T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

HETEROJEN GÖMÜLÜ HESAPLAMA KULLANILARAK İLERİ YOL ŞERİT TESPİTİ

Mustafa Emre YILMAZ

YÜKSEK LİSANS TEZİ
Elektronik ve Haberleşme Mühendisliği Anabilim Dalı
Elektronik Mühendisliği Programı

Danışman
Prof. Dr. Burcu Erkmen

Haziran, 2022

T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

HETEROJEN GÖMÜLÜ HESAPLAMA KULLANILARAK İLERİ YOL ŞERİT TESPİTİ

Mustafa Emre YILMAZ tarafından hazırlanan tez çalışması 28.06.2022 tarihinde aşağıdaki jüri tarafından Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Elektronik ve Haberleşme Mühendisliği Anabilim Dalı Elektronik Mühendisliği Programı **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Prof. Dr. Burcu Erkmen
Yıldız Teknik Üniversitesi
Danışman

Jüri Üyeleri

Prof. Dr. Burcu Erkmen, Danışman
Yıldız Teknik Üniversitesi _____

Doç. Dr. Sadiye Nergis Tural Polat, Üye
Yıldız Teknik Üniversitesi _____

Doç. Dr. Mürvet Kırcı, Üye
İstanbul Teknik Üniversitesi _____

Danışmanım Prof. Dr. Burcu Erkmen sorumluluğunda tarafımca hazırlanan Heterojen Gömülü Hesaplama Kullanılarak İleri Yol Şerit Tespiti başlıklı çalışmada veri toplama ve veri kullanımında gerekli yasal izinleri aldığımı, diğer kaynaklardan aldığım bilgileri ana metin ve referanslarda eksiksiz gösterdiğimi, araştırma verilerine ve sonuçlarına ilişkin çarpıtma ve/veya sahtecilik yapmadığımı, çalışmam süresince bilimsel araştırma ve etik ilkelerine uygun davrandığımı beyan ederim. Beyanımın aksinin ispatı halinde her türlü yasal sonucu kabul ederim.

Mustafa Emre YILMAZ

İmza

Bu çalışma Türkiye Bilimsel ve Teknolojik Araştırma Kurumu tarafından desteklenmiştir(TUBITAK).

Aileme ve yakın arkadaşlarıma

TEŞEKKÜR

Bu çalışmanın gerçekleştirilmesinde, değerli bilgilerini benimle paylaşan, kendisine ne zaman danışsam bana kıymetli zamanını ayırıp sabırla ve büyük bir ilgiyle bana faydalı olabilmek için elinden gelenden fazlasını sunan her sorun yaşadığımda yanına çekinmeden gidebildiğim, güler yüzünü ve samimiyetini benden esirgemeyen ve gelecekteki mesleki hayatımda da bana verdiği değerli bilgilerden faydalanacağımı düşündüğüm kıymetli ve danışman hoca statüsünü hakkıyla yerine getiren Prof. Dr. Burcu Erkmen'e teşekkürü bir borç biliyor ve şükranlarımı sunuyorum. Yine çalışmamda konu, kaynak ve yöntem açısından bana sürekli yardımda bulunarak yol gösteren ve gelecekteki hayatında çok daha başarılı olacağına inandığım kıymetli arkadaşım Hatice Büken'e de sonsuz teşekkürlerimi sunarım.

Çalışmayı gerçekleştirirken her türlü maddi ve manevi yönde desteklerini eksik etmeyen TÜBİTAK BİLGEM ve YongaTek ailelerine, eğitim dönemim boyunca kazandırdıkları her şey için ve beni gelecekte söz sahibi yapacak bilgilerle donattıkları için Yıldız Teknik Üniversitesi hocalarıma teker teker teşekkürlerimi sunuyorum.

Mustafa Emre YILMAZ

İÇİNDEKİLER

KISALTMA LİSTESİ

ACC	Adaptive Cruise Control
ADAS	Advanced Driver Assistant System
AI	Artificial Intelligence
API	Application Programming Interface
APU	Application Processing Unit
AXI	Advanced eXtensible Interface
BEV	Bird's Eye View
BN	Batch Normalization
BRAM	Block Random Memory Access
BUFG	Global Buffer
CLB	Configurable Logic Block
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DDR	Double Data Rate
DP	DisplayPort
DPU	Deep Learning Processing Unit
DRAM	Dynamic Random Memory Access
DRM	Direct Rendering Manager
DSP	Digital Signal Processor
FF	Flip Flop
FLC	Forward Looking Camera
FPGA	Field Programmable Gate Array
FPS	Frame Per Second

FSBL	First Stage Bootlader
GIC	Generic Interrupt Controller
GPIO	General Purpose Input Output
GPS	Global Positioning System
GPU	Graphics Processing Unit
HiL	Hardware in the Loop
IP	Intellectual Property
IR	Infrared
IRQ	Interrupt Request
JTAG	Joint Test Action Group
LIDAR	Laser Imaging, Detection, and Ranging
LRN	Local Response Normalization
LUT	Look-up Table
MMCM	Mixed-Mode Clock Manager
PC	Personal Computer
PL	Programmable Logic
PS	Processing System
QSPI	Quad Serial Peripheral Interface
ResNet	Residual Network
RPU	Real-time Processing unit
SD	Secure Digital
SDK	Software Development Kit
SoC	System on Chip
UART	Universal Asynchronous Receiver Transmitter
URAM	Ultra Random Memory Access
USB	Universal Serial Bus
VPGNet	Vanishing Point Guided Network

ŞEKİL LİSTESİ

TABLO LİSTESİ

ÖZET

Heterojen Gömülü Hesaplama Kullanılarak İleri Yol Şerit Tespiti

Mustafa Emre YILMAZ

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı
Yüksek Lisans Tezi

Danışman: Prof. Dr. Burcu Erkmen

ADAS (Gelişmiş Sürücü Yardım Sistemi), radar ve lazer gibi çevresel sensörler kullanarak araçları kontrol eden, sürücünün potansiyel olarak tehlikeli trafik koşullarını tanımasına ve tepki vermesine yardımcı olan ve esas olarak sürüş konforunu ve trafik güvenliğini artırmak için tasarlanmış sistemlerdir. 2021 yılında sadece Türkiye'de 1 milyon 168 bin 144 otomobil kazası, 283 bin 34 otomobil ile ilgili yaralanma ve 5 bin 473 motorlu araç ölümü bildirilirken, Avrupa'da yılda 40.000'den fazla zayiat ve 1.4 milyon yaralanma araçla ilgili kazalardan kaynaklanmaktadır. Araştırmacılar, bu rakamlara dayanarak, kazada hayatta kalma teknikleri ve sistemleri geliştirdiler ve bunların tüm araçlarda kurulum ihtiyacını vurguladılar. Bu sistemler bir dereceye kadar sürücünün hayatta kalmasını garanti eder.

Ülkemizin yerli otomobil üretimi sürecinde olduğu ve halihazırda birçok aracın yazılım ve donanımının argesi ve üretimi yapıldığı bu dönemde gerçek-zamanlı görüntü işleyip yol şeritlerini belirleyerek sürücüye bilgi aktaran bir sürücü asistanı sistemine büyük ihtiyaç duyulmaktadır. Bu doğrultuda bir çalışma olan olan bu sistem bir heterojen gömülü sistem olan SoC (System on Chip) üzerinde gerçeklenmiş olup, kameradan alınan gerçek-zamanlı görüntü işleme algoritmalarının, kütüphanelerin ve hafıza gerektiren işlemlerin ağır hesaplama gerektiren kısımlarını FPGA yani donanım üzerinde yaptırdığından ülkemizde bu alanda yapılmış olan diğer çalışmalara göre daha yüksek FPS (Frame Per Second) sonuç alınmaktadır.

Bu çalışmada ZynqMP Ultrascale+ SoC üzerinde ADAS uygulaması tanıtılmıştır. Bilgisayarla görü, görüntü işleme ve derin öğrenme metotlarını kullanan ZynqMP

Ultrascale+ SoC üzerinde bir ADAS uygulaması, araçtaki kameradan yol şeritlerini gerçek zamanlı olarak algılamakta ve eş zamanlı olarak yüksek FPS ile bunlar hakkında sürücüyü bilgilendirmektedir. Ön işleme uygulandıktan sonra, yol çizgilerini tespit etmek için görüntü üzerinde evrişimli bir sinir ağı gerçekleştirilir. Uygulamanın heterojen gömülü hesaplama platformu üzerinde gerçeklenmesi sistemin bir taraftan kısıtlı kaynaklar kullanılarak, düşük güç tüketimine sahip olmasına ve mümkün olan en yüksek seviyede işleme hızına sahip olmasına imkan sağlamaktadır.

Anahtar Kelimeler: ADAS, SoC, FPGA, yol şerit tanıma, evrişimsel sinir ağları

ABSTRACT

Advanced Lane Line Detection Using Heterogeneous Embedded Computing

Mustafa Emre YILMAZ

Department of Electronics and Communication Engineering
Master of Science Thesis

Supervisor: Prof. Dr. Burcu Erkmen

ADAS (Advanced Driver Assistance System) are systems that control vehicles using environmental sensors such as radar and laser, helping the driver to recognize and react to potentially dangerous traffic conditions, mainly designed to increase driving comfort and traffic safety. While 1 million 168 thousand 144 automobile accidents, 283 thousand 34 automobile-related injuries and 5 thousand 473 motor vehicle deaths were reported in Turkey alone in 2021, more than 40,000 casualties and 1.4 million injuries per year in Europe are caused by vehicle-related accidents. Based on these figures, the researchers developed crash survival techniques and systems, emphasizing the need for their installation in all vehicles. These systems to some extent guarantee the survival of the driver.

In this period when our country is in the process of domestic automobile production and the software and hardware of many vehicles are currently being developed and produced, there is a great need for a driver assistant system that processes real-time images and determines road lanes and conveys information to the driver. This system, which is a study in this direction, has been implemented on a heterogeneous embedded system, SoC (System on Chip). Compared to other studies, higher FPS (Frame Per Second) results are obtained.

In this study, ADAS application on ZynqMP Ultrascale+ SoC is introduced. Using computer vision, image processing and deep learning methods, an ADAS application on ZynqMP Ultrascale+ SoC detects road lanes in real time from the camera in the vehicle and simultaneously informs the driver about them with high FPS.

After preprocessing, a convolutional neural network is performed on the image to detect road lines. Implementation of the application on a heterogeneous embedded computing platform allows the system to have low power consumption and the highest possible processing speed, using limited resources on the one hand.

Keywords: ADAS, SoC, FPGA, lane line detection, convolutional neural networks

1
GİRİŞ

1.1 Literatür Özeti

ADAS, sürücülere yardımcı olmayı ve yol güvenliğini artırmayı amaçlayan dijital elektronik sistemdir. ADAS, güvenlik önlemlerinin uygulanması ve nihayetinde potansiyel olarak aracın kontrolünü ele geçirme yoluyla olası yol engelleri veya çarpışmalar hakkında sürücüyü uyararak kazaları önlemek için tasarlanmıştır. Aydınlatma kontrolü, uyarlanabilir hız sabitleyici, otomatik frenleme, GPS/trafik uyarıları ve sürücüyü yol şeridinde tutma, uyarlanabilir özelliklere örnektir. Şekil 1.1 'de bu uygulama örnekleri gösterilmiştir.

Şekil 1.1 Kullanılan bazı ADAS uygulamaları örnekleri

Google'ın 2009 yılında otonom bir araç tasarlama girişimlerinin ardından, Google'ın özel donanımlı araç filosu, kameralar kullanarak Kaliforniya ve Nevada'da gece gündüz 140.000 milden fazla sürüş kaydetti. Algoritmik görüntü işleme sistemleri ve LIDAR teknolojisini kullanarak gittikleri rotanın 3 boyutlu bir görüntüsünü oluşturdular [1].

Mobileye'nin çarpışma önleme teknolojisi, bir sahneyi gerçek zamanlı olarak

görüntülemek ve sürücülerin analizlerine dayalı olarak anında bir değerlendirme yapmasına yardımcı olmak için tek bir akıllı kamera ve gelişmiş bir görüş algoritması kullanır. Mobileye'nin EyeQ görüntü işlemcisi ile birleştirilen bu teknoloji, artık Volvo, BMW ve General Motors gibi otomobil üreticileri tarafından giderek değişen güvenlik özelliği uygulamalarında kullanılmaktadır [1].

Bu alanda yapılmış çalışmalar olarak, yol yüzeyini belirlemek için ufuk çizgisinin kullanıldığı [2] 'de, ufuk çizgisininin tespiti için bir şerit algılama yöntemi geliştirilmiştir. Şeritleri algılamak için görüntü üzerinde belirli kesitlerin ortalama histogram değerleri ile Bhattacharyya uzaklığı karşılaştırılarak yorumlanmıştır. CNN ve ResNet'in geleneksel Hough dönüşümüne göre avantajını vurgulandığı [3]'te yol bölütleme yöntemine göre görüntüdeki çizgileri tespit eden bir derin sinir ağı yaklaşımı kullanılmış ve deneysel sonuçlar incelenmiştir. [4] 'te, sürücü davranışı ve trafik yön işareti tespiti, derin öğrenme kullanılarak belirlendi. Geometrik yaklaşıma dayalı paralel ve dikey park etme algoritmalarının incelendiği [5] 'te oluşturulan sistem, simülasyon ortamında test edilmiş ve ardından tasarlanan mobil araçta mikrodenetleyiciye yerleştirilmiştir.

