ÇİP TASARIM YARIŞMASI

2024

MİKRODENETLEYİCİ TASARIM KATEGORİSİ

DETAY TASARIM RAPORU

TAKIM ADI:

KU RAMs

BAŞVURU ID:

1293734

İÇİNDEKİLER

1. SİSTEM TANIMI VE TEMEL TASARIM ÖZETİ 3

1.1. Motivasyon 3

1.2. Proje Kapsamında Yürütülen Faaliyetler 3

1.3. RTL, Simülasyon ve Çip Fiziksel Tasarım Aşaması ve Tamamlanma Oranları 4

2. PROJE DETAY TASARIMI 5

2.1. Sistem Mimarisi 5

2.1.1. Blok Şeması ve Sistem Uyumluluğu 5

2.1.2. Alt Bloklar ve Görevleri 6

2.1.4. Boot Sekansı 7

2.1.5. Yazılım ile İlgili Çalışmalar 7

2.1.6. Linker Script ve Yazılım Entegrasyonu 18

2.1.7. FPGA implementasyonu Aşamasında karşılaşılan zorluklar ve ASIC İmplementasyon 18

Aşamasına Geçiş Beklentileri 18

2.2. Tasarım Detayı 19

2.2.1. İşlemci ve Bus Yapısının Tasarımı 19

2.2.2. Peripheral Tasarım Detayları 20

2.2.3. FPGA Prototipleme Detayları 21

3. ÇİP TASARIM AKIŞI 22

4. TEST 23

5. TAKIM ORGANİZASYONU ve GÖREV DAĞILIMI 23

5.1. Takım Organizasyonu 23

5.2. Görev Dağılımı 24

6. İŞ PLANI VE RİSK PLANLAMASI 24

7. KAYNAKÇA 26

1. SİSTEM TANIMI VE TEMEL TASARIM ÖZETİ

1.1. Motivasyon

2010'lardan bu yana üretilen veri miktarındaki sürekli artış, yapay zeka sistemlerinin performansını iyileştirme ve hesaplamaları hızlandırma ihtiyaçlarını göstermiştir [6]. Veri hacminin büyüklüğü ve günümüz sistemlerinin temelindeki karmaşık algoritmalar göz önünde bulundurulduğunda, büyük veri kümelerinin daha hızlı işlenebilmesini sağlayacak verimli işleme yöntemlerine başvurulması gerekliliği ortaya çıkmaktadır. Bu durum, yapay zeka uygulamalarında FPGA (Field Programmable Gate Array) ve ASIC tasarımlarında hızlandırıcıların üretimindeki artışa sebep olmuştur [6]. FPGA'ler ve mikrodenetleyiciler, yüksek hızda çarpma ve toplamayı da içeren paralel hesaplamalar yapabilme yetenekleri nedeniyle bu amaç için özellikle uygundur. Ayrıca, yeniden yapılandırılabilir doğaları, belirli görevleri optimize etmek için donanımın özelleştirilmesine olanak tanır. Bu özellikler ve yapay zeka hızlandırıcılarına duyulan gereksinimdeki artış sebebiyle, bu projede OpenFPGA araçları, Vivado ve Cadence kullanılarak girdi olarak verilecek bir rakamın sınıflandırılmasını sağlamak üzere özelleştirilmiş RISC-V tabanlı bir yapay sinir ağı işlemcisi geliştirilmesine karar verilmiştir.

1.2. Proje Kapsamında Yürütülen Faaliyetler

Yapay sinir ağları, çarpma-toplama işlemleri ve önceden eğitilmiş model katsayıları kullanılarak işlevsellik kazandırılan, doğrusal olmayan genel amaçlı sınıflama, regresyon ve kestirim modelleridir. Bu algoritmaları FPGA üzerine açık kaynaklı RISC-V mimarisi kullanarak işlemek ve enerji tüketimi ile hesaplama aşamasındaki olası gecikmeleri optimize edebilmek için gerekli koşullar dikkate alınarak bir MNIST sınıflandırıcı modülü geliştirilmiştir. Bu bağlamda, ağırlık ve taraf katsayısı değerlerinin elde edilebilmesi amacıyla ilk olarak Python kullanılarak bir model geliştirilmiştir [3]. Modelin eğitimi tamamlandığında, katsayılar JSON formatında bir dosyada toplanmış, ardından Q0.15 sabit nokta gösteriminde 16 bitlik değerler haline getirilerek nicemlenmiştir (quantization)[5].

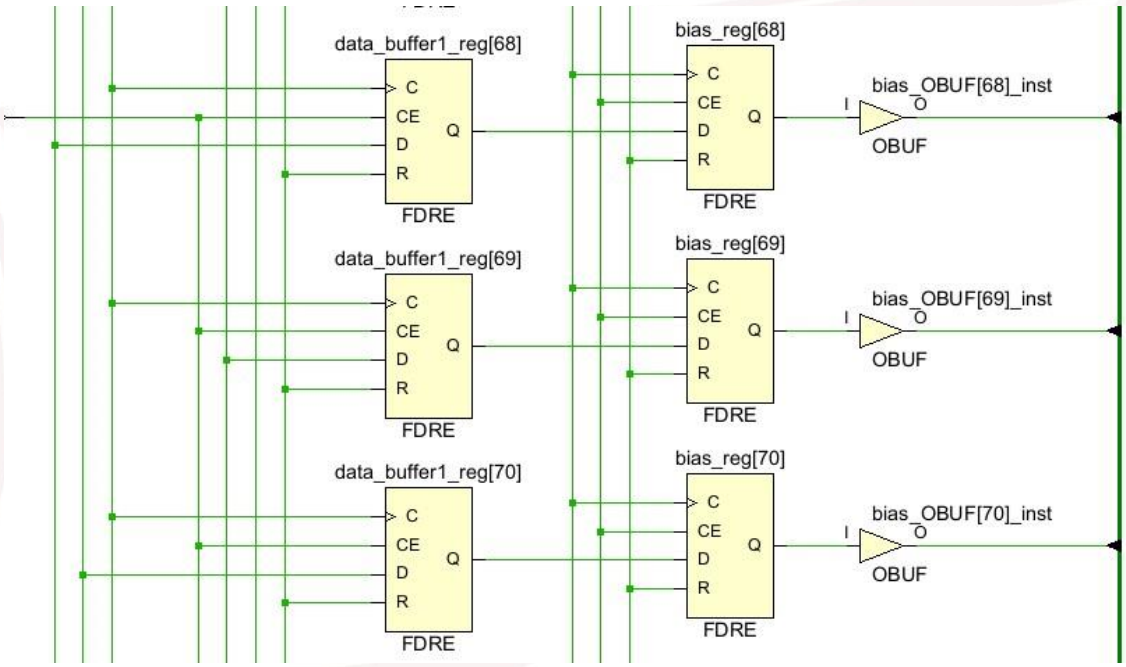
Modelin içerisindeki döngülerin ve verinin katmanlar arasındaki aktarımının daha net anlaşılabilmesi amacıyla, Python'a kıyasla daha düşük soyutlama seviyesine sahip olan C++ dili kullanılarak model yazılmıştır. C++ modelinin hafızayı kullanış biçimi ve ara hesaplamalarda ortaya çıkan matrislerin de tutulması, ne kadar hafıza gereksinimi duyulacağını göstermiştir. Ön tasarım raporunda, bir nöron modülünün ayrı bir katman modülü içerisinde birkaç defa tanımlanmasıyla oluşturulması planlanan katman yapısı terk edilmiştir [10]. Bunun sebebi, modelin içerisinde yalnızca konvolüsyon işleminin değil, havuzlama ve düzleştirme işlemlerinin de yapılmasına ihtiyaç duyulmasıdır. Model yapısı gereği her bir konvolüsyon katmanını havuzlama katmanı takip ettiğinden, iki konvolüsyon-havuzlama katmanı modülü yazılmış; bu modüllerde taraf katsayısı değerleri gömülmüş, ağırlıklar ise LUT (look-up-table) içerisinde daha sonra kullanılmak üzere depolanmıştır. Düzleştirme işlemi için ayrı bir katman yazılmasına gerek kalmaması adına elde edilen sonuçların sistem içerisinde seri olarak iletilmesi sağlanmıştır.

16 bitlik nicemlendirilmiş Q0.15 formatında ikili sayı gösterimi ile ifade edilen ağırlıklar ve taraf katsayıları kullanılarak oluşturulan modelin sentez aşaması çok uzun sürdüğünden, yazılan bir testbench aracılığı ile Vivado kullanılarak modelin doğrulanması sağlanmıştır. Öncelikle VHDL kullanılarak yazılan

model, ardından Verilog ile yazılmıştır. Modülün sentezlenebilir olması amacıyla 16 bitlik veriler 4 bite kadar düşürülmüştür. Bunun ilerleyen aşamalarda sistemin sınıflandırma performansı üzerinde sahip olabileceği etki düşünülerek daha yüksek performanslı sistemler kullanılarak daha fazla bit ile ifade edilen katsayılara sahip modellerin sentezinin yapılması gerektiği belirtilmelidir. Sentezin tamamlanmasının ardından, RISC-V mimarisi çalışılmış ve yazılmış MNIST sınıflandırıcı modülünün RISC-V çekirdeğine ait LUT'ları kullanabilmesi ve çekirdek ile iletişiminin sağlanabilmesi amacıyla UART iletişim protokolünün kullanılmasına karar verilmiştir[9]. UART alıcı ve verici modülleri yazılarak, sistemin programlanması sürecinde ağırlık değerlerinin RISC-V işlemcisi çekirdeğinde depolanması sağlanmıştır. Ek olarak, kullanıcıların fare ile rakamlar yazmasına olanak tanıyan bir kullanıcı arayüzü Python kullanılarak oluşturulmuştur, böylece model tarafından sınıflandırılmak üzere girdi görüntüleri kullanıcı tarafından üretilebilmiştir.

1.3. RTL, Simülasyon ve Çip Fiziksel Tasarım Aşaması ve Tamamlanma Oranları

Proje Kapsamında Yürütülen Faaliyetler kısmında da değinildiği üzere, ilk olarak 16-bit olarak çalışması planlanan modül, sentez aşamasının uzun süresi sebebiyle 4 bit olarak çalışabilecek şekilde düzenlenmiş ve Python modelinden çıkarılan ağırlık ve taraf katsayısı değerleri 4 bite göre tekrar nicemlendirilmiştir. Ardından, Vivado kullanılarak implementasyonu tamamlanan sistem sentezlenmiş ve Şekil 1'de gösterilmekte olan RTL şeması elde edilmiştir. RTL şeması daha detaylı bir şekilde proje Github reposundan görülebilir.



Şekil 1: Proje RTL Şemasının Yakınlaştırılmış Kesiti

Simülasyon için VHDL ve Verilog kullanılarak yazılmış iki ayrı testbench kullanılmış ve Vivado'nun kendi içerisinde sunduğu simülasyonlardan yararlanılmıştır. Yazılmış MNIST modülünün işlevselliğini ve başarılı olup olmadığını anlamak amacıyla yazılmış olan birinci testbench dosyasında, sisteme kullanıcı arayüzünden beslenecek olan girdiye ait gri tonlamalı fotoğrafın piksel değerlerine ait bilginin bulunduğu yazı dosyasının içerisindeki değerler Q0.15 formatına göre düzenlenmiş; ardından yine text dosyaları içerisinde saklanan katmanlara göre

Şekil 2: Proje RTL Şemasının Geniş Açıdan Sunulmuş Kesiti

ayrılmış ağırlık değerleri sisteme beslenmiş ve sonuç dizisindeki olasılık dağılımı incelenmiştir. En yüksek olasılığın atandığı indeks, tahmin edilen sayıyı göstermektedir. Yapılan tahminin doğruluğundan emin olunduktan sonra ara işlemlerde elde edilen matrislerin içerisindeki değerler hem VHDL/Verilog koduna ilham veren C++ kodunda hem de VHDL/Verilog kodlarında Python'da geliştirilmiş model ile kıyaslanmış ve verinin propagasyonunda ya da sistemi oluşturan katmanların herhangi birinde matematiksel veya mantıksal bir hata olup olmadığı test edilmiştir.

