FPGA İLE RISC-V TABANLI 32-BİT KRİPTO İŞLEMCİ TASARIMI

2024
YÜKSEK LİSANS TEZİ
MEKATRONİK MÜHENDİSLİĞİ

Kamer KIRALİ

Tez Danışmanı
Doç. Dr. Can Bülent FİDAN

FPGA İLE RISC-V TABANLI 32-BİT KRİPTO İŞLEMCİ TASARIMI

Kamer KIRALİ

Tez Danışmanı
Doç. Dr. Can Bülent FİDAN

T.C.
Karabük Üniversitesi
Lisansüstü Eğitim Enstitüsü
Mekatronik Mühendisliği Anabilim Dalında
Yüksek Lisans Tezi
Olarak Hazırlanmıştır

KARABÜK
Temmuz 2024

Kamer KIRALİ tarafından hazırlanan “FPGA İLE RISC-V TABANLI 32-BİT KRİPTO İŞLEMCİ TASARIMI” başlıklı bu tezin Yüksek Lisans Tezi olarak uygun olduğunu onaylarım.

Doç. Dr. Can Bülent FİDAN .....
Tez Danışmanı, Mekatronik Mühendisliği Anabilim Dalı

Bu çalışma, jürimiz tarafından Oy Birliği ile Mekatronik Mühendisliği Anabilim Dalında Yüksek Lisans tezi olarak kabul edilmiştir. 16/07/2024

İmzası

Başkan : Doç. Dr. Can Bülent FİDAN (KBÜ) .....

Üye : Doç. Dr. İsmail KOYUNCU (AKÜ) .....

Üye : Dr. Öğr. Üyesi Ali AKAY (KBÜ) .....

KBÜ Lisansüstü Eğitim Enstitüsü Yönetim Kurulu, bu tez ile, Yüksek Lisans derecesini onamıştır.

Doç. Dr. Zeynep ÖZCAN .....
Lisansüstü Eğitim Enstitüsü Müdürü

"Bu tezdeki tüm bilgilerin akademik kurallara ve etik ilkelere uygun olarak elde edildiğini ve sunulduğunu; ayrıca bu kuralların ve ilkelerin gerektirdiği şekilde, bu çalışmadan kaynaklanmayan bütün atıfları yaptığımı beyan ederim."

Kamer KIRALİ

ÖZET

Yüksel Lisans Tezi

FPGA İLE RISC-V TABANLI 32-BİT KRİPTO İŞLEMCİ TASARIMI

Kamer KIRALİ

Karabük Üniversitesi
Lisansüstü Eğitim Enstitüsü
Mekatronik Mühendisliği Anabilim Dalı

Tez Danışmanı:
Doç. Dr. Can Bülent FİDAN
Temmuz 2024, 98 sayfa

Günümüzde veri güvenliği git gide önem kazanmaktadır. Bundan dolayı, güvenli bilgi iletimi ve kullanımı için özel işlemcilerin tasarımı çok önemli olmaktadır. Bu nedenle bu çalışmada, FPGA ile RISC-V tabanlı 32-bit kripto işlemci tasarımına odaklanılmıştır. İşlemci en uygun şekilde optimize edilmeye çalışılmıştır. İşlemcinin ana özellikleri arasında, Two-Way Associative veri önbelleği ile Direct-Mapped Cache buyruk önbelleği bulunmaktadır. İşlemcimiz bilgi güvenliği gerektiren uygulamalarda kullanılmak üzere tasarlanmıştır. Veriler arasında şifreleme yapmak için önbellek ve ana hafıza arasına 8-bitlik S-box'lar yerleştirilmiştir. Bu S-box'larda, ana hafızadan gelen şifreli veriler çözülerek işlemci çekirdeğinin anlayacağı dile dönüştürülüp işlenmesi için çekirdeğe gönderilmektedir. RISCOF ortamından geçerek bütün buyrukların doğru çalıştığı ispatlanmıştır. İşlemciye, UART, SPI ve PWM gibi çevre birimleri eklenmiştir. Bu çevre birimlerinin herhangi bir haberleşme ve sinyal üretimi sırasında faydalı olacağı düşünülmektedir. Tasarlanan işlemci, herhangi bir elektronik sistemle veya bilgisayarla haberleşebilmesi için SPI ve UART protokolleri

ile desteklenmiştir. Motor kontrolü gibi uygulamalarda kullanmak üzere PWM modülü de işlemcimize eklenmiştir.

Anahtar Sözcükler : İşlemci, Kripto, RISC-V Mimarisi, Buyruklar, Verilog, FPGA, Çevre Birimleri.

Bilim Kodu **:** 90520

ABSTRACT

M. Sc. Thesis

RISC-V BASED 32-BIT CRYPTO PROCESSOR DESIGN WITH FPGA

Kamer Kırali

Karabük University
Institute of Graduate Programs
Department of Mechatronics Engineering

Thesis Advisor:
Doç. Dr. Can Bülent FİDAN
July 2024, 98 pages

Today, data security is becoming increasingly important. Therefore, the design of specialized processors for secure information transmission and use is very important. Therefore, this study focuses on the design of a RISC-V based 32-bit crypto processor with FPGA. The processor is optimized in an optimal way. The main features of the processor include Two-Way Associative data cache and Direct-Mapped Cache instruction cache. Our processor is designed to be used in applications that require information security. To encrypt data, 8-bit S-boxes are placed between the cache and main memory. In these S-boxes, encrypted data from the main memory is decrypted and converted into a language that the processor core understands and sent to the core for processing. All commands have been proven to work correctly by passing through the RISCOF environment. Peripherals such as UART, SPI and PWM were added to the processor. It is thought that these peripherals will be useful during any communication and signal generation. The designed processor is supported with SPI and UART protocols to communicate with any electronic system or computer. PWM

module has also been added to our processor to be used in applications such as motor control.

Key Word **:** Processor, Crypto, RISC-V Architecture, Instructions, Verilog, FPGA, Peripherals.

Science Code : 90520

TEŞEKKÜR

Bu tez çalışmasının, araştırılmasında, yürütülmesinde ve oluşumunda ilgi ve desteğini esirgemeyen, danışman hocam Doç. Dr. Can Bülent FİDAN başta olmak üzere, tüm hocalarıma ve arkadaşlarıma yardımları için tüm kalbimle teşekkür ederim.

KBUBAP-23-YL-034 numaralı, FPGA ile RISC-V Tabanlı 32-bit Kripto İşlemci Tasarımı projemi destekledikleri için Karabük Üniversitesi Bilimsel Araştırma Projeleri (BAP) Koordinatörlüğüne çok teşekkür ederim.

Hep yanımda olup, tüm problemlerimde bana yardımcı oldukları ve bu zorlu süreci atlatmakta verdikleri destekleri için sevgili aileme de çok teşekkür ederim.

İÇİNDEKİLER

Sayfa

Sayfa

Sayfa

ŞEKİLLER DİZİNİ

Sayfa

Sayfa

Sayfa

ÇİZELGELER DİZİNİ

Sayfa

SİMGELER VE KISALTMALAR DİZİNİ

KISALTMALAR

FPGA : Field-Programmable Gate Array (Alanda Programlanabilir kapı dizisi)
RISC : Reduced Instruction Set Computer (Azaltılmış Komut Seti Bilgisayarı)
CISC : Complex Instruction Set Computer
ASIC : Application Specific Integrated Circuit (Uygulamaya Özel Tümleşik Devre)
SOC : System On Chip (Yongada Sistem)
I/O : Input/Output (Giriş/Çıkış)
RTL : Register Transfer Level (Yazmaç)
ENIAC : Electronic Numerical Integrator and Computer
S-Box : Substitution Box
MIPS : Million Instruction Per Second
DMIPS : Dhrystone MIPS
ISA : Instruction Set Architecture
Hz : Hertz
MHz : Mega Hertz
UART : Universal Asynchronous Receiver/Transmitter
SPI : Serial Peripheral Interface
PWM : Pulse Width Modulation
IMM : Immediate (Anlık Değer)

BÖLÜM 1

GİRİŞ

Bilgisayar, tablet gibi teknolojik ürünlerde veya uçak, gemi, iha ve siha gibi askeri teknoloji alanlarında mikroçipler yaygın olarak kullanılmaktadır. Bugün için kullanılan mikroişlemciler, temel olarak toplama, çıkarma, çarpma ve bölme gibi aritmetik ve pek çok lojik işlemleri yapan ve verileri depolayan birimlerdir. Bilgi güvenliğinden hızlı işlemlere ve güç tüketimine kadar birçok alanda faaliyet gösterirler. Ancak mikroişlemci komut seti lisans ücretleri gibi durumlardan dolayı bu konu hakkında gelişim göstermek hem zor hem de pahalı olabiliyordu. Bu duruma karşı azaltılmış komut setli açık kaynaklı mikroişlemci mimarisinin çıkması günümüzde bu durumu değiştirmiştir.

17. yüzyılda insanlar matematik işlemlerinin çözülmesi işini makinelere yaptırmaya çalışmışlardır. Günümüzde kullanılan toplama, çıkartma, çarpma ve bölme gibi işlemlerin yapılması için makineler tasarlanmıştır. Bu makineleri tasarlayan matematikçiler arasında Wilhelm Schickhard, Blaise Pascal ve Gottfried Leibnitz gibi isimler vardır. Ama İlk çok amaçlı programlanabilir makineyi Charles Babbage 1823'te yapmaya başlamıştır. Ancak hiçbir zaman bitirememiştir [1].

1946 yılında ilk genel amaçlı elektronik bilgisayar olan ENIAC dünyaya tanıtıldı. ABD ordusu tarafından topçu ateşlerinin hesabını yapabilmek için geliştirildi. Pennsylvania Üniversitesi'nde John Presper Eckert ve John Mauchly tarafından tasarlanıp icat edildi. ENIAC projesi Alman askeri kuvvetlerine karşı avantaj sağlamak amacıyla başlatılmıştır. Ancak 2. Dünya Savaşı'nın sonunda tamamlanmıştır [2].

Intel 4004, 1971 yılında piyasaya sürülen tek bir çipte genel amaçlı üretilmiş ilk mikroişlemcidir [3].

Mikroişlemci talimatları daha karmaşık oldukça, daha basit talimatların daha az yer kaplayan yongalarda daha hızlı çalıştığı önerisi sunulmuştur. 1974 yılında IBM'de çalışan John Cocke tarafından RISC mimarisi geliştirilmiştir. 1980 yılında ise bu mimariden ilk yararlanan bilgisayar IBM'in geliştirdiği PC/XT oldu. Apple, Motorola ve IBM PowerPC'yi üretmek için 1991 yılında birlikte çalışmıştır [4,5].

RISC-V komut seti mimarisi 2010 yılında Berkeley'deki Carlifornia Üniversitesi'nde araştırma projesi olarak geliştirildi. Proje ileri ki zamanlarda işlemci tasarımları için temel sağlayacak açık kaynaklı bir komut seti mimarisi olarak tasarlandı. RV32I tamsayı tabanlı komut seti mimarisi 2011 yılında yayınlandı. Bu sürüm RISC ilkelerine bağlı kalmıştır. Bu ilk basit sürüm basitlik ve verimliliğe bağlı kalmıştır. Daha sonraki yıllarda RISC-V komut setini geliştirmek için yeni eklentiler eklenmiş ve işlemci komut seti mimarisi geliştirilmeye devam edilmiştir [6].

Bu tez kapsamında FPGA ile RISC-V tabanlı 32-bit kripto işlemci tasarlanmıştır. İşlemci tasarımında Verilog RTL donanım tanımlama dili kullanılmıştır. RV32IM eklentisine sahip UART, SPI, PWM ve ana hafıza ile önbellek arasında konulan S-box'lar sayesinde kripto özelliğine sahip bir işlemci oluşmuştur. İşlemci tasarımımız Vivado ortamında Zybo Z7-20 kartı ile test edilmiştir. Linux ortamında RISC-V için GNU derleyicisi ile C++ kodları derlenerek hexadecimal verilere çevrilip işlemcimizde test edilmiştir ve RISC-V RTL tasarımının debug edilebilmesi için özel ortamlar kurulmuştur. RISCOF, SPIKE gibi ortamlarda linux üzerinde debug işlemleri gerçekleştirilmiştir. Venus, RARS gibi simulasyon programlarında kendi programlarımızı yazarak tasarladığımız işlemcimizin hataları ayıklanmaya çalışılmıştır. Dhrystone Benchmark testi yapılarak işlemcinin alacağı DMIPS/MHz değeri 0.5050 DMIPS/MHz. olarak hesaplanılmıştır.

Bu tezin amacı RISC-V mimarisine sahip kripto bir işlemci tasarımının başarılmasıdır. Bunun için uygun koşullar ve uygun ortamlar sağlanmıştır. Programlama dili olarak Verilog donanım tanımlama dili seçilmiş olup uygun ortamlar sağlandıktan ve işlemci tasarımı tamamlandıktan sonra bazı testler yapılarak çıkan hatalar giderilmiştir. İleride OpenLane ortamında çalışılması için ram yapıları OPENRAM platformuna göre ayarlanmıştır.

BÖLÜM 2

BİLGİSAYAR MİMARİSİNE GİRİŞ

2.1. BİLGİSAYAR MİMARİSİ NEDİR?

Bilgisayar mimarisi, depolama, sistem yönetimi, matematiksel yöntemler gibi çeşitli kuralları tanımladığımız kurallar kümesine denir [7]. Daha ayrıntılı olarak bahsedecek olursak verilen buyrukları işleyebilen ve verilen komutları yerine getirebilen bazı aritmetik ve lojik işlemleri gerçekleştirebilen kurallar bütünüdür. Bilgisayar mimarisi kendi içinde Sistem Tasarımı, Mikro Mimari ve ISA (Komut Seti Mimarisi) olarak 3 farklı bölüme ayrılır.