[6] 'da, yüksek seviye kontrol, düşük seviye kontrol ve sensör ünitesi alt sistemlerinden oluşan otonom bir sürücü kontrol mimarisine dayalı akıllı bir ACC (Adaptif Cruise Control) sistemi önerilmiştir. Elektrikli araçlarda aktif güvenlik sistemleri için bir geri besleme bileşeni olarak kullanılabilecek düşük maliyetli bir gömülü veri toplama sistemi [7] 'de önerilmiştir. Tamamen yazılım ortamında tasarlanmış ve donanıma hiç aktarılmamış ya da gömülü yazılım olarak bir mikrodenetleyici üzerinde ardışıl olarak uygulanmış olması, [2]-[7] 'nin güç tüketiminin artmasına neden olur ve zaman-kritik bir sistemin gerçek zamanlı çalışmasını zorlaştırır.

Ayrıca [4]-[7] 'de kamera veya görüntü sensörü yerine Ultrasonik / IR sensörler kullanılması, nesneleri algılama ve tanıma yetenekleri nedeniyle daha kritik ve değerli girdi verilerinden feragat edildiğini gösterir.

Otomotiv uygulamaları için FPGA tabanlı gerçek zamanlı şerit algılama sisteminin sunulduğu [8] 'de, hesaplama gücünü ve karmaşıklığı en aza indirmek için geleneksel Canny-Hough şerit algılama algoritması, gerçek zamanlı işleme elde edecek şekilde özelleştirildi. FPGA tabanlı bir video HiL çözümü [9] 'da sunulmaktadır. Bir video sensörü (FLC) ve bir video işleme biriminden oluşan ADAS akıllı kamerayı değerlendirmek için tasarlanmıştır.

[8, 9] 'da tanıtılan iki sistem tamamen FPGA üzerinde uygulanmaktadır. Bu sayede düşük güç tüketimi ve gerçek zamanlı sistem büyük ölçüde elde edilse de, sistem geliştirme ve hata ayıklama sürecini uzatmakta ve giriş-çıkış arayüzlerinin çeşitliliği

azaltılmaktadır. Herhangi bir öğrenme algoritması kullanılmadığı için de yapay öğrenme mekanizmasından yoksundurlar. Yapay zeka ve derin öğrenme tabanlı uygulamalar, farklı ve zorlayıcı koşullar altında denendiğinde kural tabanlı sistemlere kıyasla daha iyi performans göstermektedir.

Simulink ortamında üst düzey sentez kullanılarak tasarlanan [10] 'da, gerçek zamanlı kısıtlamalar altında önden görünüm video akışı kuş bakışı (BEV) sisteminin donanım uygulaması önerilmiştir. Ancak burada Matlab aracının yüksek düzey sentezini kullanmak, sistemi optimize etmeyi zorlaştırdığı için performans düşüklüğüne sebep olmuştur.

IBM bulutundaki yerel bir Zynq terminali ve FPGA kümelerinden oluşan heterojen bir hızlandırma platformunda geliştirilen [11] 'de, bir mobil nesne çıkarma ve sınıflandırma sistemi geliştirildi. Ancak bulut ortamında çalışmak, gerçek zamanlı görüntü işleme ve görüntü aktarımı için performans darboğazlarına neden olmaktadır.

Hough dönüşümünün kullanıldığı [12] 'de, Zynq-7000 platformunda gerçek zamanlı şerit algılamanın bir donanım uygulaması geliştirilmiştir. Ancak, engebeli yollar ve şerit kayıpları gibi belirli koşullarda algılama doğruluğu azalır.

1.2 Tezin Amacı

Bu çalışmada ZynqMP Ultrascale+ SoC (System on Chip) üzerinde bir ADAS uygulaması tanıtılmıştır. Bilgisayarla görü, görüntü işleme ve derin öğrenme metotları ve kütüphaneleri kullanarak, araçtaki kameradan yol şeritlerini gerçek zamanlı olarak algılamakta ve sürücüye yüksek FPS (saniyede kare) bildirmektedir.

1.3 Hipotez

Donanım tercihi olarak ZynqMP Ultrascale+ SoC seçilen bu çalışmada gerçeklemek için Caltechlane veri kümesi ile eğitilmiş olan VPGNet CNN modeli tercih edilmiştir. SoC platformunda gerçeklenmesi için bu modelin tercih edilmesinin sebebi 17 farklı şerit ve yol bulunduran yaklaşık 20000 görüntü ile benchmark testleri yapılan VPGNet modelinin, şeritleri ve yol işaretlerini tespit edip sınıflandırabilmesidir. Yapılan testlerde, metodun gerçek zamanlı olarak yüksek performans ve doğrulukta çalıştığı görülmüştür.

Seçilen CNN modelini en optimize şekilde çalıştırabilmek için SoC'nin PL üzerinde bulunan DPU IP'si kullanılmıştır. Tamamen CNN algoritmalarını çalıştırmaya yönelik bir komut kümesi mimarisine sahip olan DPU, PL yani FPGA dokusundan oluştuğu için

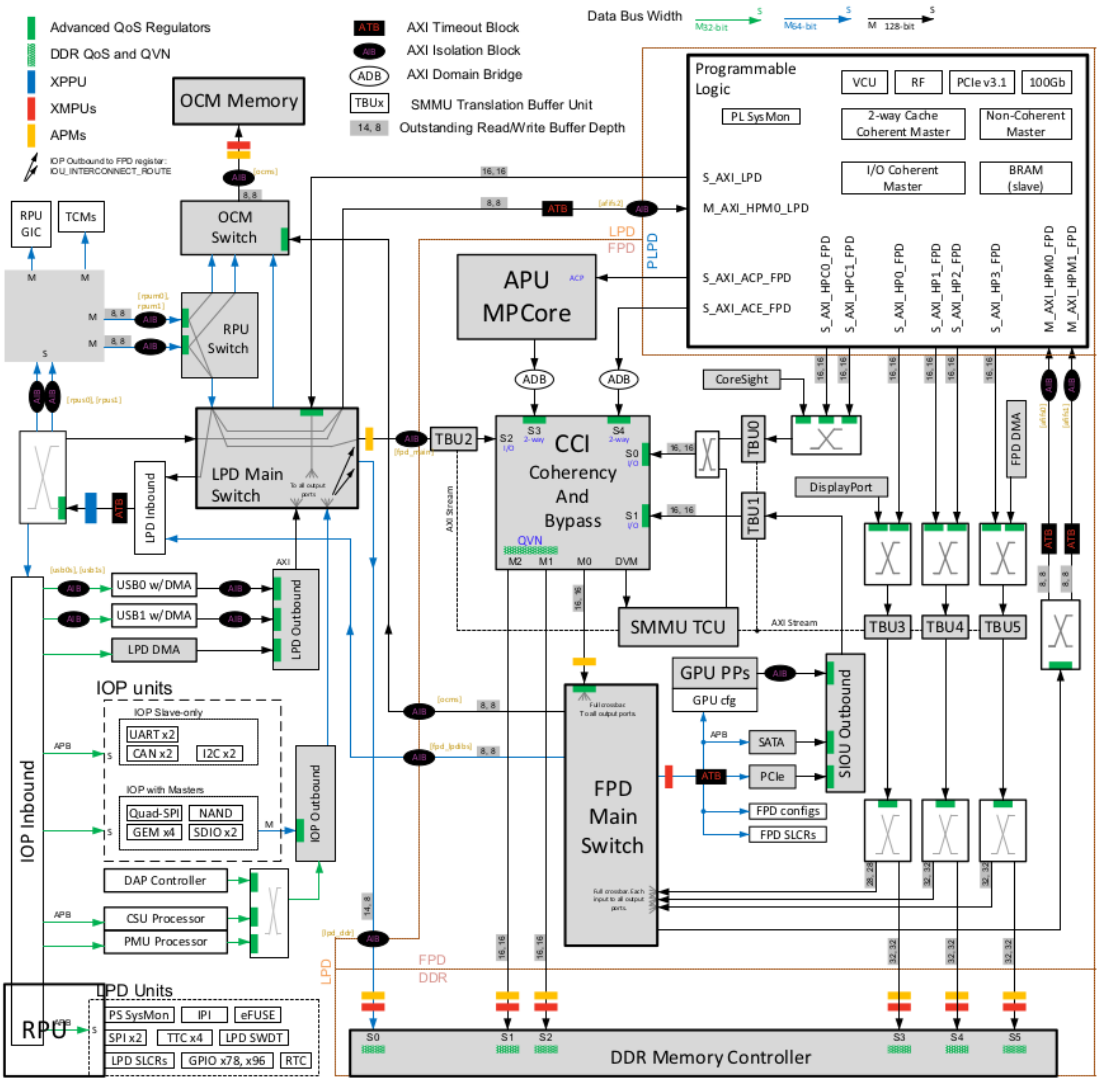
işlemleri gerçek zamanlı çalıştırabilmekte olması yanı sıra pragmalar sayesinde SoC üzerindeki PS tarafıyla yani ARM CPU ile konuşabilmektedir.

Donanımın CNN modelini anlayıp çalıştırabilmesi için Xilinx'e ait olan Vitis AI aracı kullanılmıştır. Model, Vitis AI aracı kullanılarak eğitilmiş, kuantalanmış ve derlenmiştir. Böylece düşük güç tüketen gömülü bir hesaplama platformu olan SoC üzerinde yüksek FPS'de çalışan gerçek-zamanlı bir yapay öğrenme donanım hızlandırıcısı tasarlandı.

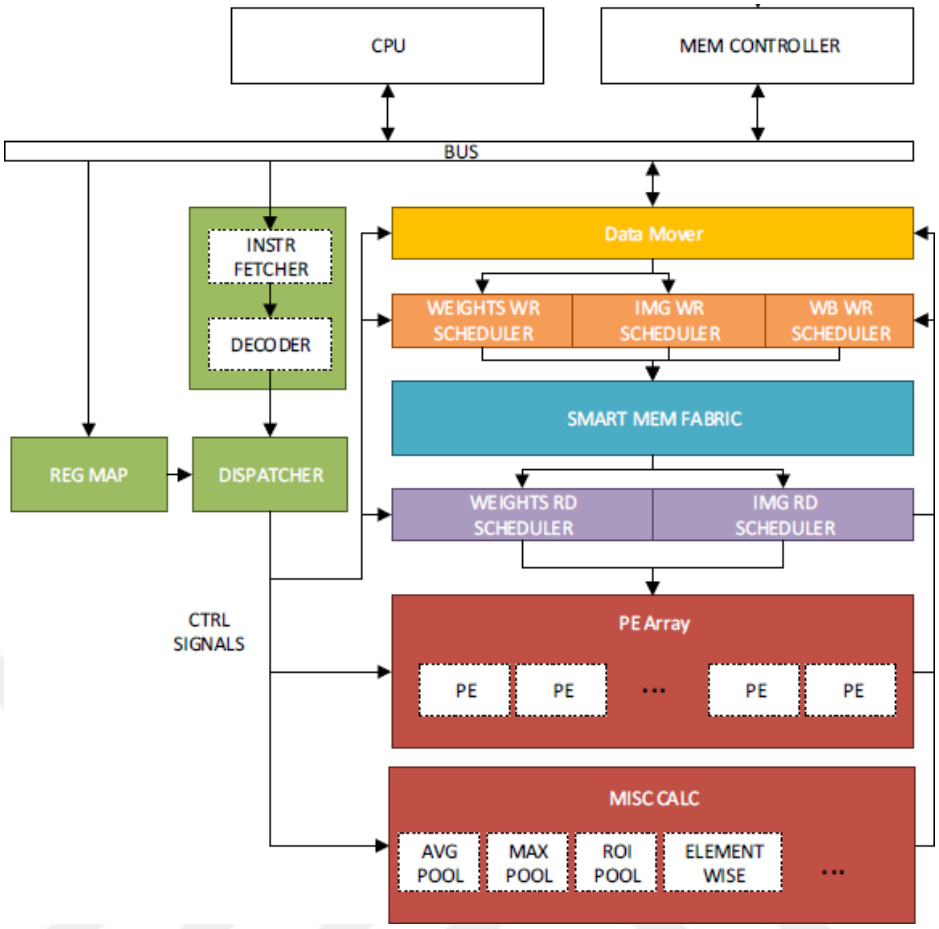
2
YONGA ÜZERİNDE SİSTEM DONANIMLARI

Bu çalışmada kameradan gerçek zamanlı alınan bir yol görüntüsünde yol şeritlerini, bir CNN modeli kullanarak tespit eden bir sistem ZynqMP Ultrascale+ SoC platformunda gerçeklenmiştir. Şekil 2.1 'de gösterildiği gibi, ZynqMP Ultrascale+'ın donanım mimarisi iki bölümden oluşmaktadır. İlk bölüm, 4 çekirdekli ARM Cortex A53 CPU (APU), 2 çekirdekli ARM Cortex R5 CPU (RPU) ve 3 çekirdekli Mali-400 MP2 GPU içeren PS'tir. PS, çevre birimlerini kullanmak için yazılım esnekliği, işletim sistemi cihaz sürücüleri ve birçok farklı türde kütüphaneden yararlanma imkanı sağlar. Diğer bölüm ise FPGA dokusunu bulunduran PL'dir. PL, tasarımcılar için fonsiyona özel mantık devreleri tasarlama imkanı sunar. Paralellik, yüksek hız ve yüksek sayıda giriş/çıkış portlarından yararlanmayı sağlar.