UART iletişim protokolü baz alınarak yazılmış alıcı ve verici kodlarının entegrasyonunun ardından, en yüksek mertebedeki modülün işleyişindeki ufak değişiklikler sebebiyle yazılmış olan testbench baz alınarak yeni bir testbench kodu yazılmıştır. Yine Vivado kullanılarak sürdürülen simülasyonlarda sistemin beklendiği gibi davrandığı gözlemlenmiştir.

Cadence sisteminin kurulumunda yaşanan teknik sorunlar nedeniyle Fiziksel Tasarım Aşamasında henüz kayda değer bir yol kat edilememiştir. Ancak, 25 Ağustos 2024 tarihine kadar tanınmış olan süre içerisinde çip tasarımı üzerine yoğunlaşılması planlanmaktadır.

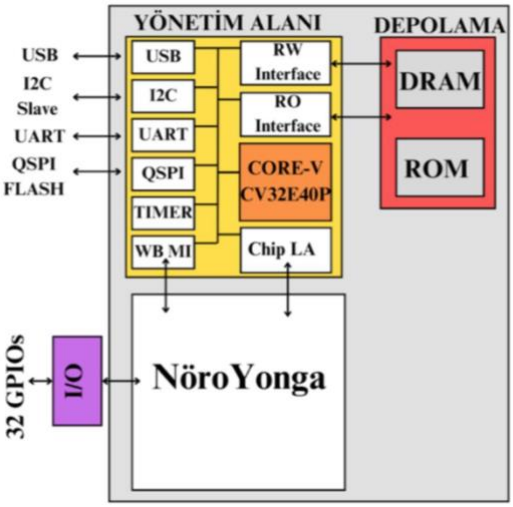
2. PROJE DETAY TASARIMI

2.1. Sistem Mimarisi

Bu bölümde, projemizin nihai tasarımının blok şeması sunulacak, tasarlanmış sistemi oluşturan alt bloklar ve blokların görevleri, Cadence kullanımına geçildiğinde kullanılması planlanan ve ilerleyen kısımlarda daha detaylı ele alınacak olan bus yapısı ve RISC çekirdeğinin çevre birimlerinin (peripherallerinin) bağlantı detayları, boot sekansı, yazılım ile ilgili Python, C++, VHDL ve Verilog kullanılarak yapılan çalışmalar, kullanıcı arayüzü ve MNIST sınıflandırıcısı için yazılmış linker script, projenin FPGA ve ASIC implementasyon aşamalarında gözlemlenmesi planlanan farklılıklar gibi konular detaylandırılacaktır.

2.1.1. Blok Şeması ve Sistem Uyumluluğu

Projemiz, yapay sinir ağlarının testlerinde sıklıkla kullanılan MNIST veri setini sınıflandırabilme yeteneğine sahip bir sistemdir. Tasarım, şu anda MNIST sınıflandırıcı modülü ve kullanıcı arayüzünü bağlayan bir UART sistemini içermektedir. Bunun yanında CV32E40P işlemcisinin çeşitli modüllerini, AXI-4 bus yapısını ve çeşitli periferallerinin de proje teslim gününe kadar tamamlanmış olması beklenmektedir. RISC-V çekirdeğinin sahip oldupu hali hazırda kayan noktalı sayılar ile işlem yapabilme kabiliyetine sahip olan Floating Point Unit'i (FPU) kullanmamakta bunun yerine, ağırlık ve taraf katsayısı değerlerinin bellekte saklandığı ve gerektiğinde alındığı, yoğun hesaplamaların ise MNIST modülünde gerçekleştirildiği bir yapıdadır [7]. Bu sayede, işlemci ve modül arasında yüksek miktarda veri transferine gerek kalmadan, verimli bir şekilde veri işleme sağlanmaktadır.



Şekil 3: Sistem mimarisi – yönetim, tasarım ve depolama alanı

2.1.2. Alt Bloklar ve Görevleri

- **CV32E40P İşlemcisi:** RISC-V tabanlı bu işlemcinin, sistemin ana kontrol ünitesi olarak görev yapması planlanmaktadır. Cadence serverlarına ulaşım sorunu yaşandığından henüz uygulamaya geçilmese de bu blok, bellekten ağırlık ve taraf katsayısı değerlerini almak ve boot sekansının bir parçası olarak MNIST modülüne göndermekle görevlendirilecektir.[7]
- **MNIST Modülü:** Sinir ağı işlemlerinin gerçekleştirildiği ana modüldür. Bu modül, gelen veriyi işler ve sınıflandırma sonucunu üretir. Yapay sinir ağı modelinin katmanlarını ve bunlar arasındaki veri akışını denetleyen bir üst modülü içermektedir.
- **AXI-4 Bus Yapısı**: İşlemci ile MNIST modülü ve diğer periferaller arasındaki veri transferlerini yönetmek için kullanılması planlnaan yüksek hızlı bus sistemidir.
- **UART Modülleri:** Veri transferi ve iletişim için kullanılan alıcı ve verici modüllerdir. Bu modüller, kullanıcıdan gelen veri girişlerini alır ve işlemciye iletir. Şu anki modelde hesaplamaları işlemesi adına direk MNIST çekirdeği ile bağlantı içerisinde olan bu modülün RISC çekirdeğine bağlanması planlanmaktadır. Böylelikle kullanıcı tarafından sisteme veri girişi olması durumunda katsayıların yüklenmesi denetlenebilecek ve tüm sistem kontrolünün denetiminin çekirdeğe atanması sağlanacaktır. [9]

2.1.3. Bus Yapısı ve Periferallerin Bağlantı Detayları

UART, I2C Master, QSPI Master, Timer, GPIO ve USB Tam Hız Cihazı çevre birimlerinin RISC-V çekirdeği ile AXI4 veri yolu kullanarak bağlanması, düzgün iletişim ve işlevselliği sağlamak için bir dizi teknik adım gerektirmektedir. Her çevre birimi, AXI4-Lite kölesi olarak entegre edilmiştir ve bu da RISCV çekirdeği tarafından verimli iletişim ve kontrol sağlanmaktadır. Aşağıda, her çevre biriminin entegrasyonu için detaylı teknik adımlar verilmiştir.

UART çevre birimi için, sistemin bellek haritasında benzersiz bir adres aralığı atanmıştır. UART çevre birimi AXI4-Lite kölesi olarak atanmıştır ve AXI arayüz sinyalleri AXI4 interconnect'e bağlanmıştır. Ayrıca, UART kesme sinyalleri (örneğin, TXRDY, RXRDY) RISC-V çekirdeğinin kesme denetleyicisine bağlanarak veri iletim ve alım olaylarına asenkron olarak yanıt vermesi sağlanmıştır [7][9].

I2C Master çevre birimi için sistem bellek haritasında belirli bir adres aralığı atanmıştır. I2C Master, AXI4-Lite kölesi olarak atanmıştır ve AXI4-Lite sinyalleri AXI4 interconnect'e bağlanmıştır. Ayrıca, I2C Master'ın SCL ve SDA hatları harici I2C cihazlarına bağlanarak bidirectional iletişim sağlanmıştır [7].

QSPI Master çevre birimi için benzersiz bir adres aralığı tanımlanmıştır. QSPI Master, AXI4-Lite kölesi olarak AXI4 interconnect'e bağlanmıştır ve tüm AXI4-Lite sinyaller düzgün şekilde yönlendirilmiştir. QSPI Master, harici QSPI flash bellek veya SPI cihazlarına bağlanarak yüksek hızlı veri transferi sağlanmıştır [7].

Timer çevre birimi için bellek haritasında benzersiz bir adres alanı ayrılmıştır. Timer, AXI4-Lite kölesi olarak bağlanmıştır ve gerekli tüm AXI4-Lite sinyalleri AXI4 interconnect'e bağlanmıştır [7]. Timer'ın kesme sinyalleri, RISC-V çekirdeğinin kesme denetleyicisine bağlanarak zamanlayıcı olaylarının işlenmesi sağlanmıştır.

GPIO çevre birimi için belirli bir adres aralığı atanmıştır. GPIO modülü, AXI4-Lite kölesi olarak iatanmıştır ve AXI4-Lite arayüz sinyalleri AXI4 interconnect'e bağlanmıştır [7]. GPIO kesme hatları, pin

durumu değişiklikleri için RISC-V çekirdeğinin kesme denetleyicisine bağlanarak kesmeler yapılandırılmıştır.

USB Tam Hız Cihazı için benzersiz bir adres aralığı atanmıştır. USB cihazı, AXI4-Lite kölesi olarak bağlanmıştır ve tüm AXI4-Lite sinyalleri AXI4 interconnect'e düzgün şekilde bağlanmıştır[7]. USB veri hatları (D+ ve D-), uygun harici USB konektörlerine bağlanarak veri iletişimi sağlanmıştır.

Her çevre birimi, sistemin üst düzey modülünde atanmış ve AXI4-Lite köle arayüz bağlantıları doğru şekilde yapılmıştır. AXI4 interconnect, birden fazla köle bağlantısını işlemek ve RISC-V çekirdeği (master) ile çevre birimleri (slave) arasında işlemleri yönlendirmek için yapılandırılmıştır. Ayrıca, çevre birimlerinden gelen kesmeleri işlemek için RISC-V çekirdeğindeki kesme denetleyicisi yapılandırılmış, böylece olay odaklı işleme sağlanmıştır. Bu adımlar izlenerek, her çevre biriminin AXI4 veri yolu aracılığıyla RISC-V çekirdeği ile doğru şekilde entegre edilmesi sağlanmış ve sistem içinde verimli iletişim ve kontrol sağlanmıştır.

2.1.4. Boot Sekansı

Boot sekansı işlemcinin başlatılması ve bellekten gerekli verilerin yüklenmesi ile başlar ve bunu ağırlık ve taraf katsayısı değerlerinin belleğe işlemci tarafından yüklenmesi takip eder. Katsayılar ve girdiye ait verilerin hazır olması durumunda MNIST modülünün içerisindeki seri işlem yapan katmanların çalışmasını başlatacak sinyal gönderilir ve girdi işlenmeye başlar. Girdi ve katsayıların hazır olmasını bekleyen ve buna göre işlemi başlatan bu süreç, sistemin hızlı işlemesini sağlamasının yanında sorunsuz bir işleme sürecine olanak tanır.

Boot sekansı, işlemcinin UART üzerinden başlatılmasıyla başlatılır ve bellekten gerekli verilerin yüklenmesiyle devam edilir. Bu süreçte, UART üzerinden bir "hazır" sinyalinin gönderilmesi sağlanır.

İlk olarak, işlemci tarafından UART'tan gelen veriler sürekli dinlenir. UART modülü tarafından bu veriler bir tampon belleğe (buffer) kaydedilir. Bu verilerin, işlemcinin başlatma kodlarını içermesi beklenir. İşlemci, belirli bir başlatma komutu aldığında, bu komutların bellekteki belirli adreslere yazılması ve ilgili işlemlerin başlatılması gerçekleştirilir.

Başlatma işlemi sırasında, UART'tan gelen her veri paketine işlemci tarafından bir "hazır" sinyali gönderilerek onay verilmesi sağlanır. Bu sinyal, genellikle bir bit veya belirli bir karakter olarak UART veriyoluna geri gönderilir. Böylece, veri gönderen taraf veri paketinin başarılı bir şekilde alındığını ve işleme alındığını anlar.

Başlatma işlemi tamamlandıktan sonra, işlemci tarafından bellekten yüklenen başlatma kodları çalıştırılır ve normal çalışma moduna geçilir. Eğer boot işlemi sırasında herhangi bir hata tespit edilirse, UART üzerinden bir hata sinyali gönderilerek işlemci tarafından bir hata durumu başlatılır. Bu şekilde, UART üzerinden boot işleminin gerçekleştirilmesi ve işlemcinin başlatma sürecinin başarılı bir şekilde tamamlanması sağlanır.

2.1.5. Yazılım ile İlgili Çalışmalar

Projemizde, yazılım geliştirme sürecinde Python ve C++ dilleri kullanılmıştır. Python kullanılarak model eğitimi ve kullanıcı arayüzü geliştirilmiş, C++ kullanılarak ise modelin daha düşük soyutlama seviyesinde uygulanması sağlanmıştır. Ayrıca, donanım ile yazılım arasındaki entegrasyonu sağlamak amacıyla çeşitli testbench kodları yazılmış ve sistemin doğruluğu test edilmiştir.