2.1.1. Sistem Tasarımı

Sistem tasarımı, hafıza kontrolcüleri, birden fazla çekirdekli işlemciler, bellek erişimi ve GPU'lar gibi fiziksel bilgisayar sistemlerinin tümüne verilen addır. Bilgisayarı oluşturan bütün donanım parçaları diyebiliriz [8].

2.1.2. Mikro Mimari

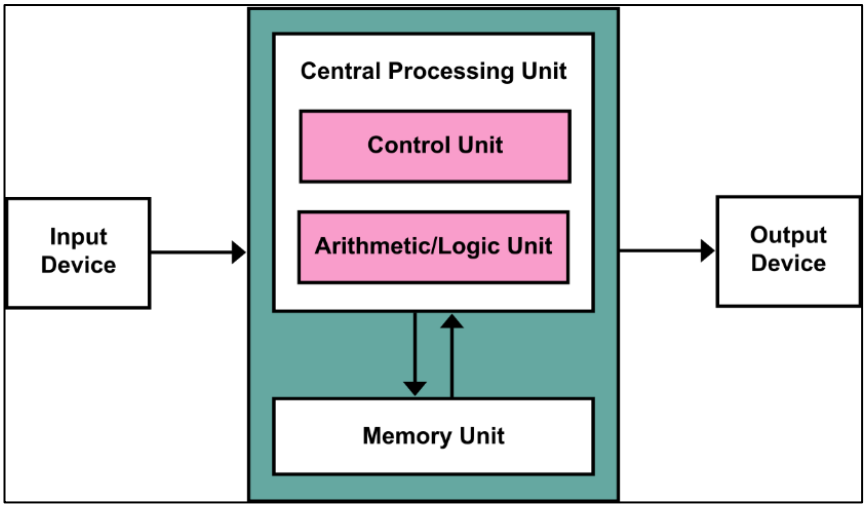
Mikro mimari, işlemci veya veri yolu tasarlanırken ISA'nın nasıl uygulanacağını belirten kurallardır [8]. Boru hattı Mimarisi, Super Scalar işlemci mikro mimariye örnek olarak verilebilir.

2.1.3. Komut Seti Mimarisi (ISA)

CPU'nun işlevleri ve yetenekleri, programlama dilleri, veri formatları, işlemci kayıt türleri ve programcılar tarafından kullanılan talimatlar dahil olmak üzere, bilgisayarı çalıştıran herhangi bir yazılımdır [9].

2.2. THE VON NEUMANN MİMARİSİ

The Von Neumann mimarisi John Von Neumann ve iş arkadaşları tarafından 1945 yılında ortaya atılan bir bilgisayar mimarisidir. Verilerin ve buyrukların aynı hafıza biriminde bulunan mimari türüdür. CPU ile hafıza birimi sadece tek bir veri yolundan haberleştiği için hafızanın hızı CPU'dan daha azdır. Buyruk çekilirken veri, veri çekilirkende buyruk beklemek zorunda kalır. Kısaca Von Neumann mimarisi veri ve buyruk paralel olarak işlenemez [10,11]. Şekil 2.1'de Von Neumann mimarisi görülmektedir.



Şekil 2.1. Von Neumann mimarisi blok diyagramı [11].

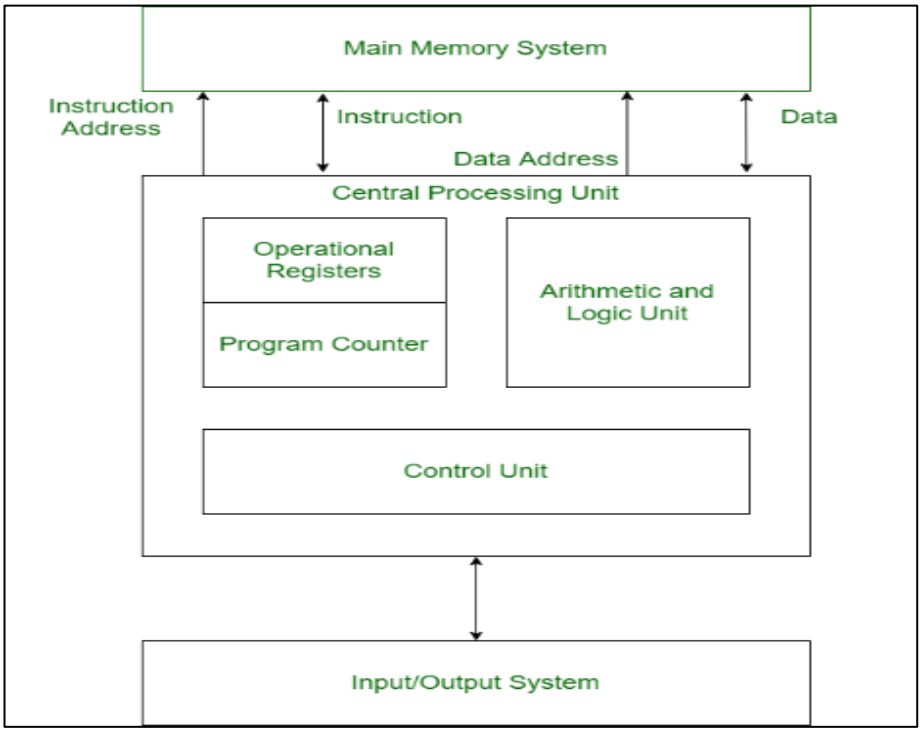
2.2.1. The Von Neumann Veri Yolu (BUS)

Bellek ile CPU arasındaki iletişim sağlayan bölgedir. Veriyi bir bölümden diğer bölgeye aktaran birimdir [10].

- Veri yolu, hafıza birimi, çevre birimleri ve işlemci arasında veri alışverişi yapar.
- Adres veri yolu, bellek ile CPU arasında gidecek olan verinin adresini iletir.
- Kontrol veri yolu, sistemdeki bütün modülleri kontrol etmek için üretilen sinyali taşır.

2.3. HARVARD MİMARİSİ

Bu mimaride veri girişleri ve buyruk girişleri ayrılmıştır. Paralel olarak çalışırlar. Bu mimari ilk defa Harvard Mark-1 bilgisayarında kullanıldı. Bu mimariyi kullanan bilgisayarlar veri ve buyruklar arasında herhangi bir köprü bulundurmazlar. Von Neumann mimarisinin yapamadığı hem komut işleyip hem veri kaydetme işlemini yapabilir. [12,13]. Şekil 2.2'de Harvard mimarisinin yapısı görülmektedir.



Şekil 2.2. Harvard mimarisi blok diyagramı [12].

2.3.1. Harvard Mimarisi Veri Yolu

Veri yolları hem sinyal yolları olarak kullanılıyor. Harvard mimarisinde hem buyruklar için hemde veriler için ayrı veri yolları bulunur [12].

- Data Bus (Veri Yolu), CPU ile hafıza birimi arasındaki veri iletimini gerçekleştirir.
- Data Address Bus (Veri address yolu), hafıza biriminde çekilecek verinin adresinin aktarıldığı veri yoludur.

- Instruction Bus (Buyruk yolu), hafıza biriminden çekilecek buyruğun veri yoludur.
- Instruction Adress Bus (Buyruk adresi veri yolu): Hafıza biriminden çekilecek buyruğun adresinin veri yoludur.

2.3.2. Harvard Mimarisinin Avantajları

Harvard mimarisi 2 tane bölünmüş veri ve buyruk veri yolu bulunur. Bu 2 veri yolu sayesinde buyruk ve veri işlerini aynı anda halledebilir. Genellikle buyruk ve veri için bölünmüş buyruk önbelleği ve veri önbelleği bulundurur. Bu mimari genel olarak ARM ve x86 işlemcilerinde kullanılır. Harvard mimarisinin avatajları aşağıdaki gibi sıralanmıştır [12,13];

- Harvard mimarisi, buyruk ve veri işlerini aynı anda parallel olarak işleyebildiği için diğer mimarilerden daha hızlıdır Bunun sebebi buyruk ve veri için ayrı belleklere sahip olmasından kaynaklanmaktadır.
- Harvard mimarisi, paralel işlem yapması, optimize edilmiş bellek ve belli uzunlukta buyruklara sahip olmasından dolayı talimatların diğer mimarilere göre daha hızlı yürütülmesine sebep olur.
- Hız ve verimlilikten dolayı Harvard mimarisi gerçek zamanlı uygulama ve gömülü sistem gibi endüstriyel uygulamalarda yaygın olarak kullanılır.
- Veri ve Buyruk önbelleğinin ayrılması ara hafıza taşması gibi saldıralara karşı az da olsa diğer mimarilere göre bir derece güvenlik sağlamaktadır.

2.3.3. Harvard Mimarisinin Dezavantajları

Harvard mimarisinin dezavantajları aşağıdaki gibi sıralandırılmıştır [12];

- Buyrukların ve verilerin ayrı bellek alanlarında olması belirli yazılım algoritmalarının uygulama olasığını zorlaştırır yada imkansız hale getirir.
- Harvard mimarisi, Von Neumann mimarisine göre daha fazla bellek gereksinimine ihtiyaç duyar buda maliyet ve güç tüketiminde artış olması demektir.

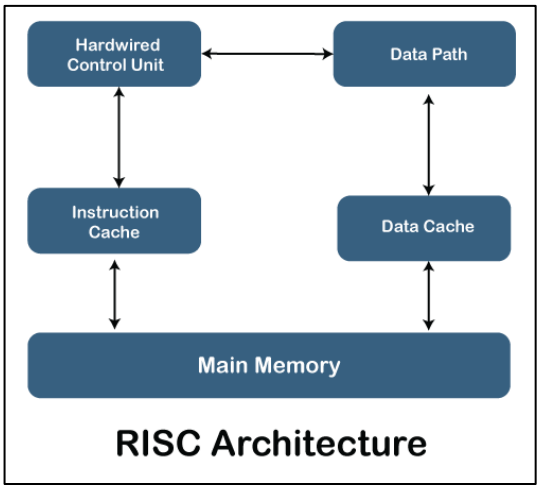
- Veri ve buyruk önbelleklerinin ayrı olması işlemci tasarımını daha kompleks hale getirir. Üretim maliyetini arttırır.
- Harvard mimarisindeki değişmeyen buyruk uzunluğu, çalıştırılabilecek komutun boyutunu sınırlayabilir ve bu da onu daha büyük kod tabanlı bazı uygulamalar için uygunsuz hale getirir.

2.4. RISC MİMARİSİ

RISC (Reduced Instruction Set Computer), diğer mimarilere göre daha basit komutlar kullanan ve komutları daha hızlı çalıştıran bir mimaridir. RISC mimarisindeki komutlar basit olduğu için diğer mimarilere göre daha hızlı çalışır. Basit komutlar ve basit decoderlara sahip olduğu için diğer mimarilere göre daha ucuz maliyet gereksinimi sağlar. Bir diğer yandan RISC mimarisinin basit komutlara sahip olduğundan CISC mimarisine göre daha fazla komut yazmak gerekecektir. Bundan dolayı RISC mimarisi hızlı ama kullanıcı dostu bir mimari değildir. RISC mimarisi CISC mimarisinden sonra çıkmış bir mimari olsada sunucu teknolojilerinde daha fazla tercih edilmektedir [14]. Şekil 2.3'de RISC mimarisi blok gösterimi görülmektedir.

Aşağıda RISC mimarisi ile yapılmış işlemci örnekleri sıralanmıştır [14];

- ARM.
- MIPS R2000.
- MOTOROLA 8800.
- IBM RS/6000.
- INTEL İ860.



Şekil 2.3. RISC mimarisinin blok gösterimi [15].

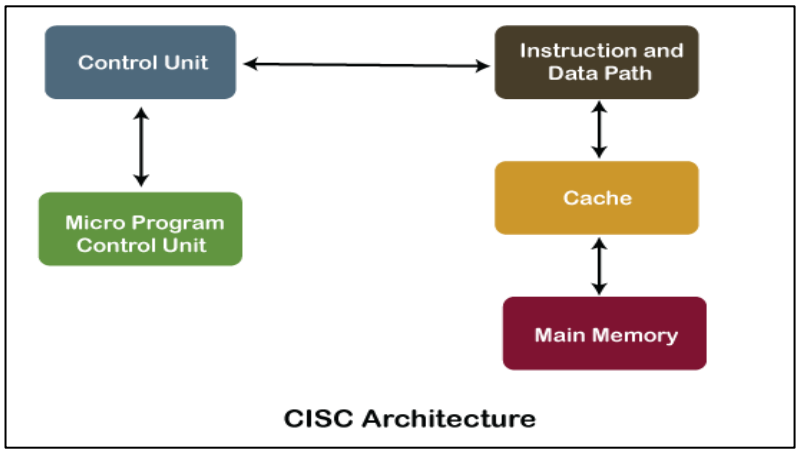
RISC işlemcilerinin avantajları şunlardır [16];

- RISC işlemcilerin performansı komut sayısı az olduğu için diğer işlemcilere göre daha iyidir.
- Tasarımı daha basit olduğu için maliyeti daha ucuzdur.
- RISC işlemci diğer mimarilere göre daha basit ve hızlı olduğu için bir işlemi tek cycle hızında yapabilir.
- Diğer mimarilere göre daha az komut seti olduğu için fiziksel donanımda daha az yer kaplar ve diğer mimarilere göre boyutları daha küçüktür.