PL ayrıca özel bir işlemci olan DPU adlı bir IP içerir. DPU, CNN algoritmalarını gerçeklemek için belirli bir komut kümesi mimarisine sahiptir. İç mimari olarak yüksek performanslı bir zamanlayıcı modülü, bir hibrit bilgi işlem dizisi modülü, bir komut getirme birimi modülü ve bir global bellek havuzu modülünden oluşur [14]. Şekil 2.2 'de DPU işlemcisinin mimarisi gösterilmiştir.



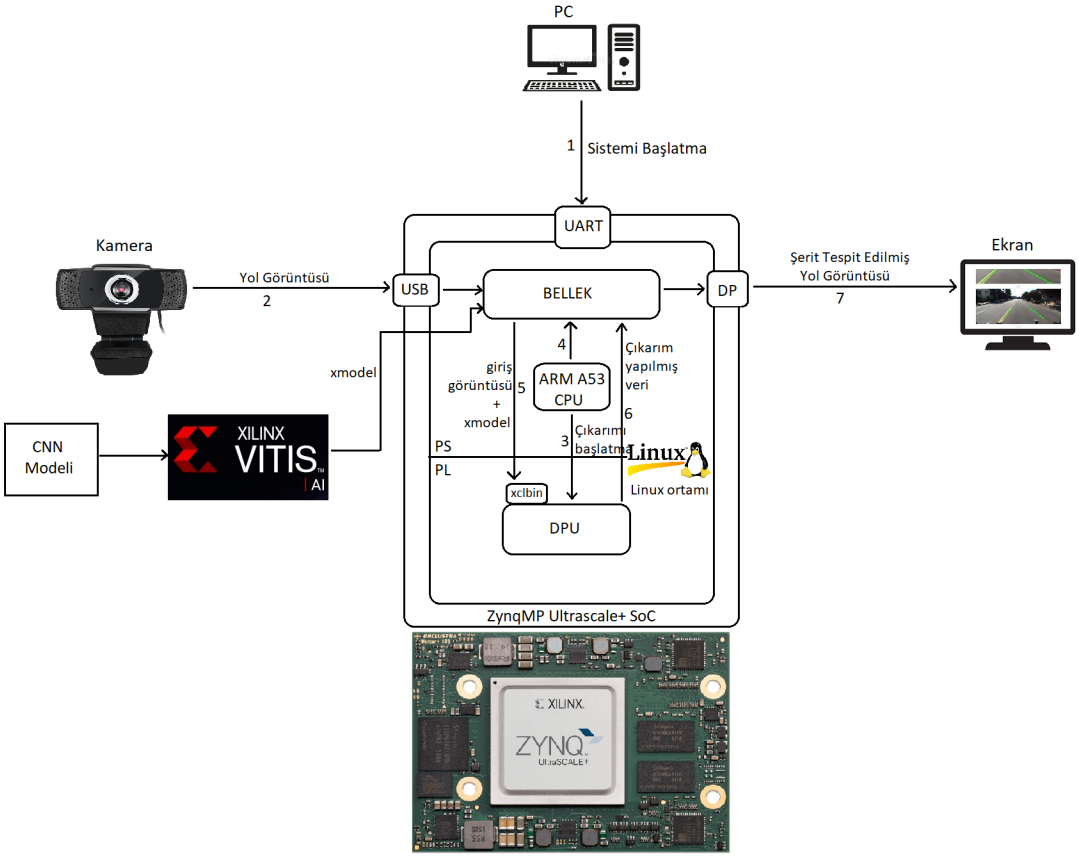
Şekil 2.1 ZynqMP Ultrascale+ donanım mimarisi[13]



Şekil 2.2 DPU IP'sinin iç mimarisi[15]

3
SİSTEMİN DONANIM VE YAZILIM BİLEŞENLERİ

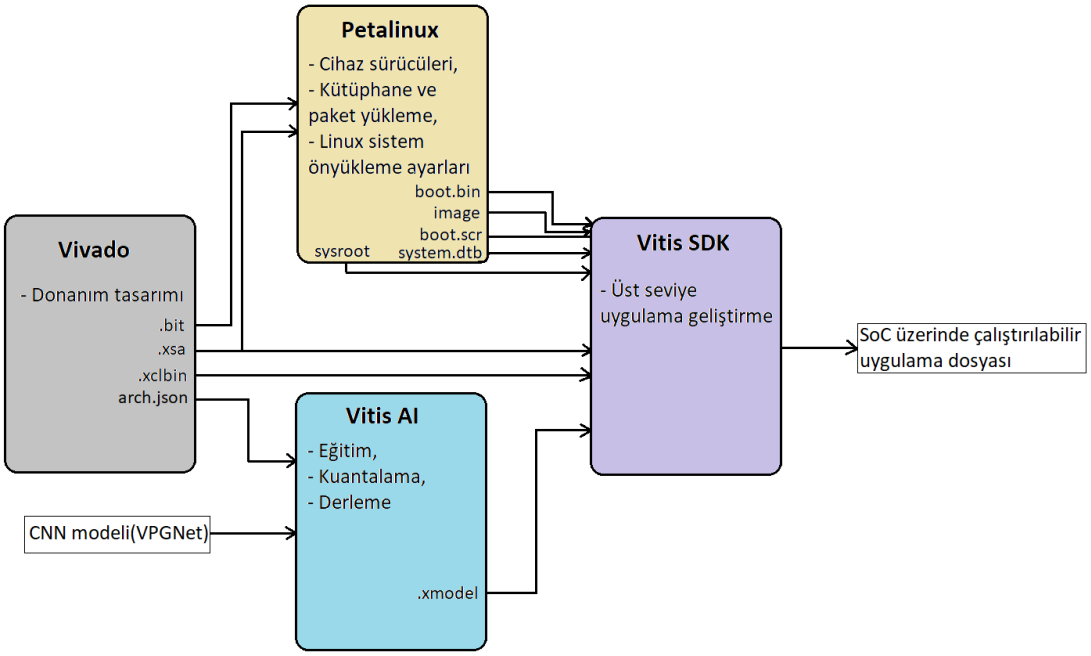
Bu bölümde sistemin bütünsel olarak her bir donanım ve yazılım tasarımı aşamaları detaylı olarak açıklanmıştır. Sistemin her bir aşamadaki girdileri ve çıktıları belirtilmiştir. Şekil 3.1 'de önerilen sistemin tasarım planı gösterilmiştir. Numaralandırılmış girdi ve çıktılar sistemin çalışma esnasındaki akış sıralamasını belirtmektedir. Vitis AI akışından çıkan xmodel dosyası çıktısı, sistem başlatılmadan öncesinde bellekte halihazırda tutulmaktadır.



Şekil 3.1 Tasarlanan sistem blok diyagramı

Bu platform üzerinde PS ve PL'in birlikte çalıştırılarak gerçeklenmiş olan tasarım akışı 4 ayrı başlık altında incelenmiştir. Bunlar;

- donanım blok tasarımının oluşturulduğu Vivado akışı,
- cihaz sürücülerinin, kütüphanelerin ve linux önyükleyici dosyalarının üretimi için gerekli konfigürasyonların yapıldığı Petalinux akışı,
- CNN modelinin, SoC üzerinde çalışabileceği formata getirmek için kullanılan Vitis AI akışı,
- Vivado, Petalinux ve Vitis AI akışlarında üretilen dosyalarla birlikte çalışacak olan üst seviye uygulama kodunun geliştirildiği Vitis SDK akışı.



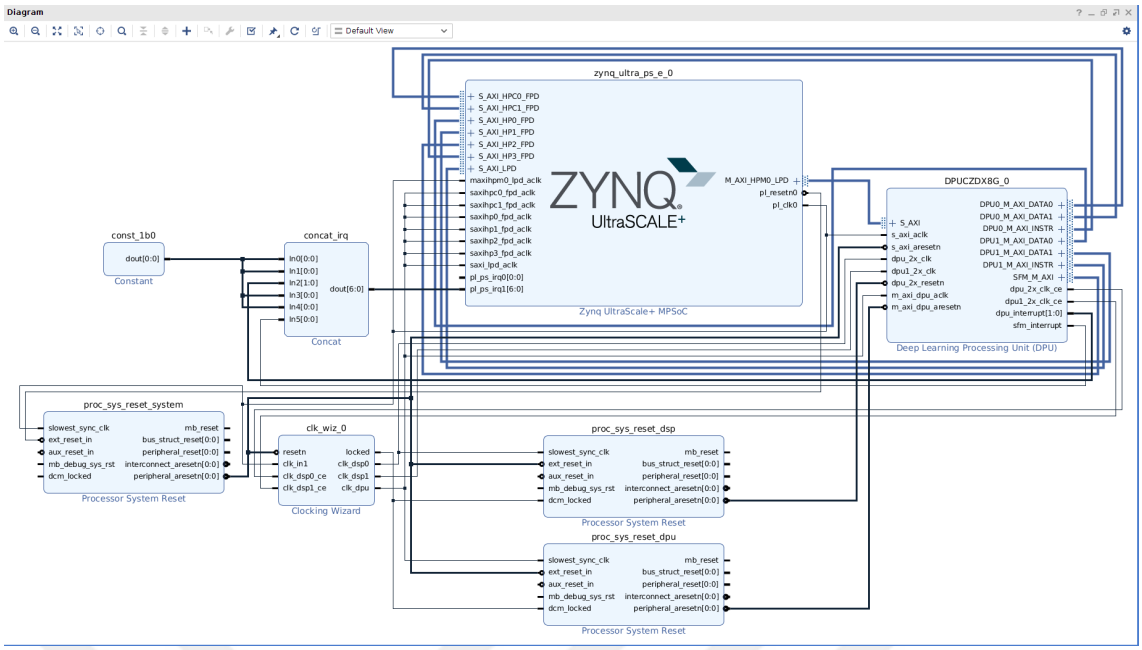
Şekil 3.2 Sistemin bütünsel akış şeması

Şekil 3.2 'de, kullanılan her bir geliştirme aracına ait tasarım akışları, girdi ve çıktılarla birlikte bloklar halinde gösterilmiştir.

3.1 Vivado Üzerinde Donanım Tasarımı Akışı

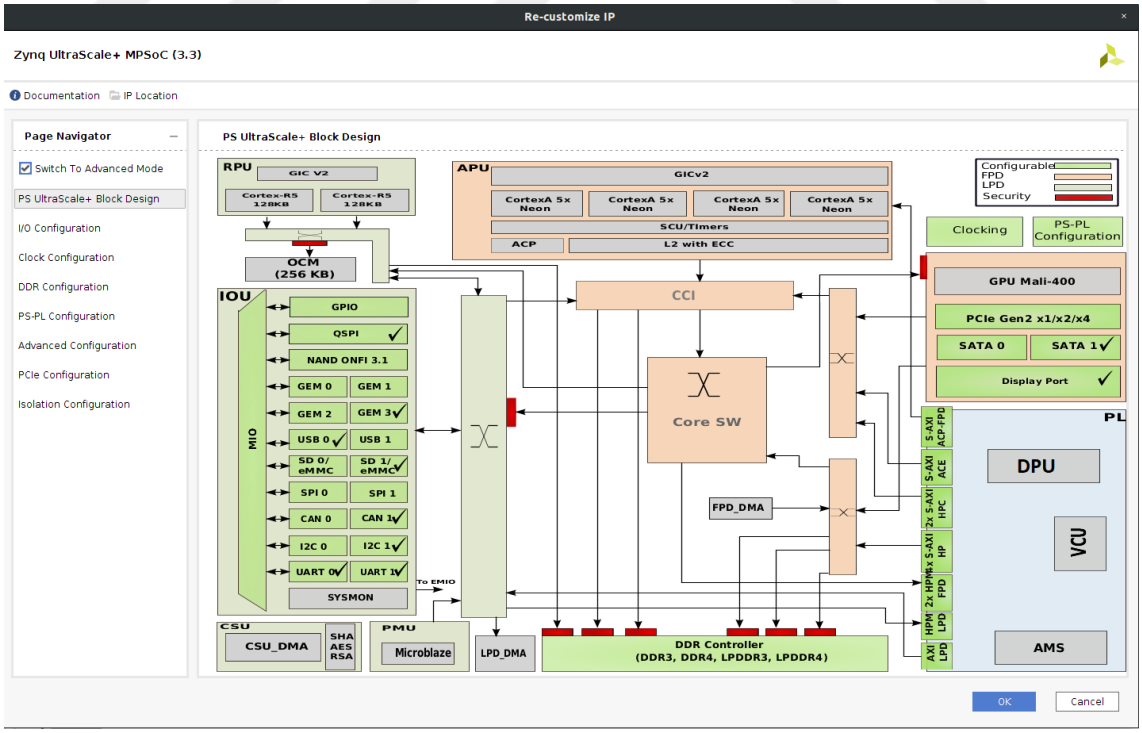
Vivado® [16] FPGA-SoC üzerinde sayısal donanım tasarımı yapmak için kullanılan, AMD Xilinx'e ait bir tasarım aracıdır. Vivado üzerinde özelleştirilmiş IP tasarımı yapılabilir, üretici firmanın sağladığı yerleşik IP'ler kullanılabilir, tasarlanan devrelerin simülasyonu, sentez ve yerleşimi yapılabilir ve zamansal-alansal analizler gerçekleştirilebilir.

