüsyon filtreleri, özellikle görüntü işleme ve sinir ağı tabanlı görüntü sınıflandırma problemlerinde yaygın olarak kullanılmaktadır. Ancak bu filtre tasarımları, karmaşık matris işlemleri ve tekrarlı süreçler nedeniyle yüksek hesaplama gücü gerektirler. Bu tez çalışmasında, bu ve benzeri uygulamaların performansını artırmak amacıyla Microblaze tabanlı bir sistem tasarlanmış; Microblaze özel olarak yapılandırılarak, konvolüsyon filtresi donanımsal olarak gerçekleştirilmiştir. Daha sonra bu tasarım Nexys4 A7-100T FPGA geliştirme kiti üzerinde uygulanmıştır. Hem FreeRTOS hem de bare-metal yaklaşımlarıyla gerçekleştirilen konvolüsyon filtresinin yazılımsal ve donanımsal uygulama süreleri karşılaştırılmıştır. Ayrıca özel yapılandırma seçeneğinin donanım üzerindeki kaynak kullanımı ve güç tüketimine olan etkisi de detaylı bir şekilde analiz edilmiştir. Donanım ve yazılım tarafında yapılan iyileştirmelerin, güç tüketimini ve uygulama süresini azalttığı görülmüştür.

.

ABSTRACT

LOW POWER CONSUMPTION FOCUSED SYSTEM IMPLEMENTATION AND PERFORMANCE ANALYSIS ON FPGA

Hardware accelerators perform better than processors, especially in applications that require high computing power. On the other hand, since processors allow different operating systems to run, they provide flexibility and modularity to systems. If hardware accelerators and processors share tasks (hardware/software co-design), advantages such as resource usage and power consumption optimization can be achieved. Today, in different areas such as industrial applications, home apliances, advanced driver assistance systems and environmental monitoring and surveillance, the concept of Internet of Things (IoT) makes all possible for objects to have the ability to collect, process, decide and share data by equipping them with sensors, software and network connections. The number of objects connected to the internet is increasing day by day, as internet use becomes more widespread and easily accessible. There are limited resources such as energy, processor and memory capacity for these distributed objects that are far from each other and often not fed from the network, and therefore these constraints need to be taken into consideration in many applications including IoT. Systems designed with low power consumption in mind will also help create a more sustainable and environmental-friendly IoT ecosystem by reducing the negative impact of devices and systems on the environment. Using open source projects to design such a system will provide various advantages. Because today, companies that produce processors sell their products with high licensing costs. However, while open source processors are transparent and open to development, and they also offer cost advantages. Considering needs such as low power consumption, hardware design on open source processors is possible. There are also many open source operating systems that have support for these hardware. By taking advantage of these projects, a whole system can be designed by using hardware/ software co-design considering specific needs. Within the scope of this thesis, hardware/software co-design for an image processing application scenario that provides low power consumption is presented. Convolution filters are widely used especially in image processing and neural

network-based image classification problems. However, these filter designs require high computational power due to complex matrix operations and repetitive processes. In this thesis, a Microblaze-based system was designed to increase the performance of the application; Microblaze was specially configured and the convolution filter was implemented on the hardware. And then Nexys4 A7-100T FPGA development kit was programmed. Execution time for the convolution filter implemented with both FreeRTOS and bare-metal approaches were compared. In addition, the impact of the special configuration option on hardware resource usage and power consumption was analyzed in detail. It has been observed that improvements made on the hardware and software side reduce power consumption and execution time.

SEMBOLLER

S (Saniye) **:** Saniye zaman birimi

W (Watt) **:** Watt güç birimi

KISALTMALAR

ASIC **:** Application-Specific Integrated Circuit
Block RAM **:** Blok Random Access Memory
CISC **:** Complex Instruction Set Computer
CPU **:** Central Processing Unit
ELF **:** Executable and Linkable Format
DSP **:** Digital Signal Processing
FF **:** Flip Flop
FPGA **:** Field Programming Gate Array
HDF **:** Hardware Description File
IoT **:** Internet of Things
IP **:** Intellectual Property
ISE **:** Integrated Synthesis Environment
IDE **:** Integrated Development Environment
LUT **:** Look Up Table
MHz **:** Megahertz
RISC **:** Reduced Instruction Set Computer
RTL **:** Register Transfer Level
Xilinx SDK **:** Xilinx Software Development Kit

ŞEKİL LİSTESİ

Şekil 2.1. RTL Proje Oluşturulması 4

Şekil 2.2. Nexys A7-100T Kartının Seçilmesi 5

Şekil 2.3. RTL Proje Özet Bilgiler 5

Şekil 2.4. Blok Tasarım Oluşturma 6

Şekil 2.5. Projeye IP Ekleme 6

Şekil 2.6. Projeye Yeni Dosya Eklenmesi 7

Şekil 2.7. Giriş/Çıkış Tanımlama 8

Şekil 2.8. Blok Tasarıma Dosya Aktarılması 9

Şekil 3.1. Sadece Gerekli Modüller ile Yapılan Donanım Tasarımı 12

Şekil 3.2. Uygulamaya Özel Eklenen IP Bloklar ile Yapılan Donanım Tasarımı 13

Şekil 3.3. Donanım Tasarımı için Uygulama Adımları 14

Şekil 3.4. Yazılım Katmanı için Uygulama Adımları 15

Şekil 3.5. Uygulamaya Yönelik Yapılan IP Blok Tasarımı 17

Şekil 3.6. Vivado IP Blok Tasarımı 19

Şekil 3.7. Nexys A7-100T Kartı 20

Şekil 3.8. FPGA Alan Kullanımı 21

Şekil 3.9. Ön Tanımlı Yapılandırma Seçenekleri 22

Şekil 3.10. Güç Tüketim Raporu 25

Şekil 3.11. Uygulamaya Özel Yapılandırma 28

Şekil 3.12. Cache Yapılandırması 29

Şekil 3.13. Debug Yapılandırması 29

Şekil 3.14. Clock Yapılandırması 30

Şekil 3.15. 2D Konvolüsyon 32

Şekil 3.16. Piksel Verilerinin Ayrıştırılması 33

Şekil 3.17. Padding Uygulanmış Dizi 34
Şekil 3.18. Benzetim Girdi Matrisi 35
Şekil 3.19. Kernel Matrisi 35
Şekil 3.20. Konvolüsyon Birinci Adım 35
Şekil 3.21. Benzetim Çıktısı 36
Şekil 3.22. Validate Design 37
Şekil 3.23. HDL Wrapper 37
Şekil 3.24. Donanımın SDK'ya Aktarılması 38
Şekil 4.1. Uygulama Projesi Oluşturma 39
Şekil 4.2. Uygulama Proje Ayarları 40
Şekil 4.3. Donanım Tanımlama Dosyasının Aktarılması 41
Şekil 4.4. Uygulamanın Derlenmesi 41
Şekil 4.5. Derleme Çıktısı 42
Şekil 4.6. Bit Dosyasının FPGA'ya Gömülmesi 42
Şekil 4.7. Uygulamanın Donanıma Yüklenmesi 43
Şekil 4.8. Terminal Penceresi 44
Şekil 5.1. Tick Okuma Metodu 45
Şekil 5.2. Gerçekleme Süresi Üzerinde İterasyonun Etkisi 46
Şekil 5.3. Kernel Boyutunun İyileşme Oranına Etkisi 47
Şekil 5.4. Uygulama FreeRTOS Kodu 53
Şekil 5.5. FreeRTOS İşlemci Kullanım Oranı 54
Şekil 5.6. FreeRTOS Uygulama Çıktısı 54
Şekil 5.7. Bare-metal Uygulama Kodu 55
Şekil 5.8. Bare-metal Uygulama Çıktısı 55
Şekil 5.9. Bare-metal Uygulama İşlemci Kullanım Yüzdesi 55

Şekil 5.10. FreeRTOS ile Konvolüsyon İşlemi CPU Kullanımı 56

TABLO LİSTESİ

Tablo 3.1. Bileşen Açıklaması 20

Tablo 3.2. Ön Tanımlı Yapılandırma Seçenekleri 23

Tablo 3.3. "Minumum Area" Seçeneği Tercih Edildiğinde Kaynak Kullanımı 24

Tablo 3.4. "Minumum Area" Seçeneği Tercih Edildiğinde Güç Tüketimi 25

Tablo 3.5. "Maximum Performance" Seçeneği Tercih Edildiğinde Kaynak Kullanımı 26

Tablo 3.6. "Maximum Performance" Seçeneği Tercih Edildiğinde Güç Tüketim..... 27

Tablo 3.7. Özel Yapılandırma Seçeneği Tercih Edildiğinde Kaynak Kullanımı 30

Tablo 3.8. Özel Yapılandırma Güç Tüketimi 31

Tablo 3.9. Yapılandırma Seçenekleri için Güç Tüketimi ve Kaynak Kullanımı..... 31

Tablo 5.1. Farklı Görüntü Boyutu ve Kernel Matrisi için İyileşme Oranları 48

Tablo 5.2. 32x32'lik Görüntü ve 5x5'lik Kernel için Ortalama İyileşme Oranı Oranları 49

Tablo 5.3. "Branch Target" Modülünün Etkisi..... 50

Tablo 5.4. "Barrel Shifter" Modülünün Etkisi..... 51

Tablo 5.5. "Mul/Div" Modüllerinin Etkisi 51

Tablo 5.6. Bare-metal ve FreeRTOS için Bellek Kullanım Karşılaştırması 56

1. GİRİŞ

Hayatımızın birçok noktasında endüstriyel uygulamalardan, sağlık uygulamalarına, tarımdan akıllı ev ürünlerine kadar çok geniş bir alanda IoT uygulamaları yer almaya başlamıştır. IoT cihazlarında high-end ve low-end kısımlar bulunmaktadır. Low-end cihazlar genellikle enerji, işlemci ve hafıza gibi kaynaklar açısından sınırlıdır [1], [2]. Sensörler, eyleyiciler ve giyilebilir cihazlar bunlara örnek verilebilir. Bu tarz uygulamalarda batarya ömrünün uzatılabilmesi ve enerjide verimlilik sağlanabilmesi önemli bir konu haline gelmektedir. Ayrıca bu tarz uygulamalarda kullanılan aygıtların çoğu kendi üzerinde bir işlemci barındırmaktadır. Günümüzde işlemci üreten büyük firmalar, ürünlerini yüksek lisans ücretleriyle satmaktadırlar. Bu alanda yapılan uygulamaların çoğu tek bir amaca hizmet etmek için tasarlanmış, donanım ve yazılım bütünü olan sistemlerdir. Çözüm sağlanmak istenen sistemler karmaşık bir hal aldığında yapılan tasarımların değiştirilmesi hem maliyet hem de karmaşıklık getirmektedir. Bu tez çalışmasında bahsedilen güç tüketimi, maliyet ve esneklik gibi kriterler için optimum çözüm sağlamak amacıyla; açık kaynak kodlu projelerden ve yeniden yapılandırılabilen projelerden faydalanarak yazılım ve donanım bütünüyle düşük maliyetli, enerji verimliliği sağlayan bir sistem tasarımı hedeflenmiştir.

Literatürde IoT uygulamaları için düşük güç tüketim odaklı çalışmalar incelendiğinde daha çok RISC-V tabanlı işlemciler kullanıldığı görülmektedir [3], [4]. Ayrıca bu çalışmaların çoğu güç tüketimi odaklı değil daha çok görüntü işleme uygulamaları ve IoT cihazlar için güvenlik uygulamaları olduğu görülmüştür. Güç tüketimi odaklı yapılan çalışmalarda, 8 bitlik işlemci, kısıtlı sayıda çevre birimi olduğu görülmektedir. Buda sisteme eklenecek haberleşme modüllerini kısıtlamakta ve işletim sistemi çalıştırmayı kısıtlamaktadır [5].

2. TEZ KAPSAMINDA KULLANILAN TEKNOLOJİLER

İşlemciler, donanım tabanlı ve yazılım tabanlı olmak üzere iki kategoriye ayrılırlar [6]. İşlemci üreten firmalar tarafından, entegre devre şeklinde sunulan işlemcilere "hard cpu" denir. Soft CPU'lar ise FPGA gibi programlanabilir mantık cihazlarının kaynaklarını (LUT, FF, Blok RAM, DSP) kullanarak oluşturulan işlemcilerdir. FPGA ise, programlanabilir mantık bloklarını içeren ve kullanıcıların dijital devreleri özelleştirerek tasarlayabileceği esnek ve yüksek performanslı bir entegre devre türüdür. Microblaze, ARM Cortex-M1, Leon, RISC-V soft CPU'lara örnek gösterilebilir.

Günümüzde işlemcilerin sahip olduğu iki farklı komut seti mimarisi bulunmaktadır. Bunlar RISC ve CISC mimarileridir. CISC mimarisinde karmaşık işlemleri yerine getirmek için geniş bir komut seti bulunur. Bu mimaride donanım yapısı daha karmaşıktır. Intel x86, Motorola 68000 gibi işlemciler bu mimariyi kullanan işlemcilere örnek verilebilir. RISC mimarisi ise daha basit komutlar kullanır ve komutları daha hızlı yerine getirir [7]. Microblaze, Xilinx firması tarafından geliştirilen ve Xilinx FPGA cihazları için özelleştirilebilir, RISC mimarisine sahip bir mikro işlemci çekirdeğidir [8].

FPGA'ları programlamak için kullanılan farklı donanım tanımlama dilleri mevcuttur. En yaygın kullanıma sahip olan donanım tanımlama dilleri VDHL, Verilog ve SystemVerilog dilleridir. Bu tez çalışmasında donanım tasarımı için Verilog donanım tanımlama dili kullanılmıştır.

Verilog, dijital mantık devrelerini, sayaçları, çeviricileri, bellek yapılarını ve daha karmaşık sayısal sistemleri tanımlamak için kullanılan bir donanım tanımlama dilidir. Bu tanımlamalar, FPGA ve ASIC'ler gibi özelleştirilebilir ve programlanabilir donanımların tasarımında kullanılmaktadır [9], [10]. Verilog, donanım tasarım sürecindeki mantıksal işlemleri, aritmetik işlemleri, zamanlama, senkronizasyon ve donanım düzeyindeki diğer bileşenleri tanımlamak için kullanılan birçok yapıya ve söz dizimi özelliğine sahiptir. Bu tanımlamalar, sonrasında sentez araçları kullanılarak fiziksel donanıma yani gerçek devrelere dönüştürülebilmektedir.

Vivado, Xilinx firması tarafından geliştirilen bir FPGA ve SoC tasarım aracıdır. Vivado, FPGA ve SoC tasarımını kolaylaştırmak için tasarlanmış bir entegre geliştirme ortamıdır [11]. Xilinx 'in önceki tasarım aracı olan ISE' nin yerini almıştır. Vivado IP (Intellectual Property) Generator ise, kullanıcılara FPGA tasarımında sık kullanılan özel IP

çekirdeklerini kolayca oluşturma ve özelleştirme imkânı sağlayan bir araçtır. IP çekirdekleri, tasarımın temel yapı taşları olarak düşünülebilir ve önceden hazırlanmış fonksiyonları içerirler. Bu IP çekirdekleri, sayısal sinyal işleme (DSP), bellek kontrolörleri, haberleşme arayüzleri, protokol dönüştürücüler ve diğer karmaşık işlevleri içerebilir. Vivado IP Generator, kullanıcıların belirli bir işlevi yerine getiren IP çekirdeklerini grafik arayüzü ile seçmelerine, yapılandırmalarını ayrıca kendi tasarımlarını sisteme entegre etmelerine olanak tanır [12].

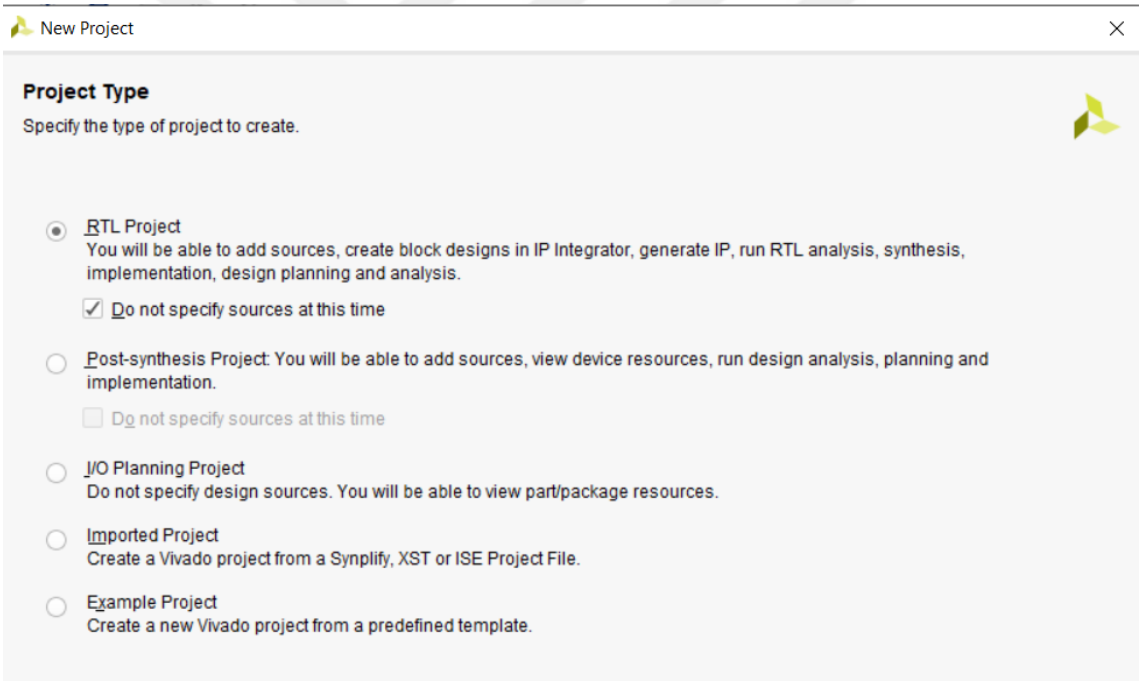
Xilinx SDK, Xilinx tabanlı işlemciler üzerinde gömülü yazılım uygulamaları geliştirmek için tümleşik geliştirme ortamıdır (Integrated Development Environment – IDE). Eclipse tabanlı açık kaynak standardını temel alır. Microblaze tabanlı işlemciler için geliştirilen yazılımın, Vivado 'da tasarlanan donanım üzerinde çalıştırılabilmesi için bu IDE'nin kullanılması gerekmektedir [13].

FreeRTOS (Free Real-Time Operating System), gerçek zamanlı gömülü sistemler için açık kaynaklı ve gerçek zamanlı bir işletim sistemidir. Özellikle düşük güç tüketimi, küçük boyut, hafiflik ve yüksek performans gibi gömülü sistemlerin ihtiyaçlarına yönelik tasarlanmıştır [14]. FreeRTOS, başta mikrodenetleyiciler olmak üzere çeşitli donanımlar üzerinde çalışabilir ve özellikle kısıtlı kaynaklara sahip olan sistemlerde etkin bir şekilde kullanılabilir. İşletim sistemi, birden fazla görev (task) çalıştırma, zaman dilimli çekirdek (preemptive kernel), hafıza yönetimi, kesme (interrupt) hizmet rutinleri ve haberleşme mekanizmaları gibi gerçek zamanlı uygulamalar için temel özellikleri sağlar. İşletim sistemin sağlamış olduğu bu özellikler aşağıda açıklanmıştır:

- Görev Yönetimi: Birden fazla görevi paralel olarak çalıştırmak için zaman paylaşımlı bir görev yönetim mekanizması sunar.
- Kesmeler (Interrupts) ve Çekirdek Desteği: Önemli kesme hizmet rutinleri için çekirdek desteği sağlar ve bu sayede sistemde kesmelerin yönetimini kolaylaştırır.
- Zaman Dilimli Çekirdek: Gerçek zamanlı sistemlerde önemli olan işlemlerin öncelik sırasına göre zamanlanması için zaman dilimli bir çekirdeği vardır.
- Kuyruk ve Yarıiletken Bellek Yönetimi: Haberleşme ve senkronizasyon için kuyruk yapıları ve yarıiletken bellek yönetimi sağlar.
- Hataya Dayanıklılık: Görevlerin kritik hatalara karşı korunmasını ve geri dönüş yapabilmesini sağlayacak mekanizmalar sunar.

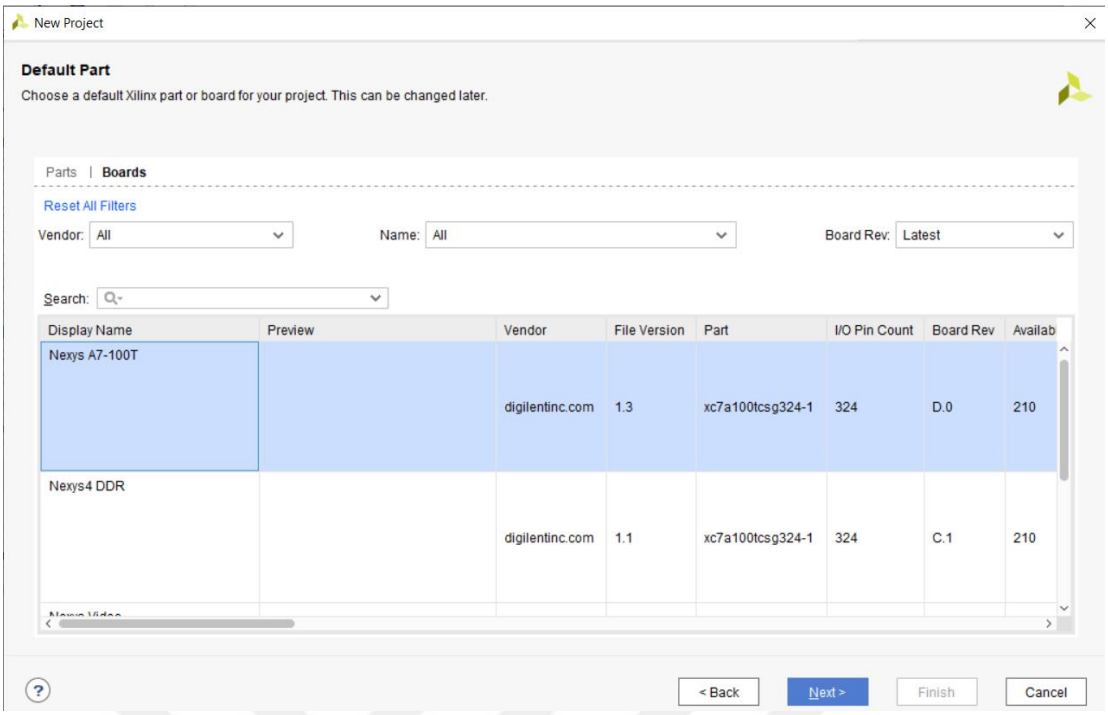
FreeRTOS, açık kaynaklı olması sayesinde ücretsizdir ve çeşitli mimarilerde ve mikrodenetleyicilerde kullanıma uygundur. Özellikle küçük boyutlu gömülü sistemlerde tercih edilen gerçek zamanlı bir işletim sistemidir [15].

Microblaze tabanlı bir sistem tasarımı yapmak için ilk olarak Vivado programında bir RTL projesi oluşturulur. Proje oluşturma aşamasında, kullanılacak geliştirme kartı seçilirse, kart üzerindeki birimler otomatik olarak bağlanacaktır ek olarak bir pin atamasının yapılması gerekmemektedir. Vivado 'da kullanılacak kart tanımlı değilse, Digilent'in web sayfasından ilgili dosyalar indirilip, Xilinx kurulum dosyalarının olduğu dosya dizininde; "SDK>2018.1>data>boards>board_files" içerisinde çıkarılmalıdır [16]. Ardından Vivado programı tekrardan başlatıldığında geliştirme kart listesi güncellenecektir. Şekil 2.1'de RTL Proje oluşturma penceresi gösterilmiştir.



Şekil 2.1. RTL Proje Oluşturulması

Tüm aşamalar bittikten sonra Şekil 2.2'te gösterildiği gibi kullanılan kart ve FPGA bilgilerinin bulunduğu bilgilendirme penceresi açılacaktır. "Finish" butonuna basılarak proje oluşturma işlemi tamamlanır.