2.1.5.1. Python ile Model Eğitimi

MNIST sınıflandırıcı modelinin geliştirilmesi için Python'da NumPy, Matplotlib ve Keras gibi çeşitli kütüphanelerden yararlanılmıştır [1]. Eğitilen model, ağırlık ve sapma değerlerinin hesaplanarak Verilog modelinde kullanılabilmesi için geliştirilmiştir. Ağırlıklar ve sapmalar kayan nokta (floating point) formatında olduğundan, FPGA hesaplama yeteneklerine daha uygun olan sabit nokta (fixed point) gösterimine dönüştürülmeleri için raporun devamında açıklanacak olan çeşitli işlemlerden geçirilmiştir. Model mimarisi, birden fazla katmandan oluşmaktadır:[2][3] Her biri havuzlama katmanı ile takip edilen iki konvolüsyon katmanı ve bu katmanların devamında bir düzleştirme katmanı ve bir yoğun katman içermektedir.

MNIST veri setinin projeye entegre edilmek üzere işlenmesi, piksel değerlerinin normalleştirilmesini ve evrişimli katman için gereken ekstra bir boyutun Keras aracılığıyla eklenerek görüntülerin 28x28 piksel boyutlarına yeniden şekillendirilmesini içerir. MNIST veri kümesindeki görüntüler gri skalaya dönüştürülmüş, piksel değerlerini tanımlamak için tek bir kanal kullanılmış ve her bir pikselin değeri 0 ile 1 arasında bir aralığa ölçeklendirilerek normalleştirilmiştir[3]. Eğitim veri kümesi işlendikten sonra model oluşturma aşaması tamamlanmıştır.

Mimari, 32 filtre ve 3x3 çekirdek boyutu içeren bir evrişim katmanı ile başlar, ardından bir ReLU aktivasyon fonksiyonu ve 2x2 maksimum havuzlama katmanı gelir. Bunu, 64 filtre içeren başka bir evrişim katmanı ve başka bir maksimum havuzlama katmanı takip eder. Çıktı daha sonra düzleştirilir ve aşırı uyuma engel olmak için bir dropout katmanından geçirilir. Aşırı uyum, modelin eğitim verilerinden öğrenmesi gerekirken verileri ezberlemesinden kaynaklı oluşan yaygın bir sorundur [6]. Dropout katmanı yalnızca eğitim sırasında kullanılır ve ana amacı eğitim performansını artırmak olduğu için Verilog model uygulamasına dahil edilmez. Son olarak, sınıflandırma için 10 nöron ve bir sigmoid aktivasyon fonksiyonuna sahip yoğun bir katman kullanılır.

2.1.5.2. Model Katsayılarının Kayan Noktalı Sayılardan Q0.15 Sayılara Dönüştürülmesi

Python kullanılarak yazılan modelin ağırlıkları ve sapmaları, modeli tanımlayan katsayılar olarak .json dosyasına aktarılır. Ancak, .json dosya formatı karmaşık bir yapıya sahip olduğundan, farklı evrişim ve yoğun katmanlara ait ağırlıklar ve sapmalar ayrı txt dosyalarına ayrılır. Bu dosyalarda yazılı olan kayan noktalı sayılar daha sonra Q0.15 sabit noktalı, ters çevrilmiş, ikili (binary) sayılara dönüştürülür. Sabit noktalı gösterimlerin kullanımı kritiktir çünkü FPGA'lar sabit noktalı sayılar üzerinde hesaplamalar yapmaya kayan noktalı sayılara göre çok daha uygundur [10]. Sabit noktalı tekniklerin veya kuantizasyonun tercih edilmesinin birkaç nedeni vardır. Sabit noktalı işlemler, daha az donanım kaynağı ve güç tüketir, bu da özellikle düşük enerji tüketimi ve kaynak verimliliği gerektiren cihazlar için avantaj sağlar. Ayrıca, sabit noktalı işlemler, kayan noktalı işlemlere kıyasla daha hızlıdır ve daha düşük gecikme sunar, bu da gerçek zamanlı sistemlerde kritiktir. Kuantizasyon, bellek ve hesaplama gereksinimlerini azaltarak sinir ağları gibi modellerin performansını artırır. Bu nedenlerle, sabit noktalı teknikler ve kuantizasyon, daha verimli ve maliyet etkin çözümler sunar. Ek olarak sabit noktalı aritmetik işlemleri (toplama, çıkarma, çarpma) donanımda uygulamanın, kayan noktalı işlemlere kıyasla karmaşıklığı ve neticesindeki saat döngü sayısının fazlalığından kaynaklı gecikmeleri azaltacağı öngörülmüştür. Daha az mantık kapısı (logic gate) ve daha düşük güç tüketimi ile FPGA kaynaklarının daha verimli kullanılmasına olanak tanıdığından bu uygulama modelde yapılan işlemleri optimize etmektedir.

JSON dosyasından okunduktan sonra, kayan noktalı katsayıların Q0.15 sabit noktalı, bitlerin tersine sıralandığı bir ikili (binary) sayıya dönüştürülmesinde izlenen adımlar aşağıdaki tabloda görülebilmektedir.

Tablo 1: Kayan Noktalı Sayıların Dönüştürülmesinde İzlenen Adımlar

Adım	Açıklama
1	Her kayan noktalı değer, Q0.15 aralığına ölçeklenmek üzere 32768 ile çarpılır.
2	Ölçeklenmiş değer, 16-bit işaretli tamsayılar olarak ifade edilebilecek sayı aralığına sığdırılacak şekilde sınırlanır. (-32768 ile 32768 arası)
3	Negatif değerler için sayının iki'nin tamamlayıcısı (two's complement) hesaplanır.
4	Sınırlanmış değer "format(scaled_value & 0xFFFF, '016b')" komutu kullanılarak ikiliye dönüştürülür. Bu satırda, & 0xFFFF işlemi değerin sadece alt 16 bitini koruyarak 16 bit içinde sığdırılmasını sağlar ve '016b', sonucu gerekirse sıfır ile doldurulmuş 16 bitlik bir ikili string olarak biçimlendirir.
5	İkili değerler, Verilog test bench verilerinin okunma ve işlenme şekli nedeniyle bu ters formatı beklediği için ters çevrilir.

2.1.5.3. Abstraksyonu Azaltmak Amaçlı Yazılmış C++ Programları

Python'dan C++'a sinir ağı katmanlarının uygulanmasına geçiş sırasında ara bir soyutlama seviyesi korunmuştur. Bu soyutlama, Verilog uygulaması tamamlanmadan önce evrişim ve havuzlama işlemlerinin anlaşılmasını kolaylaştırmaktadır. Bu geçişle, Python'un kütüphanelerinde mevcut yüksek seviyeli işlemler ile Verilog için gerekli düşük seviyeli kodlama arasındaki boşluk doldurulmuştur. Bu sayede her katmandaki veri akışı ve hesaplama süreçleri hakkında bilgi edinimi sağlanmıştır. C++ kodu, sinir ağının her katmanında yer alan temel hesaplamaların daha net bir görünümünü sağlar çünkü evrişimler, havuzlamalar ve aktivasyonlara karşılık gelen döngüler ve işlemler açıkça kodlanmıştır. Verilog'da hata tespiti ve ayıklanması, derleyicinin detaylı hata bilgisi sağlamaması nedeniyle son derece zorlayıcı olduğundan programlamanın öncelikle C++ ile yapılması, boyut uyumsuzluklarını ve hesaplama ile mantıksal hataları belirleme imkanı sunmuştur. Yazılan fonksiyonlar, giriş verilerini işlemek ve nihai sınıflandırma çıktısını üretmek için sıralı olarak çağrılır. Bu modüler yaklaşım, her katmanın ayrı bir modül tarafından temsil edildiği Verilog uygulamasını yansıtır. Ayrıca, tüm sinir ağları arasındaki bağlantıyı yöneten "main" fonksiyonu, Verilog'daki "top" modülüne benzer şekilde, ağ boyunca doğru veri akışını sağlar.

2.1.5.4. Verilog İmplementasyonu

Modelin katmanlarını temsil eden Verilog kodları takip eden bölümlerde detaylı olarak ele alınacaktır. Her katmanın işlevselliği test bench kodları aracılığıyla test edilmiştir.

2.1.5.4.1. Verilog ile Yazılmış Üst Modül (top.v)

Yoğun katman tarafından takip edilen top modülü, bir giriş görüntüsü üzerinde bir dizi evrişim işlemi gerçekleştirmek için tasarlanmış dijital bir sistem içindir. Bu modül, sistemin işlev görmesi için gereken

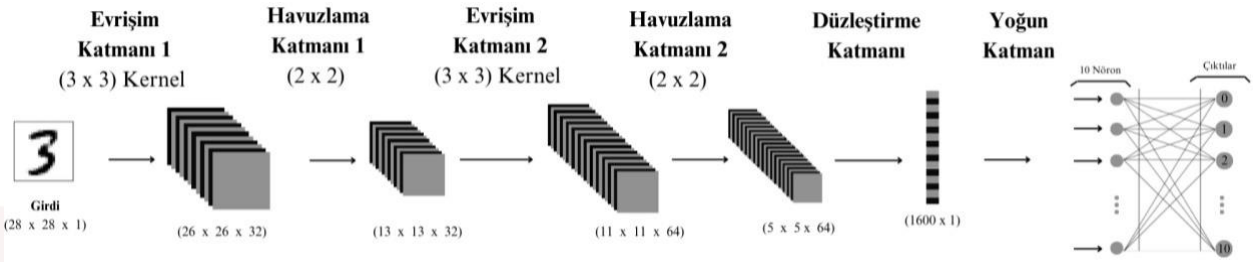
temel girişleri ve çıkışları tanımladığı için 'top' olarak isimlendirilmiştir. Bu portlar arasında saat, başlat, katsayı, sapma, görüntü ve tamamlandı bulunmaktadır.[10]

'clock' portu, sistem boyunca operasyonları senkronize etmek için kullanılan standart bir saat sinyali girişidir. 'Start' portu, evrişim sürecini başlatmak için kullanılan bir giriş sinyalidir. 'coef' portu, evrişim ve yoğun katmanlar için gereken 16-bitlik katsayıları depolamak üzere ayrılmış, 555,520 bit genişliğinde önemli bir giriştir ve gerektiğinde birden fazla katmana bölünmüştür. Benzer şekilde, 'bias' portu, çeşitli katmanlar için 16-bitlik sapmaları tutacak şekilde tasarlanmış 1,696 bit genişliğindedir. 'img' portu, her pikselin 16 bit genişliğinde olduğu bilgisiyle 28x28 görüntüyü temsil eden 12,544 bit genişliğinde bir giriştir. 'Tamamlandı' portu, evrişim ve yoğun işlemlerin tamamlandığını belirten bir çıkış sinyalidir."

"Behavioral" adı verilen mimaride Convolution, Convolution1 ve dense olmak üzere üç ana bileşen örneklenmiştir. Bu bileşenlerin her biri, sinir ağının belirli bir katmanına karşılık gelir. İlk bileşen olan Convolution, ilk evrişim işleminden sorumludur. 'clock' ve 'start' sinyallerini, 'coef' ve 'bias' vektörlerinin belirli dilimleri ile birlikte alır: sırasıyla coef(4607 downto 0) ve bias(511 downto 0). 'coef(4607 downto 0)' ifadesi, 32 adet 3x3 çekirdek için 16-bitlik ağırlıkları temsil eden 4608 bit tutar. Diğer taraftan 'bias(511 downto 0)', çekirdek başına 16-bitlik sapmaları (16x32) tutar. Bu dilimler, bu katman için gerekli katsayıları ve sapmaları sağlar. Giriş görüntüsü 'img' portu üzerinden geçirilir ve işlenmiş çıktı, ilk evrişimden sonra elde edilen görüntüyü temsil eden 173,056 bit genişliğindeki 'layer1_output' sinyalinde depolanır. Bu görüntü, evrişim süreciyle her pikselin 32 bit olduğundan 13x13x32x32 boyutlarındadır ve 32 farklı 13 x 13 özellik çıkarılmış (feature extracted) görüntü içerir.