RISC işlemcilerinin Dezavantajları şunlardır [16];

- RISC mimarisi ne kadar hızlı olsada yavaş türden anabellek kullanıldığında işlemcinin hızının yavaşlamaması için hızlı önbellek kullanmak zorunda kalınması.
- Kullanıcı diğer mimarilere göre daha karmaşık kodlamalar yapmak zorunda kalır. Kullanıcı dostu bir mimari değildir. Daha çok donanım dostu bir mimaridir.
- Diğer mimarilerde yazılan tek satırlık kodlar RISC mimarisinde birden fazla satırda yazılması gerekebilir.

2.5. CISC MİMARİSİ



Şekil 2.4. CISC mimarisinin blok gösterimi [17].

CISC (Complex Instruction Set Computer), Karmaşık assembly komutları yer aldığı için bu ismi almıştır. CISC mimarisinde pek çok aritmetik işlem için farklı buyruklar vardır. Komutların çevrimleri birden fazla saat darbesinde çalışırken komutların derlenmesi daha hızlıdır. Örnek vermek gerekirse kare alma işlemi yaparken CISC mimarisi içerisinde bunun için özel bir komut bulunur ve tek satırda bu işlem uygulanabilinirken RISC mimarisinde bunu yapabilmek için 2 satırdan fazla komut yazmak gerekir. CISC mimarilerinde tasarlanmış mikroişlemcilerde çok fazla türde komut bulunduğu için donanımda çok fazla yer kaplar ve bundan dolayı maliyeti daha yüksek bir mimari türüdür. CISC mimarisinde her eylem için bir komut tanımı yapıldığı için bir işlemler daha fazla saat darbesinde biter [14]. Şekil 2.4'te CISC mimarisinin blok gösterimi görülmektedir.

Aşağıda CISC mimarisi ile yapılmış işlemci örnekleri sıralanmıştır [14];

- Intel x86 serisi.
- IBM 370/168.
- Microvax II.
- PDP-11.
- Motorola 68000.

CISC Mimarisinin Avantajları şunlardır [16];

- Derleyiciler CISC komutlarını makine komutlarına dönüştürmek için daha az çaba harcarlar.
- Kod uzunluğu kısadır. Bundan dolayı bellek gereksinimi azdır.
- Kullanıcı dostudur. Bir işlem için bir komut gibi bir mantığa sahip olduğu için kullanıcıya daha az komut yazdırır.

CISC Mimarisinin Dezavantajları şunlardır [16];

- Çok fazla komut vardır bundan dolayı donanım çok büyük ve maliyetlidir.
- Komutlar birden fazla çevrimde çalışırlar.
- Boru hattı, CISC mimarisinin komutlarını çalıştırılmasını daha karmaşık hale getirir.
- Saat hızı yavaşladığından dolayı makinenin performansı düşer.

2.6. RISC VE CISC MİMARİLERİ ARASINDAKİ FARKLAR

Çizelge 2.1'de 2 mimari arasındaki farklar görülmektedir.

Çizelge 2.1 RISC ve CISC mimarileri arasındaki farklar [16].

RISC	CISC
Daha çok yazılıma odaklıdır.	Daha çok donanıma odaklıdır.
Buyruklar genelde bir saat darbesinde biter.	Bazı buyruklar birden fazla saat darbesinde çalışır.
Kod boyutu fazladır.	Kod boyutu azdır.
Aritmatik ve mantıksal işlemler sadece yazmaçlar kullanılarak yapılabilinir. Bellek buyrukları sadece yazmaçdan belleğe veya bellekten yazmaca veri kaydetme işlemini gerçekleştirir.	Aritmetik ve mantıksal işlemler hem bellek hem de kayıt işlenenlerine uygulanabilir.
RAM kullanımı daha verimlidir.	RAM kullanımı çok fazladır.
Başarılı boru hattı mimarisi kullanımı	Başarısız boru hattı mimarisi kullanımı

2.7. RISC-V KOMUT SETİ MİMARİSİ NEDİR?

RISC-V, Berkley Universitesi tarafından 2010 yılında geliştirilmiş bir komut seti mimarisidir. Son yıllarda oldukça ilgili duyulan bir komut set mimarisi olmuştur. Açık kaynaklı olduğu için ücretsiz bir biçimde işlemci tasarlama olanağı sağlar. RISC mimarisinin azaltılmış komut seti mantığına bağlı kalmaktadır [18].

2.7.1. RISC ile RISC-V Arasındaki Fark

Normalde RISC-V'nin RISC mimarisinin bir varyantı olarak düşünülür. Ama gerçekte RISC-V bir komut seti mimarisi iken RISC diğer bilgisayar mimarilerini içereyen bir kapsayıcı terim olmasıdır. RISC mimarisi azaltılmış komut seti mimarisidir. Bu tanım, bu mimari stilini yada bu tarz komut seti mimarisini kullanan bütün CPU'lar için geçerli genel bir tanımdır. Konuyu farklı bir yönden ele alırsak CISC mimarisininde RISC mimarisi gibi genel bir tanım olarak ele aldığımızda x86 komut setinin RISC-V komut seti gibi özel bir komut seti mimarisi olduğunu ve CISC mimarisininde bu yapı hakkındaki kapsayıcı bir tanım olduğunu söyleriz.

2.7.2. RISC-V Karakteristik Özellikleri

RISC-V'nin karakteristik özellikleri aşağıdaki gibi sıralanmıştır.

- Kanıtlanmış ISA'dır. RISC mimarisinin ilkelerine dayalıdır.
- Genelde tek çevrimde çalışan talimatlara sahiptir. (RV32I için böyle söylenir. M gibi başka extensionlar için bu değişebilir.)
- Load-Store mimarisi kullanır.
- Kararlı, basit, yazılım merkezci bir tasarıma sahiptir.
- Genişletilebilinir bir yapıya sahiptir. Donanımı özgürce geliştirilebilinir. Kısaca donanım özgürlüğü sunar.
- Esnek yapısı sayesinde, mikrodenetleyiciden süper bilgisayarlara kadar uygun bir yapıya sahiptir.
- 32-bit, 64-bit ve ondalıklı sayılar için uzantıları(extension) vardır.
- Çeşitli işletim sistemleri ve derleyiciler tarafından desteklenir.

- FPGA'lardan yongada sistemlere, mikrodenetleyicilerden sistem modüllerine kadar donanım desteği sağlar.
- IP core olarak diğer mimarilerden daha fazla örnek kod bulunur.

2.7.3. RISC-V'nin Değişkenleri

RISC-V mimarisinin 4 ana komut seti değişkenleri aşağıda verilmiştir [19].

- RV32I 32-bit Integer (Tam sayı) Buyruk Seti
- RV32E 32-bit Integer (Tam sayı) Buyruk Seti (Gömülü) 16-bit Register
- RV64I 64-bit Integer (Tam sayı) Buyruk Seti
- RV128I 128-bit Integer (Tam sayı) Buyruk Seti

2.7.4. RISC-V'nin Eklentileri (Extension)

RISC-V mimarisindeki eklentileridir. Bu eklentiler işlemciler için farklı fonkiyonel özelliker eklemek amacıyla vardır. İşlemcide kullanılacak zorunlu eklentiler değildir. Yapılacak işlemcinin özelliğine göre eklenir. Her eklentiler farklı bir komut seti mimarisi ve onlara özel register birimi içerebilir.

Aşağıda verilen eklentiler sıralanmıştır [19];

- M : Çarpma ve Bölme standart eklentisidir.
- A : Atomik buyruk kümesi standart eklentisidir.
- F : Float işlemleri için standart eklentidir.
- D : Double işleri için standard eklentidir.
- G : Temel ve üzeri eklentilerin kısaltımıdır.
- Q : 4 hassasiyetli ondalıklı sayılar eklentisidir.
- L : Decimal ondalıklı sayılar için eklentidir.
- C : Sıkıştırılmış buyruklar bulunan eklentidir.
- B : Bit Yönetimi (Bit Manupilation) standard eklentisidir.
- J : Dinamik çevrilen diller standard eklentidir..

- T : İşlemsel bellek standard eklentisidir.
- P : Packet SIMD buyurkları standard eklentidir..
- V : Vektör işlemleri için standard eklentidir.
- N : Kullanıcı seviyesinde kesme standard eklentidir.
- H : Hypervision için standard eklentidir.
- S : Supervisor buyrukları için standard eklentidir.

2.7.5. Adlandırma Kuralları

RISC-V gibi ISA'ları tanımlama kuralları vardır ve bu kurallar şu şekildedir [19].
RV [32, 64, 128] [I, M, A, F, D, G, Q, L, C, B, J, T, P, V, N]
Örnek olarak RV32IMC yada RV32IMF verilebilinir.

2.7.6. RISC-V Yazmaçları ve Özel Adlandırmaları

RISC-V mimarisinin temel değişkenine göre 32 adet register bulunmaktadır. Bu registerların bazıları derleyici üzerinde özel işler yapabilmektedirler. Her registerın x0, x1 gibi adları olmakta özel isim olarakda zero, t1 ve s2 gibi adlara sahip olmaktadırlar.

Aşağıda bazı özel registerların adları sıralanmaktadır;

- zero(x0) registerı sıfır registerı olarakta geçmektedir. Her zaman sıfır olarak kalması gerekmektedir.
- x1(ra) registerı işlemcide yapılan işlemler için return değerini tutmaktadır. Bazı derleyicilede yapılan geri dönme işleminin adresi bu registerda saklanır.
- x2(sp) registerı stack point address değerini tutan registerdır. Bazı özel stack'lenen değerlerin adresini tutar.
- x3(gp) genellikle genel amaçlı kullanılmaktadır. Derleyici tarafından çerçeve işaretleyicisi olarak kullanılır.
- x4-x11'e kadar olan registerlar genel amaçlı registerlardır.

Çizelge 2.2. RISC-V mimarisinin standart registerları.

XLEN-1	**0**
x0	
x1	
x2	
x3	
x4	
x5	
x6	
x7	
x8	
x9	
x10	
x11	
x12	
x13	
x14	
x15	
x16	
x17	
x18	
x19	
x20	
x21	
x22	
x23	
x24	
x25	
x26	
x27	
x28	
x29	
x30	
x31	
XLEN	
pc	
XLEN	

Genel amaç için 31 tane register bulunmaktadır. Bunlar 0-31 arasındanki registerlardır. Çizelge 2.2'de bu registerlar yukarıdaki gibi görülmektedirler.

2.7.7. Standart Buyruk Kümesi Formatı

Standard buyruk kümesi formatının R/I/S/U olmak üzere 4 tane temel formatı vardır. RV32, 64, 128 olsa bile bütün buyruklar standart 32-bit uzunluğundadır. Format üzerindeki opcode bilgisayarın hangi temel işlemi gerçekleştireceğini anlatan 7 bitlik bir ikili sayı sisteminden sayıdır. Yapılan işlemden sonra bir veri registera kaydedilecekse aşağıdaki formattaki rd registerın bulunduğu ikili sayı sistemindeki 5 bitlik register adresindeki registera kaydedilecektir. Burada rd'nin açılımı destination registerdır (Hedef yazmaç). func3, aynı opcode sahip olan işlemleri daha filtrelemek için kullanılan 3 bitlik ikili sayı sistemi ile yazılmış formatın bir bölümüdür. rs1 ve rs2 registerları source (kaynak) registerlar olarak gösterilir. 5 bitlik bir adresi temsil ederler. Herhangi bir komut sırasında rs1 ve rs2 registerlarının belirttiği adreslerdeki registerlardan verileri çekmek için kullanılır. İmm(immadiate), anlık değerlerdir. İşlemlerde memoryde ve registerlarda olmayan veya o an için üretilmek istenen değer imm bölümünde yazılır. func7, opcode ve func3 aynı olabilecek farklı buyrukları birbirinden filtrelemek için kullanılır [20].

Çizelge 2.3. Standart buyruk kümesi formatı [20].

31 25	24 20	19 15	14 12	11 7	6 0	
func7	rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]		rs1	funct3	rd	opcode	I-type
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[31:12]				rd	opcode	U-type

Standard komut formatları dışında ekstra B ve J tipi 2 tane daha fazla format vardır. Bunlar sıçrama ve dallanma buyurklarının format tipini belirler. Genel olarak bit sıraları B-tipi için S ile J ise U-tipi için aynı sıralamaya benzerdir. Aralarındaki tek fark immediate değerleri kendi aralarında farklı hizalarındadır. Hepsini tam olarak götermek gerekirse aşağıdaki Şekil 2.5'de gösterilmektedir.

Çizelge 2.4. Temel buyruk kümesi formatı ve varyantları [20].

31 30 25	24 21 20	19 15	14 12	11 8	7	6 0	
funct7	rs2	rs1	funct3	rd		opcode	R-type
		rs1	funct3	rd		opcode	I-type
imm[11:0]	rs2	rs1	funct3	imm[4:0]		opcode	S-type
imm[12] imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode	B-type
imm[31:12]				rd		opcode	U-type
imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd		opcode	J-type

2.7.8. RV32I Standart Buyruk Kümesi

Şekil 2.5'te görülen RISC-V mimarisine ait standart buyruk kümesidir.