Bu çalışmada Vivado 2020.2 versiyonu kullanılmıştır. Şekil 3.3 'te tasarlanan donanımın blok diyagramı verilmiştir.



Şekil 3.3 Vivado akışındaki donanımın blok diyagramı

Donanımda kullanılan IP'ler olan sistem resetleri, işlemcileri besleyen clock üreteçleri ve kesme (interrupt) denetleyicilerinin yanı sıra tasarımın merkezinde ZynqMP PS ve DPU bulunmaktadır. Şekil 3.4 'te ZynqMP PS bloğunda aktifleştirilen tüm çevrebirimleri (tikli olanlar) gösterilmektedir.

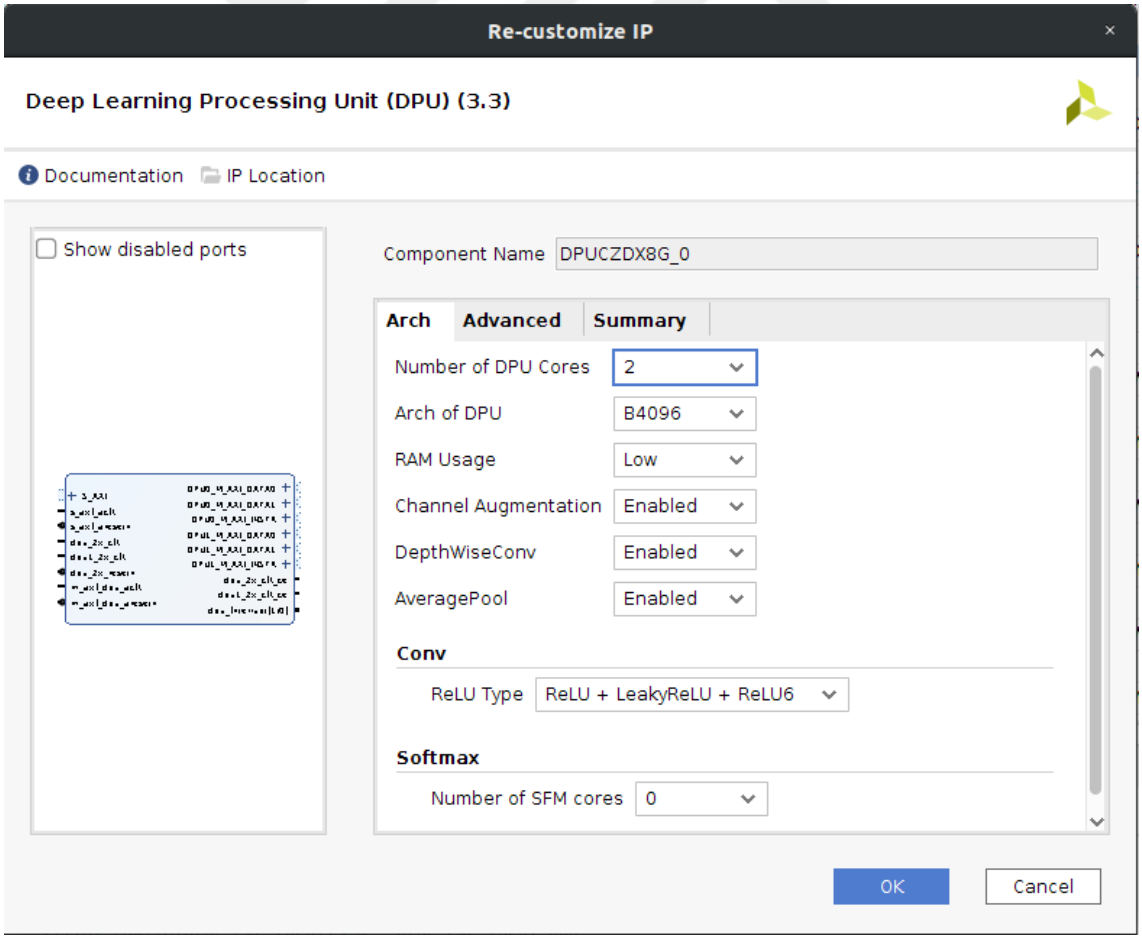


Şekil 3.4 ZynqMP Ultrascale+ bütünsel donanım mimarisi

Kamera görüntüsü almak için USB, kamera görüntüsünü basmak için DisplayPort,

SoC'u önyükleyebilmek için SD eMMC ve SoC'un bir PC ile haberleşebilmesi için UART arayüzleri bu tasarımda kullanılmıştır. UART Baud Rate 115200 olarak ayarlanmıştır. Gerçek zamanlı bir sonuç elde edilebilmesi için yüksek hızda veri aktarımı sağlayan USB ve DisplayPort arayüzleri tercih edilmiştir.

DPU IP'si, Vivado blok tasarımında kullanılırken DPU çekirdeği sayısı, DPU kaynak mimari çeşidi, RAM kullanımı, pooling çeşidi gibi konfigurasyonları yapılmalıdır. Şekil 3.5 'te gösterildiği üzere bu tasarımda DPU çekirdeği sayısı 2, kaynak mimarisi B4096 ve Block RAM kullanımı düşük olarak konfigüre edilmiştir. Çalışmada tek bir CNN uygulaması çalıştırılacağı için 2 DPU çekirdeği yeterlidir. 2 'den fazla çekirdek seçildiğinde hem kaynak kullanımında artış gösterdiği hem de performans üzerinde gözle görülür bir artış sergilemediği görülmüştür. 1 çekirdek ise kendi başına yeterince paralellik sağlayamadığından düşük FPS elde edilmiştir. B4096 kaynak mimarisi diğer seçeneklerine kıyasla daha fazla pixel paralelliği, giriş kanal paralelliği, çıkış kanal paralelliği ve clock başına işlem sayısına sahiptir. Çalışmada kart olarak Xilinx'e ait bir geliştirme kartı olan ZCU104 kullanılmıştır. Bu karttaki BRAM kaynağı az olduğu için kullanım seviyesi düşük olarak seçilmiş, onun yerine UltraRAM tercih edilmiştir.



Şekil 3.5 DPU IP'sinin konfigürasyon penceresi

2 çekirdekli DPU IP'sinin çalışabilmesi için 3 ayrı clock girişi vardır. Üretici firmanın sağladığı ürün teknik dökümanında [13] önerilen m_axi_dpu_aclk 325 MHz'tir. Her bir çekirdek için kullanılan ve DPU içindeki DSP kaynakları için gerekli olan dpu<no>_2x_clk frekansı ise bunun iki katı yani 650 MHz ayarlanmıştır. İstenilen frekanslarda clock'ları üretebilmek için Clocking Wizard IP'si kullanılmıştır.

Ayrıca DPU, PL tarafında bulunduğu için yaptığı işlemler sonrasında ZynqMP PS'in uyarılması adına kesme (interrupt) oluşturması gerekmektedir. Bu sebeple iki çekirdek bulunduran DPU IP'sinin 2-bit'lik bir kesme sinyali, ZynqMP PS üzerinde ilgili giriş portuna bağlanmıştır.

ZynqMP PS ve DPU birbirleriyle AXI arayüzünü kullanarak haberleşir. PS ve herhangi bir PL birbirleriyle haberleşirken ihtiyaca göre Full Power Domain (FPD) ve Low Power Domain (LPD) AXI bağlantılarını kullanır.

- ZynqMP PS, DPU ile veri (payload) aktarımı yaparken bu veriler kritik önem arz ettiği için AXI'nin FPD bağlantılarını kullanır. Burda DPU master rolündedir.
- ZynqMP PS, DPU ile komut aktarımı yaparken ise AXI'nin LPD bağlantılarını kullanır. Burda DPU slave rolündedir.

Donanımsal tasarım bittikten sonra sırasıyla sentez (synthesis), yerleşim&hat çizme (Place&Route) ve bit dizisi üretimi (bitstream generation) yapılır. Sentez aşaması sonrasında IP'ler, kapı seviyesinde devreye dönüştürülür. Yerleşim&hat çizme aşamasında kapı seviyesindeki devrenin, mevcut FPGA kaynakları üzerine izdüşümü yapılır ve bu kaynaklar arasında optimizasyon ayarlarına göre en optimum şekilde hat çizilerek birbirlerine bağlanır. En sonunda FPGA'yı programlayacak olan .bit uzantılı bit dizisi üretilir.

Tablo 3.1 CLB kaynaklarının kullanım durumunu gösteren sentez raporu

Kullanılan kaynak	**Kullanım Oranı (%)**
CLB LUT	47.13
LUT (Lojik)	41.96
LUT (Hafıza)	11.72
CLB yazmaçları	44.16
Flip Flop olarak yazmaçlar	44.16
Latch olarak yazmaçlar	0.00
CARRY8	13.91

Tablo 3.1 'de PL (FPGA)'e ait sadece CLB kaynaklarının kullanım yüzdeleri verilmiştir.

Tablo 3.2 RAM kaynaklarının kullanım durumunu gösteren sentez raporu

Kullanılan kaynak	Kullanım Oranı (%)
Block RAM Tile	65.06
RAMB36/FIFO	60.90
RAMB18	4.17
URAM	87.50

PL (FPGA)'e ait sadece RAM kaynaklarının kullanım yüzdeleri Tablo 3.2 'de gösterilmektedir.

Tablo 3.3 DSP kaynaklarının kullanım durumunu gösteren sentez raporu

Kullanılan kaynak	Kullanım Oranı (%)
DSP	80.67

Tablo 3.3, aritmetik işlemlerde kullanılan PL (FPGA)'e ait sadece DSP kaynaklarının kullanım yüzdelerini içermektedir.

Statik güç tüketimi tranzistörlerdeki sızıntı akımının sonucu oluşur. Dinamik güç tüketimi ise yüksek çalışma frekansının sonucu oluşur. Farklı frekanslar arasında örnekleme yapılan kaynakların birlikte kullanımı ile hesaplanır. Tablo 3.4 'te tüm işlemlerde kullanılan SoC'a ait kaynakların güç tüketimi kullanım yüzdeleri verilmiştir.

Sadece FPGA tasarımı yapılan çalışmalarda FPGA'yı programlamak için .bit dosyası yeterli olurken, hem FPGA hem de CPU bulunduran SoC'ları programlamak için başka dosyalara da ihtiyaç vardır. Bu çalışmada Vivado akışı sonrasında üretilen .bit dosyası, .xsa dosyası, arch.json dosyası ve .xclbin dosyası sonraki aşamalarda kullanılmıştır.

- .bit ve .xsa dosyaları Petalinux akışında,

Tablo 3.4 Kaynakların güç tüketimi

Güç tüketim türü	Kullanılan kaynaklar	Güç tüketimi (Watt)	Güç tüketim oranı (%)	Toplam güç tüketimi (Watt)
Dinamik	Clock'lar	2.559	18	14.528 (%95)
	Sinyaller	3.553	24	
	Lojik	1.871	13	
	BRAM	0.791	5	
	URAM	0.855	6	
	DSP	2.128	15	
	PLL	0.059	1	
	PS	2.682	18	
Statik	PL	0.691	86	0.806 (5%)
	PS	0.115	14	

- arch.json dosyası Vitis-AI akışında,
- .xsa ve .xclbin dosyası Vitis SDK akışında kullanılmıştır.

3.2 Petalinux Üzerinde Gömülü Linux Ortamı Akışı

Petalinux aracı [16], Xilinx'e ait SoC'lerde gömülü Linux komponentleri tasarlamayı ve üretmeyi sağlar. Bu çalışmada Petalinux versiyon 2020.2,

- USB, SD eMMC, DisplayPort ve UART arayüzlerinin Linux cihaz sürücülerinin (device driver) konfigürasyonu yapılmasında,
- DPU IP'si için cihaz ağacı (device tree) düzenlenmesinde,
- OpenCV, Vitis AI, XRT kütüphaneleri ve x11, v4lutils, gstreamer paketlerinin eklenmesinde,
- oluşturulan gömülü Linux imajının önyükleme (boot) ayarlarının düzenlenmesinde

kullanılmıştır. Petalinux aracı hakkında daha fazla bilgiler A.1 'de verilmiştir.

3.2.1 Petalinux Projesinin Oluşturulması

Petalinux projesi oluşturulurken ilk aşamada kullanılan donanım platformu parametre olarak kullanılır. Bunun için Liste 3.1 'de verilen komut çalıştırılmıştır.

Liste 3.1 Petalinux projesi oluşturma komutu

```
petalinux-create --type project --template <PLATFORM> --name <PROJE_ADI>
```

Bu çalışmada ZynqMP Ultrascale+ SoC kullanıldığı için <PLATFORM> parametresi olarak "ZynqMP" girilmiştir.