İkinci bileşen olan Convolution1, ilk evrişim katmanının çıktısını işler. Convolution bileşeninden 'tamamlandı' sinyali alındığında işlemine başlar.. Bu bileşen, katsayılarını coef(299519 downto 4608) ve sapmalarını bias(1535 downto 512) 'dan alır, bu katman için farklı bir parametre setini etkin bir şekilde kullanır. Bu katman için giriş görüntüsü 'layer1_output' olup işlenmiş çıktı 'layer2_output' da depolanır, bu da 76,800 (5 x 5 x 48 x 64) bit genişliğindedir ve 48-bit piksel verisiyle 64 farklı 5 x 5 görüntüden gelir. 48-bit piksel verisi, evrişim süreci ile elde edilir. (32-bit x 16-bit --> 48-bit). Bu çıktı, ikinci evrişim katmanından sonra elde edilen görüntüyü temsil eder. Son bileşen olan dense, tam bağlantılı bir katmandır ve ikinci evrişim katmanının çıktısını (layer2_output) kendi katsayı ve sapma seti ile işler. Bu katman için katsayılar coef(555519 downto 299520) 'dan ve sapmalar bias(1695 downto 1536) 'dan alınır. Dense bileşeni, nihai işlenmiş çıktıyı üretir ve 'tamamlandı' portu aracılığıyla işlemin tamamlandığını iletir.

Mimari boyunca, sinyali parçalara bölme işlemi kritik bir rol oynamaktadır. Örneğin, coef(299519 downto 4608) ikinci evrişim katmanında kullanılmak üzere coef vektöründen belirli bir bit aralığını çıkarır. Bu parçalara bölme işlemi, her bileşenin işlemleri için doğru veri alt kümesini almasını sağlar. Benzer şekilde, sapmalar her katmanın gereksinimlerine uyacak şekilde dilimlenir, bu da doğru başlangıç ve hesaplama sağlar. İşlem, sinir ağı içindeki ilgili katmanlara önceden okunan ağırlıkların ve sapmaların dağıtımını içerir. Test bench aşamasında, başlangıç giriş görüntüsü ile birlikte tüm ağırlık ve sapma seti bir kez okunur. Bu yaklaşım, verilerin tamamen yüklendiğinden ve evrişim işlemleri başlamadan önce sistem belleğinde kullanılabilir olduğundan emin olmak için izlenmiştir.



Şekil 4: Evrişimsel Model Şeması

2.1.5.4.2. Verilog ile Yazılan İlk Evrişim Katmanı ve İlk Havuzlama Katmanı (conv2d.v)

Bu Verilog kodu, bir evrişim işlemini gerçekleştirir. Bu süreci ReLU aktivasyon fonksiyonunun uygulanması, havuzlama ve çıktı görüntüsünün serileştirilmesi izler. Süreç, giriş görüntüsünün, ağırlıkların ve sapmaların ilgili matrislerine seri durumdan çıkarılmasıyla başlar. Evrişim işlemi, bu ağırlıklar ve sapmalar kullanılarak gerçekleştirilir ve sonuçta elde edilen özellik haritaları bir ReLU aktivasyon fonksiyonundan geçirilir. Bu özellik haritaları üzerinde daha sonra boyutlarını azaltmak için maksimum havuzlama işlemi uygulanır [11]. Son olarak, işlenmiş veriler daha fazla işlem veya depolama için uygun olan bir vektör formatına geri serileştirilir.

Tablo 2: İlk Havuzlama Katmanındaki Portlar ve Sinyaller

Portlar	Sinyaller
clock: Sistemin tüm operasyonlarını senkronize etmek için kullanılan standart giriş saat sinyalidir.	**image**: Seri durumdan çıkarılmış giriş görüntüsünü depolamak için kullanılan 2D dizi sinyalidir.
start: Evrişim sürecinin başlangıcını tetikleyen bir başlangıç sinyali girişidir.	**output**: Ara evrişim sonuçlarını depolamak için kullanılan 2D dizi sinyalidir.
coef: Evrişim çekirdekleri için katsayıları (ağırlıkları) içeren 4608 bit genişliğinde bir giriş vektörüdür. Her çekirdek ağırlığının 16-bit işaretli tamsayı olduğu varsayılır.	**sum**: Evrişim işlemi sonucunda toplamı biriktirmek için kullanılan bir sinyaldir.
bias: Her evrişim çekirdeği ile ilişkilendirilen sapmalar için 512 bit genişliğinde bir giriş vektörüdür, her sapmanın 16-bit işaretli tamsayı olduğu varsayılır.	**krnl**: Seri durumdan çıkarılmış evrişim çekirdeklerini depolamak için kullanılan 4D dizi sinyaldir.
img: Her pikseli 16-bit işaretli tamsayı olan 28x28 görüntüyü temsil eden 12,544 bit genişliğinde bir giriş vektörüdür.	**feature**: Havuzlama sonrası özellik haritalarını depolamak için kullanılan 3D dizi sinyalidir.
new_img: Evrişim, ReLU ve havuzlama sonrası işlenmiş görüntüyü temsil eden 173,056 bit genişliğinde bir çıkış vektörüdür, her pikseli 32-bit işaretli tamsayı olan 13x13x32 görüntü olarak biçimlendirilmiştir.	**biases**: Seri durumdan çıkarılmış sapmaları depolamak için kullanılan dizi sinyali.
done: Tüm işleme dizisinin tamamlandığını belirten bir çıkış sinyalidir.	**state**: Operasyonların akışını kontrol eden durum makinesinin mevcut durumunu temsil eden bir sinyaldir.

Tablo 3: İlk Havuzlama Katmanında Kullanılan Fonksiyonlar

Fonksiyon	Kullanım Yeri
ReLU	ReLU (Düzeltilmiş Doğrusal Birim) fonksiyonu, evrişim çıktısına bir aktivasyon fonksiyonu uygulamak üzere tanımlanmıştır. Girdi sıfırdan büyükse girdiyi geri döndürür; aksi takdirde sıfır döndürür. Bu fonksiyon, çıktı piksellerini aktive etmek için evrişim durumunda kullanılır.
Max	Max fonksiyonu, iki adet 32-bit işaretli tam sayıyı karşılaştırır ve daha büyük olan değeri döndürür. Bu fonksiyon, özellik haritalarının boyutunu, bir 2x2 bloktan maksimum değeri seçerek azaltan maksimum havuzlama işlemini gerçekleştirmek için havuzlama durumunda kullanılır.

Tablo 4: İlk Havuzlama Katmanında Durumlar ve Durum Geçişleri

Durumlar	Durum Geçişleri
serial: Çıktı özellik haritalarını new_img vektörüne serileştirmek için kullanılan durumdur.	• İlk durum olan deserial'da, giriş görüntüsü image dizisine (12,544 bit) seri durumdan çıkarılır. • Görüntü seri durumdan çıkarma işlemi tamamlandığında, durum deserial_krnl'a geçer ve evrişim çekirdeği ağırlıkları krnl dizisine (4,608 bit) seri durumdan çıkarılır. • Çekirdek ağırlıkları seri durumdan çıkarma işlemi tamamlandıktan sonra, durum deserial_bias'a geçer ve sapmalar biases dizisine (512 bit) seri durumdan çıkarılır. • Daha sonra conv durumuna geçilir, burada evrişim işlemi gerçekleştirilir. Evrişim sonuçları output dizisinde (173,056 bit) saklanır ve ReLU fonksiyonundan geçirilir. • Evrişim tamamlandıktan sonra, durum pooling'e geçer, burada çıktı dizisine maksimum havuzlama uygulanır ve sonuçlar feature dizisinde (54,144 bit) saklanır. • Havuzlama sonrasında, durum serial'e geçer, burada feature dizisi new_img vektörüne (173,056 bit) geri serileştirilir. Son olarak, finish durumuna ulaşılır ve işlem dizisinin • tamamlandığını belirtmek için 'tamamlandı' sinyali '1' olarak ayarlanır.
deserial: Giriş görüntüsünü img vektöründen seri durumdan çıkarmak için kullanılan durumdur.	
deserial_krnl: Evrişim çekirdeği ağırlıklarını coef vektöründen seri durumdan çıkarmak için kullanılan durumdur.	
deserial_bias: Sapmaları bias vektöründen seri durumdan çıkarmak için kullanılan durumdur.	
conv: Evrişim işlemini gerçekleştirmek için kullanılan durumdur.	
idle: Sistem beklerken, sayaçları sıfırlarken ve bir sonraki işlem için hazırlık yaparken kullanılan durumdur.	
pooling: Özellik haritaları üzerinde maksimum havuzlama işlemini gerçekleştirmek için kullanılan durumdur.	
finish: İşlem sürecinin tamamlandığını belirten son durumdur.	

Sayaçlar

Sayaçlar, süreç boyunca döngüler yerine kullanılır ve saat döngüsüyle senkronizasyon sağlamak için tercih edilir. Bu sayaçlar, görüntü, çekirdek ve çıktı dizileri içindeki pozisyonları izler, gerekli olduğunda artırma ve sıfırlama işlemleri yaparak her bir elemanı saat sinyali ile senkronize bir şekilde işlemek üzere düzenler. Örneğin, i_counter ve j_counter, seri durumdan çıkarma sırasında giriş görüntüsü içindeki pozisyonu takip ederken, kx_counter ve ky_counter, evrişim sırasında çekirdek matrisi içindeki pozisyonu izler.

İlk evrişim için yazılan Verilog kodu, karmaşık bir dijital sinyal işleme sistemi uygular ve durumlar ve sayaçlar kullanılarak veri akışını etkin bir şekilde yönetir. Ağırlıkların ve sapmaların önceden okunması ve dilimlenmesi sayesinde, her katmanın işleme için gerekli verileri almasını sağlar, performansı ve doğruluğu artırır. Durum makinesi yaklaşımı, seri durumdan çıkarmadan evrişime, ReLU aktivasyonundan havuzlamaya ve seri duruma kadar istenen çıktıyı üreten açık ve organize bir işlem dizisi sunar.

2.1.5.4.3. Verilog ile Yazılan İkinci Evrişim Katmanı ve İkinci Havuzlama Katmanı (conv2d_1.v)

Bu Verilog kodu, ilk havuzlama katmanının çıktısı olan [13,13,32] boyutunda üç boyutlu bir dizisi ile ikinci katmanın ağırlıkları ve sapmaları arasında bir evrişim işlemi gerçekleştirir. Bu katmanda, ağırlıklarımız [3,3,32,64] boyutunda dört boyutlu bir matristir, burada işlem 2D evrişim olarak kalır. Boyutlar, her biri 3x3 boyutunda ve 32 giriş derinliğiyle eşleşen, özellik haritalarına karşılık gelen 64 filtreyi gösterir ve bu işlem için 64 sapma bulunur. Dolayısıyla, bu 64 farklı filtrenin her biri için önceki evrişim katmanındaki işlemi tekrarlarız. Kısaca, bir 13x13 görüntü üzerinde 32 özellik haritası için bir 3x3 çekirdek kullanılarak bir evrişim işlemi uygulanır, ancak 64 farklı filtre kullanıldığı için ağırlıklar dört boyutlu hale gelir. Ağırlıklar ve sapmalarla evrişim işlemlerini gerçekleştirmek için döngüler yerine sayaçlar kullandık. Başlangıçta, girişlerimiz, ağırlıklarımız ve sapmalarımız seri olarak beslenir, evrişim işlemi için bir matris veya dizinin seri durumdan çıkarılmasına dönüştürülmesi gereklidir. Evrişim, bu ağırlıklar ve sapmalar kullanılarak gerçekleştirilir ve elde edilen özellik haritaları bir ReLU aktivasyon fonksiyonu ile işlenir. Bu işlem 64 filtre için tekrarlanır ve sonuç olarak 11x11x64 çıktısı elde edilir. Bu çıktı daha sonra boyutlarını azaltmak için bir maksimum havuzlama işlemi kullanılarak havuzlanır [3]. Son olarak, işlenmiş veriler daha fazla işlem veya depolama için uygun olan bir vektör formatına geri serileştirilir.