RV32I Base Instruction Set

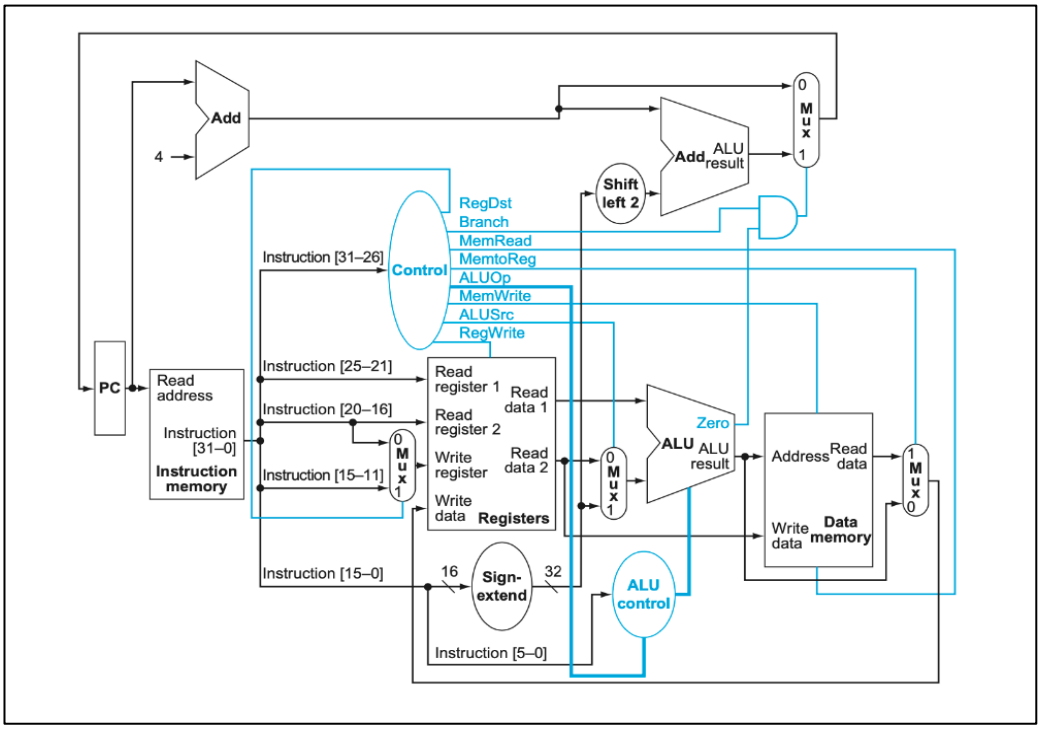
imm[31:12]			rd	0110111		LUI
imm[31:12]			rd	0010111		AUIPC
imm[20|10:1|11|19:12]			rd	1101111		JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
0000 pred	succ	00000	000	00000	0001111	FENCE
0000 0000	0000	00000	001	00000	0001111	FENCE.I
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK
csr		rs1	001	rd	1110011	CSRRW
csr		rs1	010	rd	1110011	CSRRS
csr		rs1	011	rd	1110011	CSRRC
csr		zimm	101	rd	1110011	CSRRWI
csr		zimm	110	rd	1110011	CSRRSI
csr		zimm	111	rd	1110011	CSRRCI

Şekil 2.5. RV32I standart buyruk kümesi [20].

2.8. MİKRO MİMARİ TÜRLERİ

2.8.1. Tek Çevrim İşlemci (Single Cycle Processor)

Adından anlaşıldığı gibi tek çevrim işlemciler bir komutu tek çevrimde tamamlar. Şekil 2.9'de görülen tek çevrim işlemci örneğinin veriyolunda Program Counter (PC), Registerfile, ALU, Data Memory, Control Unit ve Instruction Memory bulunur. Bu yapılar işlemcinin mikro mimarisinin temellerini oluştururlar. Bu mimaride bütün bir buyruğun işlenmesi pozitif kenar tetiklemede gerçekletirilirken Registerlar'a yada Data Memory'ye veri kaydedilirken negative kenar tetikleme ile veriler kaydedilir. Bir sonraki pozitif kenar tetiklemesinde sıradaki buyruk registerın içindeki kayıt edilmiş buyruğu çekebilir. Bu işlem yapılmazsa bir önceki buyruğun register'a kaydettiği işlenmiş veri şuanki buyrukta çekilemez. Tek çevrim işlemciler diğer mimarilere göre period süresi daha uzundur. Buyruk işlemlerini tek çevrimde gerçekleştirseler bile veri yolu uzunluğundan dolayı period süresi en uzun süren buyruğun süresine göre ayarlanmalıdır.



Şekil 2.6. Tek çevrim işlemci veriyolu [21].

2.8.1.1. Program Sayacı

Program Sayacı, bir işlemcinin Fetch (Getir) aşamasında buyrukların adreslerini çeken bir sayıcıdır. Buyrukların adreslerini tutar ve her pozitif çevrim darbesinde 1 arttırılır. Bu sayede Instruction Memory'deki (Komut Hafıza'daki) buyruklar sırasıyla çekilmiş olur. Özel durum olarak branch (dallanma) komutları kullanıldığında branch unit içinde belirlenen ve decode edilmiş anlık değer (immadiate) pc counter ile toplanarak bir sonraki buyruk adresini belirler.

2.8.1.2. Kontrol Ünitesi

Kontrol Ünitesi, işlemci çekirdeğinin bir parçasıdır. John von Neumann tarafından tarafından Von Neumann mimarisine dahil edildi. ALU (Aritmatik Mantık Ünitesi), giriş çıkış modülleri, Data Memory ve RegisterFile gibi işlemci elemanlarını kontrol etmek amacıyla kullanılır. Getir yoluyla hafızadan çekilen buyruklar kontrol ünitesinde filtrelenir ve dışardaki modülleri kontrol etmek için kontrol sinyalleri üretir. Kontrol ünitesi sabit değildir. Üreticinin kullanımına bağlı olarak istenildiği gibi ayarlanır. Kontrol ünitesine istenilen buyruklar eklenip çıkarılabilinir. Daha çok hangi göreve hizmet edeceğine bağlıdır.

2.8.1.3. Register File

Yazmaç Dosyası (Register File), registerların bulunduğu modüldür. RISC-V işlemcisine göre işlemci tasarlanacaksa 32 tane 32-bit register bulunur. Registerlar işlemcinin elleri olarak yada kısa süreli işlem yapılması için bekletilen verilerin depolandığı hafıza birimi olarak söylenebilir.

2.8.1.4. Anlık Değer Üreticisi

Sistem I-type gibi yada buyruğunda imm bölümü varsa bu birim kullanılır. Anlık değer üreticisi gelen buyruklardaki imm yerlerini decode ederek 32 bite genişletir. rs2 registerının çıkışı mux ile iptal edilir ve anlık değer üretisinden gelen veri işleme sokulur.

2.8.1.5. Aritmetik Mantık Birimi (ALU)

Aritmetik Mantık Birimi yani ALU işlemcideki aritmetik ve mantıksal işlemleri yapan birimdir. Kontrol Ünitesinden aritmetik Mantık Unitesine giden sinyal ile hangi işlem yapılacağı seçilir. 2 farklı datayı gelen buyruğa göre artimatik veya mantıksal işleme sokar ve dışarı yollar. Giden sinyal ya register file'a geri yazılması için yollanır. Yada hafızaya veri yazılması için hafızanın adres pinine bağlanır.

2.8.1.6. Dallanma Ünitesi

Dallanma Ünitesinin yapısı Aritmetik Mantık Ünitesinin yapısına çok benzer. Karşılaştırılma işlemleri yapılır bu ünitede eğer karşılaştırmalar doğru ise 1 bitlik dışarı sinyal verilir ve pc counter dallanma buyruğunun anlık değer üreticisinde belirlediği yeni adrese dallanır. B-type buyurklar bu birimi kullanırlar.

2.8.2. Boru Hattı Mimarisi

Boru hattı mimarisi işlemciyi hızlandırmak amacıyla kullanılan bir mimari türüdür. İşlemcinin bazı aşamalarının arasına registerlar koyarak bir işlemi birden fazla clock darbesinde yapılmasını sağlarlar. Amacı saat sıklığını arttırmaktır yani period süresini kısaltmaktadır. Bu sayede işlemcinin frekansı artmış olur. Boru hattı mimarisi genel olarak diğer mimarilere göre karışık bir mimari türüdür. Boru hattı mimarisi genel olarak birden fazla aşamadan oluşur. 5 aşamalı bir boru hattı mimarisinin aşamaları aşağıdaki gibi isimlendirilir;

- Getir (Fetch).
- Çöz (Decode).
- Yürüt (Execute).
- Bellek Erişimi (Memory Access).
- Geri Yaz (Write Back).

2.8.2.1. Getir (Fetch)

Getir aşaması buyrukların hafızadan çekildiği aşamadır. Program sayacı bulunur. Bu aşamada amaç buyrukların hafıza biriminden çekilmesidir. Program sayacı saymaya başlar ve her arttırıldığında, program sayacındaki arttırılan sayı, çekilecek buyruğun adresi olacaktır. ve o adresteki buyruk çekilir. Çekilen buyruk, program sayacıdaki değer ve sayacın 4 arttırılmış hali bir sonraki registera yazılır.

2.8.2.2. Çöz (Decode)

Çöz aşaması çekilen buyruğun çözüldüğü ve çözülen kısımların anlık değer üreticisi ve kontrol ünitesine gittiği yerdir. Bu bölümde kontrol ünitesi, register file ve anlık değer üreticisi bulunur. Bu alanda daha çok buyruk çözülür. 32bit olan buyruğun bitlerine ayırarak çözülür ve istenilen bitler istenilen noktalara ulaşır.

2.8.2.3. Yürüt (Execute)

Çözülen talimatın yürütülmesi. Bu aşama, aritmetik işlemleri, mantıksal işlemleri veya diğer işlemleri içerir. ALU, Çarpma Birimi, Bölme Birimi ve Dallanma bölümleri bulunur. Çözülen buyruğun istenilen talimata getirldiği yerdir.

2.8.2.4. Bellek Erişimi (Memory Access)

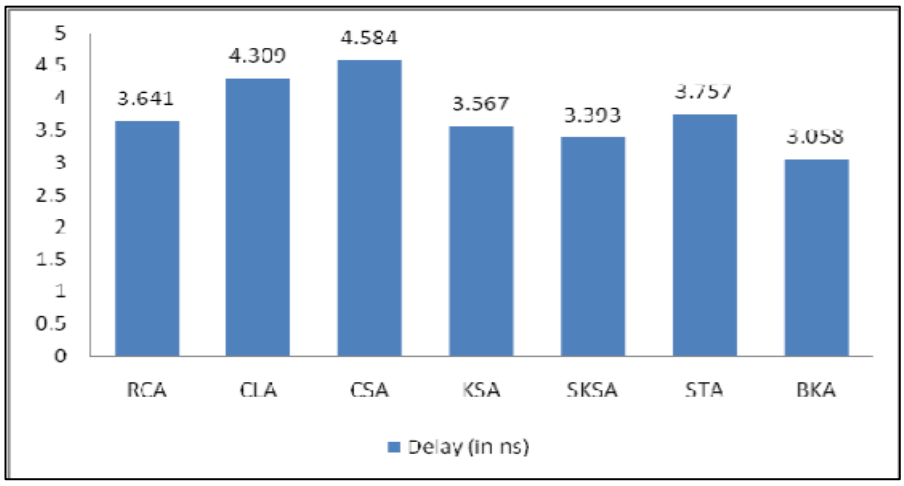
Talimatın gerektirdiği bellek erişimleri gerçekleştirilir. Bu aşama, bellekten veri okuma veya belleğe veri yazma işlemlerini içerebilir. Veri önbelleği ile iletişim kuran birimdir.

2.8.2.5. Geri Yaz (Write Back)

Bu aşamada yürütülmüş talimatlar, hafızadan çekilen veriler yada o anki program counter verisi registerlara yazılmak için yollanır.

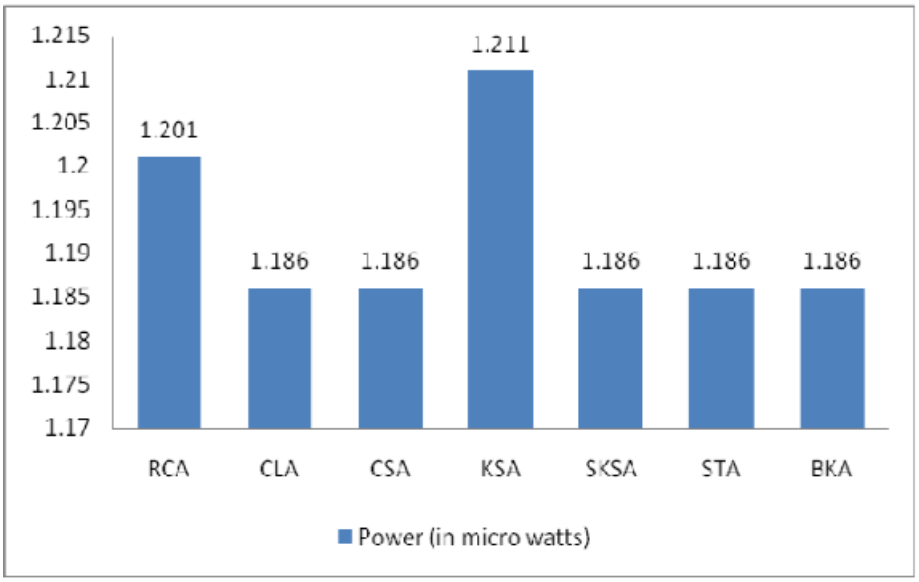
BÖLÜM 3
LİTERATÜR TARAMASI

Harish vd. yaptıkları çalışmada VSLI teknolojisinde kullanılan mimari çeşitlerini incelemiştir ve çeşitli adder topolojilerini karşılaştırmışlardır. Bir çok toplama algoritmalarının performans grafiği aşağıdaki gibi sıralanmıştır [21].