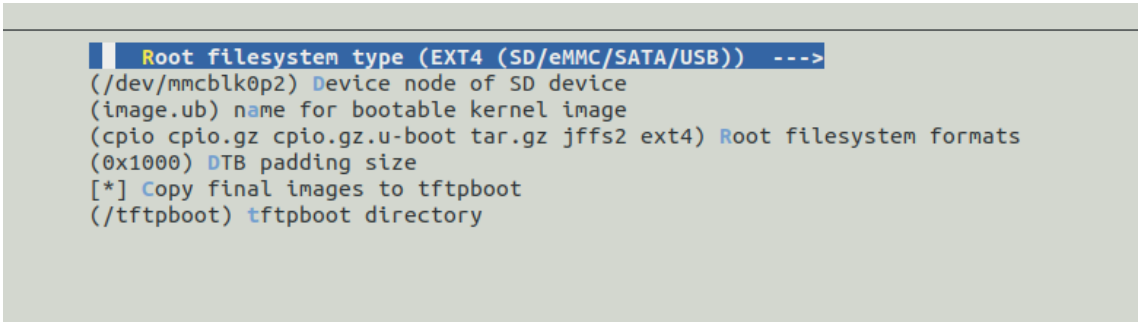
3.2.2 Donanım İlklendirme

Daha öncesinde oluşturulan Petalinux projesi, belirtilen donanım yapılandırmasını yansıtacak şekilde başlatılır. Bunun için Liste 3.2 'de verilen komut çalıştırılmıştır. Burada parametre olarak Vivado akışında üretilen .xsa dosyasını alır.

Liste 3.2 Donanım ilklendirme

```
petalinux-config --get-hw-description=<xsa\_dosyasini\_iceren\_dizin>
```

Şekil 3.6 'da, bu çalışmada yapılan önyükleyici konfigürasyon ayarları ve seçilen dosya formatları gösterilmiştir. Önyükleyici SD kart üzerinden sistemi başlatmakta ve dosya formatları olarak ext4, cpio, cpio.gz girilmiştir.



Şekil 3.6 Önyükleyici ayarları sekmesi

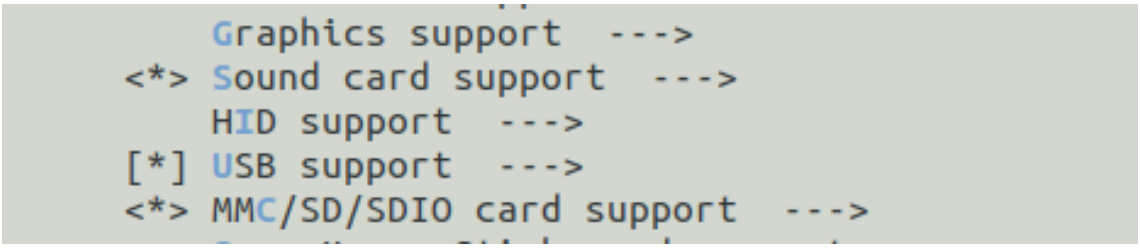
3.2.3 Cihaz Sürücülerinin Yapılandırılması

Bu aşamada Vivado akışında donanım olarak sisteme eklenen USB, DisplayPort, SD eMMC ve UART arayüzlerine üst seviye yazılımından erişebilmek için Linux cihaz sürücüleri konfigüre edilir. Bu sürücüleri konfigüre etmek için Liste 3.3 'te verilen komut çalıştırılmıştır.

Liste 3.3 Linux cihaz sürücüleri ayarlanması

```
petalinux-config -c kernel
```

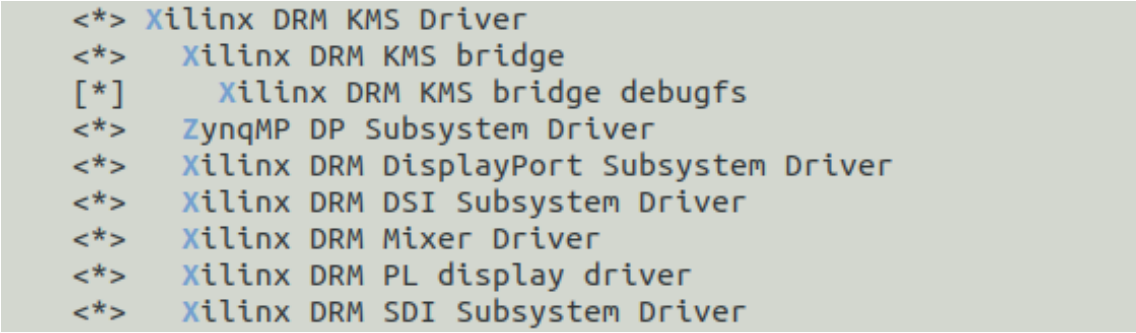
Komut çalıştırılarak konfigüre edilen arayüzler Şekil 3.7 'de verilmiştir. Burada USB, SD eMMC ve DisplayPort arayüzlerinin cihaz sürücüleri ayarları yapılmıştır.



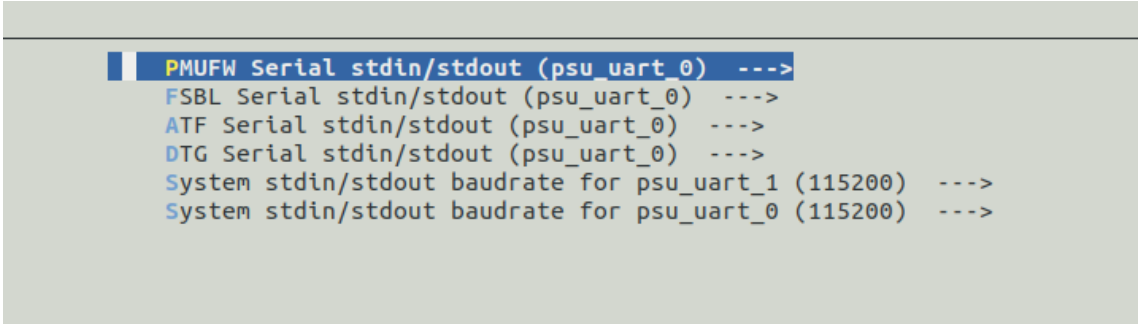
Şekil 3.7 Gerekli arayüzlerin cihaz sürücüleri ayarları sekmesi

Şekil 3.8 'de DisplayPort DRM cihaz sürücüsü ayarları yapılmıştır.

Şekil 3.9 'da UART arayüzünün cihaz sürücü ayarları yapılmıştır. Baud Rate, Vivado akışındaki gibi 115200 ayarlanmıştır.



Şekil 3.8 DisplayPort cihaz sürücü ayarları



Şekil 3.9 UART cihaz sürücü ayarları

3.2.4 Kütüphane ve Paketlerin Yüklenmesi

Bu çalışmada kütüphaneler ve paketlerin eklenmesi için Liste 3.4 'te gösterilen komut çalıştırılmıştır.

Liste 3.4 Kütüphanelerin projeye eklenmesi

```
petalinux-config -c rootfs
```

Burada projeye görüntü işleme fonksiyonları için OpenCV, CNN önişleme fonksiyonları ve sınıfları için Vitis AI, donanım ve üst seviye yazılım arasında ara katman görevi gören XRT (Xilinx Runtime) kütüphaneleri ve komplex veri yapıları gibi yapılar sağlayan bazı yardımcı paketler olan gstreamer, v4lutils ve x11 paketleri yüklenerek konfigüre edilmiştir.

3.2.5 DPU IP'si için Cihaz Ağacı Güncellemesi

Cihaz ağacı, donanımı tanımlayan bir veri yapısıdır. Bu, Linux gibi bir işletim sistemi tarafından okunabilen donanımı açıklar. Böylece makinenin ayrıntılarını kodlamasına gerek kalmaz. Petalinux aracı içerisinde bulunan cihaz ağacı üreteci (device tree generator), ZynqMP PS içindeki arayüzler için gereken cihaz ağaçlarını, "system-user.dtsi" ismindeki dosya içinde otomatik olarak oluşturur. Ama PL tarafındaki arayüzler için cihaz ağaçlarını elle oluşturmak gerekir. Oluşturulan cihaz

ağacı, üretilmiş olan "system-user.dtsi" içine eklenir. Bunun için Vivado akışındaki bilgilerden yararlanılır.

Bu çalışmada kullanılan DPU IP'si PL tarafında bulunduğu için işletim sisteminin DPU'yu tanıması adına cihaz ağacı oluşturuldu. Yazılan cihaz ağacı Liste 3.5 'te gösterilmiştir.

Liste 3.5 DPU IP'sinin cihaz ağacı

```
&amba{
    dpu{
        #address-cells = <1>;
        #size-cells = <1>;
        compatible = "xilinx,dpu";
        base-addr = <8x8F000000>;
        dpucore {
            compatible = "xilinx,dpucore";
            interrupt-parent = <&gic>;
            interrupts = <0x0 0x59 0x4 0x0 0x68 0x4>;
            core-num = <0x2>;
        };
    };
};
```

Burda "base-addr" bilgisi Vivado akışındaki adres editöründe gösterilen IP adreslerinden elde edildi. "interrupts" satırındaki bilgilerde 6 değer bulunmaktadır. Bunların 3'ü DPU'nun bir çekirdeği, diğer 3'ü de diğer çekirdeğine ait bilgilerden oluşmaktadır. İlk değer birinci çekirdek için paylaşımlı çevrebirim kesme (shared peripheral interrupt) değeri, ikinci değer birinci çekirdek için Linux kesme numarası, üçüncü değer de birinci çekirdek için örnekleme çeşidi ki DPU için bu yüksek seviye tetiklemeli örneklemedir. Satırdaki 4., 5. ve 6. değerler de ikinci çekirdek için aynı bilgileri temsil etmektedir. "core-num" bilgisi ise kullanılan DPU çekirdeği sayısını ifade eder. Bunların dışındaki satırlarda verilen değerler değiştirilmez.

Linux kesme numaraları, üretici firmanın sağladığı teknik referans dökümanında[13] verildiği üzere Tablo 3.5' teki GIC IRQ bilgilerinden hesaplanarak bulunur.

Tablo 3.5 IRQ numaraları tablosu[13]

PS Arayüzü	GIC IRQ #	Linux IRQ #
PL_PS_IRQ1[7:0]	143:136	111:104
PL_PS_IRQ0[7:0]	128:121	96:89

Hesaplama şekli şöyledir; DPU'nun birinci çekirdeği için (3.1) 'de, DPU'nun ikinci

çekirdeği için (3.2) 'de gösterildiği gibi GIC IRQ numarasından 32 çıkartılır ve hexadecimal formata dönüştürülür.

$$\begin{aligned} 121 - 32 &= 89 \\ &= (59)_{16} \end{aligned} \tag{3.1}$$

$$\begin{aligned} 136 - 32 &= 104 \\ &= (68)_{16} \end{aligned} \tag{3.2}$$

3.2.6 Linux İmaj Dosyalarının Üretilmesi

DPU IP'si için cihaz ağacı güncellendikten sonra son olarak gömülü Linux komponentlerini üretmek için Liste 3.6 'da gösterilen komutlar çalıştırılır.

Liste 3.6 Linux imaj dosyalarını üretme komutları

```
petalinux-build
petalinux-build --sdk
petalinux-package --sysroot
petalinux-package --boot --fsbl /<petalinux_projesi>/images/linux/fsbl.elf \
                         --fpga /<vivado_projesi>/src/run/system.bit \
                         --pmufw /<petalinux_projesi>/images/linux/pmufw.elf \
                         --atf /<petalinux_projesi>/images/linux/bl31.elf \
                         --u-boot /<petalinux_projesi>/images/linux/u-boot.elf
```

Bu komutlardan sonra üretilen dosyalar "boot.bin", "image", "boot.scr", "system.dtb" ve "sysroot"'tan oluşmaktadır.

- "boot.bin" dosyası önyüklemesi yapılacak ikili (binary) dosyasıdır,
- "image" dosyası Linux için (ikincil) önyüklemeyi yapacak olan dosyadır,
- "boot.scr" dosyası u-boot ilklendirmesi için gereken betik (script)lerdir,
- "system.dtb" dosyası u-boot'un sistem kurulumunu anlamak için önyükleme sırasında okuduğu cihaz ağacı bloğu (node)dur,
- "sysroot" dizini ise kullanılacak olan tüm kütüphane ve paket dosyalarını içeren dizindir.

3.3 Vitis AI Akışı

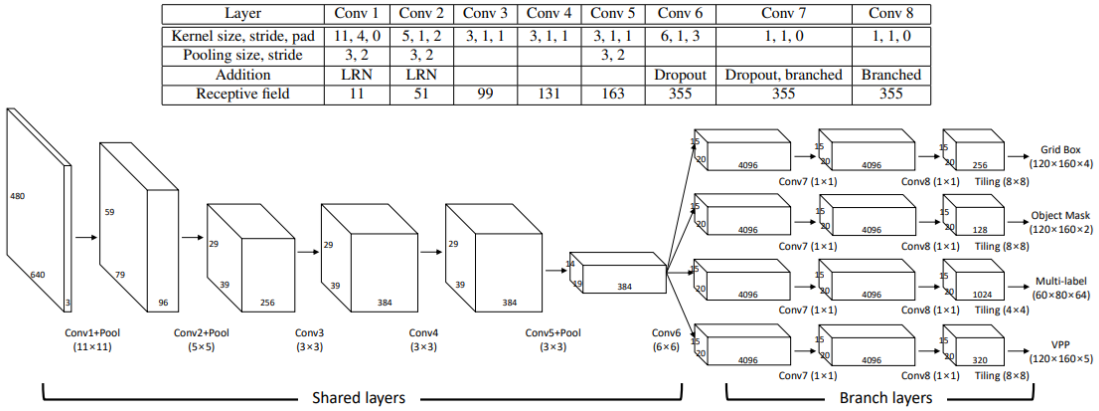
Xilinx® Vitis™ AI [17] aracı, hem uç cihazlar hem de veri merkezleri için özelleştirilmiş kartlar dahil olmak üzere Xilinx donanım platformlarında yapay zeka çıkarımı (inference) için kullanılan bir geliştirme yığınıdır. Bu çalışmada Vitis AI versiyon 1.3.2 kullanılmıştır.