Tablo 5: İkinci Evrişim ve Havuzlama Katmanındaki Portlar ve Sinyaller

Portlar	Sinyaller
clock: Sistemin tüm operasyonlarını senkronize etmek için kullanılan standart saat sinyali girişidir.	**image**: Seri durumdan çıkarılmış giriş görüntüsünü depolamak için kullanılan 3D dizi sinyalidir.
start: Evrişim sürecinin başlangıcını tetikleyen bir başlangıç sinyali girişidir.	**output**: Ara evrişim sonuçlarını depolamak için kullanılan 3D dizi sinyalidir.
coef: Evrişim çekirdekleri için katsayıları (ağırlıkları) içeren 294,912 bit genişliğinde bir giriş vektörüdür. Her çekirdek ağırlığının 16-bit işaretli tamsayı olduğu varsayılır.	**krnl**: Seri durumdan çıkarılmış evrişim çekirdeklerini depolamak için kullanılan 4D dizi sinyalidir.

bias: Her evrişim çekirdeği ile ilişkilendirilen sapmalar için 1,024 bit genişliğinde bir giriş vektörüdür, her sapmanın 16-bit işaretli tamsayı olduğu varsayılır.	**feature**: Havuzlama sonrası özellik haritalarını depolamak için kullanılan 3D dizi sinyalidir.
img: Her pikseli 32-bit işaretli tamsayı olan 13x13x32 görüntüyü temsil eden 173,056 bit genişliğinde bir giriş vektörüdür.	**biases**: Seri durumdan çıkarılmış sapmaları depolamak için kullanılan dizi sinyalidir.
new_img: Evrişim, ReLU ve havuzlama sonrası işlenmiş görüntüyü temsil eden 76,800 bit genişliğinde bir çıkış vektörüdür, her pikseli 48-bit işaretli tamsayı olan 5x5x64 görüntü olarak biçimlendirilmiştir.	**state**: Operasyonların akışını kontrol eden durum makinesinin mevcut durumunu temsil eden bir sinyaldir.
done: Tüm işleme dizisinin tamamlandığını belirten bir çıkış sinyalidir.	

Tablo 6: İkinci Evrişim ve Havuzlama Katmanında Kullanılan Fonksiyonlar

Fonksiyon	Kullanım Yeri
ReLU	ReLU (Düzeltilmiş Doğrusal Birim) fonksiyonu, evrişim çıktısına bir aktivasyon fonksiyonu uygulamak üzere tanımlanmıştır. Girdi sıfırdan büyükse girdiyi geri döndürür; aksi takdirde sıfır döndürür. Bu fonksiyon, çıktı piksellerini aktive etmek için evrişim durumunda kullanılır
Max	Max fonksiyonu, iki adet 48-bit işaretli tamsayıyı karşılaştırır ve daha büyük olan değeri döndürür. Bu fonksiyon, havuzlama durumunda kullanılır ve bir 5x5 bloktan maksimum değeri seçerek özellik haritalarının boyutunu azaltır.

Durum Geçişleri: İlk durum olan deserial'da, giriş görüntüsü image dizisine (173,056 bit) seri durumdan çıkarılır. Görüntü seri durumdan çıkarma işlemi tamamlandığında, durum deserial_krnl'a geçer ve evrişim çekirdeği ağırlıkları krnl dizisine (294,912 bit) seri durumdan çıkarılır. Çekirdek ağırlıkları seri durumdan çıkarma işlemi tamamlandıktan sonra, durum deserial_bias'a geçer ve sapmalar biases dizisine (1,024 bit) seri durumdan çıkarılır. Daha sonra conv durumuna geçilir, burada evrişim işlemi gerçekleştirilir. Evrişim sonuçları output dizisinde (76,800 bit) saklanır ve ReLU fonksiyonundan geçirilir. Evrişim tamamlandıktan sonra, durum havuzlama aşamasına geçer, burada çıktı dizisine maksimum havuzlama uygulanır ve sonuçlar feature dizisinde saklanır. Havuzlama sonrasında, durum serial'e geçer, burada feature dizisi new_img vektörüne (76,800 bit) geri serileştirilir. Son olarak, finish durumuna ulaşılır ve işlem dizisinin tamamlandığını belirtmek için 'tamamlandı' sinyali '1' olarak ayarlanır.

Sayaçlar

Süreç boyunca, döngüler yerine sayaçlar kullanılarak saat döngüsüyle senkronizasyon sağlanır. Bu sayaçlar, görüntü, çekirdek ve çıktı dizileri içindeki pozisyonları izler, gerekli olduğunda artırma ve sıfırlama işlemleri yaparak her bir elemanı saat sinyali ile senkronize bir şekilde işler. İlk sayaç olan “kz_counter”, matrisin derinliğini (veya özellik haritalarının sayısını) 32'ye kadar sayar. Sonra “kx_counter” ve “ky_counter”, evrişim sırasında çekirdek matrisi içindeki pozisyonu belirten “krnl”

sinyalinin ilk iki indisini izler. “kt_counter” sayacı, 64 olan filtre sayısını sayar. “i_counter” ve “jcounter”, çıktı özellik haritasının boyutu olan (11x11) üzerinde iterasyon yapar. “i_counter” ve “jcounter” çıktısındaki her pozisyon için, ilgili filtre ve gelecek harita kullanılarak evrişim işlemi gerçekleştirilir.

Her evrişim işlemi için toplam, “sumvar” değişkeninde saklanır. “sumvar” değişkeni, her ilgili filtrenin sapması ile başlar ve ilgili filtre numarası için evrişim işlemi tamamlandığında, sonuç ReLU fonksiyonu ile kontrol edilir ve çıktıya atanır.

Diğer sayaçlar toplama ve çarpma işlemlerini gerçekleştirmekten sorumludur. Doğrudan bir evrişim işlemi uygulanır; burada [13,13,32] giriş görüntüsü [3,3,32] çekirdeği ile evrişim yapılır. Bu, 3x3 çekirdeğin ilk havuzlamadan sonra 32 özellik haritası için 13x13 görüntü üzerinde uygulandığı anlamına gelir.

Projede, en içteki sayaç olan "kz_counter," matrisin derinliğini veya özellik haritalarının sayısını temsil eden 32'ye kadar sayar. Ardından "ixk" değişkeni, "image(i_counter + kx_counter, j_counter + ky_counter, kz_counter) * krnl(kx_counter, ky_counter, kz_counter, kt_counter)" değerini saklamak üzere başlatılır. Bu ifade, 3x3 çekirdeğin 13x13 görüntü üzerinde hareket ettiğini ve ilgili özellik haritası (kz_counter tarafından sayılır) ve filtre (64 filtre için kt_counter tarafından sayılır) için o bölgede bir çarpma işlemi gerçekleştirdiğini gösterir. Bu işlem tamamlandıktan sonra, her özellik haritası için "ixk" toplanır ve "sumvar" değişkeninde saklanır. 3x3 çekirdeğin indekslerine atıfta bulunan "kx_counter" ve "ky_counter" sayaç değişkenleri 3'e ulaşana kadar artırılır.

İkinci Havuzlama Katmanı

Evrişim katmanının tamamlanmasının ardından, işlenmiş görüntü ikinci havuzlama katmanına aktarılır. Bu katman "x", "y" ve "z" olmak üzere üç sayaca sahiptir. "x" ve "y" sayaçları, 64 özellik haritasının her biri için 11x11 görüntü üzerindeki 5x5 kareyi indeksler. "z" sayacı, toplamda 64 olan özellik haritalarının sayısını izler. 5x5 kare içindeki maksimum noktaları seçtikten sonra, çıktımızı serileştirir ve yoğun katmana beslenir.

2.1.5.4.4. Verilog ile Yazılmış Yoğun Katman (dense.v)

Yoğun Katman, projede evrişimli sinir ağı modelinin karar verme yeteneğini sağlayan kritik adımlardan biridir. Giriş katmanı, Yoğun katman ve Çıkış katmanı olmak üzere üç ana katmandan oluşur. Çıktı tahminleri, evrişimde olduğu gibi; giriş, ilişkili ağırlıkları ve sapmaları ile çarpma ve toplama işlemleri yoluyla elde edilir. Verilog implementasyonunda, yoğun katman ayrı bir Verilog dosyası olarak yazılmış ve ayrı bir FSM (Sonlu Durum Makinesi) uygulaması olarak uygulanmıştır [13].

Tablo 7: Yoğun Katmandaki Portlar ve Sinyaller

Portlar	Sinyaller
clock: Sistemin tüm operasyonlarını senkronize etmek için kullanılan standart giriş saat sinyalidir.	**image**: Seri durumdan çıkarılmış giriş görüntüsünü depolamak için kullanılan 1D dizi sinyalidir.
start: Çarpma ve toplama işlem sürecinin başlangıcını tetikleyen bir başlangıç sinyali girişidir.	**output**: Ara evrişim sonuçlarını depolamak için kullanılan 1D dizi sinyalidir.

coef: Nöronlar arasındaki işlemler için katsayıları (ağırlıkları) içeren 256,000 bit genişliğinde bir giriş vektörüdür. Her çekirdek ağırlığının 16-bit işaretli tamsayı olduğu varsayılır.	**weights**: Nöronlar arasındaki seri durumdan çıkarılmış ağırlıkları depolamak için kullanılan 2D dizi sinyalidir.
bias: Her biri 16-bit işaretli tamsayı olan sapmalarla ilişkilendirilen 160 bit genişliğinde bir giriş vektörüdür.	**feature**: Havuzlama sonrası özellik haritalarını depolamak için kullanılan 3D dizi sinyalidir.
img: "con2d_1.vhdl" dosyasının çıktısı olan serileştirilmiş bir giriş görüntü vektörünü temsil eden 76,800 bit genişliğinde bir giriş vektörüdür.	**biases**: Seri durumdan çıkarılmış sapmaları depolamak için kullanılan dizi sinyalidir.
new_img: İşlenmiş çıktı tahminini temsil eden 76,800 bit genişliğinde bir çıkış vektörüdür.	**state**: perasyonların akışını kontrol eden durum makinesinin mevcut durumunu temsil eden bir sinyaldir.
done: Tüm işleme dizisinin tamamlandığını belirten bir çıkış sinyalidir.	

Tablo 8: Yoğun Katmanda Kullanılan Fonksiyonlar

Fonksiyon	Kullanım Yeri
Sigmoid	Sigmoid fonksiyonu, yoğun katman için bir aktivasyon fonksiyonu olarak tanımlanmıştır. Verilog'da sigmoid fonksiyonunu uygulamak için, sabit noktalı sayılardan oluşan bir dizi içine
	sigmoid fonksiyon sonucunu veren sayılar yerleştirilmiş ve LUT (Arama Tablosu) kullanarak gereken sayı değerlerine ulaşılmıştır.

Durum Geçişleri:

İlk durum, VHDL kodunda "deserial_input" olarak belirtilen seri durumdan çıkarma durumudur, burada serileştirilmiş giriş görüntüsü 1-D görüntü dizisine (76,800 bit) seri durumdan çıkarılır ve yoğun algoritma için bir giriş katmanı oluşturulur. "deserial_input" aşaması tamamlandığında, durum "deserial_weights" durumuna geçer ve ağırlıklar "weights" dizisine (256,000 bit) seri durumdan çıkarılır. Çekirdek ağırlıkları seri durumdan çıkarıldıktan sonra, durum deserial_bias'a geçer ve sapmalar biases dizisine (160 bit) seri durumdan çıkarılır. Daha sonra durum conv'a geçer, burada çarpma ve toplama işlemi gerçekleştirilir. İşlemlerin sonuçları çıktı dizisinde (640 bit) saklanır ve sigmoid fonksiyonundan geçirilir.

Sayaçlar

Sayaçlar, süreç boyunca döngüler yerine kullanılır ve saat döngüsüyle senkronizasyon sağlanır. Bu sayaçlar, giriş katmanı, çıkış katmanı, ağırlıklar ve sapmaların indekslerini izler. "j" sayacı giriş katmanı nöronlarını ve "x" sayacı çıkış katmanı nöronlarını ve sapmalarını izler. Ayrıca, her iki sayaç değişkeni işlem ağırlıklarının indekslenmesi için kullanılır.

Verilog implementasyonunda giriş katmanı, serileştirilmiş maksimum havuzlanmış ikinci katmanın çıktısı olan 1600 eleman (nöron) içeren bir vektörden oluşmaktadır. Çıkış katmanı, tahmin edilen ondalık basamakları (0-9) temsil etmek üzere tasarlanmış her biri 10 elemandan oluşan 10 adet

vektörden oluşmaktadır. Sonuç olarak projede, ağırlıklar için (1600,10) boyutunda bir dizi ve sapmalar için 10 elemanlı bir dizi bulunmaktadır.