Şekil 3.1. Çeşitli toplama algoritmalarının gecikmeleri [21].

Şekil 3.2. Toplama algoritmalarının harcadıkları alan [21].



Şekil 3.3. Toplama algoritmalarının harcadığı güç [21].

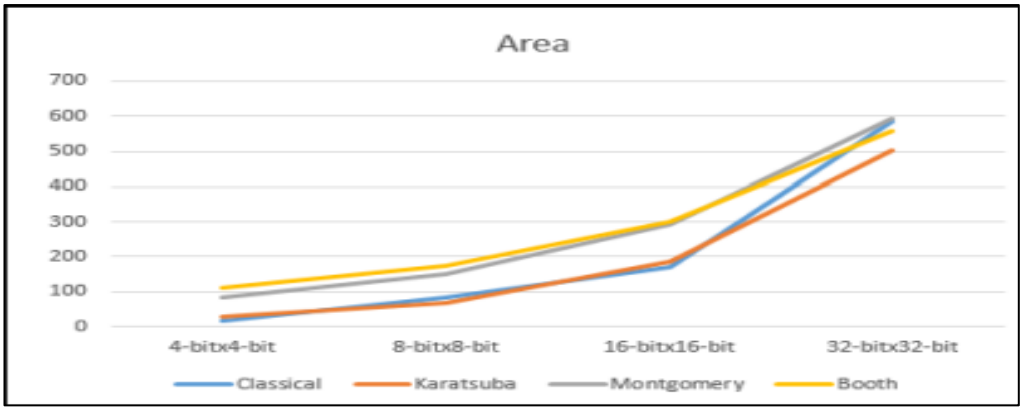
S.No	Adder Name (16 bit)	Xilinx ISE 2014.7 Tool delay (in ns)	Power in Xilinx ISE 2014.7 (in micro watts)	Device utilization (LUTs, IOBs) (69120, 640)
1	Ripple carry adder	3.641	1.201	24,50
2	Carry lookahead adder	4.309	1.186	24,50
3	Carry skip adder	4.584	1.186	29,50
4	Kogge stone adder	3.567	1.211	48,50
5	Sparse kogge stone adder	3.393	1.186	44,50
6	Spanning tree adder	3.757	1.186	35,50
7	Brent kung adder	3.058	1.186	32,50

Şekil 3.4. Toplama algortimalarının karşılaştırılması [21].

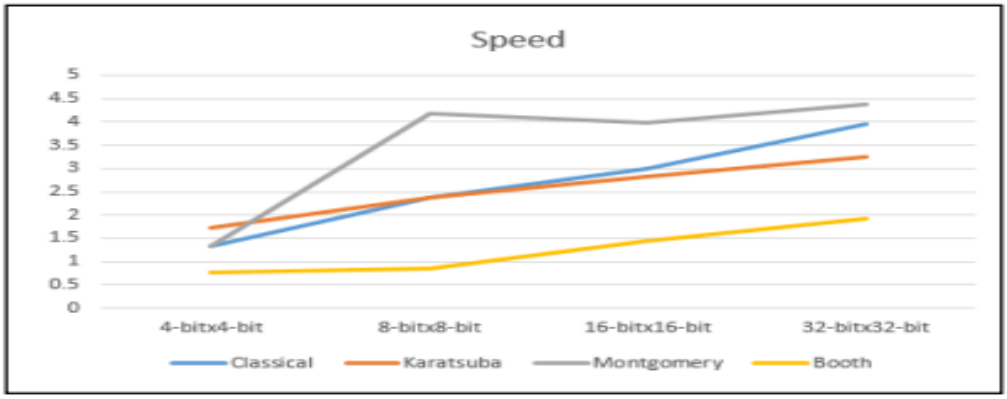
Bu makalenin sonuçlarına göre CSA'nın diğer toplayıcılara göre daha fazla gecikmesi olduğu tespit edilmiştir. RCA'nın öteki toplayıcılara göre gecikmesinin en az olduğu tesbit edilmiştir. BKA, diğer paralel toplayıcılar arasında en iyi gecikmeye sahiptir.

Sentez sonuçlarına bakıldığında dört paralel toplayıcıdan BKA'nın gecikmesi diğerlerinden daha iyi olduğu saptanmıştır [21].

Wibowo, yaptığı çalışmada çarpma işlemlerinin maliyetli ve yüksek enerji tüketen devreler olduğunu söylemiştir ve dijital uygulamalar için daha verimli çarpma algoritmalarını karşılaştırmak amacıyla 4 farklı algortima seçmiştir. Genel olarak bu makalede 2 farklı sayının FPGA ortamında çarpımı kıyaslanmıştır. Makale sonucuna göre en verimli alan Karatsuba algoritması iken en hızlı algoritma Booth algoritması olmuştur. [22].



Şekil 3.5. Çarpma algoritmalarının alanlarının kıyaslanması [22].



Şekil 3.6. Çarpma algoritmalarının hızının kıyaslanması [22].

Raveendran vd. bu makalelerinde RISC-V komut seti mimarisine uyumlu bir işlemci tasarımı gerçekleştirmiştir. Sistemi SystemVerilog ile yazılmıştır. Makelelerinde analiz sonuçları hem FPGA hemde ASIC ortamda gerçekleştirilmiştir [23].

Çizelge 3.1. FPGA üzerinde kaynak kullanım tablosu [23].

Parameter	**Proposed Core (with FPU)**	**Open RISC [17] (without FPU)**	**Rocket Core [18] (with Limited FPU ops)**
Slice Registers	18340	2280	12388
Slice LUTs	46530	6744	46256
LUT-FF pair	11759	7063	5425
DSP48E1s	32	4	22

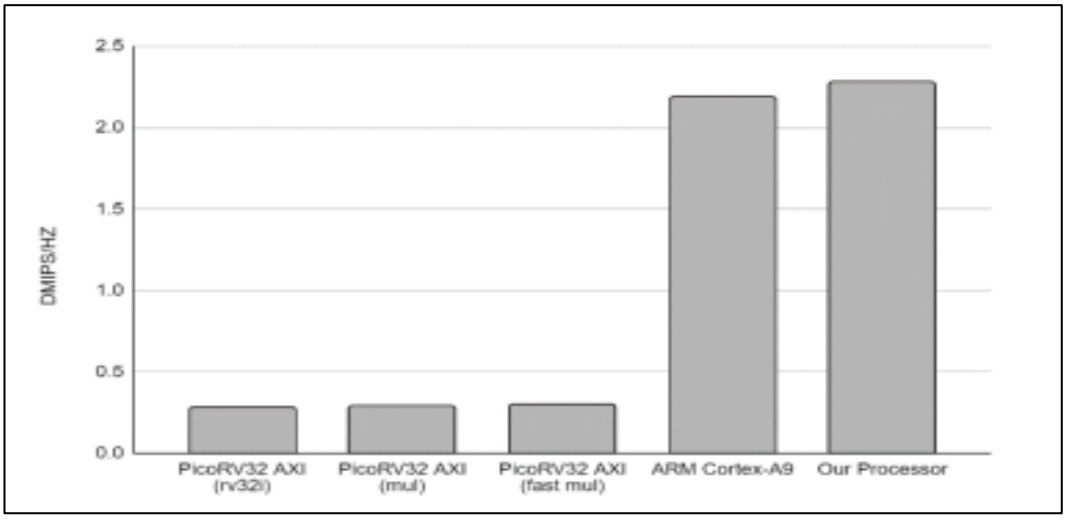
Çizelge 3.2. Çekirdeğin sentez sonucu [23].

Parameter/Technology Node		**65 nm**		**130 nm**	
		LL @ 500MHz	SP @ 500MHz	LL @ 500MHz	SP@ 500MHz
Logic Cells	Proposed (with FPU)	173 K	171 K	1424 K	1287 K
	Rocket Core [18] (with Limited FPU ops)	121 K	121 K	1328 K	1178 K
Area	Proposed	561 mm^2	557 mm^2	2539 mm^2	2408 mm^2
	Rocket Core[18] (with Limited FPU ops)	451 mm^2	465 mm^2	2118 mm^2	1824 mm^2

Sonuç olarak bu makaleye göre RISC-V mimarisine uygun, öngörücüye sahip 5 aşamalı 32bit işlemci tasarımı gerçeklendirmişlerdir. Ayrıca IEEE 754-2008 standartlarına sahip FPU sahiptir. Çekirdek Xilinx Virtex FPGA ve ASIC ortamda uygulanmıştır [23].

Birari vd. bu makalelerinde RISC-V mimarisine uygun yüksek performansı bir işlemci tasarlanmıştır. Tasarlanan çekirdek 32bit olmaktadır. 8 KB buyruk önbelleği ve veri önbelleği içermekte olup 5 aşamalı boru hattı mimarisine sahiptir. Yapılan çalışmada sanal bellek sistemini desteklemektedir. Atomic buyruklar eklenmiştir. Ondaklıklı sayılar için FPU vardır. CoreMark benchmark testi kullanılarak 3.32'lik değer elde edilmiştir. Tasarım Xilinx Virtex-7'de test edilmekte olup 60MHz saat frekansına sahiptir [24].

Poli vd. bu makalelerinde yeni başlayanlar için öğrenmesi kolay küçük FPGA'larda uygulanabilinecek bir RISC-V işlemci oluşturmayı hedeflemektedir. İşlemcilerinde gelişmiş işlemcilerde bulunan özellikler eklenmemiştir. Gerekli olmayan kavramların uygulaması en aza indirilmiştir. BASY-3 Artix-7 FPGA ile test edilmiştir. Dhrystone core testinde 2.276 DMIPS/MHz değerine ulaşılmıştır [25].



Şekil 3.7. Dhyrstone benchmark performansı [25].

Bora ve Paily, bu makalelerinde bölme işleminin işlemci üzerindeki geçikmesini azaltma hedefiyle ikili bölme yöntemi önermektedir. Donanım tasarımı olarak Verilog programlama dili kullanmıştır. Xilinx FPGA kartı üzerinde test etmiştir. 32-bitlik radix sabit nokta bölücü devresi 2,82 µW/MHz dinamik güç tüketimine sahiptir. Yol geçikmesi 4,97 ns olan 2-9 arası clock'a sahiptir. FPU için tasarlanmış olan bölücü devresi ise 2,76 µW/MHz güç tüketimi ve 4,83 ns yol geçikmesine sahiptir. [26].

Saussereau vd. bu makalelerinde RISC-V işlemcilerde boru hattı mimarisine özelleştirilebilir miktarda hat ekleyerek hem FPGA hemde ASIC tasarımda diğerleri arasındaki farkları bulmayı amaçlamıştır. Boru hattı mimarisi olmayan sistemler ile kıyasmalayı amaçlamıştır. 4 aşamalı boru hattı mimarisi ASIC tasarımında kaynak kullanı az miktar artmışken net bir performans artışı görülmüştür. FPGA sonuçları ASIC sonuçlarına göre daha çeşitli sonuçlar vermiştir. 6 aşamalı boru hattı teknolojisinde ise sistem karmaşıklığı artmasına rağmen performas avantajı sağlamamış. Boru hattı mimarisinin nihai bir çözüm olmadığından ve boru hattı

olmayan mimarilerin kaynak verimliliğin öncelik verdiklerinden diğer sistemlere göre daha ilgi çekici olduklarını dile getirmiştir [27].

Jose ve Shankar, bu makalelerinde RISC-V çekirdeği yada RISC-V çekirdeklerinin bağlı olduğu SoC modellerini geliştirmişlerdir. Saat topoloji, özel talimatlar, önbellek gibi sistemleri gibi RISC-V modüllerini iyileştirmişlerdir. Sonra SoC tasarımlarını 4 kat performans arttıracak şekilde güncellemişlerdir. [28]

Saif vd. bu makalelerinde öğrencilerin gömülü sistemin nasıl çalıştığını anlayabilmek için ve FPGA üzerinde RISC-V mimarisine dayalı bir eğitim eğitim işlemcisi tasarlamışlardır [29].

He ve Ko, bu makalelerinde, dinamik yürütme özelliğine sahip 2 yollu superscalar RISC-V işlemci tasarımı sunar. Tasarım, dallanma öngörüsü için Gshare şemasını ve out-of-order için Tomasulo Algoritmasını kullanır. Makalenin sunduğu tasarım, performans açısından farklı kıyas noktaları ile karşılaştırılır. İşlemci tasarımı, döngü başına ortalama buyruk sayısında %21,4 ve belirli bir performans oranında %9,35'lik bir iyileştirmeye ulaşır [30].

Ata ve Özkök, bu makalesinde, RISC-V mimarisine sahip işlemcileri programlamamızı sağlayan sistemleri ve bu RISC-V mimarisinin yazılım ekosistemini literatür olarak hemde uygulamalı olarak değerlendirmişlerdir. Makalede önyükleyicilerden derleyicilere, linux işletim sisteminin derlenmesinden, simülatörlere kadar birçok olay hakkında incelemeler yapmışlardır. Sonuç olarak RISC-V mimarisinin herhangi bir ticari bir ürün kullanılmadan bu mimarinin sorunsuz bir şekilde kullanılabilinecek kadar olgunlukta olduklarına karar vermişlerdir [31].

Tozlu ve Yılmaz, bu lisans bitirme tezlerinde RV32IM eklentilerine sahip, kesme ve UART gibi çevre birimleri olan RISC-V işlemcisi tasarlamışlardır. Çekirdeğin her aşamasını raporlamışlardır [32].
Öztekin, bu yüksek lisans tezinde, imkanı olmayan küçük okullarda bilgisayar mimarisi dersinde uygulama yapmanın zor olmasından ve işlemci tasarımı gibi konularda çalışmak isteyen öğrenciler için yol gösterici bir çalışma yapmıştır.