3.3.1 Yol Şerit Tespiti CNN Modeli

Bu çalışmada gerçeklemek için Caltechlane [18] veri kümesi ile eğitilmiş olan VPGNet [19] CNN modeli tercih edilmiştir. Arşiv, etiketli şeritlere ek olarak, Alice (Team Caltech'in veri toplamak için kullandığı araba)'e monte edilmiş bir kameradan alınan 1225 ayrı kareyi içermektedir. Bu modelin tercih edilmesinin sebebi, 17 farklı şerit ve yol bulunduran yaklaşık 20000 görüntü ile benchmark testleri yapılan VPGNet modelinin, şeritleri ve yol işaretlerini tespit edip sınıflandırabilmekteki başarısıdır. Yapılan testlerde, metodun gerçek zamanlı olarak yüksek performans ve doğrulukta çalıştığı görülmüştür. Şekil 3.10 'da, seçilen CNN modeline ait testlerde elde edilen sonuçlar gösterilmektedir.

Şekil 3.10 VPGNet modeli yol şerit tespiti sonuçları [19]

Modelin genel mimarisi Şekil 3.11 'de gösterilmiştir. Ağın dört görev modülü vardır ve her görev tamamlayıcı işbirliğini gerçekleştirir: gridbox regresyonu, nesne algılama, çok etiketli sınıflandırma ve ufuk noktasının tahmini[19].



Şekil 3.11 CNN modelinin mimarisi[19]

3.3.2 Vitis AI Ortamı

Vitis AI aracı, Docker konteyner [20] ortamı aracılığıyla çalıştırılır. Bu sayede her bir proje birbirinden izole edilmiş ve bağımsız halde çalışabilmektedir. Vitis AI Tensorflow, Caffe, Pytorch, Darknet ve Neptune gibi çalışma ortamlarını (framework) destekler. Her çalışma ortamının Vitis AI aracı içinde farklı bir çalışma akışı vardır. Liste 3.7 'de Vitis AI aracı açıldığında çıkan çalışma ortamları gösterilmektedir. Modelin çeşidine göre içlerinden bir seçilerek Vitis AI akışı sürecine başlanır.

Liste 3.7 Vitis AI ortamı

```
==========================================


__      ___ _   _                    _____
\ \    / (_) | (_)             /\   |_   _|
 \ \  / / _| |_ _ ___ ______   /  \    | |
  \ \/ / | | __| / __|______/ /\ \   | |
   \  /  | | |_| \__ \     / ____ \ _| |_
    \/   |_|\__|_|___/    /_/    \_\_____|

==========================================

Docker Image Version:  latest
Build Date: 12-02-2022
VAI_ROOT: /opt/vitis_ai

For TensorFlow 1.15 Workflows do:
     conda activate vitis-ai-tensorflow
For Caffe Workflows do:
     conda activate vitis-ai-caffe
For Neptune Workflows do:
     conda activate vitis-ai-neptune
For PyTorch Workflows do:
     conda activate vitis-ai-pytorch
For TensorFlow 2.3 Workflows do:
     conda activate vitis-ai-tensorflow2
For Darknet Optimizer Workflows do:
     conda activate vitis-ai-optimizer_darknet
For Caffe Optimizer Workflows do:
     conda activate vitis-ai-optimizer_caffe
For TensorFlow 1.15 Workflows do:
     conda activate vitis-ai-optimizer_tensorflow
For LSTM Workflows do:
```

```
     conda activate vitis-ai-lstm
Vitis-AI /workspace >
```

VPGNet modeli, Caffe ortamı ile çalıştığından "conda activate vitis-ai-caffe" ile Caffe çalışma akışı üzerinden devam edildi. Seçilen CNN, Caffe'de eğitildikten sonra, Vitis AI Quantizer ile kayan nokta ağırlıklarının caffemodel dosyası kuantalandı ve 8 bitlik bir tamsayı gösterimi (INT8) oluşturuldu. Bundan, Vitis AI Compiler ile DPU için mikro komutlar içeren .xmodel dosyası oluşturuldu ve SoC üzerinde VART (Vitis AI Runtime) C++ API'leri aracılığıyla çalışma zamanında yürütüldü.

3.3.3 CNN Model Eğitimi

CNN modeli[21] parametrelerinin bazıları FPGA üzerinde işlemsel yük, hafıza kaynakları, hafıza bantgenişliği ve DPU'nun desteklediği katman çeşitleri,kernel, stride ve padding boyutları gibi kısıtlamalardan dolayı güncellendi. Yapılan değişiklikler:

- giriş boyutları olarak 640x480x3 yerine 512x288x3 ayarlandı,
- kernel boyutu 11x11x1 yerine maximum 8x8x1 olarak ayarlandı,
- stride boyutu maximum 1 olarak ayarlandı,
- padding boyutu maximum 1 olarak ayarlandı,
- LRN katmanları, BN katmanları ile değiştirildi.

VPGNet ağının güncellenmiş mimarisi Tablo 3.6 'da verilmiştir.

Tablo 3.6 Ağın güncellenmiş mimarisinin parametreleri

Katman	**Conv 1**	**Conv 2**	**Conv 3**	**Conv 4**	**Conv 5**	**Conv 6**	**Conv 7**	**Conv 8**
Kernel boyutu, stride, pad	8, 1, 0	5, 1, 1	3, 1, 1	3, 1, 1	3, 1, 1	6, 1, 1	1, 1, 0	1, 1, 0
Pooling boyutu, stride	3, 1	3, 1			3, 1			
Toplama	BN	BN				Dropout	Dropout, Branched	Branched

3.3.4 Modelin Kuantalanması

Kayan nokta kullanmak teoride daha yüksek hassasiyet sağlar. Bu nedenle, çoğu zaman CNN'leri kayan nokta konusunda eğitmek daha yüksek performans sağlar,

ancak dezavantajı, kayan nokta aritmetiği yapacak donanım tasarımının daha karmaşık olmasıdır. Kuantalama, bit uzunluklarını azaltır ve dolayısıyla bellek kullanımını azaltır. 32-bit kayan noktadan sabit 8-bit geçmek, bellek kullanımını dört kat azaltır.

Seçilen CNN, Caffe'de eğitildikten sonra, Vitis AI Quantizer kullanılarak kayan nokta ağırlıklarının caffemodel dosyası kuantalandı ve 8 bitlik bir tamsayı gösterimi ("INT8" olarak adlandırılır) oluşturuldu. Kuantalama komutu Liste 3.8 'de gösterilmiştir.

Liste 3.8 Vitis AI Quantizer çalıştırma

```
vai_q_caffe quantize -model float.prototxt -weights float.caffemodel
```

Kuantalama komutu için 3 parametre kullanılır.

- kayan nokta CNN modelini tanımlayan .prototxt dosyası,
- CNN'nin kayan noktadaki ağırlıklarını bulunduran .caffemodel dosyası,
- kalibrasyon veri seti. Kalibrasyon seti, genellikle eğitim setinin veya gerçek uygulama görüntülerinin bir alt kümesidir. .prototxt dosyasının içinde çağrılır.

Kuantalama yapıldıktan sonra, iki çıktı dosyası oluşturulur. Bunlar daha sonra Vitis AI Compiler'ın girdileri olacaktır:

- kuantalanmış CNN modelini tanımlayan yeni "deploy.prototxt" dosyası,
- sabit nokta kuantalanmış ağırlıkları bulunduran "deploy.caffemodel" dosyası.

3.3.5 Modelin Derlenmesi

Vitis AI Compiler, kuantalanmış CNN modelini ayrıştırır (parse) ve bir veri akışı ve bir kontrol akışından oluşan bir hesaplama grafiği üretir. Daha sonra BN yoluyla veya verileri yeniden kullanarak veri ve kontrol akışını optimize eder. En sonunda çalıştırılacak kodu üretir. Model, bazıları DPU'da, bazıları ise CPU'da çalıştırılacağı şekilde çekirdeklere (kernel) bölünür. Modelin derlenmesi için kullanılan komut Liste 3.9 'da gösterilmiştir.

Liste 3.9 Vitis AI Compiler çalıştırma

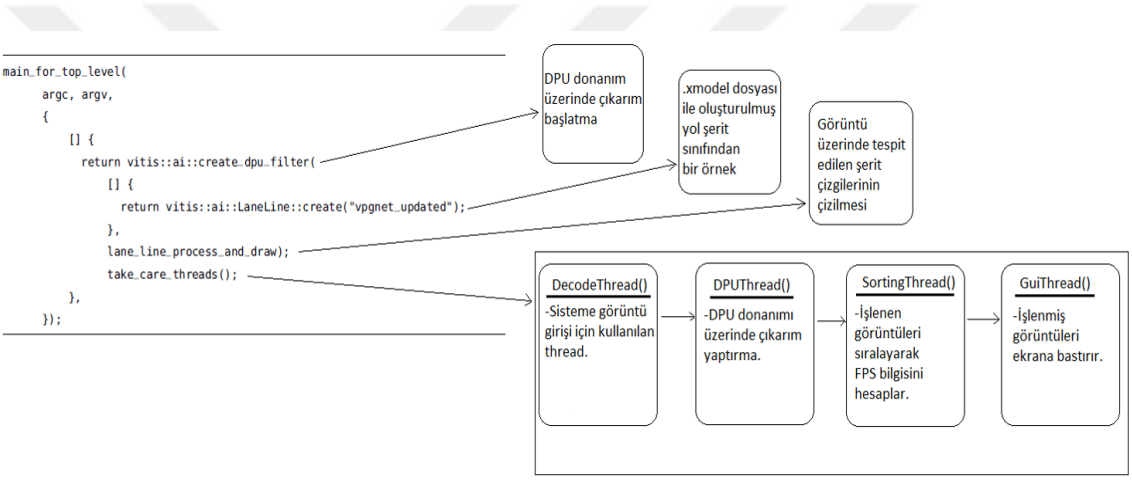
```
vai_c_caffe -p /<dosyanin>/<bulundugu>/<dizin>/deploy.prototxt \
            -c /<dosyanin>/<bulundugu>/<dizin>/deploy.caffemodel \
            -a /<dosyanin>/<bulundugu>/<dizin>/arch.json \
            -o /<kaydedilecegi>/<dizin> -n netname
```

Bu komutun parametreleri Vitis AI Quantizer'ın daha önce ürettiği "deploy.prototxt", "deploy.caffemodel" ve Vivado akışında üretilen "arch.json" dosyalarıdır. Derleme sonrasında DPU'nun çalıştıracağı komutları bulunduran .xmodel dosyası üretildi.

3.4 Vitis SDK Akışı

Bu aşamada, daha önceki akışlarda üretilen dosyalar kullanılarak (donanım tanımlama dosyaları, gömülü Linux imaj dosyaları ve model tanımlama dosyası) C++ ile bir üst-seviye uygulama geliştirildi. Vitis SDK 2020.2 versiyonu kullanıldı.

Vitis SDK [16] aracının bulundurduğu özel bir derleyici olan v++ derleyicisi kullanılarak tüm bu dosyalar ile SoC üzerinde çalıştırılabilir bir uygulama geliştirilmesini sağlar.



Şekil 3.12 Üst seviye uygulama ana fonksiyonu ve görevleri

Ana fonksiyon ve hiyerarşisi Şekil 3.12 'de gösterilmiştir.

- *create_dpu_filter* fonksiyonu DPU üzerinde çıkarım (inference) görevini başlatır. Vitis AI kütüphanesinden gelir. İki parametre alır:
  - Vitis AI akışından elde edilmiş olan .xmodel dosyası (*vpgnet_updated*) kullanılarak oluşturulan yol şeridi sınıfından (*LaneLine*) bir örnek (instance) veri yapısı. Uygulama, .xmodel dosyası üzerinden sinir ağındaki tensor parametrelerine ulaşır.
  - Bu veri yapısı kullanılarak oluşturulan çizgilerin, kameradan alınan görüntü üzerinde çıkarım (inference) yapılarak yol şeritlerinin üzerine çizildiği görüntü (*lane_line_process_and_draw*).
- Uygulama, kamera görüntülerini thread denen paralel programlama yapıları yoluyla aynı zaman biriminde komplex veri yapılarını kullanarak depolar

ve işler. Thread'ler PS tarafında bulunan ARM Cortex A53 işlemcisinin 4 çekirdeği ve PL tarafında bulunan DPU'nun 2 çekirdeği olmak üzere toplamda 6 çekirdeğin üzerinde paralel bir yazılım mimarisi tasarlamayı sağlar. Thread ve komplex veri yapıları kütüphaneleri, Petalinux akışında projeye yüklenmiştir. *take_care_threads()* fonksiyonu bunun için kullanıldı. Fonksiyon içerisinde,

- kameradan giriş görüntülerini almak için *DecodeThread*,
- işlenen görüntüleri ekrana bastırmak için *GuiThread*,
- asıl işlem olan çıkarımın çalışması için DPU'nun ilklendirmesini yapan *DpuThread*,
- işlenmiş ve işlenecek olan görüntülerin sıralamasını yaparak FPS hesaplamasını yapmak için *SortingThread* yapıları oluşturuldu.
- Karelere (frame) ait bilgileri (channel_id, frame_id, fps, width, height) tutmak için *FrameInfo* veri yapısı,
- görüntü kanalları yaratmak için threadlerden gelen bilgileri kullanan (çözülmüş threadler, görüntü bastıracak olan threadler, DPU ilklendirecek olan threadler, sıralanmış olan threadler) *Channel* veri yapısı oluşturuldu.