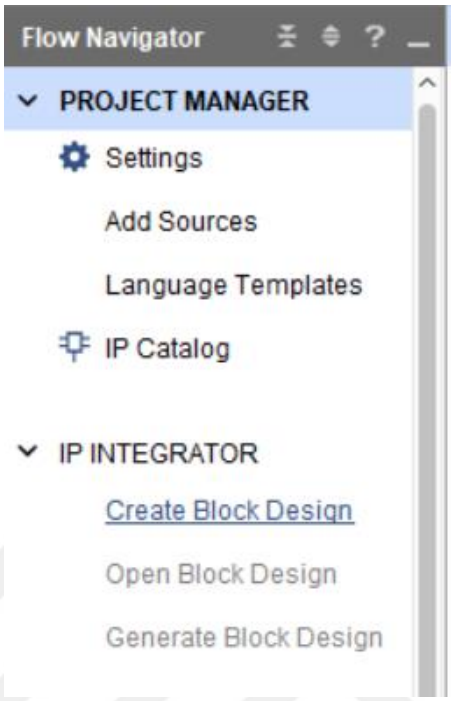
Şekil 2.2. Nexys A7-100T Kartının Seçilmesi

Şekil 2.3'de oluşturulan projeye ait özet bilgiler görülmektedir.



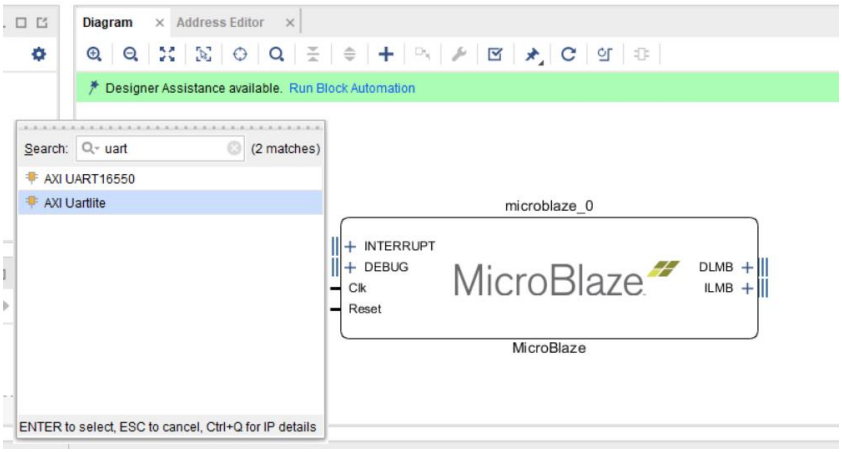
Şekil 2.3. RTL Proje Özet Bilgiler

Bundan sonraki adımda, IP tasarımı yapmak için Şekil 2.4'te gösterildiği gibi "create block design" butonuyla yeni bir blok tasarımı oluşturulur.



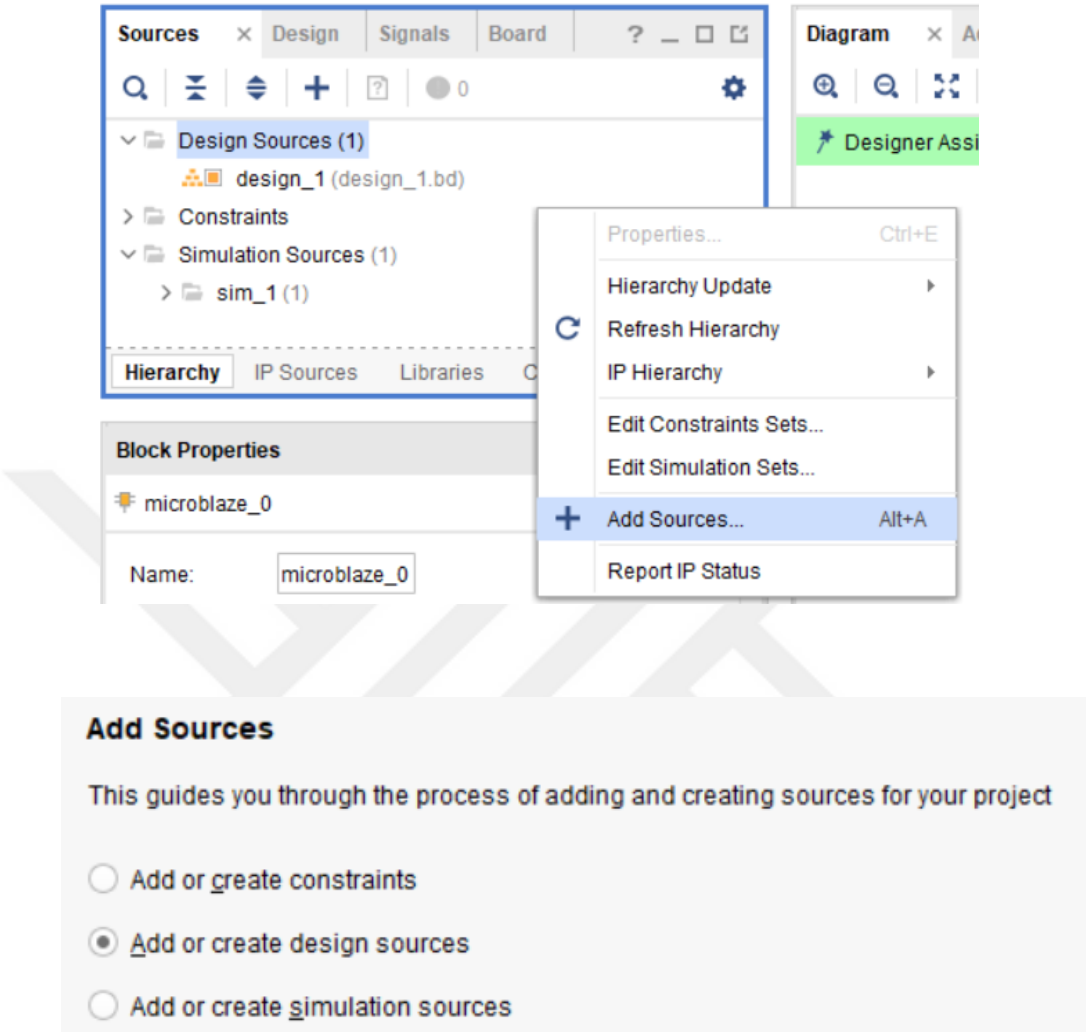
Şekil 2.4. Blok Tasarım Oluşturma

Proje oluşturulduktan sonra açılan pencerede bulunan "+" ikonuyla IP'ler projeye eklenebilir. Şekil 2.5'de örnek olarak IP'lerin tasarıma nasıl eklendiği gösterilmiştir.



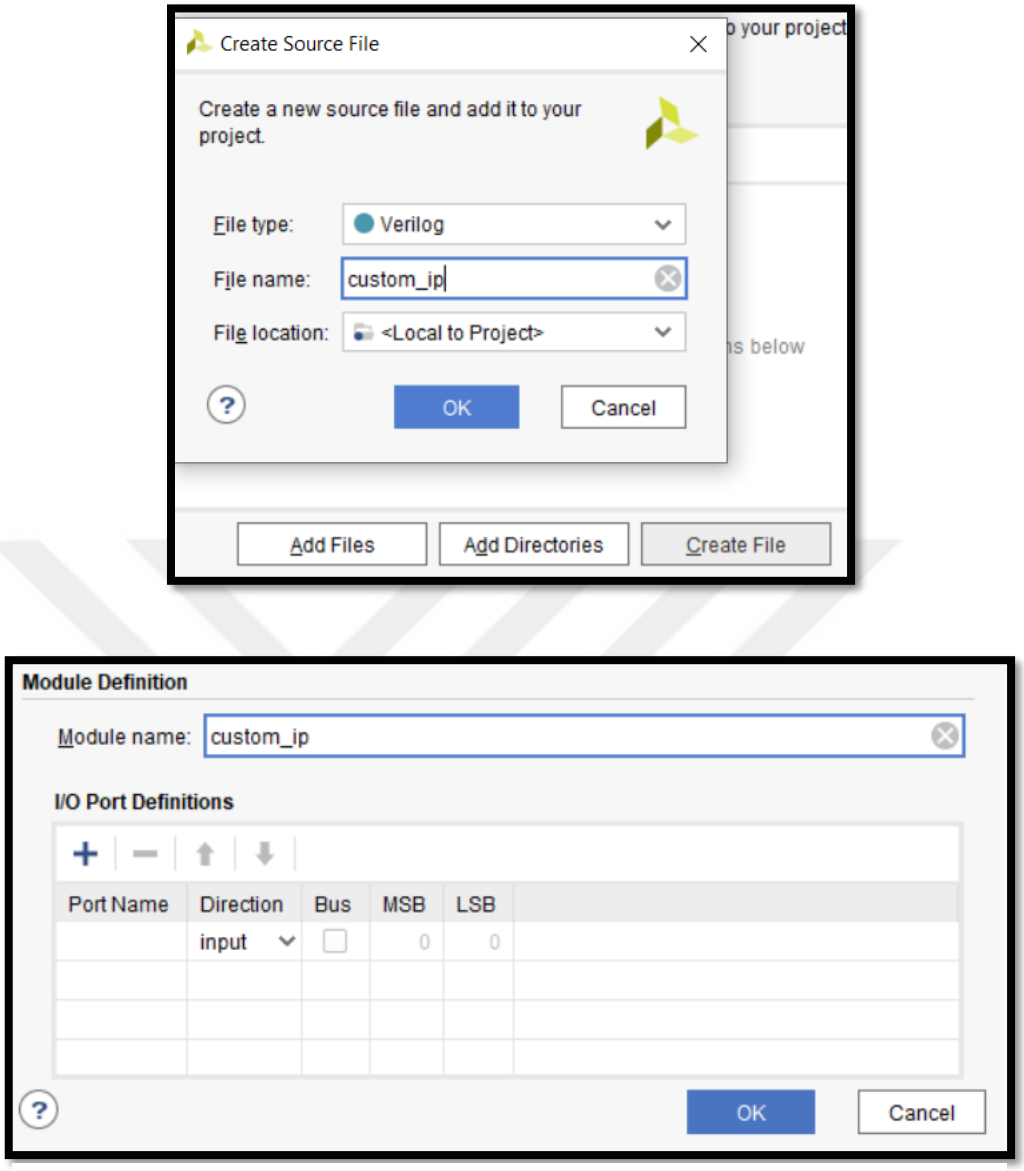
Şekil 2.5. Projeye IP Ekleme

Donanım tanımlama dili kullanılarak yapılan bir tasarımın, blok tasarıma aktarılması için ilk olarak Şekil 2.6'da gösterildiği gibi "add sources" seçeneği kullanılmalıdır.



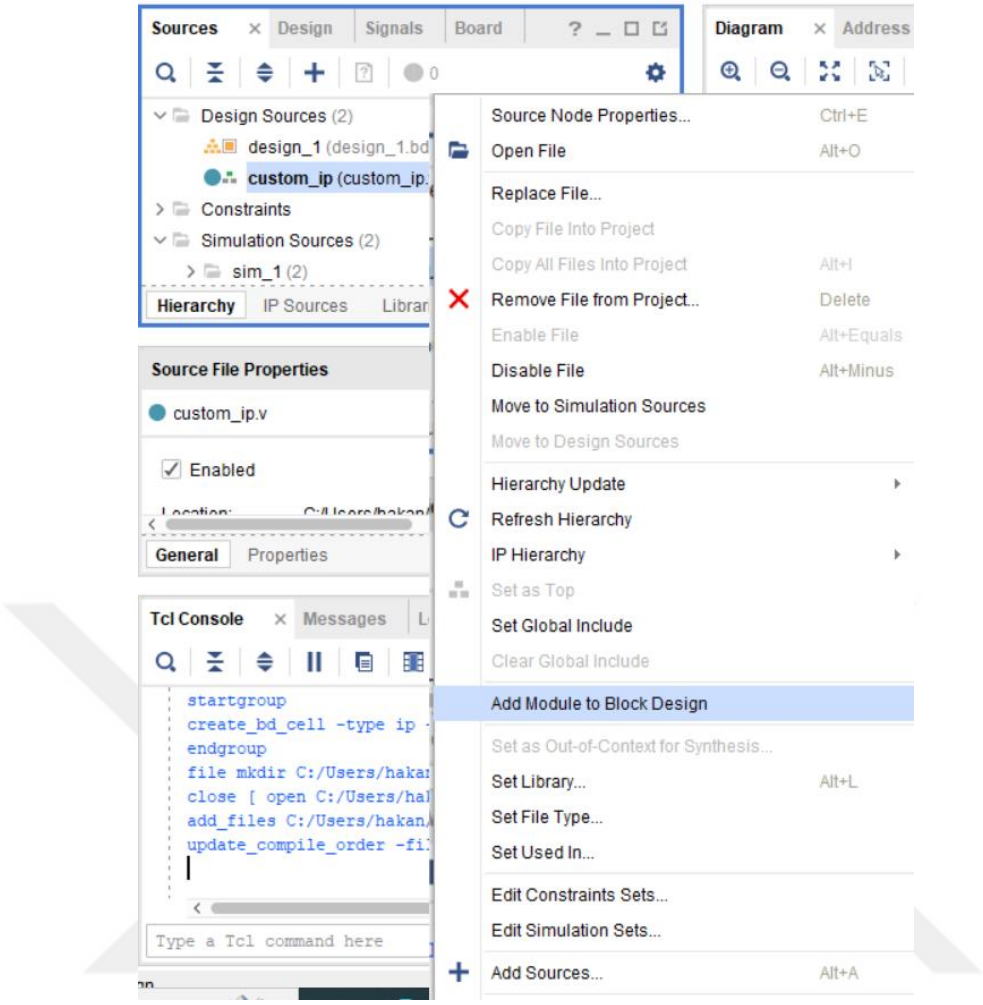
Şekil 2.6. Projeye Yeni Dosya Eklenmesi

Daha sonra Şekil 2.7'de görüldüğü gibi dosya ismi ve konumu seçilir ve sonra "input/output" tanımları arayüzden verilebileceği gibi doğrudan dosya içerisinde de verilebilir.



Şekil 2.7. Giriş/Çıkış Tanımlama

Dosya eklendikten sonra "sources" menüsü altında gözükecektir. Dosya içeriği ihtiyaca göre yazılıp tasarım tamamlandıktan sonra Şekil 2.8'de gösterildiği gibi blok tasarıma aktarılır.



Şekil 2.8. Blok Tasarıma Dosya Aktarılması

Tasarım tamamlandıktan sonra gerekli “validate desing”, “synthesis”, “implementation”, “simulation”,” bit stream” oluşturma gibi işlemler ilerleyen bölümlerde detaylı biçimde ele alınmıştır.

Bir işlemin yazılım yerine donanım tarafından gerçekleştirilerek; bu işlemin fiziksel bir cihaz tarafından daha hızlı yapılabilmesi için gerçekleştirilen tasarımlara "Hardware accelerated" denilmektedir [17], [18]. Geleneksel olarak, birçok işlem yazılım tabanlıdır ve merkezi işlem birimi tarafından yürütülür. Ancak bazı işlemler çok fazla hesaplama gücü veya zaman gerektirir ve yalnızca yazılım tarafından yürütüldüğünde yeterince hızlı olamayabilirler. Bu tür durumlarda, özel donanım yapıları veya yardımcı işlemciler kullanılarak işlemleri donanım tarafından gerçekleştirmek, performansı önemli ölçüde artırabilir. Donanım hızlandırma, yazılım tarafından gerçekleştirilmesi zor veya zaman alan işlemleri daha hızlı ve verimli bir şekilde yapmak için kullanılır. Bu, genellikle daha

yüksek performans, daha düşük güç tüketimi ve daha iyi verimlilik sağlar. Özellikle yapay zeka, nesnelerin algılanması, veri analitiği ve kriptografi gibi alanlarda donanım hızlandırmaları yaygın olarak kullanılır [19], [20]. Bu tarz uygulamalarda konvolüsyon işlemi oldukça yaygın kullanılmaktadır. Temelde bir konvolüsyon, bir girdi matrisi üzerine bir filtre matrisinin yerleştirilmesi ve matris elemanlarının çarpılması işlemiyle yeni bir matrisin oluşturulması işlemidir. Bu işlem özellikle görüntü işleme, sinyal işleme, derin öğrenme uygulamalarında ve diğer veri işleme uygulamalarında sıkça kullanılır. Görüntü filtreleme, kenar tespiti, nesne tespiti ve görüntü düzeltme gibi işlemler, 2 boyutlu konvolüsyon işlemiyle etkili bir şekilde gerçekleştirilebilir. Özellikle yoğun hesaplama gerektiren işlemler, özel donanım hızlandırma teknikleri ile daha verimli ve hızlı bir şekilde gerçekleştirilebilir. Konvolüsyon işlemi de yoğun hesaplama gerektiren bir filtredir. Dolayısıyla bu işlemin FPGA üzerinde donanımsal olarak gerçekleştirilmesi de sistem performansını oldukça iyileştirecek ve güç tüketimini azaltıcı etki gösterecektir [21].

2.1. Soft CPU Kullanılarak Yapılan Benzer Çalışmalar

Literatürde mevcut yeniden sentezlenebilir işlemciler kullanılarak düşük güç tüketimi ve görüntü işleme ile ilgili yapılan çalışmalar incelenmiştir. İnceleme sonucunda daha çok RISC-V tabanlı işlemciler kullanılarak çalışmaların yapıldığı görülmüştür. Ayrıca bu çalışmaların çoğu güç tüketimi odaklı değil daha çok görüntü işleme uygulamaları ve IoT cihazlar için güvenlik uygulamalarına yönelik çalışmalar olduğu görülmüştür. Güç tüketimi odaklı yapılan çalışmalarda ise, uygulamaların gömülü sistem uygulamaları için daha az yetenekli sayılabilecek; 8 bitlik işlemci, kısıtlı sayıda çevre birimi barındıran donanım tasarımlarının yapıldığı görülmektedir. Buda sisteme eklenecek haberleşme modüllerini kısıtlamakta ayrıca işletim sistemi ihtiyacı olan çalışmaların yapılmasını mümkün kılmamaktadır.

Gholizadehazari vd [22], görüntü işleme uygulamaları için özel komut seti içeren RISC-V tabanlı bir SoC'i FPGA üzerinde gerçeklemişlerdir. FPGA geliştirme kartı olarak Nexys 4 DDR kullanılmıştır. Kamera modülü olarakta OV7670 CMOS kullanılmıştır. Görüntülere Laplacian filter uygulanmış ve sistemin genel performansını iyileştirmek adına RISC-V komut setini genişletmişlerdir.

B. Sürer [23], çalışmasında; içerisinde RISC-V tabanlı bir işlemci olan LowRISC SoC ’yi, Nexys4-DDR FPGA kartında gerçeklemiştir. Donanım üzerinde Linux işletim sistemi çalıştırılmıştır. Ethernet bağlantısı kullanarak masaüstü bilgisayardan FPGA kartına bir görüntü aktarılmış ve bu görüntü üzerinde temel görüntü işleme filtreleri uygulanmıştır.

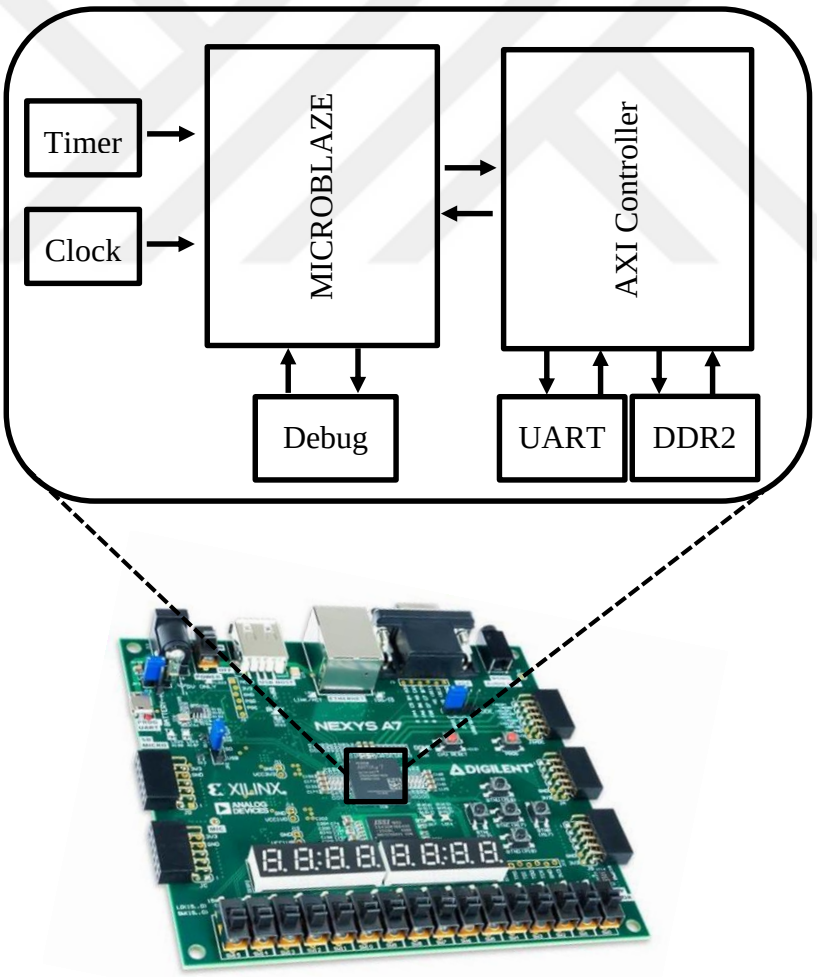
R Garcia-Ramirez vd [24], bu çalışmada; giyilebilir uygulamaları hedefleyen kompakt, düşük güçlü bir SoC tasarımı hedeflemişlerdir. RISC- V RV32I mimarisi barından bir donanım kullanmışlardır. Standart çevre birimlerine ek olarak biyolojik dokular için entegre algılama ve haberleşme arayüzü sunulmuştur.

Skliarova’nın [25], bu çalışmasında; düşük güç tüketimi ve FPGA tabanlı uygulamalar konusuyla ilgili olan nüfus sayımı uygulamasını hızlandırmak için bir MicroBlaze çekirdeğinin ve donanım ortak işlemcisinin kullanımı tartışılmaktadır.

R. Serrano vd [26], bu çalışmada; IoT uygulamaları için düşük güçlü, düşük alan kaplayan bir SoC tasarımı amaçlamışlardır. Daha küçük bir RISC-V çekirdeği kullanmışlardır. 1 kB Ram, GPIO, Timer, Debug Modülü ve SPI çevre birimi desteği sunulmuştur. 1 V ve 32kHz için 2.17 uW güç tüketimi rapor edilmiştir.