Çalışmada Multimedia Logic programında bir tür sanal bilgisayar tasarlanmıştır. Visual Studio .Net ortamında bir assembler programı programlanmış ve bu assembler programında yazılan program makine diline çevrilebilmektedir. Yapılan işlemci, kullanıcının işlemcide yapılan işlerin adım adım nasıl çalıştığını göstermektedir [33].

Ramezani, bu bitirme projesinde çarpma ve toplama işlemleri için tasarlanmış algoritmaları incelemiştir. Bu işlemeleri yaparken VHDL donanım tasarım dilini kullanmıştır. Program olarakta Xilinx ISE 11 programını kullanmıştır. Toplamda 5 farklı toplama devresini ve 2 farklı çarpma devresini kıyaslamıştır. Sonuçları aşağıdaki tablolarda sunulmuştur [34].

Çizelge 3.3. Toplayıcı devrelerinin karşılaştırması [34].

Toplayıcı	**Gecikme (ns)**	**Alan (CLB)**
Ripple Carry Adder	36.700	37
Carry Look Ahead Adder	28.043	48
Conditional Sum Adder	18.730	82
Carry Select Adder	23.381	64

Çizelge 3.4. Çarpma devrelerinin karşılaştırılması [34].

Çarpıcı (16-bit)	**Gecikme (ns)**	**Alan (CLB)**
Ardışıl Çarpıcı	31.421	46
Dizin Çarpıcı	27.672	65

Baysal, bu yüksek lisans tezinde kripto işlemleri için RTL düzeyde çarpma devresi tasarlamıştır. Tez aşamasında look-up tabloları kullanılmıştır. FPGA kartı olarak Virtex 5 FPGA kartı kullanılmıştır. Sonuç olarak kısmı parçalara ayrılarak yapılan çarpma işleminin 16 döngü sonrası bittiği görülmüştür [35].

Kumari vd. bu makalelerinde bellek tasarlamışlar ve hatalı modeler için test etmişlerdir. Bellek tasarımı ChipScope pro kullanılarak simülasyonu yapmışlardır. FPGA olarak ise Spartan 3E FPGA kullanmışlardır. Makalenin sonucunda ModelSim kullanılarak simülasyonu yapılmıştır. Hataları ve başarısızlık durumlarını test etmişlerdir [36].

Omran ve Amory, bu makalelerinde önbellek tasarlamak için VHDL kullanmışlardır. Önbelleği en iyi şekilde tasarlayıp işlemci performansını ve önbellek mimarisini geliştirmeyi amaçlamışlardır [37].

Awedh ve Mueen, bu makalelerinde FPGA'da bulunan Microprogrammed Controller denetleyicisi ile UART haberleşmesi yapmışlardır. Sistemi test edebilmek için Basy2 ve Spartan3E FPGA kartları kullanmışlardır. 2 tane, farklı ortalama 60MB olan dosyayı 2 farklı kart aracılığı ile birbirlerine aktarılmıştır. Karşlılaştırma yazılımları ile bir problem oluşmamıştır [38].

Deng vd. bu makalelerinde Xilinx FPGA ve VHDL kullnarak PWM oluşturmayı denemişlerdir. FPGA'ların daha verimli çözümler sunduğunu açıklamaktadır [39].

Gazziro vd. yaptıkları çalışmada Baby8 adı verdikleri CISC mimarisine sahip bir softcore işlemci tasarlamışlardır. FPGA üzerinde çalışasına uygun programlanmıştır. Tasarlanan işlemci diğer softcore işlemciler ile karşıtırılmıştır. Sonuç olarak tasarlanan işlemcinin 57 Mhz frekansa sahip olduğu ve 2 mW'lik güç tüketimine sahip olduğu görülmüştür. [40].

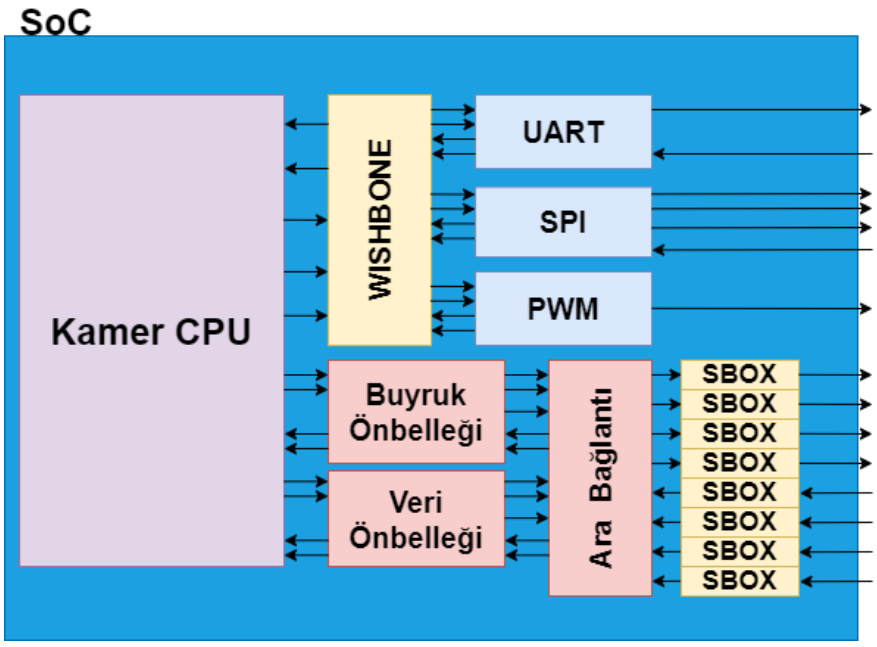
Chang vd. bu çalışmalarında 5 aşamalı boru hattı mimarisine ve RV32IM eklentilerine sahip bir işlemci tasarımı yapılmıştır. Tasarlanan çekirdek farklı olaylar için farklı düzenlebilir modüller bulunur. 2 farklı moda sahip olan tasarlanılmış çekirdek düşük güç modunda sadece integer işlemler yaparken yüksek güç modunda çarpma ve bölme işlemleri yapabilmektedir. İşlemci çekirdeği user ve supervisor gibi 2 farklı ayrıcalık modu ile donatılmıştır. Tasarlanan işlemci modülleri tamamiyle verilator ve gtkwave gibi açık kaynaklı simülasyon araçları ile tasarlanmıştır. Tasarlanan işlemcinin benchark skorları, Coremark skoru olarak 3.51 CoreMark/MHz, Dhrystone skoru olarak ise 1.48 DMIPS/Mhz olarak bulunmuştur. Tasarımları FPGA ile doğrulanmıştır. Sonuç olarak Cortex M3'ün performansını aşmaktadır [41].
Toker , bu makalesinde Vivado ile HLS aracı kullanarak RISC-V RV32I çekirdeği tasarımı sunmuştur. Önerilen tasarım 3 farklı seviyeden oluşmaktadır. İlki, C++ programlama dili ile yapılan bir HLS tasarımı ve simülasyonudur. İkincisi, HLS ile

oluşturulan Verilog CPU çekirdeğinin buyruk kümesi ile simülasyon yapılması. Üçüncü ve son seviye ise 100Mhz saat hızına sahip bir FPGA kartı ile gerçekleştirilmiştir [42].

BÖLÜM 4

MATERYAL VE METOT

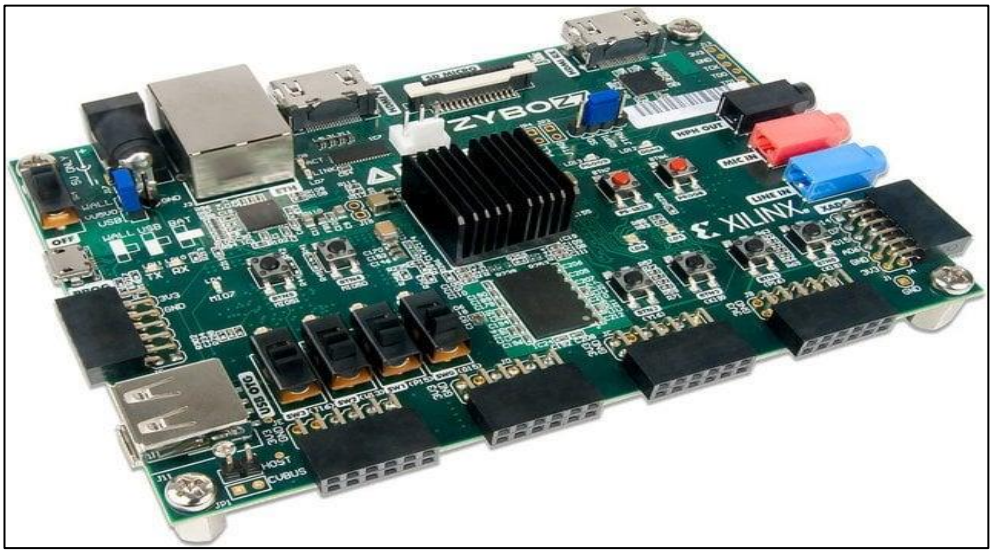
RISC-V mimarisi gömülü sistemler, otomasyon ve askeri projeler gibi bir çok alanda bize kolaylık sağlayacak bir mimaridir. CISC mimarisine göre daha hızlı ve lisans ücreti bulunmaması sayesinde işlemci üretimi artık bazı kurumsal firmalardan daha çok akademik ortamda geliştirilmesine olanak sağlamıştır. RISC-V sayesinde işlemci gelişimi dünya üzerinde daha hızlı bir şekilde artacaktır. Bu tez projesinde, RISC-V mimarisinde RV32IM eklentilerine sahip, UART, SPI ve PWM gibi çevre birimleri bulunan 4 aşamalı boru hattı mimari ile tasarlanmış bir işlemci yapısı sunulacaktır. CPU ile çevre birimleri arasında Wishbone Bus ile haberleşmesi sağlanmıştır. Buyruk ve veri önbelleği adında 2 farklı önbellek bulunmaktadır. Buyruk ve veri önbelleği ana memory ile haberleşmesi sağlanılacak ve arada S-box bulunacaktır. Bütün verilerin S-Box'da şifrelenmesi sağlanmıştır. Şekil 4.1'de tasarlanan işlemci şeması görülmektedir.



Şekil 4.1. KamerSoC yapısı.

4.1. MATERYAL

Bu projede tasarlanan işlemci, FPGA ortamında çalıştırılması için modeli Xilinx Zybo Z7-20 olan bir FPGA kartı kullanılmıştır. Zybo Z7-20 FPGA kartı, Zynq-7000 türünde bir FPGA kartıdır. Birçok farklı özelliğe sahip, gömülü sistem yazılımları ve dijital tasarım gibi konularda fayda sağlayan geliştirme kartıdır. Şekil 4.2'de kullanılan FPGA kartı görülmektedir. Özellikleride Çizelge 4.1'de yazmaktadır.

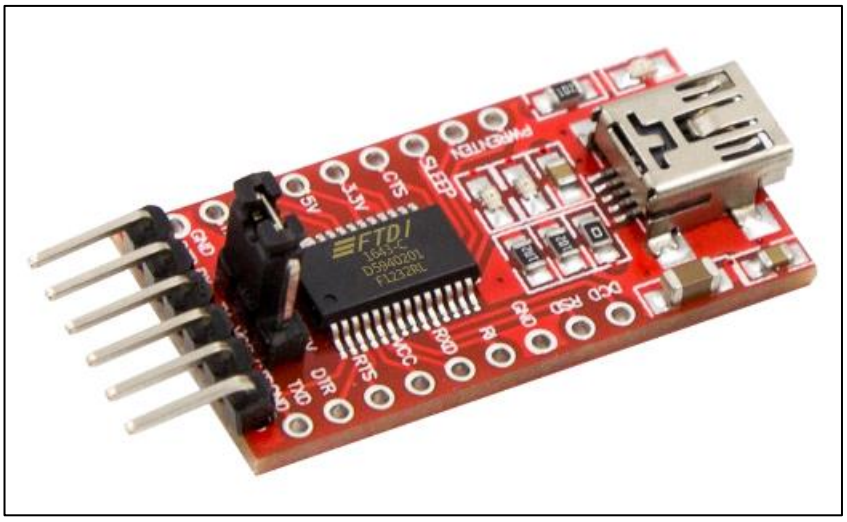


Şekil 4.2. Zybo Z7-20 FPGA kartı.

Çizelge 4.1. Zybo Z7-20 FPGA özellikleri.

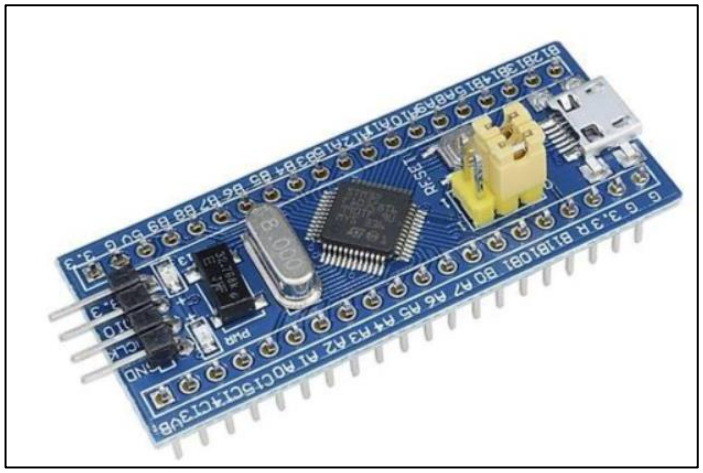
XC7Z020-1CLG400C	**Miktar**
Logic slices	13.300
6 input LUTS	53.200
Flip-Flops	106.400
Block Ram	630KB
DSP Slices	220
Harici clock	125MHZ

UART testi için USB TTL UART dönüştürücü modülü olan FT232R kullanışmıştır. Kartın 3.3V desteklemesinden dolayı tercih edilmiştir. Bilgisayar ve USB iletişimi sağlayabilmesi ve 3.3V ile 5V arasında geçiş yapabilmesinden dolayı tercih edilmiştir.