"conv" aşamasında, çarpma ve toplama işlemlerini gerçeklemek için ve eleman bazında çarpma işlemini kolaylaştırmak amacıyla bir sayaç diziyi endekslemek üzere kullanılır. Çarpma sonuçları "ixk" değişkeninde, bu sonuçların toplamı "sumvar" değişkeninde saklanır. "ixk", giriş katmanı elemanlarının (nöronların) ağırlıklar ile eleman bazında çarpımını tutar, "sumvar" ise bu çarpım sonuçlarının toplamını biriktirir. Bu işlem tamamlandığında, sonuca sapmalar eklenir ve ardından aktivasyon fonksiyonuna beslenir. Bu uygulamada kullanılan aktivasyon fonksiyonu sigmoid fonksiyonudur. Sigmoid fonksiyonu ile çıktı dizisi (64-bit) işaretli 16-bit tamsayılara eşlenmektedir.

2.1.5.4.5. Verilog'da Sigmoid Fonksiyonunun Uygulanması

Verilog'da sigmoid fonksiyonunu uygulamak için, LUT (Arama Tablosu) kullanılır [11] ve sigmoid fonksiyonunun sonuçları sabit noktalı sayılar olarak bir diziye yerleştirilir. Proje, her biri Q0.15 formatında 16-bit genişliğinde olan 256 katsayı içeren "sigmoid_lut" adında bir dizi içermektedir. Tüm gerçek giriş değerleri için tanımlanan sınırlı bir gerçek fonksiyon olan sigmoid fonksiyonu, aşağıdaki matematiksel formülle ifade edilebilir:

$$\frac{1}{1 + e^{-x}}$$

Denklem 1: Sigmoid Formülü

"sigmoid_lut" dizisinin katsayılarını (elemanlarını) oluşturan kod Python ile yazılmıştır. LUT girişleri -4 ile 4 arasında eşit olarak dağıtılmış 256 nokta arasında haritalanmıştır. Bu amaçla, -4 ile 4 aralığı 256'ya bölünür ve her nokta sigmoid fonksiyon formülüne beslenir. Elde edilen değerler daha sonra sabit noktalı Q0.15 formatına dönüştürülür.

```python
def sigmoid(x):
    """ Sigmoid function """
    return 1 / (1 + math.exp(-x))

def generate_lut():
    lut_size = 256  # Number of entries in the LUT
    lut = []

    # Mapping range - Assume the index maps linearly from -4 to 4
    min_x = -4
    max_x = 4

    # Generate LUT values
    for i in range(lut_size):
        x = min_x + (max_x - min_x) * (i / (lut_size - 1))
        sig_value = sigmoid(x)
        #
        fixed_point_value = int(sig_value * (2**15))
        lut.append(f"to_signed({fixed_point_value}, 16)")
    return lut

def print_lut(lut):
    """ Print the LUT values in a format that can be copied into VHDL """
    print("type sigmoid_lut_type is array(0 to 255) of signed(15 downto 0);")
    print("constant sigmoid_lut : sigmoid_lut_type := (")
    print(", ".join(lut), ");")

# Generate and print LUT
lut = generate_lut()
print_lut(lut)
```

Şekil 5: LUT'taki Sigmoid Değerleri için Yazılmış Python Kodu

Katsayılar belirlendikten sonra, "sigmoid_lut" dizisine sabit kodlu olarak yerleştirilirler. Daha sonra Verilog'da "sigmoid" adında bir fonksiyon tanımlanır. Bu fonksiyon 64-bit değerleri kabul eder ve "sigmoid_lut" dizisi için bir indeks değeri olarak en önemli 8 biti kullanır. Son olarak, "sigmoid_lut" dizisindeki karşılık gelen değer nihai sonuç olur.

2.1.5.4.6. Çıktının Tahmini

Çıktı sigmoid fonksiyonu ile eşlendikten sonra, nihai tahmin sonucu elde edilmektedir. Doğrulama süreci, Vivado ve Cadence ile yapılan simülasyon ve sentezler ile uygulanmıştır. Simülasyon sonucunda, en yüksek 16-bit işaretli tamsayı değeri kullanıcıya CNN'in ana tahminini sunmaktadır. Örneğin, çıktı dizisinde "8" indeksi en yüksek değer ise, ağın verilen giriş için tahmini "8"dir.

2.1.6. Linker Script ve Yazılım Entegrasyonu

Projede linkerscript kullanımı yazılımın yerleştirilme ve çalışma prosedürlerini açıklayıcı nitelikte olduğu için önem taşımaktadır. Linkerscript dosyası, bellek bölümlendirmesi ile yazılımın bellek yönetiminin sağlanmasını desteklemektedir. Dosyada Flash hafıza ve RAM gibi farklı bellek bölümleri tanımlanmıştır. Flash bellek kod ve sabit veriler için kullanılırken, RAM dinamik veri saklama ve işlem yapma için ayrılmıştır. Bu tanımlar sayesinde kod ve verilerin fiziksel olarak nerede saklanacağı belirlenmiştir.

Linkerscript dosyasında .text, .data, .bss gibi farklı seksiyonlar özel bellek bölgelerine yerleştirilmiştir. Bu yerleştirme işleminde .text'in flash belleğe, .data ve ve .bdd değişkenlerinin ise RAM'e yerleştirilmesi uygun görülmüştür. Bu sayede bellek kullanımı optimize edilebilmekte ve işlemci performansına katkı sağlanabilmektedir. Bunların yanında .python_ui_data adında kullanıcı arayüzü verilerini depolamak için özel bir bölüm tasarlanmıştır. Kesinti vektörlerinin korunması ve uygulamanın giriş noktasının (ENTRY(main)) belirtilmesi, mikrodenetleyicinin başlangıçta doğru işlemi yürütmesini ve sistem kesintilerine doğru yanıt vermesini sağlar. Yazılım geliştirme sürecinde, C++ ve Verilog gibi diller kullanılarak yazılan modüller, belirlenen bellek adreslerine ve seksiyonlara uygun şekilde yerleştirilmektedir.

2.1.7. FPGA implementasyonu Aşamasında karşılaşılan zorluklar ve ASIC İmplementasyon Aşamasına Geçiş Beklentileri

Yapılan tasarım başarıyla sentezlenebilmiş, fakat Nexys A7 kartı içi entegrasyon sırasında yetersiz slice register, flip-flop cell ve FDRE cell hatası alınmıştır. Bu sorun, FPGA'nın limitli kaynaklarının, gereken işlem kapasitesini karşılayamamasından kaynaklanmaktadır. Bu tür bir durumda, kaynakları daha verimli kullanmak için tasarımın optimizasyonu gerekir. İlgili tasarımın daha az kaynak tüketmesini sağlamak amacıyla, lojik hücrelerin birleştirilmesi, gereksiz kayıtların (registers) azaltılması ve pipeline yapılarının gözden geçirilmesi gibi çeşitli yöntemler devreye sokulabilir. Ayrıca, daha fazla kaynak sunan bir FPGA kartına geçiş yapmak da bir çözüm olabilir. Bu tür optimizasyonlar ve donanım seçimleri, FPGA üzerinde başarılı bir implementasyon yapılmasını sağlayacak adımlardır. Bahsedilen bu adımlar doğrultusunda gerçeklenebilir bir tasarım ve Cadence araçları ile nihai ASIC tasarım yapılacaktır.

2.2. Tasarım Detayı

2.2.1. İşlemci ve Bus Yapısının Tasarımı

Projedeki bus yapısı tasarımında AXI-4 iletişim protokolünden yararlanılması tercih edilmiştir. AXI4 (Advanced eXtensible Interface 4) iletişim protokolü, ARM tarafından geliştirilmiş, yüksek performanslı ve yüksek bant genişlikli veri iletimi için özelleştirilmiştir. Sistem içerisindeki farklı bileşenler arasında veri transferini optimize eden bir bus yapısına sahip olduğundan dolayı bu methoddan yararlanılmıştır. AXI-4 kullanımı, yüksek veri aktarım hızı, esnekliği ve düşük gecikme süresi ile tahmin sürecini hızlandırmada önemli rol oynamaktadır.

AXI-4 protokolü çerçevesinde veri transfer işlemi; okuma, yazma ve adresleme olmak üzere üç ana kanal üzerinden gerçekleştirilir. Bu kanallar sayesinde işlemler birbirlerinden bağımsız olarak tamamlanabilmekte ve artan verimlilik ile birlikte yüksek veri aktarım hızı sağlanabilmektedir.

BUS yapısında kullanılan AXI-4 protokolü, 32 veya 64 bit adresleme modunda çalışabilir ve eş zamanlı veri transferini destekler. Okuma ve yazma işlemleri için sağlanan bağımsız kanallar, ortalama veri aktarım hızını arttıran en önemli etmenlerdir. AXI-4 bus yapısı, birden fazla veri ögesinin ardışık şekilde transfer edildiği veri patlamalarını (burst) desteklemektedir. Kullanıcı tarafından girilen rakam girdisinin mümkün olan en kısa sürede bir tahmin ile sonuçlandırılabilmesi için düşük gecikme süresi kritiktir. Girdinin hızlı analizini büyük veri miktarlarının hızlı işlenmesini gerçekleştirerek sağlayan yüksek bant genişliği ile düşük gecikme hızına sahip olması AXI-4 BUS yapısını proje için önemli kılar.

Yapay zeka uygulamaları için önemli bir gereksinim olan yüksek bant genişliği, farklı sistem gereksinimlerine uyum sağlayabilen esneklik ve ölçeklenebilirliği gibi sebeplerden dolayı projeye AXI-4 bus yapısının entegrasyonu gerçekleştirilmiştir. Ayrıca endüstride sıkça kullanılan bir protokol olmasından dolayı, geniş IP kütüphanesi ile projenin ilerleyen aşamalarında getirilebilecek donanımsal yeniliklerle de bu bus yapısının uyum sağlayacağı düşünülmektedir.

CV32E40P işlemcisi, mikrodenetleyici içinde Wishbone bus yapısı kullanılarak çevre birimlerine bağlanmıştır. İşlemci Wishbone master olarak yapılandırılmış ve çevre birimleri (UART, SPI, PWM, vb.) Wishbone slave olarak bağlanmıştır. Bağlantılar master ve slave portları üzerinden gerçekleştirilmiş, veri transferi Wishbone arayüz sinyalleri ile sağlanmıştır. Çevre birimleri, Wishbone veri yolu üzerindeki sinyalleri kullanarak işlemci ile iletişim kurar. Bu sinyaller arasında wishbone_data_i, wishbone_data_o, wishbone_addr, wishbone_we gibi kontrol ve veri hatları bulunmaktadır.

Wishbone bus yapısı, modülerlik ve esneklik sağlar, bu da çevre birimlerinin eklenmesini ve sistemin genişletilmesini kolaylaştırır. Ancak, yüksek performans gerektiren uygulamalarda AXI4 gibi daha gelişmiş bus yapılarına göre performans düşebilir ve kaynak verimliliği açısından karmaşıktır. Wishbone bus yapısının avantajları arasında basitlik ve uyumluluk bulunurken, dezavantajları arasında düşük performans ve yüksek kaynak kullanımı sayılabilir.

Zamanlama kısıtlamaları, sinyal gecikmeleri ve sinkronizasyon sorunları analiz araçları kullanılarak optimize edilmiştir. Bu optimizasyonlar, veri yolu üzerindeki sinyal gecikmelerini ve uyumsuzlukları minimize ederek sistemin kararlılığını artırmıştır. Farklı çevre birimlerinin Wishbone bus ile uyumlu hale getirilmesi sırasında yaşanan uyumsuzluk sorunları, çevre birimlerinin tasarımında Wishbone bus protokolüne uygun arayüzler kullanılarak çözülmüştür.

Sonuç olarak, CV32E40P işlemcisi ve çevre birimleri, Wishbone bus kullanılarak verimli şekilde bağlanmış ve performans optimizasyonu sağlanmıştır. Bu yapı, mikrodenetleyici içinde çevre birimlerinin entegrasyonunu kolaylaştırırken, yüksek performans gerektiren uygulamalarda dikkatli tasarım ve optimizasyon gerektirmiştir.

2.2.2. Peripheral Tasarım Detayları

Şu ana kadar geliştirilmiş ve bize tanına son tarihe kadar geliştirilmesini tamamlamayı planladığımız çevre birimlere ait tasarım detaylarını ve tasarımların mevcut durumlarını aşağıda açıklanacaktır.