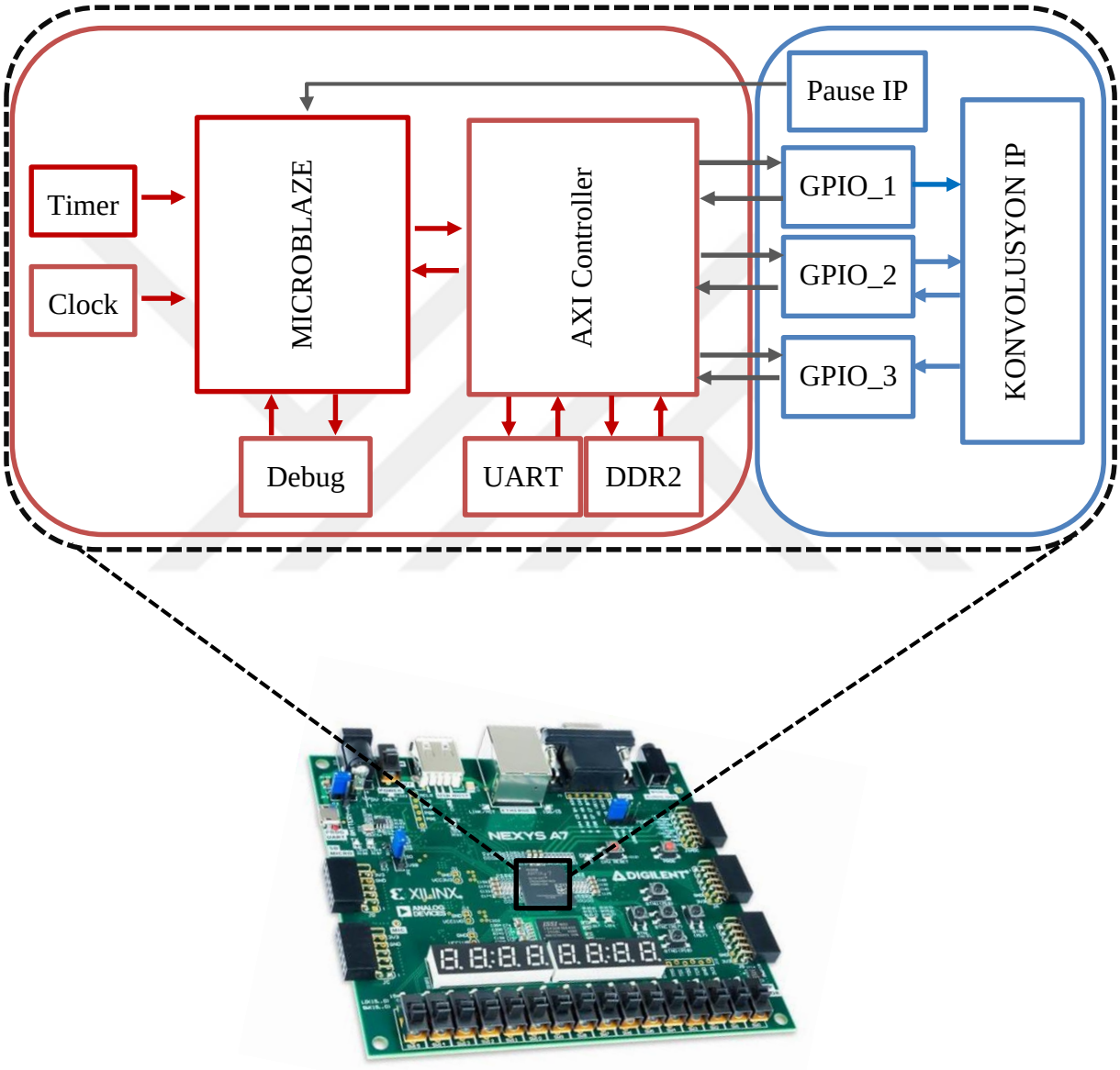
3. SİSTEM MİMARİSİ

Bu bölümde, tez kapsamında tasarlanan donanım ve yazılımla birlikte tüm sistem genel hatlarıyla açıklanmıştır. Tasarlanan sistem; donanım ve uygulama katmanından oluşmaktadır. İlk olarak uygulamaya özel herhangi bir donanım eklemeden, genel kullanım için zorunlu olan modüllerden bir donanım tasarımı yapılmıştır. Bu tasarımda, 32 bitlik Microblaze işlemcisi, geliştirme aşamasında program yüklemek ve hata ayıklamak amacıyla "debug" modülü, performans karşılaştırma sonuçlarının görüntülenebilmesi için "uart" modülü, işlemcinin ve diğer çevre birimlerinin çalışması için gerekli olan "clock" ve "timer" modülleri, büyük bellek ihtiyacı duyabilecek uygulamalar için "DDR2" modülü eklenmiştir. Bu modüller herhangi bir uygulamayı çalıştırmak için gerekli olan donanım modülleridir. Söz konusu bu donanıma ait şema Şekil 3.1'de görülmektedir.



Şekil 3.1. Sadece Gerekli Modüller ile Yapılan Donanım Tasarımı

Donanım tasarımının ikinci adımında, Şekil 3.1'de belirtilen gerekli modüllere ek olarak, tez kapsamında gerçekleştirilecek uygulamaya yönelik; "pause IP", "konvolüsyon IP" ve bunların işlemci ile haberleşebilmesi için 3 adet GPIO modülü donanıma dahil edilmiştir. Eklenen bu modüllerle ilgili detaylı açıklamalar bir sonraki bölümde anlatılmıştır. Şekil 3.2'de uygulamaya özel eklenen modüllerle birlikte yeni donanıma ait şema görülmektedir.

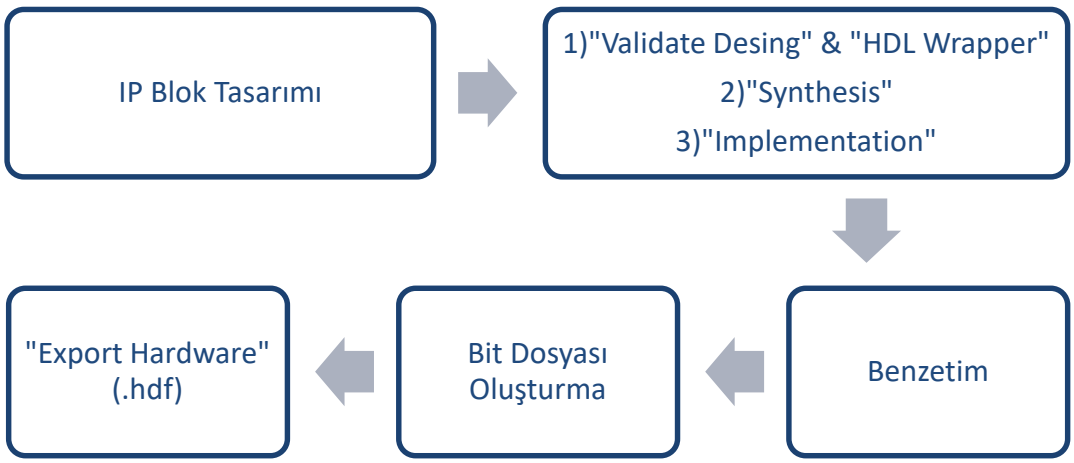


Şekil 3.2. Uygulamaya Özel Eklenen IP Bloklar ile Yapılan Donanım Tasarımı

Microblaze, RISC mimarisine sahip bir işlemci çekirdeğidir bu yüzden basit ve hızlı bir işlemci tasarımı sağlar. Microblaze, genellikle gömülü sistemlerde ve FPGA tabanlı yüksek performanslı hesaplama uygulamalarında kullanılır [27]. Özellikle düşük güç

tüketimi gerektiren ve yüksek paralel işlem gücüne ihtiyaç duyan uygulamalarda tercih edilir [28], [29]. Harici bellek birimleriyle iletişim kurabilir bir veri yolu arabirimi sağlar. IoT uygulamalarında Microblaze kullanımının önemi, düşük güç tüketimi, esneklik, yüksek performans, hızlı prototipleme ve FPGA entegrasyonu gibi avantajlar sunmasından kaynaklıdır. Bu özellikler, özellikle özelleştirilmiş ve enerji verimli IoT çözümleri geliştirmek için çok değerlidir [30], [31], [32]. Microblaze, FPGA üzerinde çalışırken düşük güç tüketimi sağlamak için optimize edilebilir [33]. Bu, IoT cihazları için önemli bir özelliktir, çünkü çoğu IoT cihazı genellikle pil veya batarya ile çalışır ve enerji verimliliği bu sistemler için kritik bir faktördür. Microblaze kullanılarak yapılan düşük güç tüketimi odaklı sistemlerin tasarlanması, IoT cihazlarının pil ömrünü uzatabilir ve enerji tasarrufu sağlayabilir.

Tez kapsamında yapılan konvolüsyon filtre uygulaması ilk olarak Şekil 3.1'de gösterilen donanım üzerinde gerçekleştirilmiştir. Bu donanımda, Microblaze varsayılan yapılandırma seçenekleriyle kullanılmıştır. İkinci adımda ise Şekil 3.2'de gösterilen donanım tasarlanmıştır. Eklenen donanımlara ek olarak, yukarıda bahsedilen Microblaze'in yeniden yapılandırılabiliyor özelliği kullanılmıştır. İşlemci güç tüketimi göz önüne alınarak, performansı iyileştirecek şekilde yapılandırılmıştır. Bu yapılandırma seçenekleri ve performansa etkisi bir sonraki bölümlerde detaylı şekilde ele alınmıştır. Donanımın tasarlanma sürecini özetleyen akış Şekil 3.3'de gösterilmiştir.



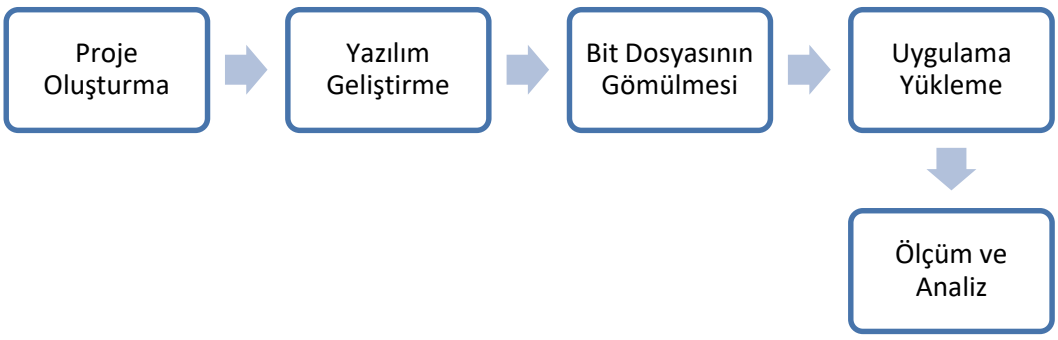
Şekil 3.3. Donanım Tasarımı için Uygulama Adımları

1. **IP Blok Tasarımı:** Xilinx'in sağlamış olduğu IP blok kütüphanesi kullanarak, Şekil 3.1 ve Şekil 3.2'de gösterilen donanımlarda bulunan IP'lerin birbirleriyle haberleşebilmesi için gerekli pin bağlantıları ve yapılandırma seçeneklerinin gerçekleştirildiği adımdır.
2. **"Validate Design" ve "HDL Wrapper":** Gerçekleştirilen donanım tasarımının doğruluğunun kontrol edildiği ve IP blok tasarımın HDL koda dönüştürüldüğü adımdır.

   "**Synthesis":** Donanım tasarım dilinin derlenip, FPGA veya ASIC gibi bir uygulama teknolojisine eşlendiği bir süreçtir.

   "**Implementation":** FPGA üzerinde hangi fiziksel kaynakların, hangi mantıkla programlanacağını ve bunların nasıl bağlanacağını (yönlendirileceğini) belirleme işlemidir
3. **Benzetim:** Bu adım opsiyonel olarak gerçekleştirilir. Donanımın doğru çalışıp çalışmadığını test etme sürecidir.
4. **Bit Dosyası Oluşturma:** FPGA'nın yapılandırma bilgilerini içeren bir dosyadır.
5. **"Export Hardware":** Donanıma ait bilgileri içeren ".hdf" uzantılı dosyanın oluşturulma adımıdır.

Xilinx tabanlı işlemciler üzerinde gömülü yazılım uygulamaları geliştirmek için Xilinx SDK IDE'sini kullanmak gerekmektedir. Bu IDE'de tasarlanan donanıma yönelik bir proje oluşturulur. Bunun için Şekil 3.3'de verilen işlemlerin tamamlanmış olması gerekmektedir. Şekil 3.4'de uygulama geliştirmek için yapılan işlemleri özetleyen akış görülmektedir.



Şekil 3.4. Yazılım Katmanı için Uygulama Adımları

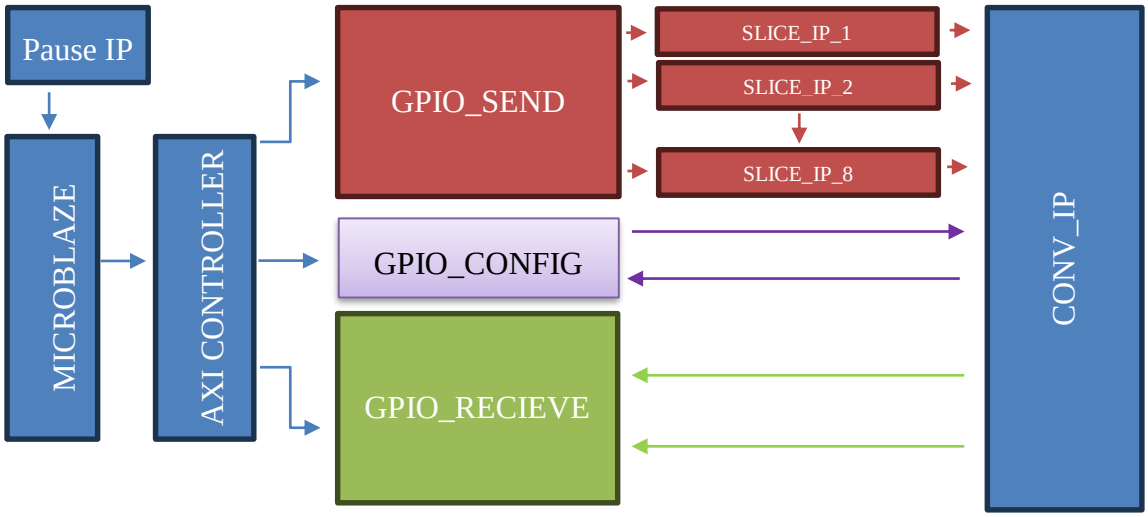
1. **Proje Oluşturma:** Vivado programında hazırlanan ve donanıma ait bilgiler içeren “hdf” uzantılı dosyayı kullanarak, donanıma yönelik proje oluşturma adımıdır.
2. **Yazılım Geliştirme:** Uygulama kodunun yazıldığı ve derlenerek işlemciye yüklenecek “elf” formatında ki dosyanın üretildiği adımdır.
3. **Bit Dosyasının Gömülmesi:** Soft işlemciyi ve diğer donanım bilgilerini içeren “bit” dosyasının FPGA gömüldüğü adımdır. Bu adım daha öncede yapılabilir. Bu adımla birlikte artık tasarlanan soft işlemci FPGA üzerinde çalıştırılabilir hale gelmektedir.
4. **Uygulama Yükleme:** Adım 2’de üretilen “elf” dosyasının FPGA’ye yüklendiği adımdır.
5. **Ölçüm ve Analiz:** Uygulama içerisinde alınan ölçümlerin “uart” modülüyle terminalde görüntülendiği adımdır.

Tez kapsamında yapılan donanım tasarımı ve yapılandırma seçeneklerinin her biri için Şekil 3.3 süreç işletilerek bir “hdf” uzantılı dosya üretilmiştir. Her bir “hdf” dosyası için Şekil 3.4’de verilen adımlar gerçekleştirilerek ölçüm ve analizler tekrar alınmıştır. Bu ölçüm ve analizlere ait detaylı karşılaştırmalar bölüm 5’de açıklanmıştır.

3.1 IP Tasarım

FPGA kaynaklarını kullanarak yazılımsal olarak tasarlanan işlemcilere Soft CPU denir. Bu tarz işlemciler amaca uygun değiştirilebilir ve tekrar sentezlenebilir. Bu çalışmada donanım tasarımı, Digilent firması tarafından üretilen, Nexys A7-100T FPGA geliştirme kartı üzerinde, Verilog donanım tanımlama dili ve Xilinx firmasının sunmuş olduğu IP blok tasarım servisleri kullanılarak yapılmıştır. Bu çalışma kapsamında yapılan donanım tasarımında, gömülü sistem uygulamalarına yönelik özellikle IoT uygulamalarında sistem performansını iyileştirecek bazı donanımsal özellikler aktifleştirilmiştir. Bu tasarımın etkisini görebilmek için bir uygulama gerçekleştirilmiştir. Örnek uygulama olarak iki boyutlu konvolüsyon işlemi gerçeklenmiştir.

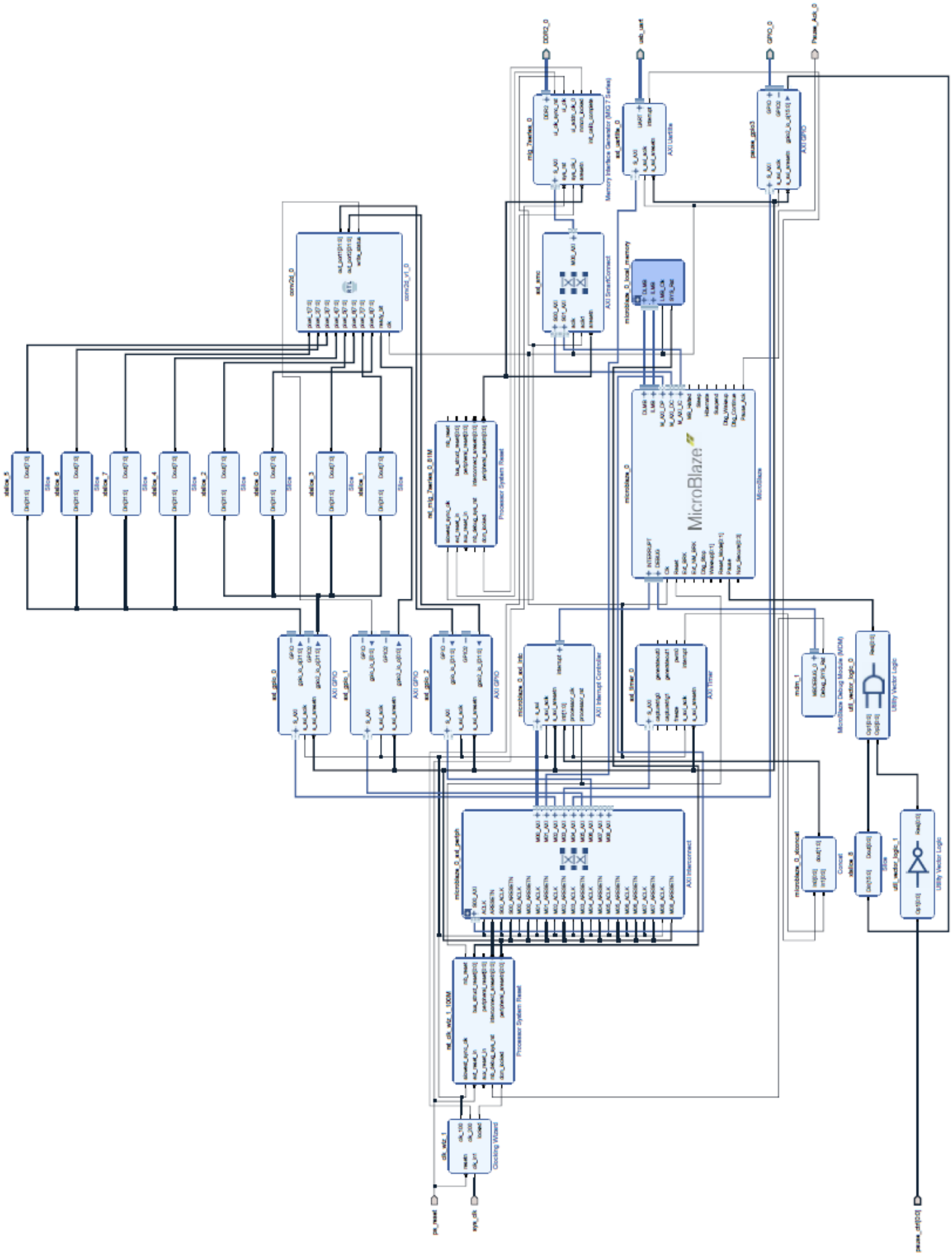
Konvolüsyon işlemi, nesne algılama, yüz tanıma, hasta hareket takibi gibi görüntü işleme uygulamalarından, tıbbi görüntülerin işlenmesinde hastalık teşhis aşamalarına, akıllı ulaşımdan tarım ve bitki sağlığı gibi endüstride birçok alanda kullanılmaktadır. Bu tür uygulamaların bir kısmı batarya ile çalışırken bir kısmı şebekeye bağlı çalışırlar. Bu tarz uygulamalarda enerji verimliliğini sağlamak hem sürdürülebilir enerji dostu bir yaşam sunmak hemde sistemlerin daha uzun süreli çalışmalarına imkân sağlayacaktır. Uygulamaya özel tasarlanan donanımın işlemci ile haberleşmesini açıklayan akış Şekil 3.5'de görülmektedir.



Şekil 3.5. Uygulamaya Yönelik Yapılan IP Blok Tasarımı

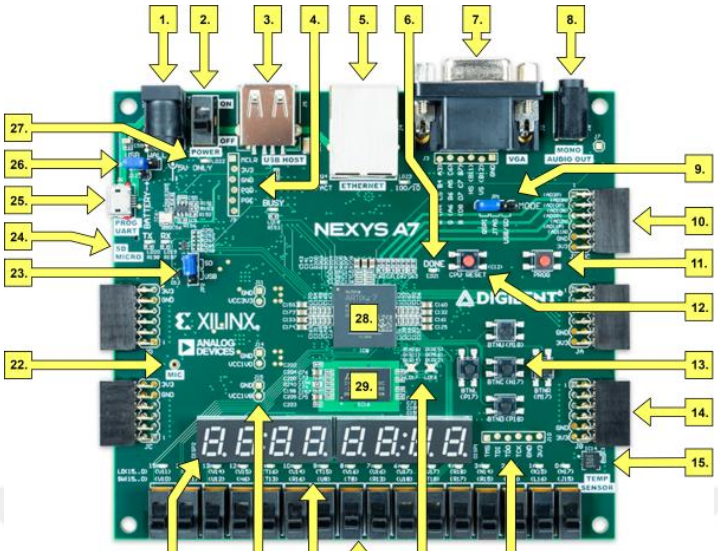
İşlemci ile özel donanım arasındaki iletişim, tasarımın basit ve güç tüketimin az olması için GPIO portları üzerinden sağlanmıştır. Gönderilecek her piksel için bir I/O pini kullanılmaktadır. Tez kapsamında kullanılan geliştirme kartının sahip olduğu pin sayısı göz önünde bulundurularak, pikseller 8'erli gruplar halinde gönderilmiştir. GPIO modülünün 2 adet portu bulunmaktadır. Bu portlar üzerinden 32 bitlik verilerle işlem yapılabilmektedir. Yani bir GPIO modülünde 64 bitlik veri transferi sağlanabilir. Gönderilecek 8 piksel değeri uygulama katmanında, 4'lü pikseller şeklinde 32 bitlik iki değişkende tutulur. 32 bitlik değer tutan bu değişkenler, GPIO_SEND modülünün sahip olduğu 2 port kullanılarak konvolüsyon IP'ye gönderilir. Konvolüsyon IP'ye göndermeden önce 4'erli gruplar halinde olan bu piksel değerleri "Slice IP" kullanılarak anlamlı birer piksele dönüştürülür ve konvolüsyon IP'sine girdi olarak iletilir. Tek seferde toplamda 8 tane piksel değeri konvolüsyon IP'ye iletilir. Veri transferinde IP ile işlemcinin senkron olması gerekmektedir. Bunun için "GPIO_CONFIG" IP'si kullanılmıştır. Bu IP sayesinde, veri gönderme ve alma işlemlerinin senkronizasyonu için çift yönlü iletişim sağlanır. Konvolüsyon işlemi tamamlandıktan sonra, hesaplanan pikseller GPIO_RECIEVE IP'si üzerinden işlemciye geri gönderilir.

"Pause IP", işlemciyi boşta çalıştığı durumlarda uyku moduna almak için tasarlanmıştır. Tez kapsamında konvolüsyon işlemi bare-metal ve FreeRTOS kullanılarak gerçekleştirilmiştir. FreeRTOS kullanılan senaryoda, piksel okuma işlemi için bir görev tanımlanmıştır. Bu görev belirli aralıklarla yeni piksel değerlerini beklemektedir. Bu beklemenin olduğu durumda işlemci başka bir iş yapmamaktadır. Bu durumlarda işlemcinin uyku moduna geçirilerek düşük güç tüketimi sağlanması amaçlanmıştır. Tüm bu IP bloklar kullanılarak yapılan tasarımın genel görünümü Şekil 3.6'da görülmektedir.



Şekil 3.6. Vivado IP Blok Tasarımı

Tez'de kullanılan Nexys A7-100T geliştirme kartı ve sahip olduğu özellikler Şekil 3.7'de görülmektedir.