İşlenecek olan ve işlenmiş olan görüntüler, OpenCV kütüphanesi ve gstreamer paketi yoluyla USB ve DisplayPort alt-seviye cihaz sürücüleriyle konuşur. OpenCV ve gstreamer, Petalinux akışında projeye yüklenmiştir.

Uygulama kodu dosyası, Vitis SDK üzerinde v++ derleyicisi ile önceki akışlarda üretilen dosyalarla birlikte derlenerek bir Linux-çalıştırılabilir (executable) dosya üretildi.

Uygulama, platform üzerinde çalıştırılırken gerekli tüm dosyalar bir SD karta yüklenir. Yüklenen dosyalar;

- "boot.scr" dosyası,
- "boot.bin" dosyası,
- .xclbin dosyası,
- "image" dosyası,
- "system.dtb" dosyası,
- .xmodel dosyası,

- Linux-çalıştırılabilir üst-seviye uygulama dosyası,
- ve kütüphane ve paketlerin bulunduğu "sysroot" dizinidir.

Her bir dosyanın ne işe yaradığı ve nerde üretildiği önceki akışlarda anlatılmıştır.

.xclbin dosyası DPU donanım bilgilerini (komut kümeleri, register adresleri) bulundurur. Uygulamada kullanılan sinir ağı modeli, bu bilgilere erişmek için aşağıdaki hiyerarşiyi izler;

- Uygulama, .xclbin dosyasına *xcl_*()* fonksiyonları üzerinden,
- *xcl_*()* fonksiyonlarına XRT (Xilinx Runtime) kütüphanesinden,
- XRT kütüphanesine XIR (Xilinx Intermediate Representation) kütüphanesinden
- ve XIR kütüphanesine de Vitis AI kütüphanesinden erişilir.

Uygulamayı çalıştırmak için SoC, bir PC'ye USB üzerinden bağlandı. PC de seri bir haberleşme sağlamak için Putty aracı[22] kullanıldı. UART arayüzünden 115200 Baud Rate bağlantı hızında COM portları üzerinden seri bir haberleşme kurularak uygulama çalıştırıldı.

4
DENEYSEL SONUÇLAR

Çalışmayı test etmek amacıyla giriş olarak kameradan gerçek zamanlı alınan 512x288 boyutunda yol görüntüleri, üzerinde FPGA ve ARM işlemci bulunduran ZynqMP Ultrascale+ SoC platformuna giriş olarak verilmiş ve yaklaşık 30 FPS performansında yol şeritleri tespit edilerek çıkış alınmıştır.

Şekil 4.1' de bir monitörde oynatılan bir yol videosunun kameradan giriş görüntüsü olarak alındığı gösterilmektedir.

Şekil 4.2 'de gösterildiği üzere kameradan alınan görüntü, üzerinde ZynqMP Ultrascale+ SoC bulunduran platforma USB yoluyla aktarılarak uygulama çalıştırılmış ve DisplayPort üzerinden görüntü başka bir monitöre basılmıştır.

Şekil 4.3 'te alınan çıkış görüntüsü ve FPS değeri verilmiştir. Görüldüğü üzere yaklaşık 30 FPS performansıyla birlikte şerit çizgilerinin saptandığı bir sonuç elde edilmiştir.

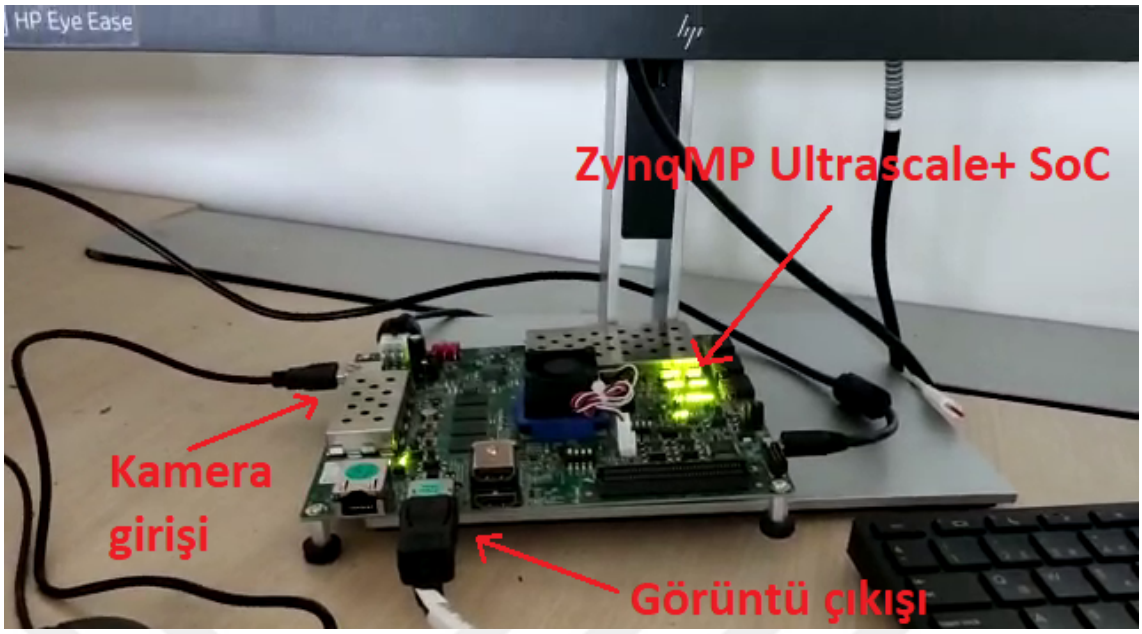
Yol şeritlerinin eskiyerek soluklaştığı, virajların olduğu ve güneş ışığının sürekli değiştiği daha zorlayıcı videolar da giriş görüntüsü olarak denenmiş ve sonuçlar alınmıştır. Şekil 4.4 'te görüldüğü üzere virajın bulunduğu bir yola ait video görüntüsü kamera yoluyla giriş olarak alınmıştır.

Çıkışı alınan görüntüde (Şekil 4.5) düz çizgi tespit edilirken daha fazla eğikliğe sahip olan kesikli çizgileri algılamakta zorlandığı görülmektedir. Burada viraj açısının derecesi ile birlikte ve şerit çizgilerinin sürekliliğinin, şerit tespitini etkileyen parametreler olduğu kanıtlanmaktadır.

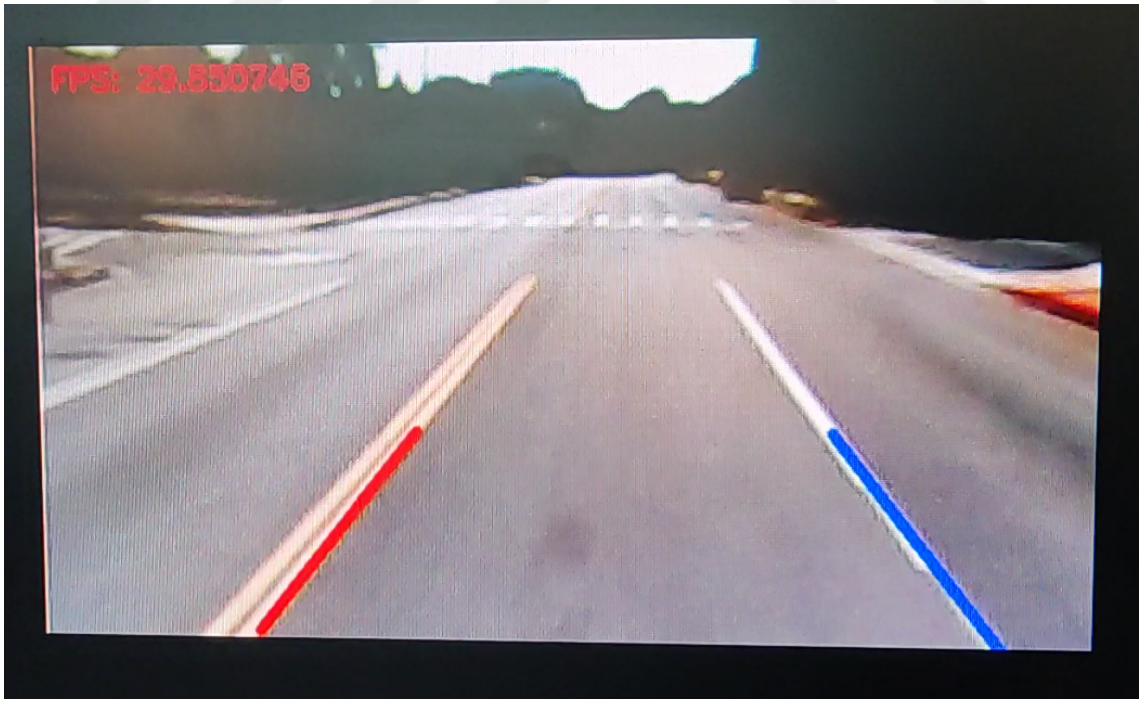
Bazı yollarda zamana bağlı olarak asfalt üzerinde eskimeler meydana gelebilir. Şekil 4.6 'da şerit ortasına kadar asfaltı yenilenmiş bir yola ait video görüntüsü kamera yoluyla giriş olarak alınmıştır.

Şekil 4.7 'de görüldüğü üzere eski ve yeni asfaltın kesiştiği yeri de ayrıca şerit olarak algılanmaktadır. Buradan yolun ve şerit çizgilerinin eskime derecesinin, şerit tespit

Şekil 4.1 Giriş görüntüsü olarak bir yol videosunun kameradan alınması



Şekil 4.2 Görüntünün SoC platformuna aktarılması



Şekil 4.3 Alınan çıkış görüntüsü

Şekil 4.4 Virajın oluğu bir yol görüntüsü çıkışı

Şekil 4.5 Virajın oluğu bir yol görüntüsü çıkışı

Şekil 4.6 Eski ve yeni asfaltın bulunduğu yol görüntüsü

etmede etkili olan parametreler olduğu görülmektedir.

Şekil 4.7 Asfaltı yenilenmiş yol görüntüsünün çıkışı

Tablo 4.1 'de bu alanda yapılmış bazı çalışmaların giriş-çıkış arayüzleri, işlenen görüntü boyutları, kullanılan donanımlar ve performansları kriter alınarak yapılmış kıyaslamaları görülmektedir. Önerilen çalışmada diğer çalışmalara kıyasla giriş ve çıkış arayüzleri olarak hem yaygın bulunan hem de yüksek hızlı veri aktarımı sağlayan arayüzler seçilmiş, gerçek zamanlı bir işlem hattı (pipeline) sağlanmış ve yaklaşık 30 FPS gibi iyi bir performans elde edilmiştir.

Tablo 4.1 Farklı çalışmaların kıyaslamaları

	Giriş arayüzü	**Çıkış arayüzü**	**İşlenen görüntü boyutu**	**FPS**	**Platform**
Hwang et al. [8]	D5M kamera (Altera bağımlı)	GPIO LCD Ekran(240x320)	-	24	FPGA
Komorkiewicz et al. [9]	Hafızada kayıtlı video	DVI	1280x960	30	FPGA
Li et al. [11]	On-board kamera	-	1080p	Çıkartım için 30, sınıflandırma için 1	SoC+bulut
Khongprasongsiri et al. [12]	-	-	480x270	130	SoC
Önerilen çalışma	USB kamera	Displayport	512x288	30	SoC

5
SONUÇLAR VE ÖNERİLER

Bu çalışmanın sonunda FPGA ve ARM Cortex A53 işlemcisinin görev paylaşımı yaparak çalıştığı ZynqMP Ultrascale+ SoC platformu üzerinde bir evrişimli sinir ağı yoluyla yol şerit tespiti yapan bir uygulama geliştirilmiştir. Çalışmada adı geçen platformların, araçların ve metotların kullanılma sebebi;

- FPGA dokusu sayesinde sayısal donanım kaynakları ile paralel veri işleme, yüksek yoğunlukta veri girişi ve yüksek sayıda giriş/çıkış arayüzleri ile baş edebilme imkanı sağlaması,
- tamamen CNN algoritmalarını çalıştırabilmek için özel bir komut kümesi mimarisi barındıran derin öğrenme işleme birimi (DPU) kullanılarak CNN modelini en optimize şekilde çalıştırabilmek,
- Linux cihaz sürücülerinden faydalanarak USB, SATA, DisplayPort gibi yüksek hızlı arayüzlerden faydalanma imkanı sağlaması ve yazılımsal esneklik kazandırması,
- CNN modellerini donanım üzerinde çalıştırabilmek için gerekli dönüşüm ve derleme imkanlarını sağlaması ve
- bu tarz derin öğrenme hızlandırma çalışmaları için yeniden uyarlanabilir bir çerçeve sunması olarak sayılabilir.