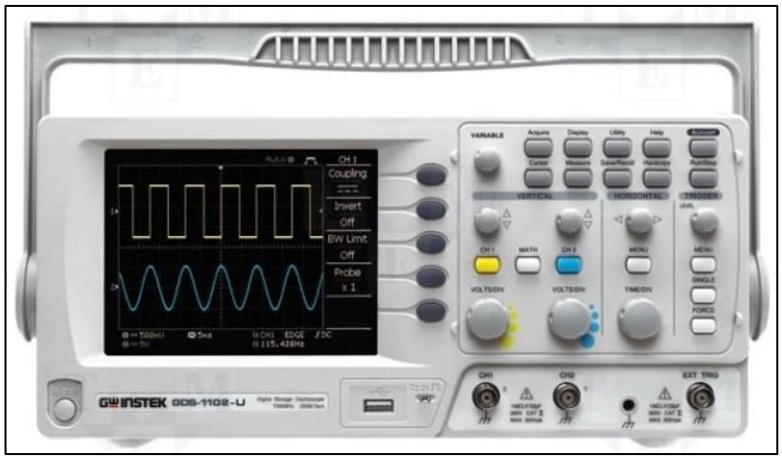
Şekil 4.3. FT232R UART modülü.

SPI testi için ARM Cortex M3 tabanlı STM32F103C8T6 mikrodenetleyici geliştirme kartı kullanılmıştır.



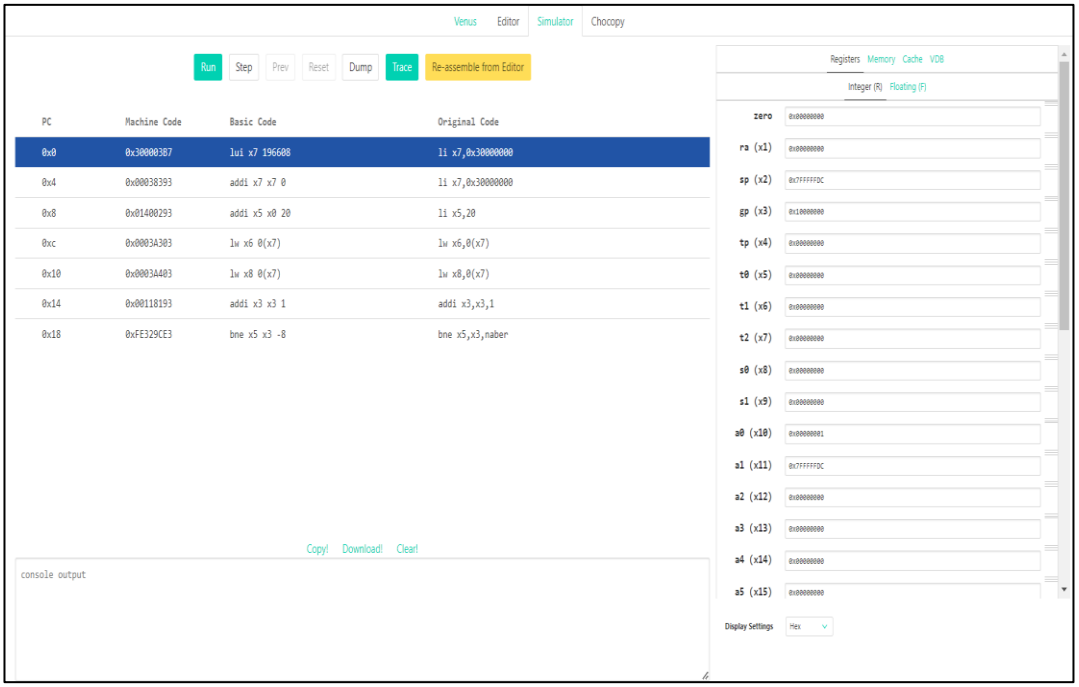
Şekil 4.4. STM32F103C8T6 geliştirme kartı.

PWM sinyallerinin testini yapmak için GWINSTEK osiloskop kullanılmıştır.



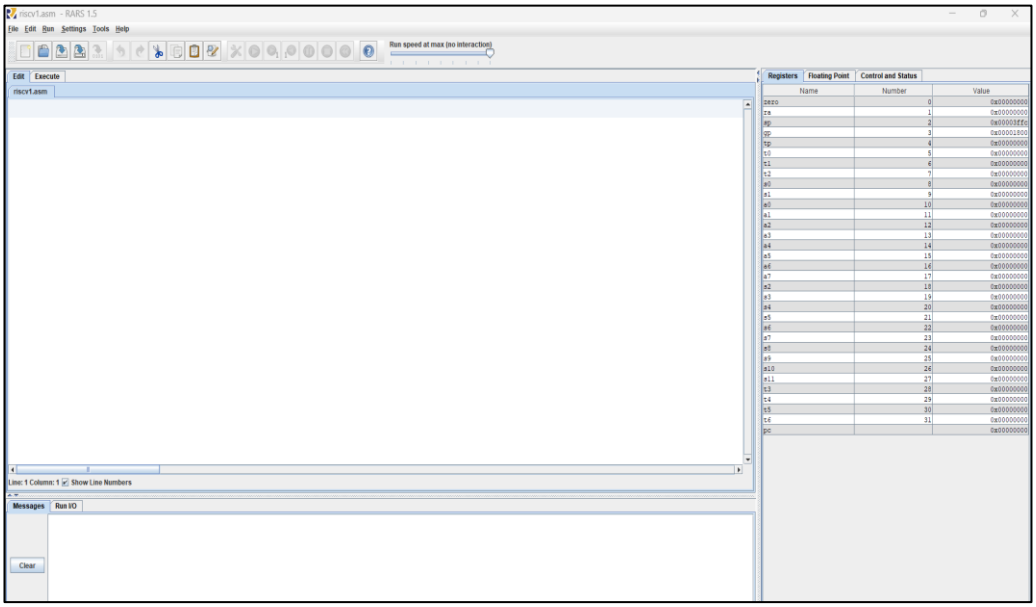
Şekil 4.5. GWINSTEK GDS-1102-U osiloskop.

Venus, RISC-V buyruk seti simülasyon programıdır. Bu program browser ortamında online bir şekilde yazılan buyrukların sonuçlarını registera kayıt edebilir ve istenilen adrese gidip gitmediğini test edebilir. Tasarlanan işlemci bu ortamlarda test kodları ile işlemcinin doğru işi yapıp yapmadığını anlamak için bu program kullanılmıştır. Basit arayüze sahiptir register kısımları net bir şekilde görünür. RV32IM buyruklarına kadar desteklenir. Başlangıç için hızlı ve ideal bir programdır.



Şekil 4.6. Venus simülasyon programı

RARS, RV32IMFDN eklentilerini destekleyen bir simülasyon programıdır. Desteklenen buyrukların ne işe yaradıklarını ve nasıl kullanıldıklarını gösterir. Venus'e benzer özellikleri vardır. Venusten daha gelişmiştir. Bazı CSR türlerini destekler. Memory yapısının adreslerini belirli seçeneklerle değiştirmemize izin verir. Veri önbelleği simülasyonu, klavye, buyruk sayacı gibi özellikleride mevcuttur. Yazılan programı hex koduna dönüştürüp kendi işlemcimizin içinde testimizi yapabiliriz. Şekil 4.7'de RARS program görülmektedir.



Şekil 4.7. RARS

İşlemci tasarımında Verilog programlama dili tercih edilmiştir. Donanım modellemelerini daha iyi kavrayabilmek, daha az kod gereksinimi, diğer dillere göre daha fazla donanıma uygun bir programlama dili olması, kısa ve öz olması gibi durumlar bu tercihin sebepleri arasındadır.

Tasarlanan işlemcide C gibi programlama dillerinin de kullanması ve RISC-V ile alakalı bütün toolların indirilmesi için RISC-V GNU Toolchain indirilmiş ve kurulmuştur. Bu sayede işlemciyi assembly seviyesinde programlamak yerine gerektiği zaman C gibi orta seviyeli dil ile de programlamak mümkün olmaktadır.

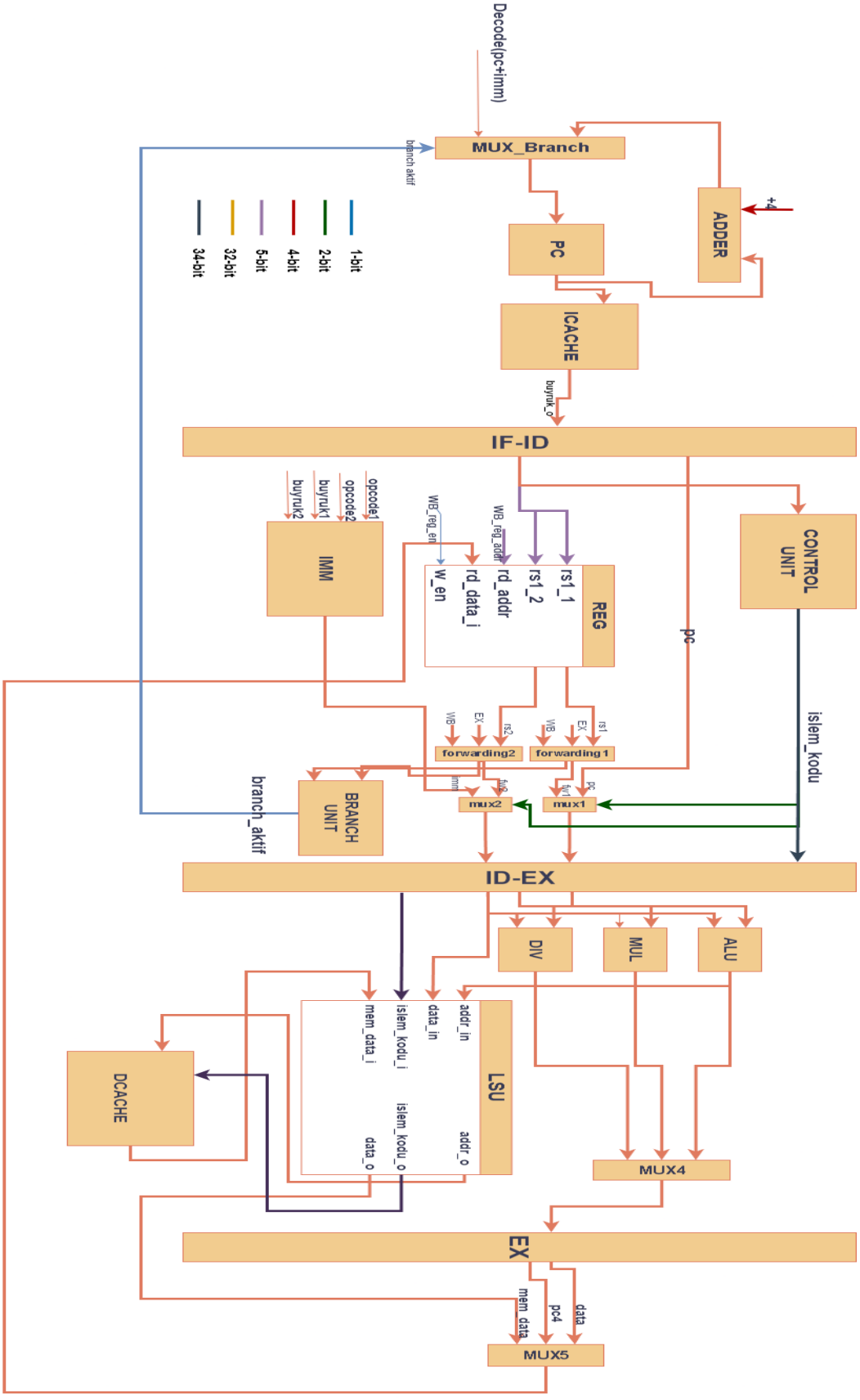
İşlemcimizi test ederken kullanacağımız derleyici ve bazı SPIKE gibi simülasyon programlarının Linux ortamında çalışabilmesi için. WSL'den Ubuntu linux kurulmuştur.

RISCOF, Ubuntu ortamına kurulmuştur. RISCOF bir çekirdek test programıdır. Hazır halde bulunan paketlerini kullanarak hedef gösterilen işlemcide bu paketleri dener. Aynı paketleri referans bir işlemci modelinde test ederek 2 modelin sonuçlarını kıyaslar. RISCOF'dan kısaca bahsedecek olursak bir çekirdek doğrulama frameworkudur.