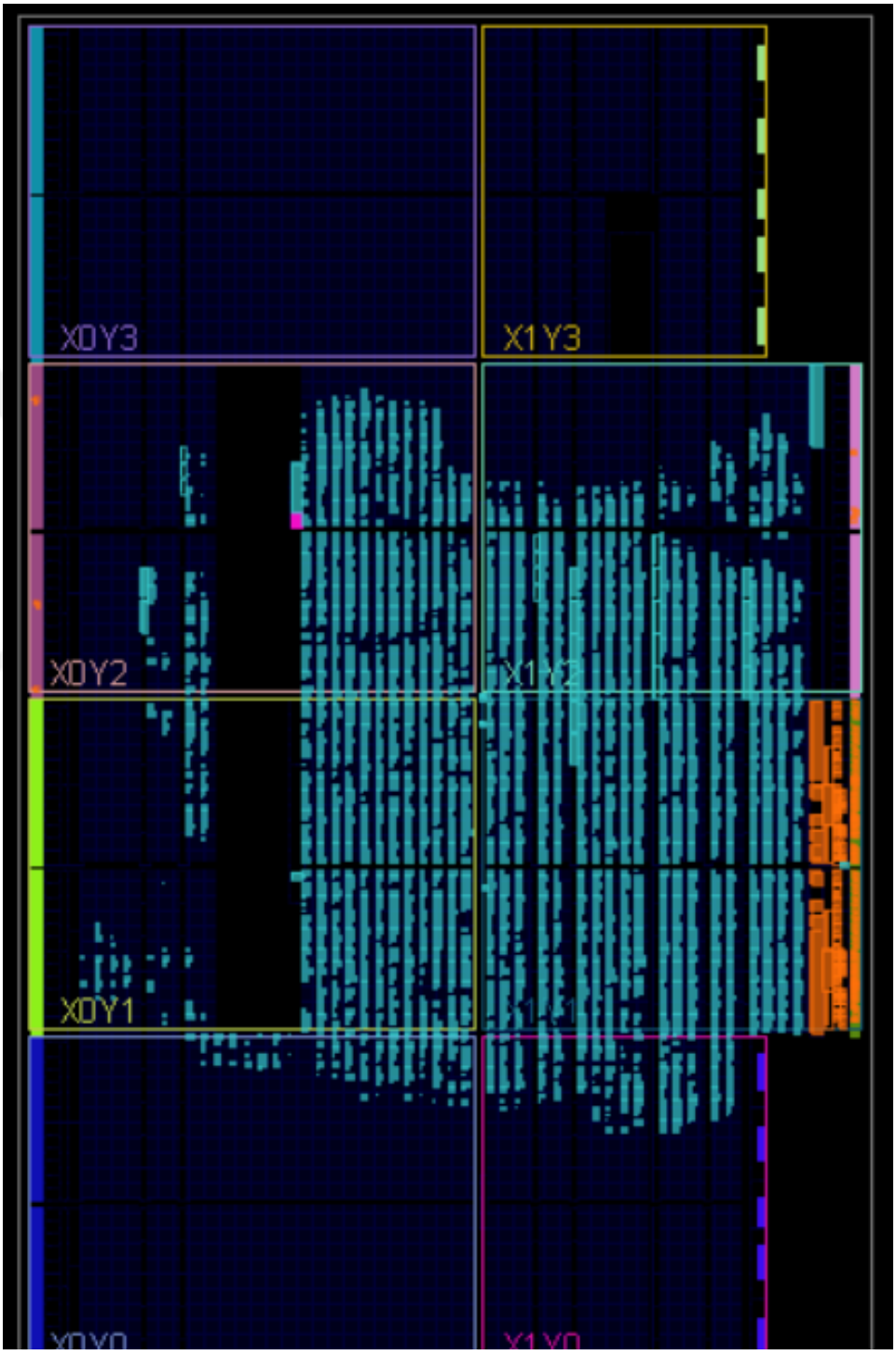
Şekil 3.7. Nexys A7-100T Kartı

Tablo 3.1'de geliştirme kartının sahip olduğu bağlantılar ve açıklamaları detaylandırılmıştır [34].

Tablo 3.1. Bileşen Açıklaması

No	**Bileşen Açıklaması**	**No**	**Bileşen Açıklaması**
1	Güç Kablosu	**16**	Harici Kablo için JTAG Port
2	Güç Anahtarı	**17**	RGB Led
3	USB Bağlantı Ucu	**18**	Kaydırmalı Anahtarlar (16)
4	PIC24 Programlama Ucu	**19**	LED (16)
5	Ethernet Bağlantısı	**20**	Güç Kaynağı Test Noktası
6	FPGA Programlama Ledi	**21**	7 Segment Display
7	VGA Bağlantı	**22**	Mikrofon
8	Audio Bağlantı	**23**	SD/USB Seçimi
9	Programlama Modu Seçeneği	**24**	MicroSD Kart Yuvası
10	Analog Sinyal Pmod Bağlantı	**25**	UART/JTAG/USB Bağlantı Ucu
11	FPGA Yapılandırma Reset Buton	**26**	Güç Seçme Ucu ve Batarya Başlığı
12	İşlemci Reset Buton	**27**	Güç Durum Ledi
13	5 adet Buton	**28**	Xilinx Artix-7 FPGA
14	Pmod Uçları	**29**	DDR2 Bellek
15	Sıcaklık Sensörü		

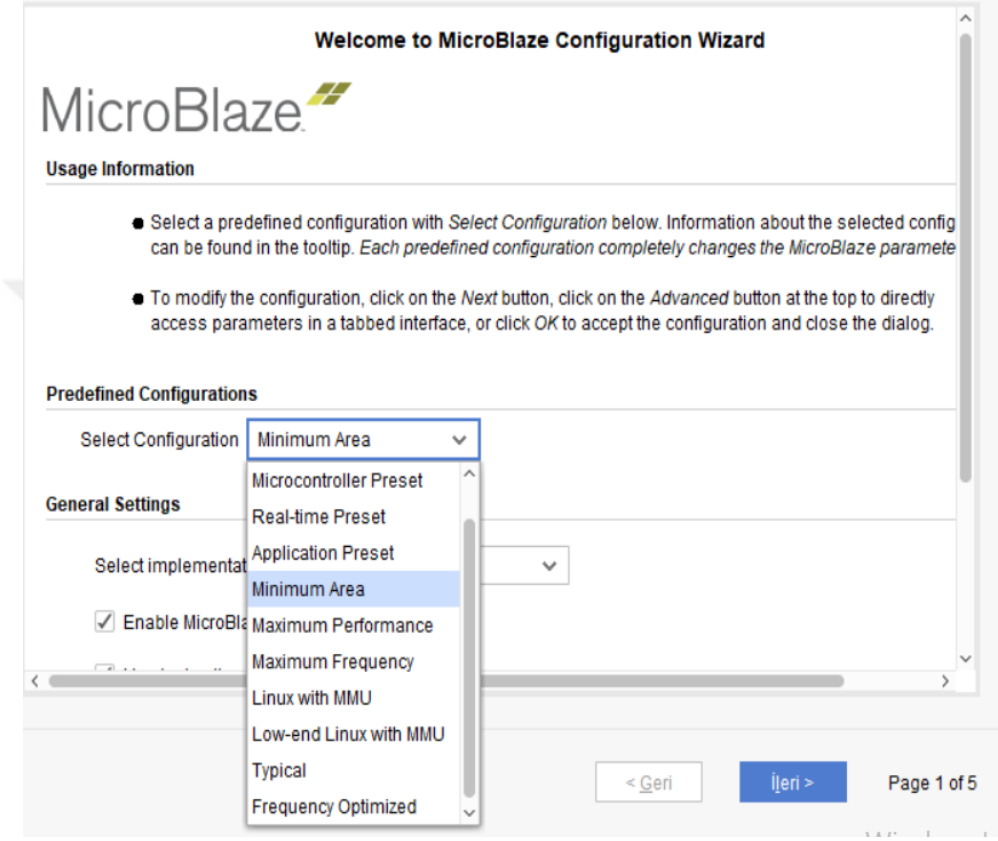
Donanımın FPGA üzerinde kaynak kullanımı Şekil 3.8‘de görülmektedir. Burada kullanılan kaynak sayıları yapılandırma seçeneklerine göre değişiklik göstermektedir. Her yapılandırma seçeneğine ait LUT, FF gibi kaynak kullanımlar ilgili yapılandırma seçeneğinin açıklandığı bölümde tablo olarak verilmiştir.



Şekil 3.8. FPGA Alan Kullanımı

3.2 Microblaze Yapılandırma Seçenekleri

Microblaze için önceden tanımlı bazı yapılandırma seçenekleri bulunmaktadır. Bu yapılandırma seçenekleri farklı modülleri donanıma dahil ederek, bazı özellikleri ön plana çıkarmaktadır. Microblaze'de var olan ön yapılandırma seçenekleri Şekil 3.9'da görülmektedir.



Şekil 3.9. Ön Tanımlı Yapılandırma Seçenekleri

Bu tezde "minumum area" ve "maximum performance" seçenekleri kullanılarak sentezleme işlemleri gerçekleştirilmiştir. Ancak bu seçeneklerin uygulama ihtiyacını tam olarak karşılayamadığı görülmüştür. Örneğin "minumum area" tercihi kaynak kullanımını azaltmaktadır ancak bunu yaparken gerekli birçok özelliği de devre dışı bırakmaktadır. Diğer taraftan "maximum performance" da uygulamada gerekli olmayan birçok özelliği aktif etmektedir. Bu da gereksiz kaynak ve güç tüketimi anlamına gelmektedir. Microblaze'de var olan ön tanımlı sentezleme seçenekleri ve her birinin izin verdiği donanım modülleri Tablo 3.2'de görülmektedir.

Tablo 3.2. Ön Tanımlı Yapılandırma Seçenekleri [35]

Parametre	Mikrodenetleyici	Gerçek Zamanlı	Uygulama	Minimum Alan	Maksimum Performans	Maksimum Frekans	Linux MMU	Low-end Linux MMU	Tipik	Frekans Optimize
C_ALLOW_DCACHE_WR	1	1	1	1	1	1	1	1	1	1
C_ALLOW_ICACHE_WR	1	1	1	1	1	1	1	1	1	1
C_AREA_OPTIMIZED	1	0	0	1	0	0	0	0	0	2
C_CACHE_BYTE_SIZE	4096	8192	32768	4096	32768	4096	16384	8192	8192	16384
C_DCACHE_BYTE_SIZE	4096	8192	32768	4096	32768	4096	16384	8192	8192	16384
C_DCACHE_LINE_LEN	4	4	4	4	8	4	4	4	4	4
C_DCACHE_USE_WRITEBACK	0	1	1	0	1	0	0	0	0	1
C_DEBUG_ENABLED	1	1	1	0	1	0	1	1	1	1
C_DIV_ZERO_EXCEPTION	0	1	1	0	0	0	1	0	0	1
C_M_AXI_D_BUS_EXCEPTION	0	1	1	0	0	0	1	1	1	1
C_FPU_EXCEPTION	0	0	1	0	0	0	0	0	0	1
C_FSL_EXCEPTION	0	0	0	0	0	0	0	0	0	0
C_FSL_LINKS	0	0	0	0	0	1	0	0	0	0
C_ICACHE_LINE_LEN	4	4	8	4	8	4	8	4	8	8
C_ILL_OPCODE_EXCEPTION	0	1	1	0	0	0	1	1	0	1
C_M_AXI_I_BUS_EXCEPTION	0	1	1	0	0	0	1	1	0	1
C_MMU_DTLB_SIZE	2	2	4	2	4	2	4	4	4	4
C_MMU_ITLB_SIZE	1	1	2	1	2	1	2	2	2	2
C_MMU_TLB_ACCESS	1	2	2	0	1	1	1	1	2	1
C_MMU_ZONES	2	2	2	2	2	2	2	2	2	2
C_NUMBER_OF_PC_BRK	1	2	2	0	1	1	1	1	2	1
C_NUM_OF_RD_ADDR_BRK	0	0	1	0	0	0	0	0	0	0
C_OPCODE_0x0_ILLEGAL	0	1	1	0	0	0	1	1	0	1
C_PVR	0	0	2	0	0	0	2	0	0	2
C_UNALIGNED_EXCEPTIONS	0	1	1	0	0	0	1	1	0	1
C_USE_BARREL	1	1	1	0	1	0	1	1	1	1
C_USE_DCACHE	0	1	1	0	1	0	1	1	1	1

Güç tüketimi etkisi sadece donanımın kaynak kullanımıyla ilgili değil aynı zamanda uygulamanın gerçekleme süresine de bağlıdır. Bu yüzden sistem tasarlanırken sadece donanımın kaynak tüketimi değil, uygulama gerçekleme süresinin iyileştirilmesi de amaçlanmalıdır. Örneğin önbellek kullanımı genellikle tüm sistemlerde iyileştirici bir performans sunar. Bunun yanı sıra kod geliştirirken hata ayıklama işlemleri ve uygulamanın karta yüklenmesi için debug birimine ihtiyaç duyulur. Aynı şekilde işlemciyi boşta çalışma sırasında düşük güç tüketim moduna alabilmek için Microblaze'de "discrete port" modülünün aktifleştirilmesi gerekmektedir. Tablo 3.2'den görüleceği üzere bu özellikler "Minumum Area" seçeneğinde aktif edilememektedir. "Maximum Performance" seçeneğinde ise "Minumum Area" seçeneğinin aksine ihtiyaç duyulmayan birçok modül donanıma dahil edilmektedir.

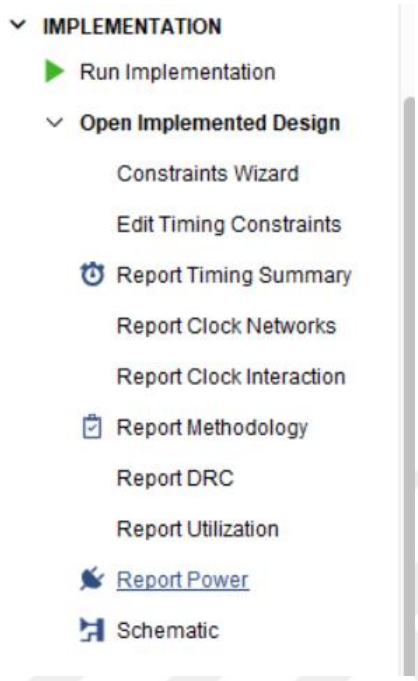
3.2.1 Minimum Alan

Bu seçenekte asıl hedef işlemcinin genel performansını göz önünde bulundurmadan sadece kullanılan kaynak sayısını azaltmaktır. Bu seçenekte işlemci 3 aşamalı boru hattı kullanır. "Branch prediction" ve "cache" gibi uygulama performansını önemli ölçüde iyileştiren birçok özellik bu seçenekle uyumlu değildir. Ayrıca bu seçenek ile oluşturulmuş donanım üzerinde koşturulan yazılımlar için bellek kullanımı lokal bellek kapasitesi ile sınırlıdır. Bu değer de maksimum 128kB olabilmektedir. "Minumum Area" seçeneği kullanılarak yapılan sentezleme sonucu FPGA üzerinde kullanılan kaynak sayısı ve oranları Tablo 3.3'de görülmektedir.

Tablo 3.3. "Minumum Area" Seçeneği Tercih Edildiğinde Kaynak Kullanımı

Kaynak Tipi	**Kullanım Sayısı**	**Kullanılabilir Kaynak Sayısı**	**Kullanım %**
LUT	11914	63400	18.79
LUTRAM	1630	190000	8.58
FF	12335	126800	9.73
BRAM	12	135	8.89
IO	52	210	24.76
BUFG	5	32	15.63
MMCM	1	6	16.67
PLL	2	6	33.33

Vivado ‘da “implementation” yapıldıktan sonra donanıma ait güç tüketimi Şekil 3.10’da gösterildiği gibi “Report Power” seçeneğiyle elde edilebilir.



Şekil 3.10. Güç Tüketim Raporu

Elde edilen raporda donanım içinde bulunan tüm bileşenlerin ne kadar güç tükettikleri ayrı ayrı yazmaktadır. “Minumum Area” seçeneği için elde edilen dinamik ve statik güç raporları Tablo 3.4’de görülmektedir.

Tablo 3.4. “Minumum Area” Seçeneği Tercih Edildiğinde Güç Tüketimi

Kaynak	**Güç Tüketimi (W)**
Clocks	0.063
Signals	0.031
Logic	0.022
BRAM	0.013
PLL	0.115
MMCM	0.113
Phaser	0.110
I/O	0.531
XADC	0.004
PL Static	0.108

Eşitlik (3.1) kullanılarak “Minumum Area” seçeneği için toplam güç tüketimi 1.11 W olarak hesaplanır.

$$Toplam\ Güç\ Tüketimi = Dinamik\ Güç\ Tüketimi + Statik\ Güç\ Tüketimi \quad (3.1)$$

3.2.2 Maksimum Performans

“Minumum Area” seçeneğinin aksine “Maximum Performance” seçeneğinde 5 seviye “pipeline” kullanılmaktadır. “Branch prediction”, “cache” vb. donanımlar bu seçenekte donanıma dahil edilebilmektedir. Alan olarak daha fazla yer kaplasa da uygulama performansı için en iyi seçeneklerden biridir. Fakat bu projede ihtiyaç duyulmayan “floating point”, “pattern comparator” vb. diğer donanım modülleri gereksiz kaynak tüketmektedir. “Maximum Performance” sentezleme yöntemine ait kaynak kullanım sayısı ve oranları Tablo 3.5’de görülmektedir.

Tablo 3.5. “Maximum Performance” Seçeneği Tercih Edildiğinde Kaynak Kullanımı

Kaynak Tipi	**Kullanım Sayısı**	**Kullanılabilir Kaynak Sayısı**	**Kullanım %**
LUT	14944	63400	23.26
LUTRAM	1711	190000	8.58
FF	15647	126800	9.73
BRAM	27	135	8.89
DSP	6	240	2.50
IO	52	210	24.76
BUFG	5	32	15.63
MMCM	2	6	33.33
PLL	1	6	16.67

“Maximum Performance” seçeneği için elde edilen güç raporu Tablo 3.6’da görülmektedir.

Tablo 3.6. "Maximum Performance" Seçeneği Tercih Edildiğinde Güç Tüketim

Kaynak	**Güç Tüketimi (W)**
Clocks	0.072
Signals	0.046
Logic	0.029
BRAM	0.072
DSP	0.004
PLL	0.115
MMCM	0.209
Phaser	0.110
I/O	0.530
XADC	0.004
PL Static	0.110

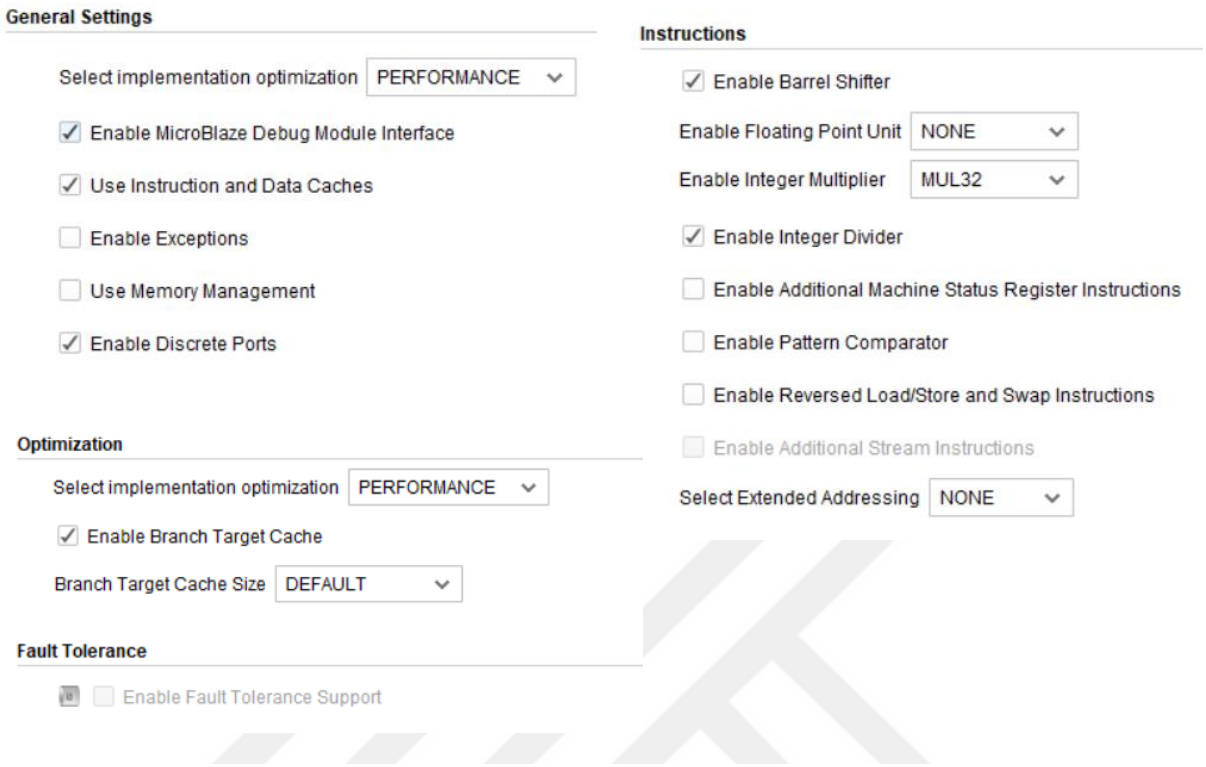
Eşitlik (3.1) kullanılarak toplam güç tüketimi 1.301 W olarak hesaplanır.

3.2.3 Uygulamaya Özel Donanım Yapılandırması

Bu bölümde, yukarıda açıklanan kapsamı en büyük ve en düşük olan 2 yapılandırma seçeneğine alternatif olarak uygulamaya özel bir yapılandırma detayları açıklanmıştır. Bu yeni konfigürasyon aşağıda verilen seçenekler ile oluşturulmuştur.

- Program yükleme ve ayrıca oluşan hataları ayıklamak için debug modülü eklenmiştir.
- IoT uygulamalarının büyük bir kısmında, çeşitli filtreler kullanılmaktadır. Bu filtreler doğası gereği matematiksel işlemlere ihtiyaç duymaktadır. Özellikle bu çalışmada yapılan örnek uygulamada olduğu gibi, kompleks matematiksel işlemler içeren durumlar için "çarpma modülü" ve "bölme modülü" eklenmiştir.
- Uygulamalarda veriler anlamlandırılırken, paketlenip çözülürken aritmetik kaydırma operatörü kullanılır. Eklenen "barrel shifter" sayesinde bu işlemler daha az saat işaretinde gerçekleştirilebilmektedir.

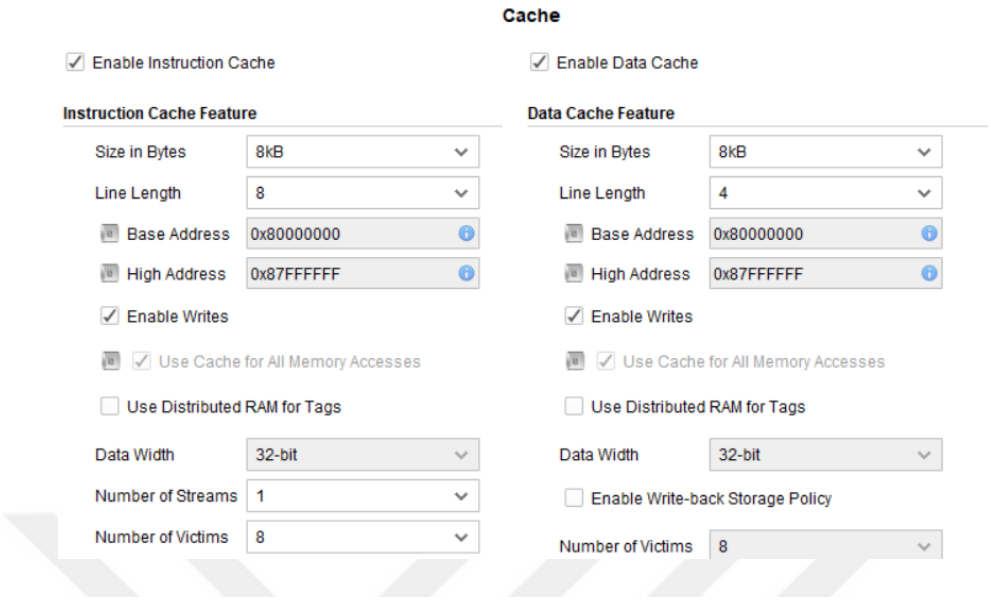
Bu modüllerin donanıma aktif edilebilmesi için yapılan güncellemelere ait arayüz Şekil 3.11'de görülmektedir.