Bölüm 4 'te aktarıldığı üzere, temiz yol videolarında benzer performans sergileyen uygulamanın, yol şeritlerinin eskiyerek soluklaştığı, sürekli virajların olduğu, orman içi gibi ışığın sürekli değiştiği zorlayıcı videolarda yol şeritlerini tanımada zorlandığı görülmüştür. Bu durum, kullanılan donanım kaynaklarının kısıtlaması sonucu modelin parametrelerinin düşürülerek güncellenmesinden kaynaklanmaktadır. Model parametrelerini yeniden daha optimize ederek güncelleme, modeli eğitmek için farklı veri kümeleri kullanma veya tamamen farklı bir modele yönelme gibi bu durumu kompanze edecek ilerleyici çalışmalar yapılacaktır.

KAYNAKÇA

[1] J. Kim, H. Shin, *Algorithm & SoC design for automotive vision systems*. Springer, 2014.

[2] A. Küçükmanisa, O. Urhan, "Road surface detection on a single image using lane detection," in *2016 24th Signal Processing and Communication Application Conference (SIU)*, IEEE, 2016, pp. 2153–2156.

[3] S. E. Mutluoğlu, T. Ölmez, "A comparison of hough transform and deep neural network methods on road segmentation," in *2019 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, IEEE, 2019, pp. 1–5.

[4] M. Karaduman, H. Eren, "Deep learning based traffic direction sign detection and determining driving style," in *2017 International Conference on Computer Science and Engineering (UBMK)*, IEEE, 2017, pp. 1046–1050.

[5] A. Kızıl, A. C. Kutlucan, C. Doğan, S. Koçak, V. Sezer, "Design and implementation of autonomous parallel and vertical parking mobile vehicle," in *2018 6th International Conference on Control Engineering & Information Technology (CEIT)*, IEEE, 2018, pp. 1–8.

[6] I. Kılıç, A. Yazıcı, Ö. Yıldız, M. Özçelikors, A. Ondoğan, "Intelligent adaptive cruise control system design and implementation," in *2015 10th System of Systems Engineering Conference (SoSE)*, IEEE, 2015, pp. 232–237.

[7] D. Dogan, S. Bogosyan, P. B. Baykas, "A low-cost embedded data collection system for traction control systems in electric vehicles," in *2019 IEEE International Conference on Mechatronics (ICM)*, IEEE, vol. 1, 2019, pp. 513–518.

[8] S. Hwang, Y. Lee, "Fpga-based real-time lane detection for advanced driver assistance systems," in *2016 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, IEEE, 2016, pp. 218–219.

[9] M. Komorkiewicz, K. Turek, P. Skruch, T. Kryjak, M. Gorgon, "Fpga-based hardware-in-the-loop environment using video injection concept for camera-based systems in automotive applications," in *2016 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, IEEE, 2016, pp. 183–190.

[10] M. Bilal, "Resource-efficient fpga implementation of perspective transformation for bird's eye view generation using high-level synthesis framework," *IET Circuits, Devices & Systems*, vol. 13, no. 6, pp. 756–762, 2019.

[11] Z. Li, J. Jin, L. Wang, J. Yang, J. Lu, "A moving object extraction and classification system based on zynq and ibm supervessel," in *2016 International Conference on Field-Programmable Technology (FPT)*, IEEE, 2016, pp. 307–310.

[12] C. Khongprasongsiri, P. Kumhom, W. Suwansantisuk, T. Chotikawanid, S. Chumpol, M. Ikura, "A hardware implementation for real-time lane detection using high-level synthesis," in *2018 International Workshop on Advanced Image Technology (IWAIT)*, IEEE, 2018, pp. 1–4.

[13] A. Xilinx, *Zynq UltraScale+ Device Technical Reference Manual*. AMD Xilinx, 2020. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug1085-zynq-ultrascale-trm.

[14] [Online]. Available: https://docs.xilinx.com/r/3.2-English/pg338-dpu/Introduction?tocId=7o8Y673V3imhiImm8i69dg.

[15] [Online]. Available: https://docs.xilinx.com/r/3.2-English/pg338-dpu/Hardware-Architecture.

[16] [Online]. Available: https://www.xilinx.com/support/download.html.

[17] [Online]. Available: https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html.

[18] M. A. E.-D. Aly, *Caltech lanes dataset*, Dec. 2020 [Online]. [Online]. Available: http://www.mohamedaly.info/datasets/caltech-lanes.

[19] S. Lee *et al.*, "Vpgnet: Vanishing point guided network for lane and road marking detection and recognition," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 1947–1955.

[20] [Online]. Available: https://www.docker.com/.

[21] [Online]. Available: https://github.com/SeokjuLee/VPGNet.

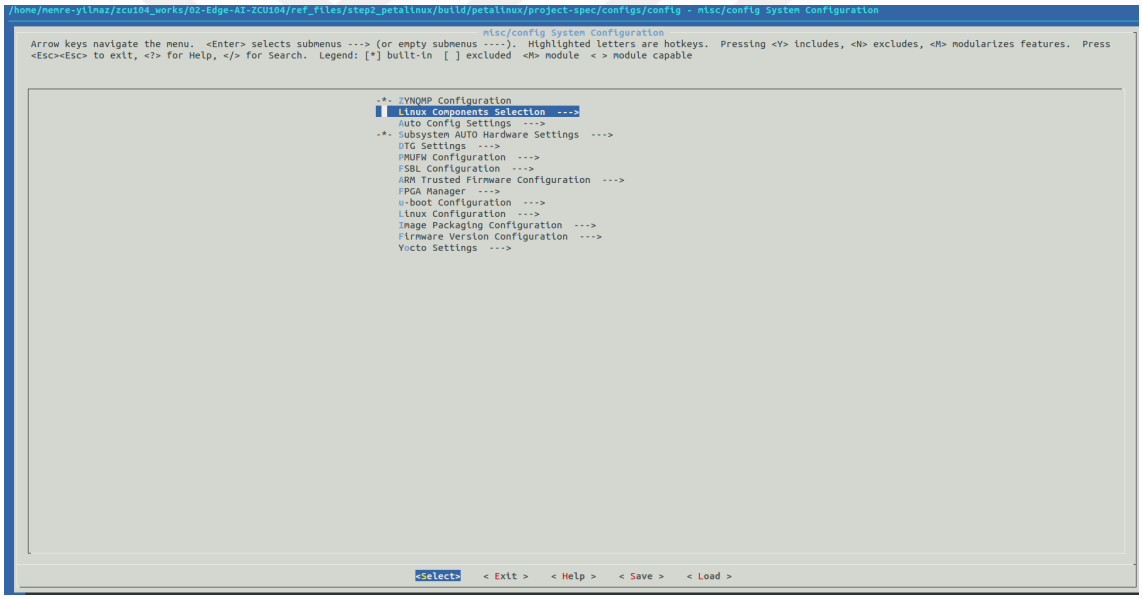
[22] [Online]. Available: https://www.putty.org/.

[23] M. E. Yılmaz, B. Erkmen, "Advanced lane line detection using heterogeneous embedded computing," in *2021 International Conference on INnovations in Intelligent SysTems and Applications (INISTA)*, IEEE, 2021, pp. 1–5.

A
EK-1

A.1 Petalinux Konfigurasyon Ekranları

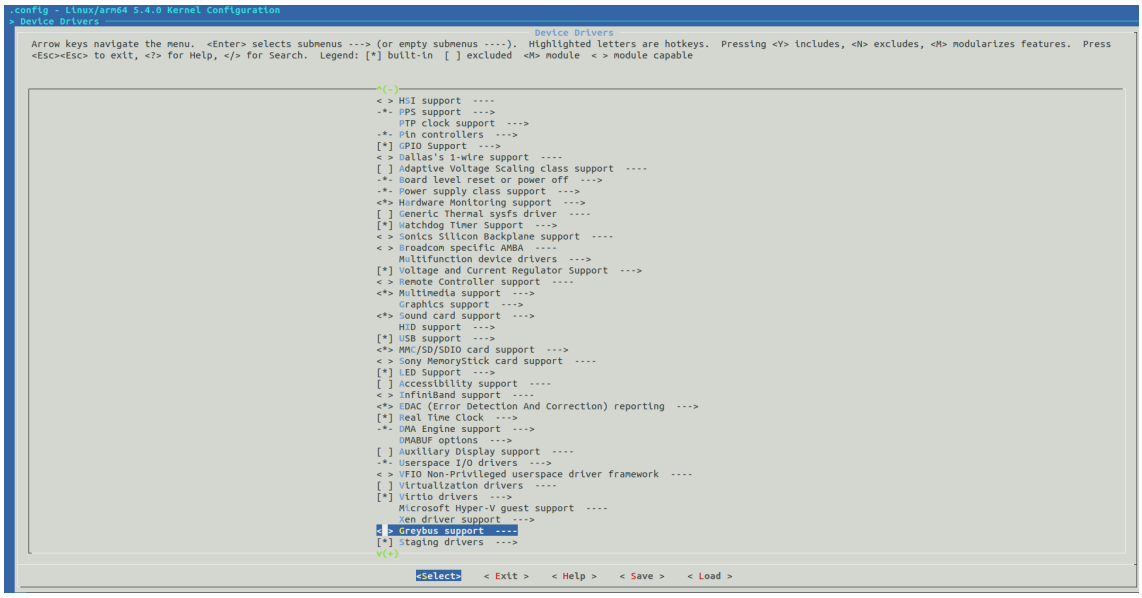
Liste 3.2 'de verilen komut sonrası çıkan konfigurasyon penceresi Şekil A.1 'de gösterilmiştir. Bu komut sonrasında konfigurasyon grafiksel arayüzü açılır. Bu süreçten etkilenen bileşenler, FSBL (ilk seviye önyükleyici) konfigürasyonunu, U-boot (Linux önyükleyicisi) seçeneklerini, Linux çekirdeği seçeneklerini ve Linux cihaz ağacını içerebilir.



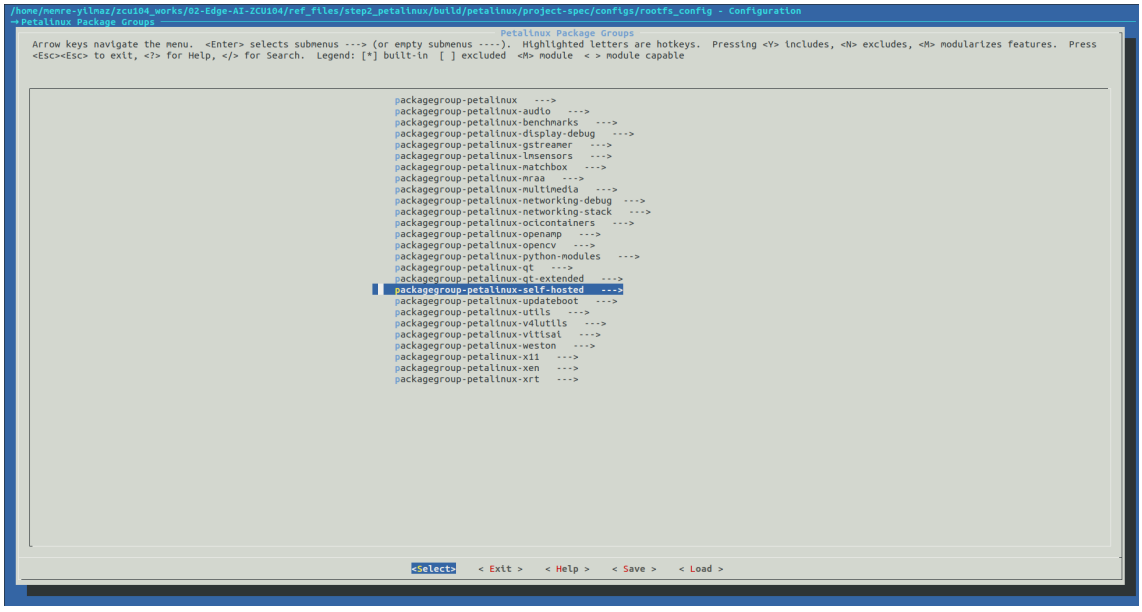
Şekil A.1 Donanım ilklendirme grafiksel arayüzü

Cihaz sürücüleri, işletim sistemi çekirdeği (kernel) seviyesinde bulunan kodlardır. Bu aşamada Vivado akışında donanım olarak sisteme eklenen arayüzlere üst seviye yazılımından erişebilmek için Linux cihaz sürücüleri konfigüre edilir. Liste 3.3 'de verilen komut çalıştırıldıktan sonra çıkan konfigürasyon grafiksel arayüzü Şekil A.2 'de verilmiştir.

Gömülü Linux projelerinde, yararlanılan kütüphaneler ve paketler root filesystem seviyesinde yüklenir. Liste 3.4 'te verilen komut çalıştırıldıktan sonra çıkan konfigurasyon penceresi Şekil A.3 'de verilmiştir.



Şekil A.2 Linux cihaz sürücüleri ayarları sekmesi



Şekil A.3 Kütüphane ve paketler listesi

TEZDEN ÜRETİLMİŞ YAYINLAR

Konferans Bildirisi

1. Advanced Lane Line Detection Using Heterogeneous Embedded Computing[23], 2021, Kocaeli, Türkiye,