Projemizde UART, I2C Master, QSPI Master, Timer, GPIO ve USB Tam Hız Cihazı çevre birimlerini içermektedir. Her bir çevre birimi için detaylı tasarım aşamaları planlanmış ve UART çevre birimi için C driverları ve headerlar yazılarak, çevre biriminin çalışırlığı doğrulanmıştır. Tüm çevre birimleri tasarım durumları ve doğrulama durumları ile beraber aşağıdaki tabloda gösterilmektedir [8].

Tablo 9: Çevre Birimler ve Tasarım ile Doğrulama Durumları

Çevre Birimi	Tasarım Durumu	Doğrulama Durumu
UART	Ağırlık ve taraf katsayısı değerlerinin reboot işlemi sırasında sistemin hafıza birimlerine yüklenmesinde kullanılan UART çevre biriminin gerekli donanım modülleri (alıcı ve verici) yazılmış ve entegrasyonu sağlanmıştır. Alıcı ve verici için FIFO yapısını benimseyen kodlar da yazılmış ancak seri bir şekilde yalnızca program başlangıcında MNIST modülüne ve işlem tamamlandıktan sonra çevre birimlere iletim yapılacağından buna gereksinim olmayacağı düşünülerek ilk tasarıma geri dönülmüştür.	UART modülünün doğrulama çalışmaları Vivado simülasyon ortamında başarılı bir şekilde gerçekleştirilmiştir. C driver'ı yazılmış ve temel fonksiyonlar (başlatma, veri gönderme, veri alma) test edilmiştir.
I2C	Şartnamede belirtildiği için sisteme eklenen I2C Master çevre birimi için donanım tasarımı tamamlanmıştır. Aktif olarak sistem ile entegrasyonunun sağlanması üzerine uğraşılmaktadır.	Bu için doğrulama protokobu aşamada l modülü, yazılmamıştır.
QSPI	Şartnamede belirtildiği için sisteme eklenen QSPI Master çevre birimi tasarlanmış ve sistem ile entegre edilmiştir.	Bu için doğrulama protokobu aşamada l modülü, yazılmamıştır.
Timer	Şartnamede belirtildiği için sisteme eklenen Timer modülü için donanım tasarımı ve entegrasyonu tamamlanmıştır.	Bu için doğrulama protokobu aşamada l modülü, yazılmamıştır.

GPIO	Şartnamede belirtildiği için sisteme eklenen GPIO modülünün donanım tasarımı tamamlanmış ve entegrasyon sağlanmıştır.	Bu için doğrulama protokobu aşamada l modülü, yazılmamıştır.
USB Tam Hız Cihazı	Şartnamede belirtildiği için sisteme eklenen USB Tam Hız Cihazı için donanım modülleri tasarlanmış ve sistem ile entegre edilmiştir.	USB modülünün doğrulama süreci devam etmekte olup, C driver'ı henüz tamamlanmamıştır.

UART C Driver'ı İçeriği: UART için yazılmış driver, modülü başlatma, veri gönderme ve veri alma işlemlerini içermektedir. Driver'ın içeriğinde, "uart_init" fonksiyonu ile UART modülü belirlenen baud rate ile başlatılarak gerekli başlangıç ayarlarının yapılması sağlanmıştır. Veri gönderme işleminin test edilebilmesi adına veri gönderim kuyruğunun hazır olup olmamasını kontrol ettikten sonra veri gönderimine başlanmasını sağlayan "uart_send_byte" fonksiyonu yazılmıştır. Veri alımı için ise yazılmış olan "uart_receive_byte" fonksiyonu veri alım kuyruğunda olup olmadığını kontrol etmesi ve verinin alınması amacıyla yazılmıştır. Bu işlem sırasında veri olup olmadığının kontrol edilmesi amacıyla "uart_data_available" yardımcı fonksiyonu yazılmıştır.

Driver içeriğinde bulunan, "test_uart_transmission" isimli test fonksiyonu, UART modülünü başlatarak belirli bir test verisi gönderir ve ardından bu verinin doğru bir şekilde alınıp alınmadığını kontrol eder. Test sırasında, gönderilen ve alınan veriler karşılaştırılarak veri bütünlüğünün doğrulanması sağlanmış olur.

Çevre Birimleri için Yazılması Planlanan C Driverları Tarafından Test Edilmesi Planlanan Durumlar: I2C Master modülü için yapılan denemelerde, modülün başlatılmasının testinin yanı sıra I2C üzerinden doğru veri gönderimi ve alınması, I2C cihazları arasında iletişimin sağlandığı ve veri bütünlüğünün korunduğu doğrulanması planlanmaktadır. QSPI Master modülünde ise, modül başlatılarak okuma ve yazma işlemlerinin gerçekleştirilmesinin ardından QSPI üzerinden verinin doğru bir şekilde okunup yazıldığı, hızlı veri transferinin sağlandığı ve veri bütünlüğünün korunduğu test edilmesi amaçlanmaktadır. Timer modülünde yapılan denemelerde, belirlenen zaman dilimlerinde kesmeler (interrupt) oluşturulacak ve zamanlayıcının doğru çalışıp çalışmadığı, belirlenen süre sonunda kesme sinyallerinin üretilip üretilmediği incelenerek doğrulanacaktır. GPIO modülü için yapılan denemelerde, pinler giriş ve çıkış modunda kullanılarak veri alımı ve gönderimi gerçekleştirilecek alınan ve gönderilen veriler karşılaştırılacak bu şekilde GPIO pinlerinin doğru şekilde yapılandırılıp yapılandırılmadığı kontrol edilecektir. USB Tam Hız Cihazı modülü için yapılan denemelerde ise USB üzerinden veri gönderimi ve alımı test edilmesi planlanmaktadır.

2.2.3. FPGA Prototipleme Detayları

FPGA prototiplemesi aşamasında yarışma şartnamesinde de belirtildiği üzere.Nexys A7 FPGA boardu kullanılmıştır. Ancak, projenin şu an elimizde bulunan halinin büyük kaynak gereksinimleri, mevcut FPGA boardun kaynak kapasitesinin üzerindedir. Karşılaşılan bu zorluğa bir çözüm bulabilmek amacıyla, projedeki işlemci ve veri boyutlarını optimize edilmesi üzerinde çalışılmaktadır. Projedeki tüm yazılımlar ve çevre birimleri, geliştirme sürecinin bir parçası olarak yoğun bir şekilde test edilmiştir. Bu

testler, C++ ile yazılmış olan sinir ağı modelinin algoritmasının doğruluğunu kanıtlamıştır ancak FPGA'e fiziksel entegrasyon aşamasında yaşanılan zorluklar sebebiyle FPGA üzerinde henüz herhangi bir yazılım koşturulamamıştır. Yine de çevre birimlerinin testi sistemle uyumlu bir şekilde çalıştığından emin olmak için ayrı ayrı gerçekleştirilmiştir.

FPGA üzerinde doğrudan prototipleme aşamasından önce Vivado simülasyon ortamında katsayı boyutları 16 bitten 4 bite indirilerek geniş çaplı testler gerçekleştirilmiş ve başarılı bir sentez alınmıştır. FPGA prototipleme aşamasında gerekirse katsayı boyutu 4 bitten 2 bite indirilerek projenin board'a entegrasyonu gerçekleştirilecektir.

FPGA Prototipleme aşamasında karşılaşılan en büyük zorluk Nexys A7 board'ın sınırlı kaynakları sebebiyle projenin gereksinimlerini karşılayamamış olması olmuştur. Çözüm olarak, veri boyutlarını ve işlemci kaynaklarını 16 bitten 4 bite indirgeyerek kaynak kullanımını azaltılması düşünülmüştür. Bu aşamada edinilen tecrübe projenin ASIC implementasyon aşamasında karşılaşabilecek sorunlara fayda sağlayacaktır.

3. ÇİP TASARIM AKIŞI

Tasarım sürecine işlemci çekirdeğini Verilog dilinde tasarlayarak başlanmış ve çeşitli test senaryolarıyla simülasyonlarını yaparak modelin doğruluğunu sağlanılmıştır. RTL tasarımının doğrulanmasından sonra, Verilog kodunu sentetik kapılara dönüştürmek için Vivado kullanarak sentez aşamasını gerçekleştirilmiştir. Sentez işlemi sırasında zamanlama sorunlarıyla karşılaşılmış, bu sorunları çözmek için clock frekansını optimize ederek bazı yol gecikmelerini düzeltilmeye çalışılmıştır. Ayrıca, kapı (gate) seviyesinde fazla güç tüketimi olan kısımlar tespit edilerek düşük güç tüketimli kapılarla değiştirilmiştir.

Sentez aşamasını takiben, fiziksel yerleştirme ve yönlendirme (placement and routing) aşamalarına geçilmiştir. Cadence araçlarını kullanarak tasarım yerleştirilmiş ve yönlendirilmiştir. Bu aşamada karşılaşılan en büyük sorunlardan biri, yoğun bağlantılardan dolayı bazı sinyallerin yönlendirilememesi olmuştur. Bu sorunu çözmek için tasarımı yeniden bölümlendirmek ve yönlendirme kurallarını optimize etmek gerekmiş, bu sayede sorunlar giderilmiştir. Ayrıca, güç dağıtım ağı optimize edilerek IR düşüşleri minimize edilmiştir.

Yerleştirme ve yönlendirme aşamasından sonra zamanlama analizi yapılmış ve kaynak kullanım raporları incelenmiştir. Zamanlama analizinde, kritik yol gecikmelerini belirlenip bu yolları optimize edilerek zamanlama ihlallerini çözülebilmiştir. Kaynak kullanım analizinde, bazı modüllerin fazla alan kapladığı tespit edilerek, bu modülleri yeniden tasarlanmıştır.

GDSII layout resimlerini Cadence araçlarına erişim problemi devam etmekte olduğundan sunulamamaktadır. Ancak, tasarım sürecinde karşılaşılan sorunlar ve çözümleri detaylandırılarak genel bir bakış sağlanmıştır. Örneğin, yerleştirme ve yönlendirme sırasında yoğun bağlantılardan kaynaklanan yönlendirme sorunlarını çözmek için tasarımın bölümlendirilmesi ve yönlendirme kurallarının optimize edilmesi gibi yöntemlerin kullanımı uygun görülmüştür. Tasarım sürecinde Slack kanalları gibi kaynaklardan yararlanılarak, karşılaşılan sorunlara çözümler bulunmuş ve tasarım sürecimizi hızlandırılabilmiştir. Çip tasarımının bazı aşamaları diğerlerine göre daha fazla problemin oluşmasına sebebiyet vermiştir; özellikle yerleştirme ve yönlendirme aşaması üzerine harcanılan zaman düşünüldüğünde kritik olmuştur ve iş planında çeşitli aksaklıklar yaşanmasına neden olmuştur. Süreç boyunca öğrenilen teorik bilgiler pratik uygulamalarla birleştirerek başarılı bir tasarım elde edilebilmiştir.

4. TEST

Her katmanın işlevselliği test bench kodları aracılığıyla test edilmiştir. İlk test benchten yola çıkılarak yazılan kod, sisteme daha fazla katman eklendikçe geliştirilmiştir, bu nedenle sadece tek bir test bench uygulaması bulunmaktadır. Ancak, model gelişim sürecinde ilerleme kaydedildikçe, metin dosyalarının giriş ve çıkışı, uzunlukları ve içerikleri, Verilog kodunda Python ve C++'da tanımlanan model davranışlarının doğru bir şekilde yansıtıldığından emin olmak için her aşamada test bench aracılığıyla detaylı olarak incelenmiştir.

16 bitte sentez çok uzun sürdüğünden ve kullanılacak olan FPGA modeli tarafımıza geç iletildiğinden veri besleme konusunda ön taslak raporunda belirtildiği üzere üç farklı senaryonun incelenmesi mümkün olmamış, yapılan değerlendirmeler üzerine bu fikirden vazgeçilmiştir. Bunun yerine taraf katsayılarının sistemin içine gömülmesi, ağırlıkların ise RISC-V çekirdeğindeki LUT'larda tutulması uygun görülmüştür.

Vivado ile sentez tamamlandıktan sonra test ile doğrulama aşaması tamamlanmış, ancak Cadence uygulamasında yaşanan ve hala çözülememiş durumda olan server hataları sebebiyle Cadence ile test gerçekleştirilememiştir.