Şekil 3.11. Uygulamaya Özel Yapılandırma

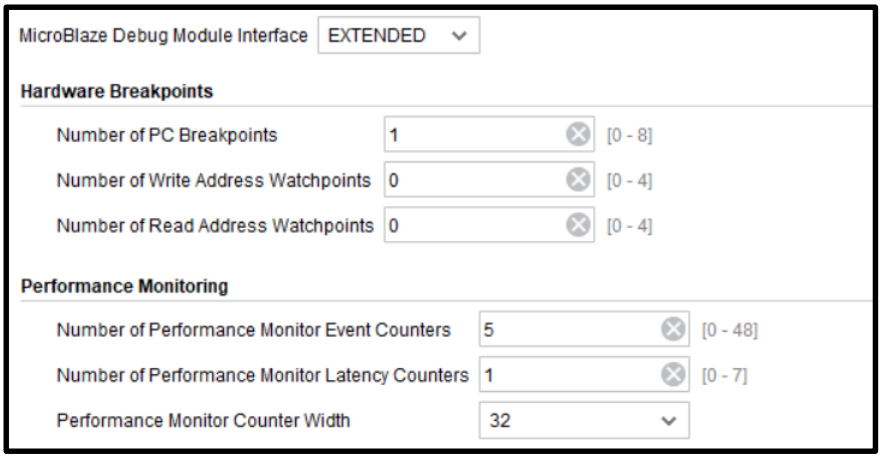
CPU önbelleklerinin iki ana amacı vardır: 1) Ana bellekten veri ve komut getirme işlemlerinin ortalama gecikmesini azaltmak. 2) CPU'nun veri veya komut beklediği süreyi azaltarak genel CPU performansını artırmak. Bu nedenlerle donanım tasarımına önbellek eklenmiştir.

Önbellek aktifleştirmek için yapılan seçimler Şekil 3.12'de görülmektedir.



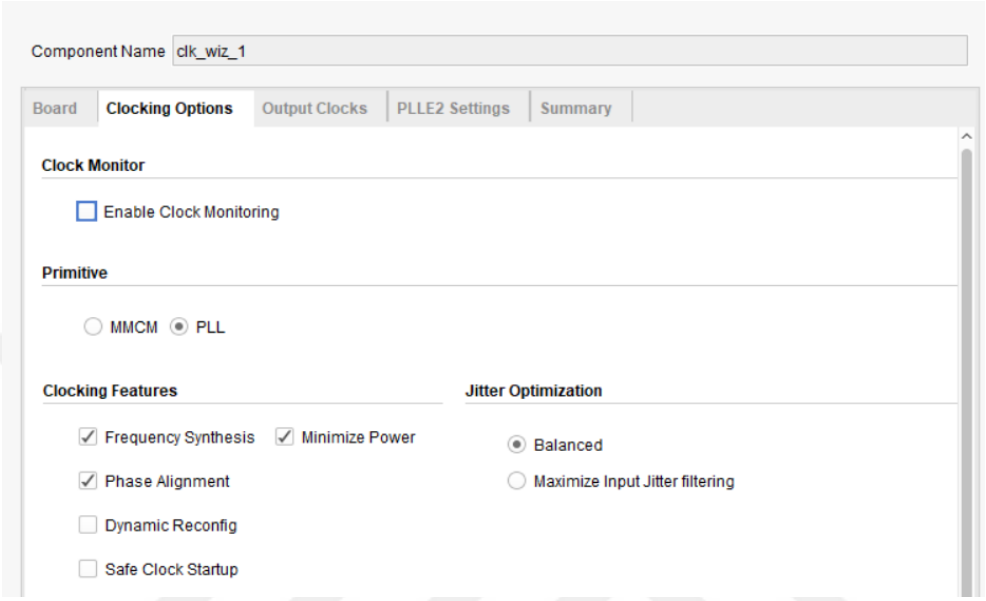
Şekil 3.12. Cache Yapılandırması

Uygulamanın CPU kullanımını analiz etmek için "Debug Extended" modülü donanıma eklenmiştir. Bu seçenek sayesinde işlemci üzerinde gerçekleşen olaylar analiz edilebilmektedir. "Debug Extented" modülü için yapılan seçimler Şekil 3.13'de görülmektedir.



Şekil 3.13. Debug Yapılandırması

Son olarak, saat işareti kaynağı olarak varsayılan seçenek “MMCM” gelmektedir. Diğer bir alternatif ise PLL’dir. PLL, MMCM ‘nin alt kümesi gibi düşünülebilir. Donanımsal olarak daha basit yapıya sahiptir. MMCM kullanımında FPGA kaynak tüketimi PLL’ye göre çok fazla artmaktadır. Bu nedenle clock kaynağı PLL olarak güncellenmiştir. Clock için yapılan seçimler Şekil 3.14’de görülmektedir.



Şekil 3.14. Clock Yapılandırması

Yapılan özel donanımın sentezleme sonrası FPGA üzerinde kaynak kullanım sayısı ve oranları Tablo 3.7’de görülmektedir.

Tablo 3.7. Özel Yapılandırma Seçeneği Tercih Edildiğinde Kaynak Kullanımı

Kaynak Tipi	Kullanım Sayısı	Kullanılabilir Kaynak Sayısı	Kullanım %
LUT	14909	63400	23.52
LUTRAM	1686	190000	8.87
FF	15425	126800	12.16
BRAM	17	135	12.59
DSP	7	240	2.92
IO	52	210	24.76
BUFG	5	32	15.63
MMCM	1	6	16.67
PLL	2	6	33.33

Özel yapılandırma için elde edilen güç raporu Tablo 3.8'de görülmektedir.

Tablo 3.8. Özel Yapılandırma Güç Tüketimi

Kaynak	**Güç Tüketimi (W)**
Clocks	0.073
Signals	0.037
Logic	0.026
BRAM	0.030
DSP	0.002
PLL	0.115
MMCM	0.113
Phaser	0.110
I/O	0.530
XADC	0.004
PL Static	0.109

Eşitlik (3.1) kullanılarak toplam güç tüketimi 1.149 W olarak hesaplanmıştır.

Yukarıda anlatılan 3 farklı yapılandırma seçeneğine göre donanım Vivado Design Suite 'de sentezlemiş ve her bir duruma ait güç raporları oluşturulmuştur. Güç raporlarından görüldüğü üzere statik ve dinamik olmak üzere 2 tane güç tüketim tipi vardır. Bu 3 yapılandırma seçeneğine ait raporların karşılaştırılması Tablo 3.9'da görülmektedir.

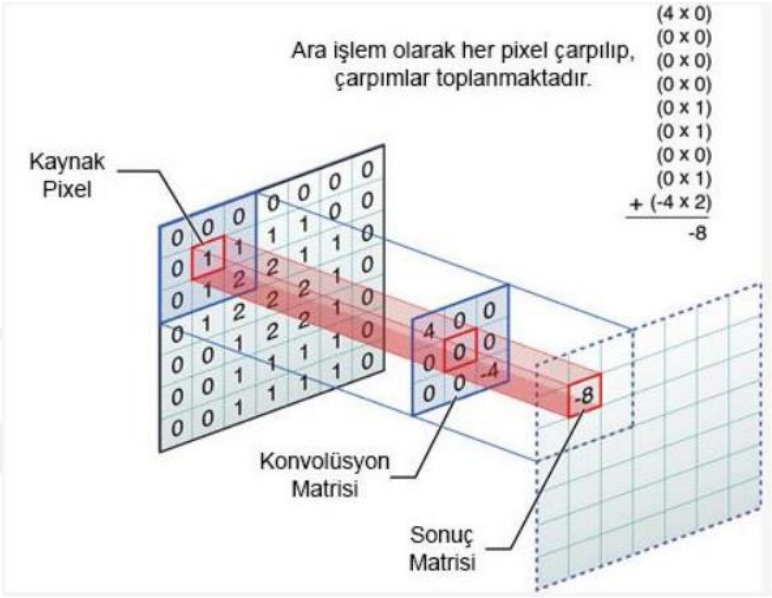
Tablo 3.9. Yapılandırma Seçenekleri için Güç Tüketimi ve Kaynak Kullanımı

Yapılandırma Seçeneği	**LUTs**	**FFs**	**Güç Tüketimi(W)**
Minumum Area	11914	12335	1.11
Maximum Performans	14944	15647	1.301
Uygulamaya Özel	14909	15425	1.149

Bu tezdeki uygulamaya yönelik özel yapılandırılan donanım seçeneğinde ise 1,149 Watt ile hem "Minumum Area" 'ya yakın bir güç tüketimi hemde iyi bir performans elde edilebilmiştir.

3.3 Konvolüsyon İşlemini Yapan Donanımın Gerçeklenmesi (IP Tasarımı)

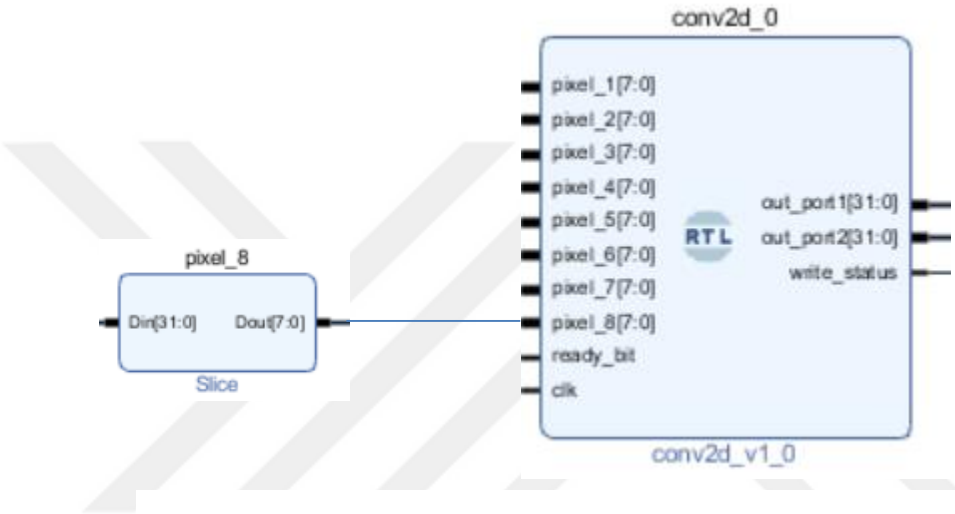
Bu tezde örnek uygulama olarak 2 boyutlu konvolüsyon işlemi ele alınmıştır. Konvolüsyon işlemi, iki boyutlu bir dizi üzerinde, bir kernel matrisinin dolaştırılarak bazı özelliklerin çıkarılmasını sağlar. Bu durumu açıklayan örnek Şekil 3.15'de gösterilmiştir.



Şekil 3.15. 2D Konvolüsyon

Bu çalışmada güç tüketimine odaklanıldığı için örnek uygulama olarak seçilen konvolüsyon işlemi içinde bir donanım hızlandırma devresi tasarlanmıştır. Bu sayede işlem süreleri önemli ölçüde azaltılarak toplam çalışma süresi dolayısıyla güç tüketiminin azaltılması amaçlanmıştır. Tasarlanan donanım, Microblaze soft işlemcisi ile I/O kullanarak haberleşmektedir. Nexys A7-100T geliştirme kartında 210 tane sınırlı sayıda port bulunur. Dolayısıyla tasarım yapılırken kartın sahip olduğu I/O port sayısı göz önünde bulundurulmalıdır. Matematiksel işlemlerin rahat yapılabilmesi ve donanım karmaşıklığını artırmamak için pikseller 8'erli gruplar halinde donanıma gönderilir. Bu tasarımda hem işlem kolaylığı sağlar; hemde yeterli sayıda I/O kullanım imkanı sunar. Bu yüzden uygulama, her defasında 8 piksel gönderimi yapacak şekilde tasarlanmıştır. Her bir piksel 8bit ile ifade edildiği için toplam 64 I/O pin kullanılmaktadır. Benzer şekilde hesaplanan değerlerin işlemciye geri gönderilmesi için de 64bitlik başka bir çıkış kullanılmıştır. 2 adet de I/O pini yapılandırma amacıyla kullanılmaktadır.

Tasarlanan işlemci 32 bitlik olduğu için I/O portlar yapılandırılırken maksimum 32 bit olabilmektedir. AXI GPIO bloğunda aynı anda en fazla 2 adet port aktif edilebilmektedir. Her 2 port aktif edildiğinde toplam 64 bitlik işlemler yapılabilmektedir. 32 bitlik mimariler için C dilinde "int" tipindeki veriler 32 bitlik alan kaplar. Her bir piksel 8 bitlik olduğundan," int" türünde tanımlanan bir değişken 4 adet piksel değeri tutabilir. Paketlenen bu veriler, Şekil 3.3'de bahsedildiği gibi "SLICE IP" bloğu ile 8 bitlik bölümlere ayrılır. Şekil 3.16'de bu kullanıma örnek olarak bir pikselin ayrılma işlemi gösterilmektedir.



Şekil 3.16. Piksel Verilerinin Ayrıştırılması

Process bloklar, işlemleri FPGA üzerinde eş zamanlı gerçekleştirilebilmektedirler. Bu tezde tasarlanan donanımda 3 tane process bloğu vardır: Birinci blok, saat işaretinin her yükselen kenarı için yeni bir veri gelip gelmediği kontrol etmektedir. Eğer veri geldiyse, 8 tane piksel değeri buffer'a alınır. Buffer'a alma işlemi bitene kadar yeni veri alınmayacağı işlemciye bir "ready_bit" ile bildirilir. Tüm veriler alındıktan sonra, "konvolusyon_basla" değişkeni set edilir. Bu değişken, ikinci process tarafından kontrol edilir. Eğer değişken set edildiyse, ikinci process 'de konvolüsyon işlemi başlatılır. Konvolüsyon işlemi bittiğinde ise "gönderim_basla" değişkeni set edilir ve hesaplanan değerler, işlemciye gönderilir. Gönderme işlemi üçüncü process tarafından gerçekleştirilir.

Gönderim algoritmasını göstermek amacıyla padding uygulanan örnek bir görüntü dizisi Şekil 3.17'de verilmiştir. Her kutucuk gray-scale'de bir piksel değerini ifade etmektedir. Bu görüntü dizisi, kernel matrisiyle konvolüsyona sokulacaktır. İlk adım olarak, bu

veriler donanıma aktarılmalıdır. Her iterasyonda, birinci satır sabit tutularak ilk 8 sütun değeri donanıma aktarılmaktadır. Donanım her 8'lik veri gurubunu için bir "buffer"'a doldurmaktadır. İkinci iterasyonda eğer sütun sonuna gelinmediyse, bir sonraki 8 sütunun değeri aktarılır. Bu işlem tüm sütun bitene kadar devam eder. Sütunda sona gelindiyse satır değeri bir artırılır ve aynı işlem tüm dizi elemanları aktarılana kadar devam eder. Tüm pikseller okunduktan sonra konvolüsyon işlemi yapılmaktadır. Ardından hesaplanan piksel değerleri işlemciye geri gönderilmemektedir.

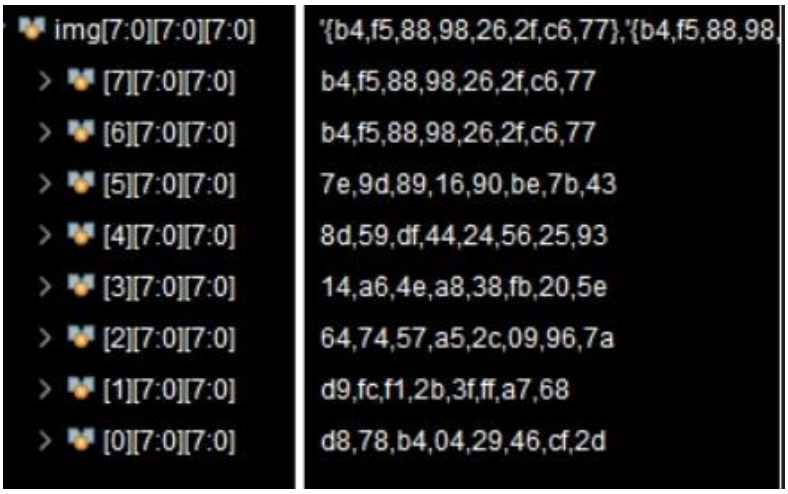
3.4 Konvolüsyon IP Tasarımının Doğrulanması

Oluşturulan konvolüsyon donanımının çalışmasını doğrulamak için bir test bench yazılmıştır. Sistemi doğrulamak için rastgele sayılardan oluşan temsili bir görüntüye ait iki boyutlu 8x8'lik bir dizi oluşturulmuştur. Konvolüsyon işlemi sonrasında dizi boyutunun değişmemesi için "padding" işlemi uygulanmıştır. "Padding" işlemi uygulanmış dizi Şekil 3.17'de görülmektedir.

0	0	0	0	0	0	0	0	0	0
0	2d	cf	46	29	4	b4	78	d8	0
0	68	a7	ff	3f	2b	f1	fc	d9	0
0	7a	96	9	2c	a5	57	74	64	0
0	c4	af	15	28	a4	e9	57	db	0
0	5e	20	fb	38	a8	4e	a6	14	0
0	93	25	56	24	44	df	59	8d	0
0	43	7b	be	90	16	89	9d	7e	0
0	77	c6	2f	26	98	88	f5	b4	0
0	0	0	0	0	0	0	0	0	0

Şekil 3.17. "Padding" Uygulanmış Dizi

Oluşturulan görüntü dizisi üzerinde konvolüsyon işleminin doğru yapıldığını doğrulamak için Vivado programında "testbench" hazırlanmıştır. Hazırlanan girdi dizisinin görüntüsü Şekil 3.18'de görülmektedir. Şekil 3.19'da ise konvolüsyon işlemi uygulanan 3x3'lük örnek matris ve kernel matrisi görülmektedir.



Şekil 3.18. Benzetim Girdi Matrisi

0	0	0
0	2d	cf
0	68	a7

×

1	1	1
1	1	1
1	1	1

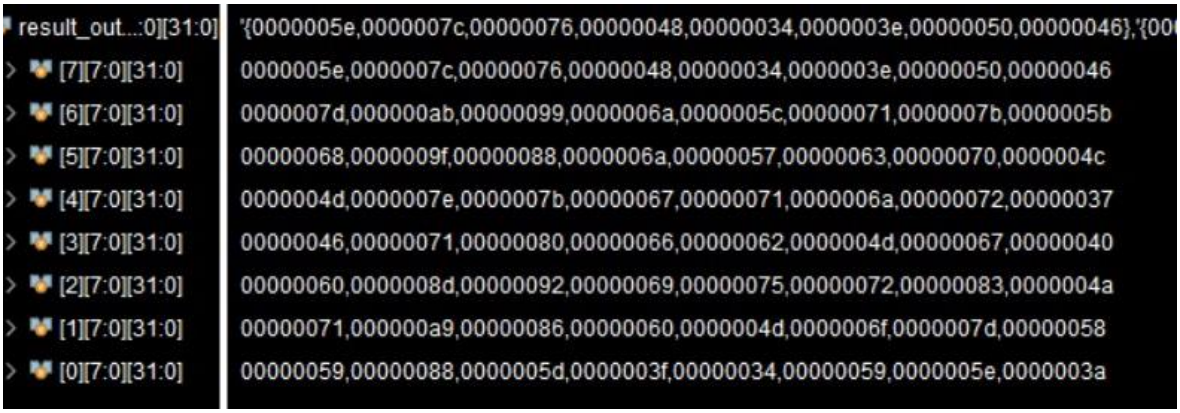
Şekil 3.19. Kernel Matrisi

Test senaryosunda kernel matrisi, her elemanı 1 olacak şekilde 3x3'lük tanımlanmıştır. Konvolüsyon işlemi için bazı örnek iterasyonlar Şekil 3.20'de gösterilmiştir.

$$Hedef\ Piksel = \begin{bmatrix} ((0*1)+(0*1)+(0*1))+ \\ ((0*1)+(0x2D*1)+(0xCF*1))+ \\ ((0*1)+(0x68*1)+(0xA7*1)) \end{bmatrix} /9$$

Şekil 3.20. Konvolüsyon Birinci Adım

İşlemin sonucu 0x20B çıkmaktadır. Bu değer kernel matrisi elemanlarını değerleri toplamına bölündüğünde sonuç 0x3A çıkmaktadır. Şekil 3.21'de Vivado benzetim ekranında sonuç matrisi görülmektedir. İlk piksel değeri 0x3A olarak hesaplanmıştır. Diğer değerlerinde doğru hesaplandığı benzer yöntemle doğrulanmıştır.



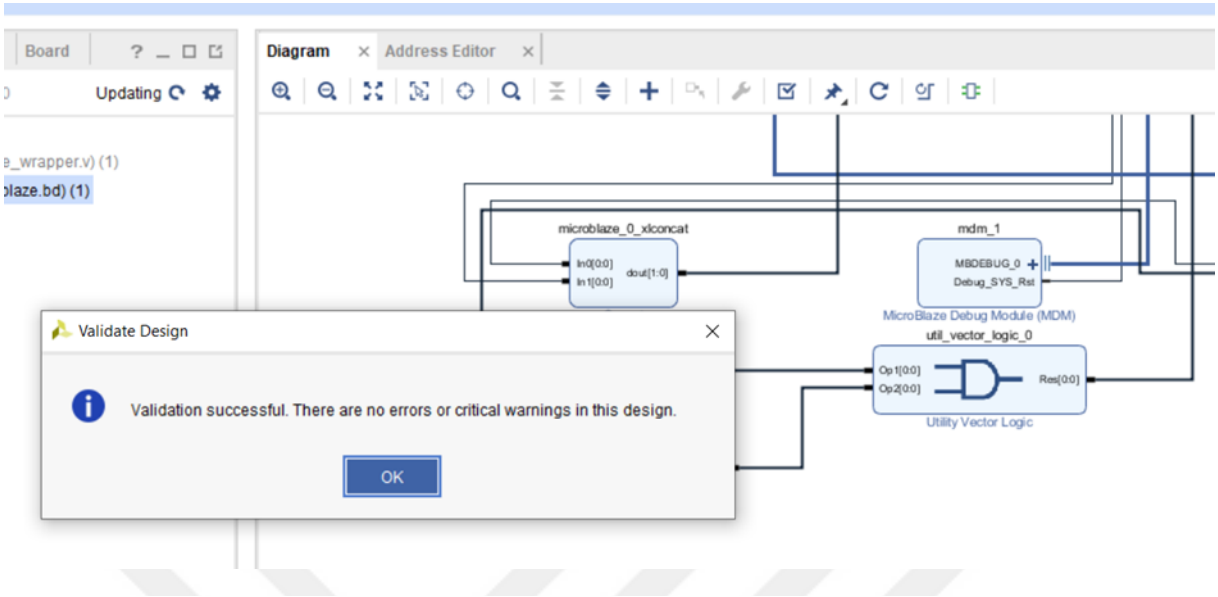
Şekil 3.21. Benzetim Çıktısı

Benzetim aşamasında tasarlanan donanımın doğru çalıştığı görüldükten sonra bir sonraki adımda "bitstream" dosyası oluşturulmalıdır. Bir sonraki bölümlerde üretilen "bit" dosyasının FPGA'ya nasıl gömüleceği üzerine detaylı bilgiler verilmiştir.

3.5 Bitstream Dosya Oluşturma

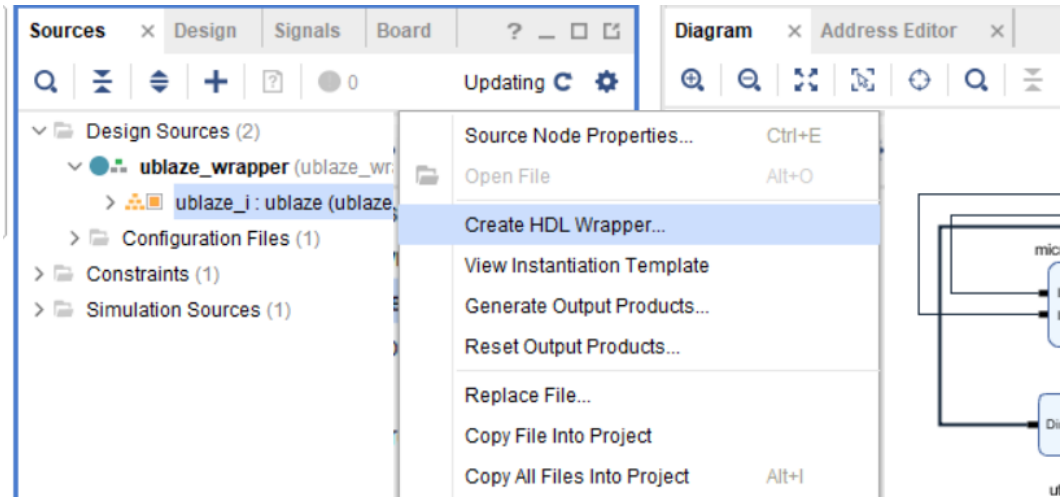
Vivado içerisinde bulunan "IP GENERATOR" sekmesi altında bir proje oluşturulur. Yukarıda bahsedilen gerekli tüm donanım tasarımlarının yapılması ve 3 farklı duruma göre sentezleme ve gerçekleştirme işlemi sonrasında 3 donanım elde edilir. FPGA gömülecek dosyanın oluşturulması için aşağıdaki işlemler sırayla yürütülür:

1- "Validate design" seçeneğiyle tasarımda bir hata olup olmadığı kontrol edilir. Şekil 3.22'den de gözüktüğü gibi herhangi hata olmadığında başarılı mesajı elde edilmiş olur.



Şekil 3.22. "Validate Design"

2- Şekil 3.23'de gösterildiği gibi "Generate HDL Wrapper" seçilerek yapılan blok tasarımı, donanım tanımlama diline dönüştürülür.

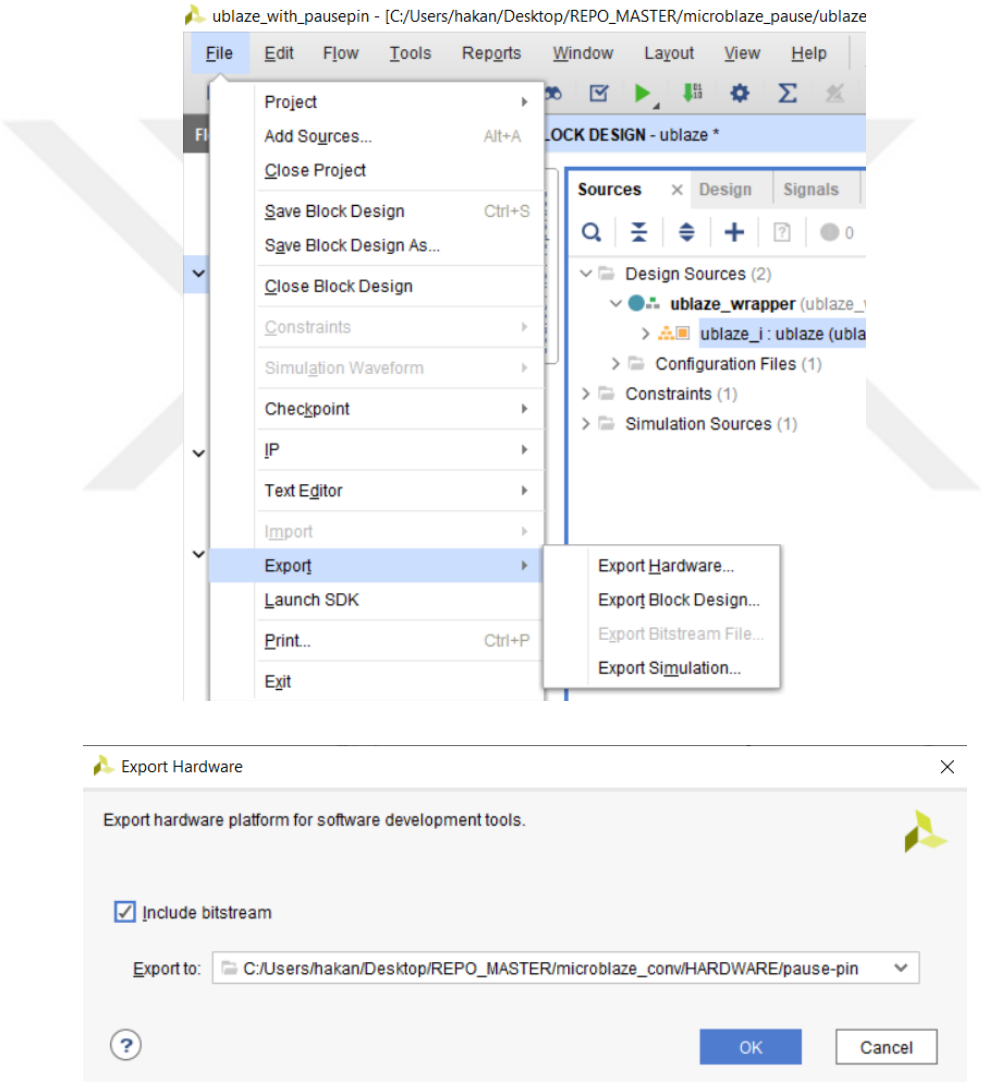


Şekil 3.23. "HDL Wrapper"

3- Bu adımlar tamamlandıktan sonra sırasıyla, “Run Synthesis”, “Run Implemantation” ve “Generate Bitstream” aşamaları çalıştırılır. Bu işlemlerde başarıyla tamamlandıktan sonra “bit” dosyası elde edilir.

3.6 Donanımın SDK’ya Aktarılması

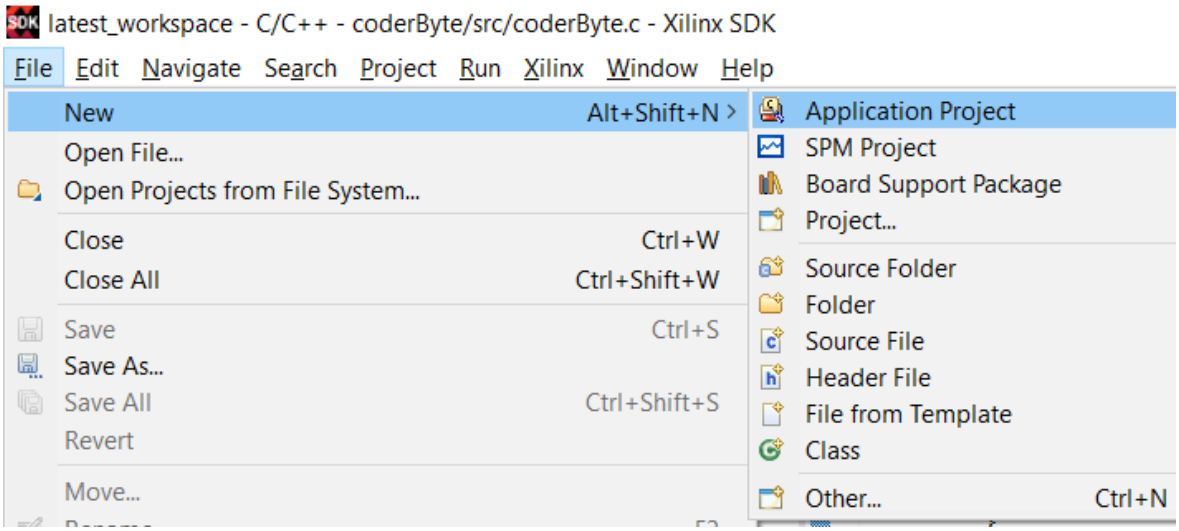
Bölüm 3.5’de bahsedilen adımlar tamamlandıktan sonra, donanımın FPGA ’de çalışabilmesi için Şekil 3.24’de gösterildiği gibi projenin SDK’ya aktarılması gerekmektedir.



Şekil 3.24. Donanımın SDK'ya Aktarılması

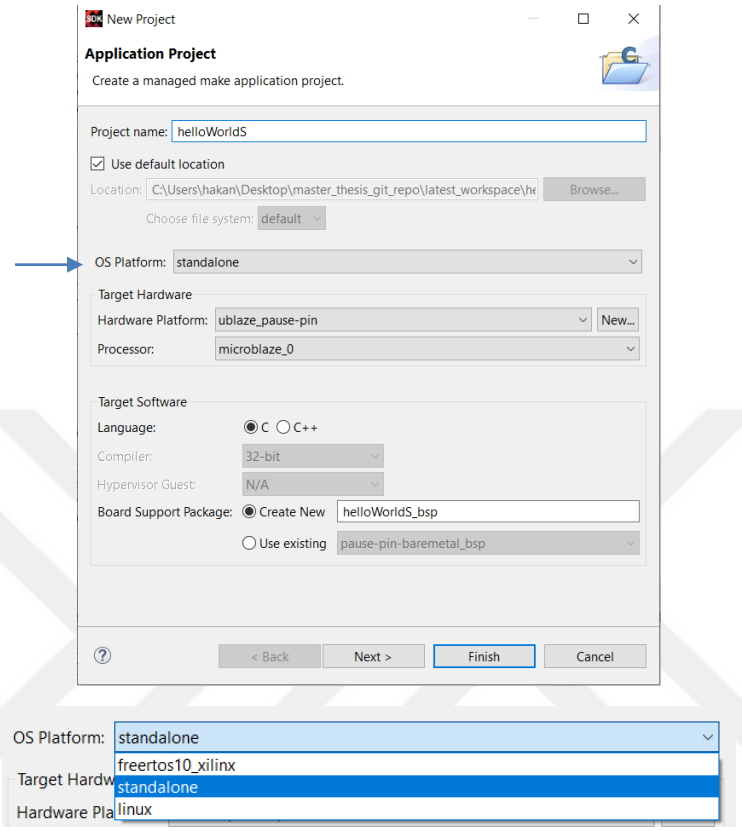
4. UYGULAMANIN GERÇEKLENMESİ

Bu bölümde, tasarlanılan konvolüsyon IP bloğunun ve yeniden yapılandırılan microblaze donanımın güç tüketimi açısından etkisini inceleyebilmek için yapılan işlemler anlatılmıştır. Uygulamanın, tasarlanan donanım üzerinde çalıştırılabilmesi için Xilinx SDK'nin kullanılması gerekmektedir. Xilinx SDK, Xilinx tabanlı işlemciler üzerinde gömülü yazılım uygulamaları geliştirmek için tümleşik geliştirme ortamıdır (Integrated Development Environment – IDE). Böyle bir uygulamayı microblaze'de çalıştırabilmek için öncelikle Şekil 4.1'den de görüldüğü gibi bir "Application Project" oluşturulur.



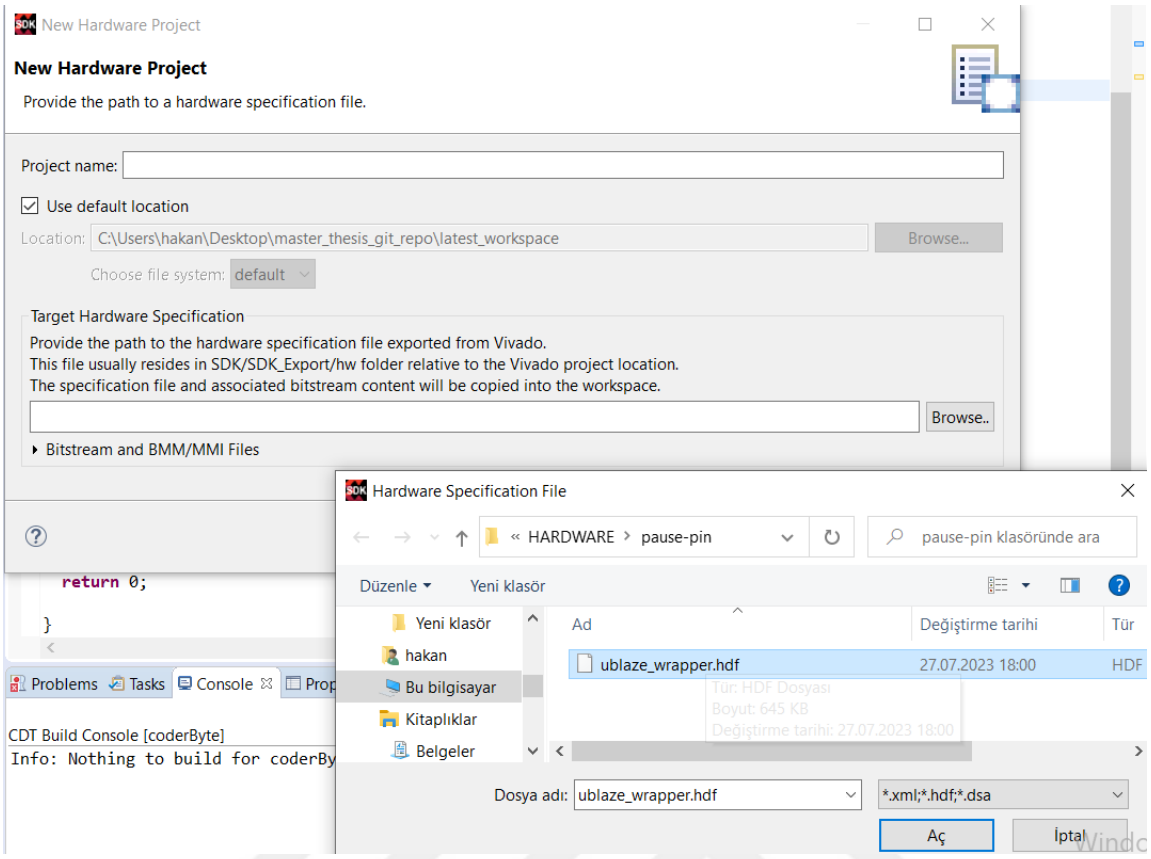
Şekil 4.1. Uygulama Projesi Oluşturma

Şekil 4.2'de verilen arayüz ekranında proje ismi isteğe göre verilebilir. OS platform altında, 3 tane seçenek bulunmaktadır: 1) "Standalone"; bare-metal olarak isimlendirilen ve herhangi bir ara katman olmadan, uygulama kodlarının doğrudan işlemci üzerinde çalıştırılması durumudur. 2)"Freertos10_Xilinx" kurulan FreeRTOS kütüphaneleri sayesinde, projenin FreeRTOS servislerini kullanma imkanının olduğu durumdur. 3)" Linux" ise Xilinx 'in işlemcilerine yönelik hazırlanan "petalinux" işletim sistemi kullanımına olanak sağlayan çalışma durumudur. Bu tez kapsamında standalone ve FreeRTOS seçenekleriyle uygulama geliştirilmiştir.



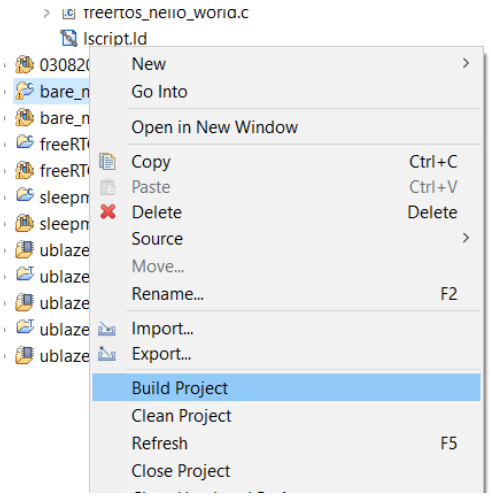
Şekil 4.2. Uygulama Proje Ayarları

Şekil 4.3'de görüldüğü gibi yeni bir donanım projesi oluşturulurken, "Hardware Platform" seçeneğinde Vivado 'dan aktarılan "hdf" uzantılı dosyayı seçmek gerekmektedir. Bu "hdf" dosyası yapılan donanıma ait bilgileri içerir ve o donanıma ait donanım destek paketi dosyası oluşturulur. Bu sayede donanımda bulunan uart, timer vb. modüller kullanılabilir hale gelmektedir.



Şekil 4.3. Donanım Tanımlama Dosyasının Aktarılması

Bu işlemlerden sonra, Vivado 'da tasarlanılan bsp'ye ait bir proje oluşturulmuş olur. Kodu derlemek için Şekil 4.4'de görüldüğü gibi "Build Project" butonuna basılır.



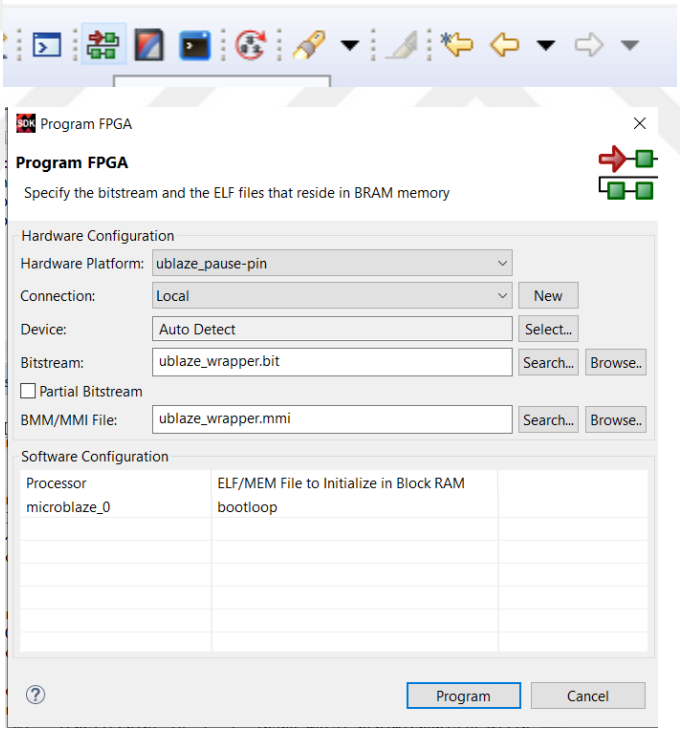
Şekil 4.4. Uygulamanın Derlenmesi

Derleme işleminde sonra eğer bir hata yok ise kod, Microblaze mimarisine yönelik derlenir ve “elf” formatında bir dosya oluşturulur. Derleme çıktısına ait başarılı olma bilgisi Şekil 4.5’de görülmektedir.

```
'Invoking: MicroBlaze Print Size'
mb-size bare_metal.elf  |tee "bare_metal.elf.size"
   text    data     bss     dec     hex filename
   2308     324    3112    5744    1670 bare_metal.elf
'Finished building: bare_metal.elf.size'
' '

01:26:25 Build Finished (took 1s.313ms)
```

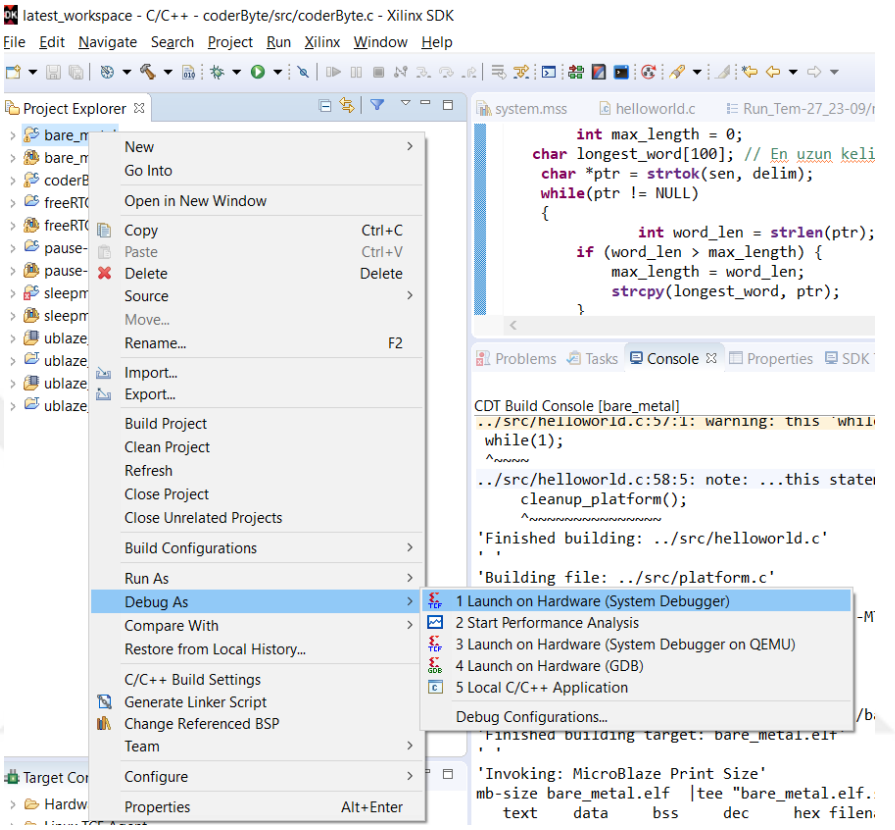
Şekil 4.5. Derleme Çıktısı

Uygulamayı çalıştırmadan önce ilk olarak “bit” dosyası FPGA’ya gömülmelidir. Bu işlemden sonra artık FPGA içerisinde tasarlanılan donanım çalışıyor olacaktır. Bunun için Şekil 4.6’da gösterildiği gibi “Program FPGA” menüsü kullanılır.



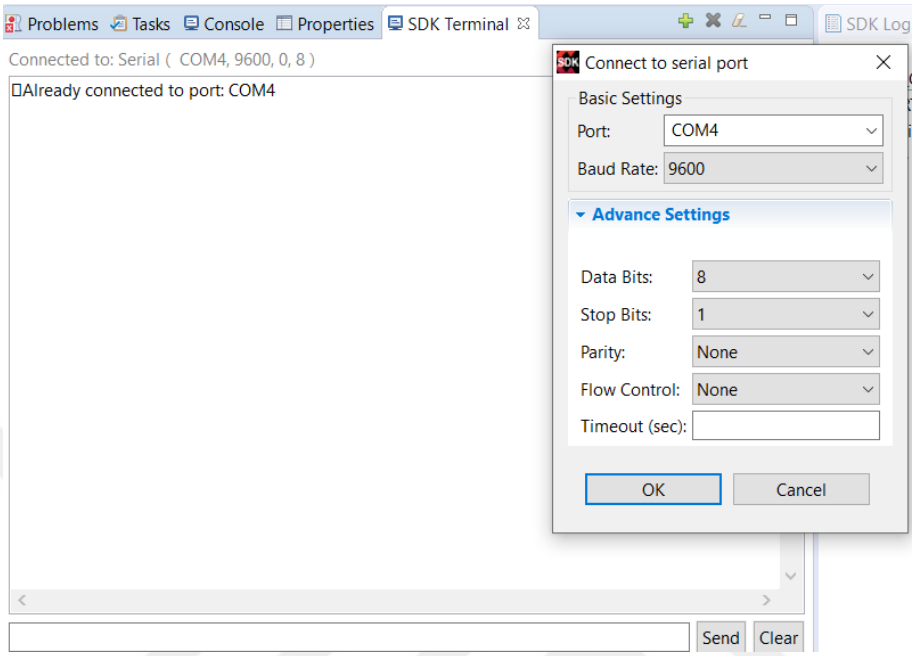
Şekil 4.6. Bit Dosyasının FPGA’ya Gömülmesi

Bit dosyası FPGA gömüldükten sonra artık tasarlanan donanım üzerine, geliştirilen uygulama yazılımı atılabilir. Şekil 4.7'den görüldüğü gibi uygulama, debug arayüzü kullanılarak Microblaze' e yüklenir.