Tasarım ve doğrulama sürecimiz, RTL seviyesinde fonksiyonel simülasyonlar, CV32E40P işlemcinin detaylı doğrulanması, UART peripheral için UVM/SystemVerilog ile yapılan kapsamlı doğrulama çalışmaları ve static timing analysis ile maksimum frekans hesaplamalarını içermektedir. Bu süreçlerde, bize sağlanan Cadence araçlarına bağlantı sorunları yaşamamıza rağmen, tasarımın doğruluğunu ve performansını değerlendirmek için alternatif yöntemler geliştirdik. Özellikle, Vivado'nun sunduğu simülasyon ve doğrulama kapasitelerini kullanarak alt modüllerdeki simülasyonları başarıyla gerçekleştirdik. Verification ortamımız, hem FPGA hem de çip tasarım doğrulaması için kullanılan ileri düzey testbench kodları ve çeşitli verification araçları ile desteklenmektedir. Bu zorluklara rağmen, projemizde kayda değer ilerlemeler kaydetmeyi sürdürüyoruz ve tasarım sürecimizin her aşamasında yüksek standartlarda çıktılar elde etmeye odaklanıyoruz. Bu yaklaşım, projemizin başarıyla tamamlanmasını ve endüstri standartlarında sonuçlar üretmesini sağlamaktadır.

5. TAKIM ORGANİZASYONU ve GÖREV DAĞILIMI

5.1. Takım Organizasyonu

KU RAMs, mikrodenetleyicilerde yapay zekâ uygulamaları alanında çalışmalarını sürdüren üç lisans öğrencisi tarafından Haziran 2023 tarihinde Doç. Dr. Mehmet Cengiz Onbaşlı danışmanlığında kurulmuştur. Proje ekip üyeleri İsmet Erdem, Damla Görgülü ve İdil Görgülü, Koç Üniversitesi'nde bilgisayar mühendisliği ile çift anadal yapan üçüncü sınıf elektrik elektronik mühendisliği lisans öğrencileridir.

5.2. Görev Dağılımı

Tablo 10: Takım Üyeleri ve Katkıları

Takım Üyeleri	Rol/ Görev	Katkıları
İsmet Erdem	Takım Kaptanı	Takım organizasyonu, RISC-V alt modüllerinin kodlanması, veri giriş alt modüllerinin tanımlanması,, C++ ile yazılan katman kodlarının Verilog'a aktarımı, Vivado ile doğrulama ve sentezin tamamlanması, UART modulünün hazırlanması, raporlama
Damla Görgülü	Üye	Python ile CNN kodlanması ve eğitimi, RISC-V alt modüllerinin çalışılması, veri giriş alt modüllerinin tanımlanması, C++ ile katman kodlarınn yazılması ve VHDL'e aktarımı, Vivado ile doğrulama ve sentezin tamamlanması , doğrulama ve raporlama
İdil Görgülü	Üye	Python ile CNN kodlanması ve eğitimi, ağırlık ve sapma değerlerinin belirlenmesi, RISC-V alt modüllerinin çalışılması, kullanıcı arayüzü kodunun Python ile yazılması, katman kodlarının C++ ile yazılması, doğrulama ve raporlama
M. C. Onbaşlı	Danışman	Akademik ve teknik danışmanlık, proje aşama kontrolü ve koordinasyonu

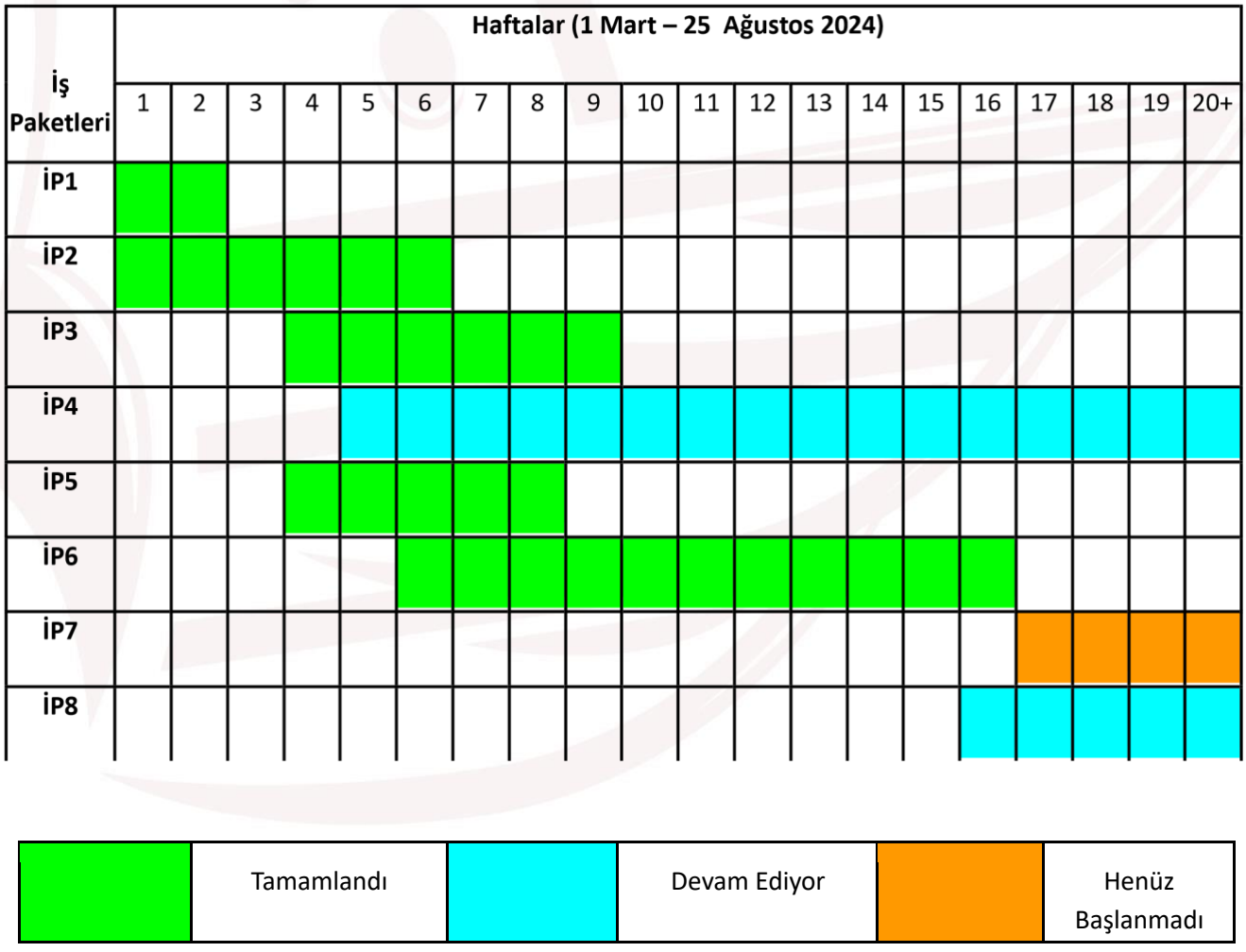
6. İŞ PLANI VE RİSK PLANLAMASI

Tablo 11: İş Paketleri ve Tanımları

İş paketi ve adı	Açıklama	Görev
İP1: MNIST sınıflama modelinin Python ile eğitilip test edilmesi	Çok katmanlı evrişimsel sinir ağlarının (CNN) Python'da kodlanması, eğitilmesi ve testleri	İdil - Damla
İP2: İsterler ile fonksiyonel blokların belirlenmesi	Model ve RISC-V işlemci ile entegre edilecek alt modüllerin tanımlanması	Takım
İP3: Sınıflama modelinin Verilog ile kodlanması ve doğrulanması	Katman kodlarının önce C++ ile, ardından Verilog ile yazılması ve gerekli testbench hazırlanarak doğrulanması	Takım
İP4: RISC-V işlemci çekirdeğinin kodlanması ve entegrasyonu	RISC-V mimarisinin çalışılması, alt modüllerin, kodlanması ve model ile entegrasyonu	İsmet

İP5: Veri besleme modüllerinin Verilog ile yazılması, sentezlenmesi	CNN modelleri ve veri besleme koşullarının gerçekleştirilmesi için gerekli veri giriş alt modüllerinin tanımlanması	Damla - İsmet
İP6: Vivado ile sentez, çip planının tasarımı, analizler (güç ve gecikme)	Vivado, OpenLane veya Synopsis araçları kullanılarak azami güç ihtiyacının ve gecikmenin 3 senaryo için hesaplanması	İdil - İsmet
İP7: Cadence ile sentez, çip planının tasarımı, analizler (bellek ve alan)	Vivado ile tamamlanmış olan sentezin Cadence ile de yapılarak doğrulanması ve analizlerin yürütülmesi	Takım
İP8: Fiziksel Gerçekleme ve Doğrulama	Fonksiyonel doğrulama (verification)	Takım

Tablo 12: İş Paketlerinin Zaman Çizelgesi Üzerinde Gösterimi



	Tamamlandı		Devam Ediyor		Henüz Başlanmadı

7. KAYNAKÇA

[1] Andreas Zinonos. “Beginner Deep Learning Tutorial | MNIST Digits Classification Neural Network in Python, Keras", YouTube, Aug. 11, 2020 [Video file].
Available:https://www.youtube.com/watch?v=BfCPxoYCgo0 [Accessed: May. 3,

2024]

[2] codebasics. “Neural Network For Handwritten Digits Classification | Deep Learning Tutorial 7 (Tensorflow2.0)”, YouTube, Jul. 18, 2020 [Video file]
Available: https://www.youtube.com/watch?v=iqQgED9vV7k&t=26s. [Accessed: May.

3, 2024]

[3] Edureka!. “Handwritten Digit Recognition on MNIST dataset | Machine Learning

Projects 5 | ML Training | Edureka”, YouTube, Jul. 29, 2021 [Video file]

Available: https://www.youtube.com/watch?v=L2cAjgc1-bo. [Accessed: May. 3, 2024]

[4] Nathanne Isip, “N2CMU (Neural Network Coprocessing Microcontroller Unit)”,GitHub repository, [Online]. Available:

https://github.com/nthnn/n2cmu [Accessed: May. 4, 2024]

[5] P. Huang et al., “Towards Efficient Verification of Quantized Neural Networks.” arXiv, 2023. Doi: 10.48550/ARXIV.2312.12679. Available:
https://arxiv.org/abs/2312.12679

[6] R. Campagnoli, "Implementation of a Convolutional Neural Network Algorithm on

FPGA Using High-Level Synthesis," M.S. thesis, Department of Electronics and

Telecommunications., PoliTO., Torino, 2021. [Online].

Available:https://webthesis.biblio.polito.it/secure/19272/1/tesi.pdf. [Accessed: May.

3,2024]

[7] “OpenHW Group CV32E40P User Manual — CORE-V CV32E40P User Manual v1.8.3 documentation,” *docs.openhwgroup.org*. https://docs.openhwgroup.org/projects/cv32e40p-user-manual/en/latest/index.html (accessed June. 6, 2024).

[8] RISC-V Verification, "RVVI," GitHub repository, [Online].

Available: https://github.com/riscv-verification/RVVI. [Accessed: Mar. 10, 2024].

[9] S. Campbell, “Basics of UART Communication,” Circuit Basics, Feb. 13, 2016.

Available: https://www.circuitbasics.com/basics-uart-communication/

[10]Vipin Kizheppatt. “Neural Networks on FPGA”, YouTube, Jun. 1, 2020 [Video file]

Available: https://youtu.be/rw_JITpbh3k?si=yqUTEHhRW_wvc7-w. [Accessed: Jan. 3,

2024]

[11] Vipin Menon, "neuralNetwork," GitHub repository, [Online]. Available: https://github.com/vipinkmenon/neuralNetwork. [Accessed: Jan. 4, 2024]

[12] W. Rawat and Z. Wang, “Deep Convolutional Neural Networks for Image Classification: A Comprehensive review,” Neural Computation, vol. 29, no. 9, pp.
2352–2449, Sep. 2017, doi: 10.1162/neco_a_00990.

[13]Y. Hao, "A General Neural Network Hardware Architecture on FPGA," arXiv preprint arXiv:1711.05860, 2017.