Şekil 4.7. Uygulamanın Donanıma Yüklenmesi

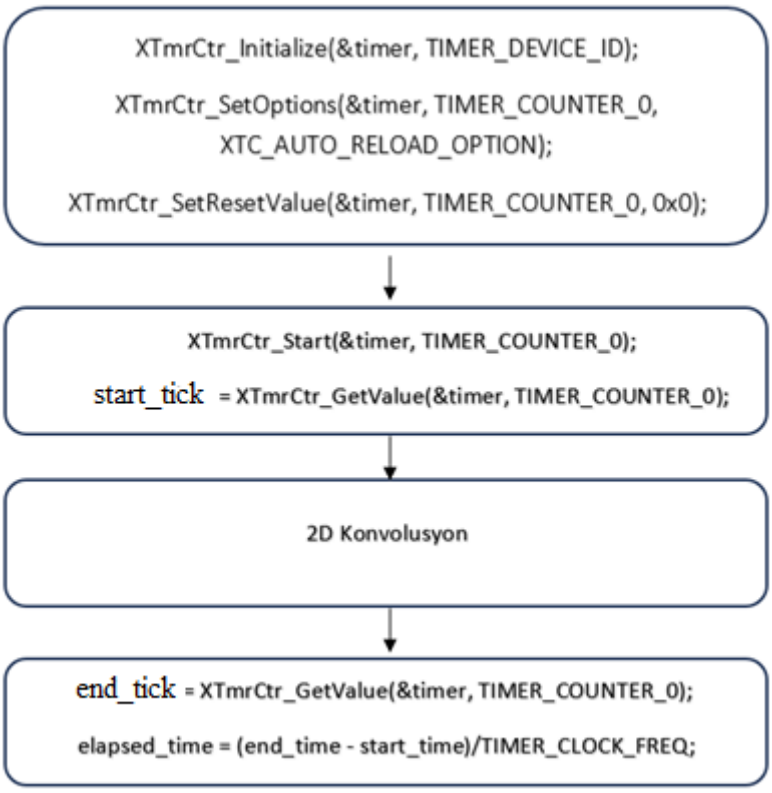
UART üzerinden çıktıları görüntüleyebilmek için SDK Terminal menüsünden Şekil 4.8'de gösterildiği gibi yeni bir bağlantı açılmalıdır. Uygun port ve baudrate seçildikten sonra çıktılar bu arayüzden görüntülenebilir.



Şekil 4.8. Terminal Penceresi

5. GÜÇ ÖLÇÜMLERİ VE KARŞILAŞTIRMALAR

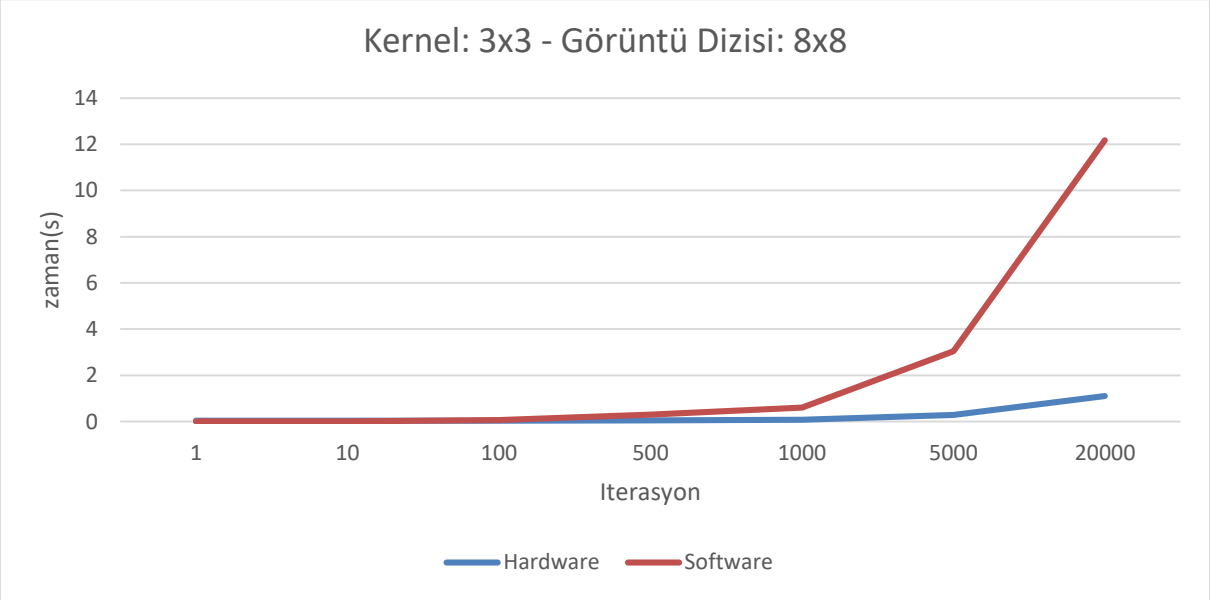
Konvolüsyon uygulaması iki farklı şekilde geliştirilmiştir: 1) yazılımsal olarak, 2) donanımsal olarak. Yazılımsal olarak geliştirilen uygulamada konvolüsyon işlemi, C dilinde kodlanan ve konvolüsyon işlemini yazılımsal olarak yerine getiren durumdur. Donanımsal uygulamada ise konvolüsyon işlemi, Verilog donanım tanımlama dili ile oluşturulmuş IP blok ile gerçekleştirilen durumdur. Her iki durum içinde görüntü boyutu, kernel boyutu ve iterasyon parametreleri değiştirilerek uygulama tekrarlanmış ve çalışma süreleri ölçülmüştür. Uygulama süresini ölçmek için işlemcinin sistem "clock'unu referans alan timer'lar kullanılmıştır. 100 MHz'lik timer, saniyede 100.000.000 defa saymaktadır. Her iki durumda da uygulama gerçekleme süreleri Şekil 5.1'de görülen yöntem ile ölçülmüştür. Bu "tick" ölçümleri eşitlik (5.1) kullanılarak saniye türünde ifade edilmiştir. Şekil 5.1'de verilen "elapsed_time" ifadesi konvolüsyon işlemi için geçen zamanı saniye cinsinden ifade eder.



Şekil 5.1. Tick Okuma Metodu

$$elapsed\ time(s) = (end_tick - start_tick)/TimerClockFreq \quad (5.1)$$

8x8'lik görüntü boyutu sabit tutularak, iterasyon sayısı 1, 10, 100, 500, 1000 ve 5000 olmak üzere 6 farklı değer verilerek hem yazılımsal hemde donanımsal tasarımlara ilişkin gerçekleme süresi ölçümleri yapılmıştır. Donanımsal ve yazılımsal uygulamalarda alınan ölçümler eşitlik (5.2)'de kullanılarak yüzdelik olarak iyileştirme oranı cinsinden ifade edilmiştir. Elde edilen iyileştirme oranlarına göre sonuçlar Şekil 5.2'de görülmektedir. Gerçekleme süresi, donanımsal olarak tasarlanan konvolüsyon bloğu işlemci ile I/O port üzerinden haberleştiği için düşük iterasyonlarda yazılımsal tasarıma göre daha uzun olmuştur. Çünkü yazılımsal tasarımda herhangi bir I/O port veya bekleme olmadan doğrudan konvolüsyon işlemi gerçekleştirilmektedir. Fakat gerçek kullanım senaryosunda konvolüsyon işlemi bir defa değil, n defa tekrarlanacağı için bu durum yazılımsal tasarımın üstünlüğünü göstermemektedir.

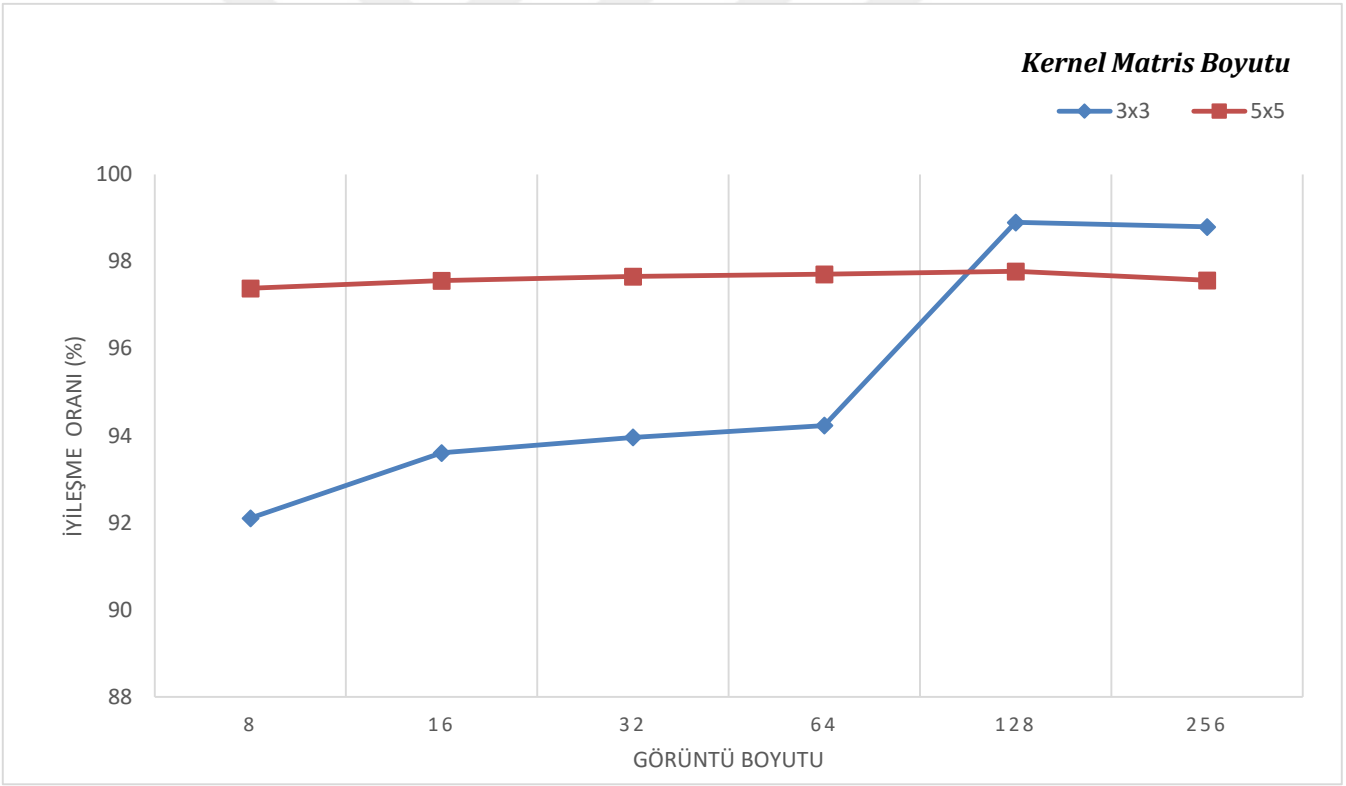


Şekil 5.2. Gerçekleme Süresi Üzerinde İterasyonun Etkisi

$$\text{İ}yileşme\ Oranı = \frac{Yazılımsal_Konvolüsyon - Donanımsal_Konvolüsyon}{Yazılımsal_Konvolüsyon} * 100 \qquad (5.2)$$

Eşitlik (5.2)'te verilen matematiksel işlemin sonucunun (+) pozitif çıkması, donanımsal uygulamanın yazılımsal uygulamaya göre daha az sürede gerçekleştiği ve iyileşme

sağladığını ifade eder. (-) negatif çıkması ise, yazılımsal uygulamanın donanımsal uygulamaya göre daha iyi performans gösterdiği anlamına gelir. Bu ölçüm senaryosunda ise iterasyon sayısı artırılarak, farklı görüntü boyutlarında (8x8, 16x16, 32x32, 64x64, 128x128 ve 256x256) ve kernel boyutu değiştirilerek (3x3, 5x5) için 36 adet farklı duruma ait gerçekleme süreleri ölçülmüştür. Sonuçlar Şekil 5.3'de gösterilmiştir. Kernel boyutu küçüldüğünde, görüntü üzerinde yapılacak iterasyon sayısı arttığı için daha fazla çarpma işlemi gerçekleştirilmektedir. Dolayısıyla küçük kernel boyutu kullanılan durumlarda iyileşme, büyük kernel boyutu kullanımına göre daha az olduğu görülmüştür. Şekil 5.3'de dikey eksende ifade edilen iyileşme oranı için eşitlik 5.2 kullanılarak; donanımsal konvolüsyon işleminin yazılımsal konvolüsyon işlemine göre iyileştirme oranları hesaplanmıştır. Şekil 5.3'de kernel matrisinin boyutu arttığında, işlem gücü artacağından bu durumun uygulama süresine etkisi gösterilmiştir.



Şekil 5.3. Kernel Boyutunun İyileşme Oranına Etkisi

Yüksek iterasyon sayılarında yazılımsal ve donanımsal gerçekleşme süresi oranı çok fazla değişmemektedir. Bu nedenle belirli bir değerden sonra iterasyon sayısını artırmak daha fazla iyileşme sağlamadığından işe yaramamaktadır.

Tablo 5.1. Farklı Görüntü Boyutu ve Kernel Matrisi için İyileşme Oranları

Görüntü Boyutu	Kernel Boyutu	Iterasyon	Konvolüsyon Uygulama Süresi (s)		İyileşme Oran (%)	Ortalama
			Yazılımsal	Donanımsal		
8x8	3x3	1800	1,095449	0,089961	91,787	92,194
		3600	2,190857	0,159218	92,732	
		5400	3,417802	0,271232	92,064	
	5x5	1800	2,60135	0,067398	97,409	97,382
		3600	5,12544	0,134785	97,370	
		5400	7,689917	0,20249	97,366	
16x16	3x3	1800	4,541113	0,298659	93,423	93,602
		3600	9,082153	0,576625	93,651	
		5400	13,62319	0,853551	93,734	
	5x5	1800	11,1959	0,27001	97,588	97,559
		3600	21,99136	0,540027	97,544	
		5400	32,987	0,810043	97,544	
32x32	3x3	1800	18,57479	1,130427	93,914	93,960
		3600	37,14938	2,239064	93,972	
		5400	55,72394	3,34666	93,994	
	5x5	1800	46,57457	1,080014	97,681	97,653
		3600	91,49097	2,160015	97,639	
		5400	137,2364	3,240017	97,639	
64x64	3x3	1800	77,07729	4,454063	94,221	94,231
		3600	154,0716	8,885487	94,232	
		5400	231,2318	13,31524	94,241	
	5x5	1800	190,584	4,319789	97,733	97,706
		3600	374,3828	8,639528	97,692	
		5400	561,5737	12,95927	97,692	
128x128	3x3	1800	309,6661	3,432399	98,891	98,895
		3600	619,3323	6,847352	98,894	
		5400	928,9899	10,20255	98,901	
	5x5	1800	784,6793	17,27732	97,798	97,771
		3600	1541,424	34,55444	97,758	
		5400	2312,129	51,83156	97,758	
256x256	3x3	1800	1241,392	15,03353	98,788	98,793
		3600	2482,789	29,99065	98,792	
		5400	3724,143	44,68607	98,800	
	5x5	1800	3145,629	75,67277	97,594	97,565
		3600	6179,268	151,3447	97,550	
		5400	9268,897	227,0166	97,550	

Örnek olarak 32’lik bir input için 1800, 3600 ve 5400 iterasyonluk ölçümlerin oranları Tablo 5.2‘de verilmiştir. Görüldüğü üzere iyileşme oranı çok ufak hassasiyetle değişmektedir.

Tablo 5.2. 32x32’lik Görüntü ve 5x5’lik Kernel için Ortalama İyileşme Oranı

Iterasyon	**Konvolüsyon Uygulama Süresi (s)**		**İyileşme Oranı (%)**
	Donanımsal	***Yazılımsal***	
5400	3,24	137	97,63
3600	2,16	91	97,64
1800	1,08	47	97,68

Görüntü boyutu, kernel matris boyutu ve iterasyon sayısına ek olarak, donanıma eklenen “branch target”, “mul-divider”, “barrel shifter” gibi modüllerin etkisini incelemek için bir dizi ölçüm daha yapılmıştır.

"Branch target," bilgisayar bilimlerinde bir işlemcinin komut akışını kontrol etmek için kullanılan bir terimdir. İşlemciler, bir programı yürütürken sıralı olarak komutları işlerler. Ancak, koşullu dallanmalar (conditional branches) veya atlamalar (jumps) gibi komutlar program akışını değiştirebilir. Örneğin, "if-else", “switch-case” ya da “loop” ifadeleri bir programın farklı bölümlerine atlamasına neden olabilir. Bu kavramlar her programda kullanımına ihtiyaç duyulan yapılardır. "Branch target," bu tür koşullu dallanma veya atlamaların hedef adresini ifade eder. Yani, bu komutların nereye gitmesi gerektiğini belirtir. İşlemci, bu hedef adresi bilmek zorundadır çünkü hangi komutun işleneceğini belirlemelidir. Ancak, koşullu dallanmaların hedefleri programın çalışma zamanında belirlendiğinden, işlemci önceden bu hedef adresini tam olarak bilemez. İşte burada "branch target prediction" devreye girer. İşlemci, geçmiş koşullu dallanma davranışını analiz ederek veya belirli algoritmaları kullanarak bu hedef adresini tahmin etmeye

çalışır. Eğer tahmin doğruysa, işlemci doğru yoldaki komutları işleyerek yürütme hızını artırır. Ancak tahmin yanlışsa, işlemci bu yanlış yolu düzelterek zaman kaybedebilir.

“Branch Target” modülünün devrede olmasının etkisini görebilmek için yukarıda ki örnekte de açıklandığı gibi uygulama katmanında bulunan “if-else”, “switch-case” ve “loop” döngülerinin yapıldığı kısımların ölçümleri alınmıştır. Programda bu ifadelerin kullanıldığı kısımlar için, “branch prediction” aktif ve pasifken “tick” olarak ölçülmüştür. Ölçümlere ait veriler Tablo 5.3’de görülmektedir.

Tablo 5.3. “Branch Target” Modülünün Etkisi

Iterasyon	**Donanım Durumu**		**İyileşme Oranı (%)**
	Pasif	***Aktif***	
1	333	166	50,150
100	2931	2752	6,107
500	13732	12562	8,520
1000	27232	24800	8,930
5000	135144	122737	9,180
20000	540144	490237	9,239

“Barrel Shifter” modülü, verilerin belirli bir sayıda bit sağa veya sola kaydırılmasını gerçekleştirir. Bu tür kaydırma işlemleri, aritmetik, mantıksal ve veri manipülasyonu gibi çeşitli işlemlerde kullanılır. “Barrel shifter”'lar, işlemcilerin ve diğer dijital sistemlerin performansını artırmak ve veri işleme yeteneklerini optimize etmek için kullanılır. “Barrel shifter”'lar genellikle işlemcilerde, sayısal sinyal işleme devrelerinde, veri işleme birimlerinde ve çeşitli dijital tasarım uygulamalarında kullanılır. Veri işleme hızını artırmak ve işlemci performansını optimize etmek istendiğinde “barrel shifter”'lar etkili bir şekilde kullanılabilir. Uygulama içerisinde veri paketleme, çözümleme işlemlerinde aritmetik shift ve lojik işlemler sık yapıldığı için bu kısımların ölçümü “tick” olarak yapılmıştır. Tablo 5.4’de ölçüm sonuçları görülmektedir.

Tablo 5.4. "Barrel Shifter" Modülünün Etkisi

Iterasyon	Donanım Durumu		İyileşme Oranı (%)
	Pasif	***Aktif***	
1	3674	4684	21,562
100	308237	391777	21,323
500	1539036	1955360	21,291
1000	3077533	3909860	21,287
5000	15385536	19545892	21,285
20000	61540537	78180881	21,284

Çarpma işlemi işlemcinin en temel matematiksel operasyonlarından biridir. Özellikle büyük veri setlerinde veya yoğun hesaplama gerektiren uygulamalarda çarpma işleminin hızlı yapılması önemlidir. "Mul32" ve "divider" donanım modülleri, işlemcilerde matematiksel işlemleri gerçekleştirmek için kullanılan önemli bileşenlerdir. Her ikisi de hesaplama yoğun uygulamalarda ve genel olarak sayısal işleme gerektiren birçok durumda kritik rol oynar. Bu modüllerin aktif ve pasif olduğu durumlarda; bu modüle kullanıma ihtiyaç duyan programlar için "tick" cinsinden alınan ölçümler Tablo 5.5'de görülmektedir.

Tablo 5.5. "Mul/Div" Modüllerinin Etkisi

Iterasyon	Donanım Durumu		İyileşme Oranı (%)
	Pasif	***Aktif***	
1	96249	63573	33,949
100	9377839	6119433	34,745
500	46879035	30587837	34,751
1000	93755554	61173330	34,752
5000	468767651	305857200	34,752
20000	1875062674	1223422190	34,752

5.1 FreeRTOS ve Bare-metal için İşlemci Kullanım Yüzdeleri

FreeRTOS (Real-Time Operating System), gömülü sistemlerde gerçek zamanlı uygulamaları kolaylaştırmak ve yönetmek için kullanılan bir işletim sistemidir. Bare-metal (donanım üzerinde) programlama ise işletim sistemi olmadan doğrudan donanım üzerinde çalışmayı ifade eder. Bu bölümde, konvolüsyon işlemi bare-metal ve FreeRTOS yaklaşımlarıyla gerçekleştirilip bu iki yaklaşımın ölçümleri alınmış ve sonuçlar karşılaştırılmıştır.

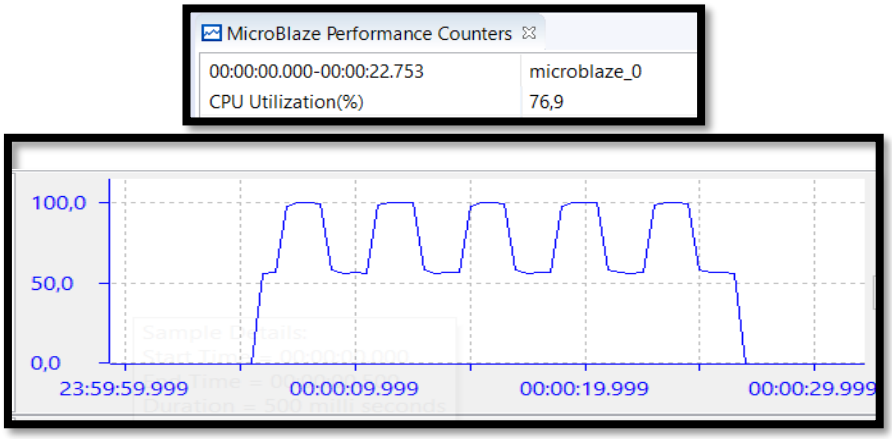
- FreeRTOS ’da çekirdek görevleri, zaman dilimlerine ve önceliklere göre işlenir. Bare-metal ’de ise işletim sistemi olmadan doğrudan donanım üzerinde çalışmayı içerir. Uygulama doğrudan işlemciye erişir.
- FreeRTOS, farklı öncelik seviyelerine sahip görevler arasında geçiş yapabilir. Genellikle tek bir ana iş parçası veya döngü bulunur. Çoklu görev desteği sağlanmaz.
- FreeRTOS, öncelikli görevlere daha fazla işlemci zamanı ayrılabilir. Bu, enerji verimliliğini artırabilir. Bare-metal programlamada işlemci, sürekli olarak ana döngüyü çalıştırır ve görev öncelikleri yoktur. Bu, işlemcinin belirli görevlere daha fazla veya daha az zaman harcamasını zorlaştırabilir ve enerji verimliliğini düşürebilir.
- FreeRTOS kullanımı, görevleri ve iş parçacıklarını yönetme karmaşıklığını azaltabilir. Ancak, işletim sistemi kodu daha fazla bellek ve kaynak tüketebilir. Bare-metal programlama, daha düşük seviyede kontrol sağlar, ancak iş parçacığı yönetimi ve senkronizasyonu gibi görevleri manuel yönetmek gerekebilir.

Multithread bir uygulamada ortak kaynak kullanımının uygulama süresine etkisini göstermek adına örnek bir uygulama yapılmıştır. Bare-metal uygulamada 2 farklı fonksiyon tarafından ekrana bir mesaj yazılmakta ve 1 saniye boyunca bekleme yapılmaktadır. Bu işlem n defa tekrarlanmaktadır. Aynı işi FreeRTOS ile gerçekleştirmek içinde 2 tane task oluşturulup, 1 saniyelik uyumayla işlemler tekrarlanmıştır. FreeRTOS 'da görev oluşturma için yapılan işlemler Şekil 5.4'de gösterilmiştir.

```
static void prvTxTask( void *pvParameters )
{
const TickType_t x1second = pdMS_TO_TICKS( DELAY_1_SECOND );

    for( ;; )
    {
        /* Delay for 1 second. */
        vTaskDelay( x1second );

        /* Send the next value on the queue.  The queue should always be
        empty at this point so a block time of 0 is used. */
        xQueueSend( xQueue,          /* The queue being written to. */
                    HWstring, /* The address of the data being sent. */
                    0UL );          /* The block time. */
    }
}

/*-----------------------------------------------------------*/
static void prvRxTask( void *pvParameters )
{
char Recdstring[15] = "";

    for( ;; )
    {
        /* Block to wait for data arriving on the queue. */
        xQueueReceive(  xQueue,             /* The queue being read. */
                        Recdstring, /* Data is read into this address. */
                        portMAX_DELAY );    /* Wait without a timeout for data. */

        /* Print the received data. */
        xil_printf( "Rx task received string from Tx task: %s\r\n", Recdstring );
        RxtaskCntr++;
    }
}
```

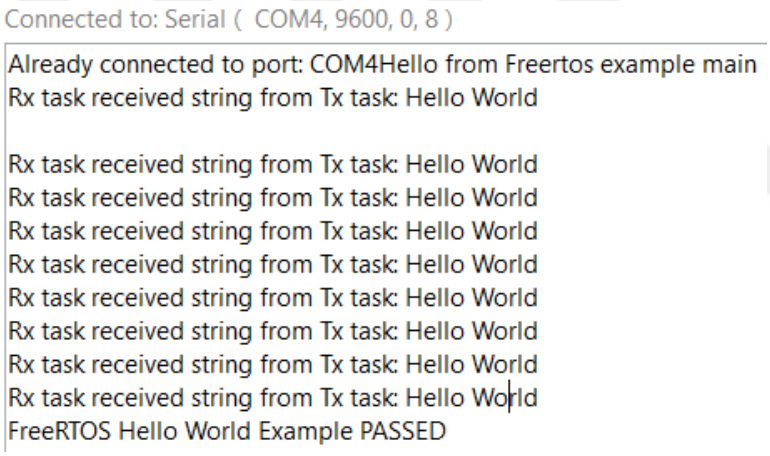
Şekil 5.4. Uygulama FreeRTOS Kodu

FreeRTOS ile hazırlanan uygulamaya ait işlemci kullanım yüzdesi Şekil 5.5'de gösterilmiştir.



Şekil 5.5. FreeRTOS İşlemci Kullanım Oranı

Uygulamanın seri çıktısı Şekil 5.6'da gösterilmiştir.



Şekil 5.6. FreeRTOS Uygulama Çıktısı

Bare-metal olarak örnek uygulamayı gerçekleştirmek için "a_print" ve "b_print" isminde 2 fonksiyon oluşturulmuştur. Main fonksiyon içerisinde 5 iterasyonluk bir döngü kurulup, bu iki fonksiyon döngü içerisinde toplamda 5 defa çağrılmıştır. İlk fonksiyon ekrana mesajını bastırdıktan sonra 1 saniye uyumaktadır. Diğer fonksiyonunu başlaması için ilk fonksiyonun uykudan uyanması gerekmektedir. Dolayısıyla uygulamanın çalışması için minimum 5*2 = 10 s geçmesi gerekmektedir. Şekil 5.7'de uygulama kodu, Şekil 5.8'de bare-metal uygulmaya ait seri çıktısı ve Şekil 5.9'da ise işlemci kullanım yüzdesi gösterilmiştir.

```c
int main()
{
    init_platform();

    /* Enable non-critical exceptions */
    Xil_ExceptionEnable();

    int iter = 5;
    while (iter--)
    {
        a_print(iter);
        b_print(iter);
    }

    xil_printf( " END\r\n" );


    cleanup_platform();
    return 0;
}

void a_print(int index)
{
    xil_printf( "\r\n a print: index: %d",index);
    sleep(1);
}

void b_print(int index)
{
    xil_printf( "\r\n b print: index: %d",index);
    sleep(1);
}
```

Şekil 5.7. Bare-metal Uygulama Kodu

```
Already connected to port: COM4
a print: index: 4
b print: index: 4
a print: index: 3
b print: index: 3
a print: index: 2
b print: index: 2
a print: index: 1
b print: index: 1
a print: index: 0
b print: index: 0 END
```

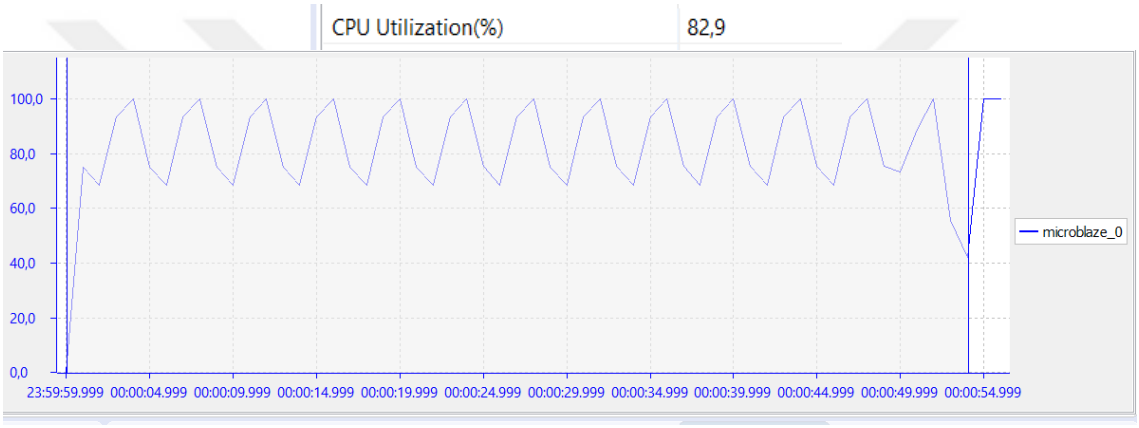
Şekil 5.8. Bare-metal Uygulama Çıktısı

00:00:00.000-00:00:32.029	microblaze_0
CPU Utilization(%)	99,8
CPU Cycles Per Instruction	2,39
Data Cache Access per ms	0,13
Data Cache Miss Rate(%)	10,4
CPU Stall Cycles Per Instruction	0,00
Minimum Instruction Cache Read L...	0,00
Maximum Instruction Cache Read ...	118,0

Şekil 5.9. Bare-metal Uygulama İşlemci Kullanım Yüzdesi

5.2 Konvolüsyon İşleminin FreeRTOS ile Gerçekleştirilmesi

Giriş dizisi 32x32'lik boyuta 1800 iterasyonluk bir konvolüsyon işlemi FreeRTOS kullanılarak gerçekleştirilmiştir. Bunun için 2 tane task oluşturulmuştur. İlk task, 4 saniyede bir yeni görüntü verisinin gelip gelmediğini kontrol etmektedir. Eğer yeni görüntü verisi geldiyse, giriş dizisine aktarılmaktadır ve verinin alındığına dair global bir değişkenin değeri ayarlanır. Bu değişken diğer task tarafından erişilebilir olduğu için sürekli olarak ayarlanıp ayarlanmadığı kontrol edilir. Eğer değişkenin değeri ayarlandıysa ikinci task konvolüsyon işlemi gerçekleştirir. Konvolüsyon işlemi için gerçekleştirilen işlemci kullanım oranı Şekil 5.10'da gösterilmiştir. Toplam uygulama süresi boyunca CPU kullanım oranı %82,9 olarak ölçülmüştür.



Şekil 5.10. FreeRTOS ile Konvolüsyon İşlemi CPU Kullanımı

Bare-metal ve FreeRTOS ile gerçekleştirilen durumlarda oluşan "hex" dosyasını boyutu ve "byte" cinsinden bellek kullanım oranları Tablo 5.6'da gösterilmiştir.

Tablo 5.6. Bare-metal ve FreeRTOS için Bellek Kullanım Karşılaştırması

	Text	**Data**	**Bss**	**Toplam (Byte)**
FreeRTOS	35060	1460	81904	118424
Bare-metal	20504	1480	16552	38536

6. SONUÇ

Bu tez çalışmasında, Xilinx firmasının sunmuş olduğu IP blok servisleri kullanılarak düşük güç tüketimi odaklı bir sistem tasarlanmıştır. Bu sistem, Digilent firmasının üretmiş olduğu Nexys A7-100T Kartı geliştirme kartı üzerinde gerçeklenmiştir. Tasarlanan sistemin güç tüketimi üzerindeki etkisini göstermek adına örnek bir filtre (2 boyutlu konvolüsyon) uygulaması gerçekleştirilmiştir. Ancak sistem, farklı filtrelerinde sisteme entegrasyonu kolay olacak şekilde tasarlanmıştır. Bu uygulama C dili kullanılarak Xilinx SDK platformunda gerçeklenmiştir. Güç tüketiminde donanım ve uygulamanın ayrı ayrı etkileri bulunmaktadır. Uygulamaya özel yapılan tasarımların hem donanım tarafındaki güç tüketimine etkisi hemde uygulama süresine etkisi detaylı şekilde ele alınmıştır.

Temelde iki farklı donanım tasarlanmıştır. İlki herhangi bir uygulama çalıştırmak için gerekli olan modüllerden oluşmaktadır. İkinci donanım ise zorunlu olan donanımlara ek olarak, uygulamaya yönelik tasarlanan IP blokları da içermektedir. Donanımda işlemci olarak 32 bitlik Microblaze Soft CPU kullanılmıştır. Microblaze işlemcisine yönelik 10 tane ön tanımlı yapılandırma seçeneği bulunmaktadır. Bu seçeneklerden en az kaynak kullanarak güç tüketimini azaltan "minimum area" seçeneği ve performans odaklı yapılandırma seçeneği olan "maximum performance" seçenekleri denenmiştir. Bu seçeneklerin birbirine göre avantaj ve dezavantajları bulunmaktadır. Örneğin "minumum area" seçeneğinde, asıl hedef işlemcinin genel performansını göz önünde bulundurmadan sadece kullanılan kaynak sayısını azaltmaktır. Bu yapılandırmada işlemci 3 aşamalı "pipeline" kullanır. "Branch prediction" ve "cache" gibi uygulama performansını önemli ölçüde iyileştiren birçok özellik bu seçenekle uyumlu değildir. "Maximum Performance" seçeneğinde ise 5 seviye "pipeline" kullanılmaktadır. "Branch prediction", "cache" vb. donanımlar bu seçenekte donanıma dahil edilebilmektedir. "Miniumum Area" seçeneğine göre, "Maximum Performance" uygulama performansı için daha iyi bir seçenek olsa da daha fazla FPGA kaynağı kullanır. Örneğin bu projede ihtiyaç duyulmayan "floating point", "pattern comparator" vb. diğer donanım modülleri gereksiz kaynak tüketmektedir ve bu durum güç tüketimine olumsuz etki etmektedir. Donanım tasarımındaki güç tüketimi ise, statik ve dinamik güç tüketimi olmak üzere iki grupta incelenebilir. Statik güç tüketimi; transistörlerde meydana gelen sızıntı akımlarının sebep olduğu güç tüketimidir. Dinamik güç tüketimi ise; transistörlerin anahtarlama sırasında meydana

gelen güç tüketimidir. Toplam güç tüketimini hesaplamak için bu iki tüketim değeri toplanır. “Minimum area” seçeneği için statik güç tüketimi 0.108 W, dinamik güç tüketimi ise 1.002 W olarak ölçülmüştür. “Maximum Performance” seçeneği için statik güç tüketimi 0.110 W, dinamik güç tüketimi ise 1.196 W olarak ölçülmüştür. Yani toplam güç tüketimi “minumum area” için 1.11 W, “maximum performance” için 1.306 W olarak hesaplanmıştır. Uygulamaya özel geliştirilen özel yapılandırma seçeneğinde ise statik güç tüketimi 0.109 W, dinamik güç tüketimi ise 1.04 W olarak ölçülmüştür. Toplam güç tüketimi ise 1.149 W olarak hesaplanmıştır. Görüldüğü gibi yeni geliştirilen donanım yapılandırması “minumu area” seçeneğinden daha fazla, “maximum performance” seçeneğine göre ise daha az güç tüketimi sağlamıştır. Tabi bu değişikliklerin uygulama süresine etkisi göz önünde alındığında yeni yapılan tasarımın büyük bir avantaj sağladığı görülmüştür. Uygulama süresi ölçülürken, işlemcinin saat işareti kullanılmıştır. İlk uygulama senaryosunda konvolüsyon işlemi; 8x8 görüntü dizisi üzerinde sırasıyla 1, 10, 100, 500, 1000, 5000 ve 20000 gibi farklı iterasyon sayıları için uygulanmıştır. 10 ve altı olan iterasyonlarda yazılımsal konvolüsyon daha hızlı gerçekleştirilmiştir. Çünkü donanımsal konvolüsyon işleminde, I/O port üzerinden haberleşme yapılmaktadır. Fakat gerçek uygulama senaryosunda düşük iterasyon değil daha yüksek iterasyonlarda konvolüsyon işlemi uygulanacaktır. 100 iterasyon uygulanan senaryoda donanımsal olarak gerçekleştirme süresi yazılımsal gerçekleşme süresine göre %39,47 daha hızlı gerçekleştirilmiştir. Bu oradan 500 iterasyonda %82.78, 1000 iterasyonda %87,98, 5000 iterasyonda %90.44 ve 20000 iterasyonda ise %90.95 olarak hesaplanmıştır. Bu ölçüm senaryosunda, konvolüsyon işleminin kaç iterasyon uygulandığının önemi gösterilmiştir. Düşük iterasyonlarda bir avantaj sağlamazken, belirli bir iterasyon sonrası büyük bir avantaj sağladığı görülmektedir. Benzer şekilde belirli bir değerden sonrada doyuma ulaşmakta ve çok fazla artış görülmemiştir.

İkinci senaryoda ise 1800, 3600 ve 5400 iterasyon sayıları kullanılmıştır. Yani ortalamada 3600’lük bir iterasyon uygulanmıştır. Görüntü boyutu olarak 8x8, 16x16, 32x32, 64x64, 128x128 ve 256x256’lık boyutlar kullanılmıştır. Her görüntü boyutu için kernel matrisi 3x3 ve 5x5’lik olmak üzere ölçümler tekrarlanmıştır. Kernel matris boyutu arttığında, konvolüsyon işleminin tamamlanması için gerekli işlem sayısı azalmaktadır. 3x3’lük kernel matrisi için 8x8, 16x16, 32x32, 64x64, 128x128 ve 256x256’lık görüntü matrisleri için iyileşme oranı sırasıyla %92.194, %93.602, %93.960, %94.231, %98.895 ve %98.793 olarak hesaplanmıştır. 5x5’lik kernel matrisi için ise oranlar sırasıyla; %97.382,

%97.559, % 97.653, %97.706, %97.771 ve % 97.565 olarak hesaplanmıştır. 128x128'den küçük olan görüntü boyutlarında 5x5'lik kernel matrisi daha iyi sonuç gösterirken 128x128 ve 256x256'lık görüntülerde 3x3'lük kernel matrisiyle daha iyi ölçümler alınmıştır. Çünkü kernel matris boyutunun değişmesi görüntü üzerinde yapılacak iterasyon sayısını da değiştirmektedir. 5x5'lik kernel matrisi kullanılırken bir pikselin konvolüsyonu hesaplanırken 25 çarpma işlemi gerekir. 3x3'lük kernel matrisinde ise 9 çarpma işlemi gerekir.

Yukarıda bahsedilen ölçüm senaryolarında bare-metal programalama yaklaşımı izlenilmiştir. Buna ek olarak; FreeRTOS kullanılarak konvolüsyon işlemi multitask şekilde gerçeklenmiştir. FreeRTOS ve Bare-metal yaklaşımları kullanılarak uygulamanın gerçekleme süresi ve cpu kullanım oranları ölçülmüştür. CPU kullanımını gözlemleyebilmek için Microblaze'de "Debug Extended" özelliği aktif edilmiştir. FreeRTOS ile gerçekleştirilen konvolüsyon işleminde işlemci kullanım oranı %82.9 olarak ölçülmüştür. Ayrıca FreeRTOS ve Bare-metal uygulamalar için bellek kullanım oranları da karşılaştırılarak analiz edilmiştir. FreeRTOS bellekte, toplamda 118424 byte'lık yer kaplarken, bare-metal uygulama 38536 byte'lık yer kaplamaktadır. Çünkü FreeRTOS kernel kodu da bellekte yer tutacaktır.

KAYNAKLAR

[1] Guegan, L., & Orgerie, A. C. (2019, December). Estimating the end-to-end energy consumption of low-bandwidth IoT applications for WiFi devices. In *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)* (pp. 287-294). IEEE.

[2] Augustin, A., Yi, J., Clausen, T., & Townsley, W. M. (2016). A study of LoRa: Long range & low power networks for the internet of things. *Sensors*, *16*(9), 1466.

[3] Mezger, B. W., Santos, D. A., Dilillo, L., Zeferino, C. A., & Melo, D. R. (2022). A survey of the RISC-V architecture software support. *IEEE Access*, *10*, 51394-51411.

[4] Dörflinger, A., Albers, M., Kleinbeck, B., Guan, Y., Michalik, H., Klink, R., ... & Berekovic, M. (2021, May). A comparative survey of open-source application-class RISC-V processor implementations. In *Proceedings of the 18th ACM International Conference on Computing Frontiers* (pp. 12-20).

[5] Kalomiros, J. A., & Lygouras, J. (2008). Design and evaluation of a hardware/software FPGA-based system for fast image processing. *Microprocessors and Microsystems*, *32*(2), 95-106.

[6] (2023). Comparative study of risc and cisc architectures. International Journal of Computer Applications Technology and Research, 500-503.

[7] MicroBlaze Processor Reference Guide." UG984 (v2023.1) June 05, 2023 https://docs.xilinx.com/v/u/en-US/ug984-vivado-microblaze-ref(accessed Sep. 15, 2023)

[8] Cavanagh, J. (2017). Digital design and Verilog HDL fundamentals. CRC press.

[9] Ramachandran, S. (2007). Digital VLSI systems design: a design manual for implementation of projects on FPGAs and ASICs using Verilog. Springer Science & Business Media.

[10] Feist, T. (2012). Vivado design suite. White Paper, 5, 30.

[11] "Vivado Design Suite Tutorial" UG995 (v2022.1) April 26, 2022 https://www.xilinx.com/support/documents/sw_manuals/xilinx2022_1/ug995-vivado-ip-subsystems-tutorial.pdf(accessed Sep 15, 2023)

[12] "Getting Started with Xilinx SDK" (accessed September 29) https://www.xilinx.com/htmldocs/xilinx2019_1/SDK_Doc/sdk_getting_started/sdk_getting_started.html#sdk_getting_started

[13] Simonović, M., & Saranovac, L. (2013). Power management implementation in FreeRTOS on LM3S3748. *Serbian Journal of Electrical Engineering*, *10*(1), 199-208.

[14] https://github.com/FreeRTOS.

[15] https://digilent.com/reference/software/vivado/board-files (Accessed August 23)

[16] Wang, W., Bolic, M., & Parri, J. (2013, September). pvFPGA: Accessing an FPGA-based hardware accelerator in a paravirtualized environment. In *2013 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)* (pp. 1-9). IEEE.

[17] Del Sozzo, E., Baghdadi, R., Amarasinghe, S., & Santambrogio, M. D. (2017, November). A common backend for hardware acceleration on FPGA. In *2017 IEEE International Conference on Computer Design (ICCD)* (pp. 427-430). IEEE.

[18] Talib, M. A., Majzoub, S., Nasir, Q., & Jamal, D. (2021). A systematic literature review on hardware implementation of artificial intelligence algorithms. *The Journal of Supercomputing*, *77*, 1897-1938.

[19] Park, H., & Kim, S. (2021). Hardware accelerator systems for artificial intelligence and machine learning. In *Advances in Computers* (Vol. 122, pp. 51-95). Elsevier.

[20] Saegusa, T., Maruyama, T., & Yamaguchi, Y. (2008, September). How fast is an FPGA in image processing?. In *2008 International Conference on Field Programmable Logic and Applications* (pp. 77-82). IEEE.

[21] Gholizadehazari, E., Ayhan, T., & Ors, B. (2021, June). An FPGA implementation of a RISC-V based SoC system for image processing applications. In *2021 29th Signal Processing and Communications Applications Conference (SIU)* (pp. 1-4). IEEE.

[22] Akçay, L., Sürer, B., & Yalçın, B. Ö. (2022, May). Implementation of a SoC by Using lowRISC Architecture on an FPGA for Image Filtering Applications. In *2022 30th Signal Processing and Communications Applications Conference (SIU)* (pp. 1-4). IEEE.

[23] Garcia-Ramirez, R., Chacón-Rodríguez, A., Castro-Gonzalez, R., Arnaud, A., Miguez, M., Gak, J., ... & Rimolo-Donadio, R. (2020, February). Siwa: a risc-v rv32i based micro-controller for implantable medical applications. In *2020 IEEE 11th Latin American Symposium on Circuits & Systems (LASCAS)* (pp. 1-4). IEEE.

[24] Skliarova, I. (2021). Accelerating Population Count with a Hardware Co-Processor for MicroBlaze. *Journal of Low Power Electronics and Applications*, *11*(2), 20.

[25] Serrano, R., Sarmiento, M., Duran, C., Nguyen, K. D., Hoang, T. T., Ishibashi, K., & Pham, C. K. (2021, October). A low-power low-area SoC based in RISC-V processor for IoT applications. In *2021 18th International SoC Design Conference (ISOCC)* (pp. 375-376). IEEE.

[26] Azmi, H. FPGA Acceleration of Tree-based Learning Algorithms.

[27] Afifi, S. M., Verdier, F., & Belleudy, C. (2014, March). Power estimation method based on real measurements for processor-based designs on fpga. In *2014 International Conference on Computational Science and Computational Intelligence* (Vol. 2, pp. 260-263). IEEE.

[28] Azzouzi, O., Anane, M., Koudil, M., & Issad, M. (2022). Flexible and Parallel Architectures for Optimal Ate pairing on FPGA.

[29] Balasubramanian, L. S. (2020). *ASIC-Implemented Microblaze-Based Coprocessor for Data Stream Management Systems*. Purdue University.

[30] Choudhury, A., Srinivasan, A., Kattimani, A. R., Siddamal, S. V., & Hiremath, S. B. (2021, September). Microblaze Implementation of GPS. In *3rd International Conference on Integrated Intelligent Computing Communication & Security (ICIIC 2021)* (pp. 296-302). Atlantis Press.

[31] El-Hadedy, M., Hua, R., Saqib, S., Yoshii, K., Hwu, W. M., & Margala, M. (2023, September). BLTESTI: Benchmarking Lightweight TinyJAMBU on Embedded Systems for Trusted IoT. In *2023 IEEE 36th International System-on-Chip Conference (SOCC)* (pp. 1-6). IEEE.

[32] Abid, F., & Izeboudjen, N. (2022, July). Zynq-SoC vs. MicroBlaze Based SoC for Forest Fires Prediction: Comparison Study. In *International Conference on Computing and Information Technology* (pp. 279-289). Cham: Springer International Publishing.

[33] https://www.xilinx.com/support/documents/university/XUP%20Boards/XUPNexys4DDR/documentation/Nexys4-DDR_rm.pdf (accessed 13 July, 2023)

[34] MicroBlaze Processor Reference Guide.” UG984 (v2023.1) June 05, 2023 https://docs.xilinx.com/v/u/en-US/ug984-vivado-microblaze-ref(accessed Sep. 15, 2023)

ÖZGEÇMİŞ

Adı Soyadı: Hakan Gürbüz

Lisans: Marmara Üniversitesi / Elektrik-Elektronik Mühendisliği (3.27/4.0)