4.2. METOT

4.2.1. İşlemcinin Çekirdek Tasarımı

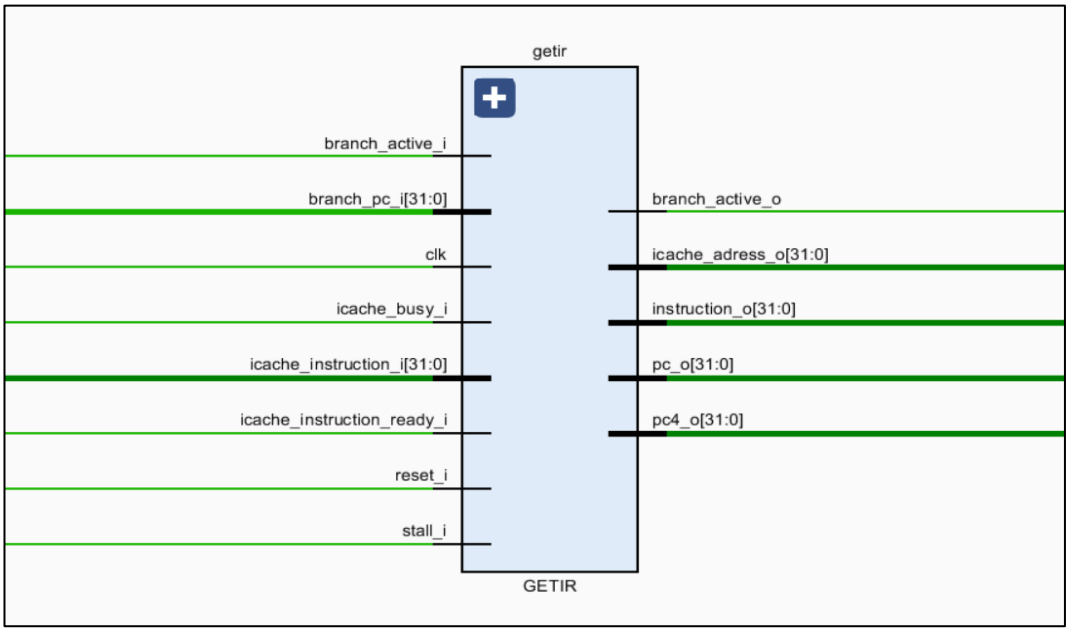
İşlemci çekirdeği 4.8'deki şekilde görüldüğü gibi 4 aşamalı boru hattı mimarisine göre tasarlanmıştır. Fetch (Getir), Decode (Çöz), Execute (Yürüt) ve Write Back (Geri Yaz) olmak üzere 4 tanedir. RV32IM eklentisine sahiptir. Çekirdek 32-bit'tir. RV32IM içersindeki 32 rakamı işlemcinin bitini temsil etmektedir. Tasarlanan işlemcide CSR (Control Status Register) bulunmamaktadır. CSR bulunmadığı için CSR'leri etkileyen komutlar haricinde bütün RV32I komutlarını çalıştırabilmektedir. M eklentisi bulunduğu için çarpma ve bölme işlemi yapabilmektedir. Çarpma algoritması için Unisgned Binary Multiplication algoritması kullanılmıştır. Bölme algoritması için ise Non-Restoring Division algoritması kullanılmıştır. Çekirdeğin bütün yapısı, Verilog programlama dili programlanmıştır. Doğrulama yapılırken Venus online derleyiciden buyruklar yazılarak test edilmiştir. Çekirdeği daha ayrıntılı tanımlamak için bütün aşamalar tek tek farklı başlıklar halinde anlatılacaktır.



Şekil 4.8. İşlemci çekirdeğinin veri yolu.

4.2.1.1. Fetch.

Getir aşaması işlemcinin buyruk çekmesini sağlayan bölümdür. Bu aşamada önbelleğe, çekilmesi gereken buyruğun adresi yollanır ve istenilen adresdeki veri Fetch aşamasına ulaşması beklenir. İstenilen veri, buyruk önbelleğinden Fetch aşamasına yollandığını anlatmak için icache_instruction_ready_i sinyalini 1'e çeker ve buyruk Fetch tarafından kabul edilmiş olur.



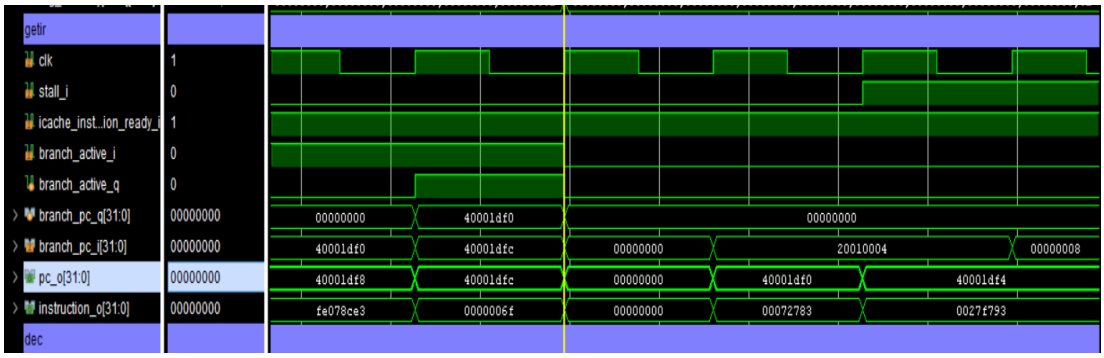
Şekil 4.9. Getir aşamasının şematik gösterimi.

Şekil 4.9'daki şematiğin gösteriminde Fetch aşamasındaki giriş ve çıkış sinyalleri net bir şekilde görülmektedir. Bu sinyaller arasında clk_i sinyali sisteme giren clock sinyalini belirtir. Clock sinyali sistemde program sayacının artması ve boru hattı mimarisinin bir aşamasında kurulan flip flop sistemi için önemli bir sistemdir. Sistem clock sinyalinin pozitif kenar tetiklemesine göre kurulmuştur.

Program sayacının başlangıç değeri 4000_0000 olarak ayarlandı. Sisteme her reset_i sinyali geldiğinde mevcut program sayaç değeri 4000_0000'uncu adrese döndürülür. Sıfırlama işlemi bu şekilde olması için ayarlanmıştır. Program sayacı her buyruk kabulü aldığında 4 arttırılır. Bunun sebebi her 1 sayısının 1-byte olarak ifade edilmesi ve hafıza birimindeki her veri satırının 4-byte olmasıdır. Bundan dolayı her satırın 4

ve 4 katı olacak şekilde ayarlanmıştır. Örnek verecek olursak ilk satır adresi 40000000 ise ikinci satır 40000004 olarak ifade edilir. Bundan dolayı her buyruk kabulünde 4 arttırılır.

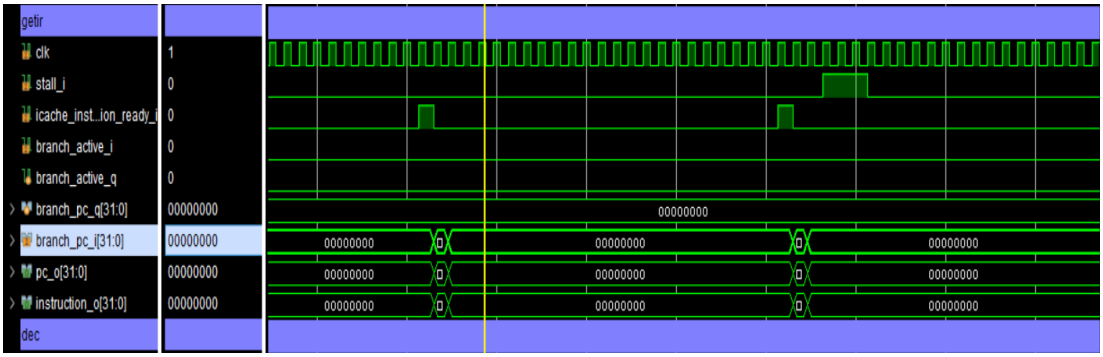
Fetch modülünün içinde dallanma için ayrı bir register bulunur. Clock sinyalinin pozitif tetiklemesinde 1 cycle gecikmeli olarak çalışır. Bunlardan birinin sebebi dallanmanın aktif olması ve bu değeri dallanasıya kadar tutması için bu tarz bir sistem tasarlanmıştır. Dışarıdan gelen stall sinyali ile sistem durdurulursa dallanacak adresin ve dallanmanın aktif olarak belirten sinyalinin korunması ile ilgili yapılmıştır. Burada tasarlanan register ile verinin stall yani sistemi durdurma işleminin bitimini bekleyesiye kadar dallanmanın korunması sağlanmıştır.



4.10 Fetch aşamasının Vivado simülasyonu.

Yukarıda şekil 4.10'da görüldüğü gibi Fetch aşamasına giren çıkan sinyaller görülmektedir. Kaydedilen branch_active_q ve branch-pc_q sinyalinin bir sonraki clock darbelerinde yeni adrese dallandıkları ve bu verileri bir sonraki boru hattı mimarisindeki registerlara kaydedip bir sonraki aşama olan Decode aşamasına yollandığı görülmektedir.

Şekil 4.11'da görülen icache_instruction_ready_i sinyalinin sürekli 1 olduğu görülmektedir. Bunun sebebi ise buyruk kümesinde, verilerin dolu olduğunu ve ana hafızadan veri çekmediğini gösterir. Şekil 4.11'de ise bu olayın daha buyruk önbelleğinde kayıtlı buyruk olmamasının ve buyruk önbelleğine verinin kayıt edilesiye kadar icache_instruction_ready_i sinyalinin 0 olmasıyla alakalıdır.

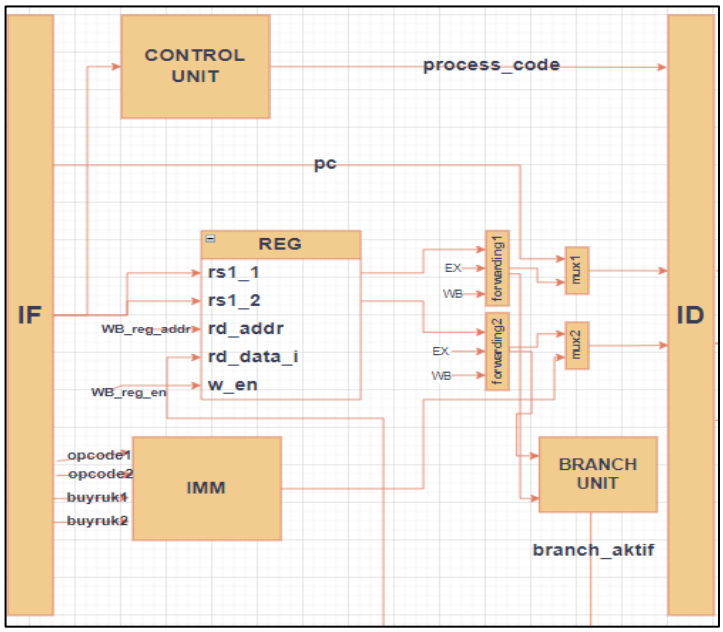


4.11. Fetch aşamasının Vivado simülasyonu.

Bu durum Fetch aşamasında bulunan boru hattını oluşturan register değerinin bir sonraki boru hattı aşamasına 32 bitlik 0 değeri yollandığını gösterir. Bundan dolayı icache_instruction_ready_i sinyalinin 1 iken gelen buyruk ve o buyruğun bulunduğu program sayaç değerinin bir sonraki aşamaya yollandığı ama 0 iken sadece 0 verisinin yollandığı görülmektedir.

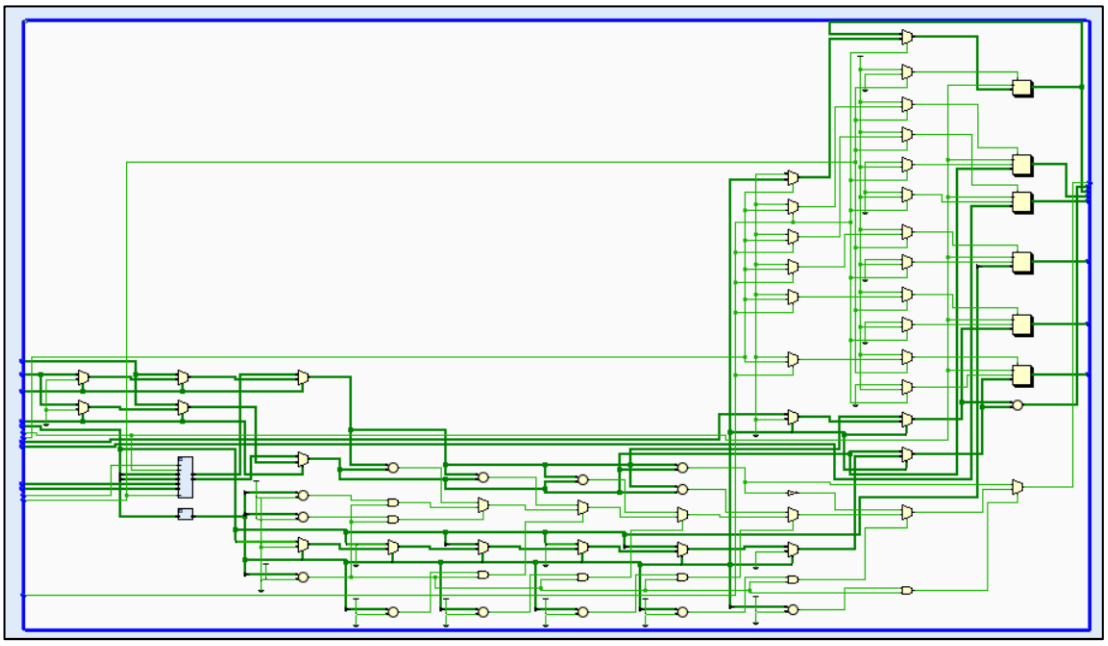
4.2.1.2. Decode

Decode aşaması Fetch aşamasından gelen buyruğun çözüldüğü aşamadır.



Şekil 4.12. Veri yolundaki Decode aşaması.

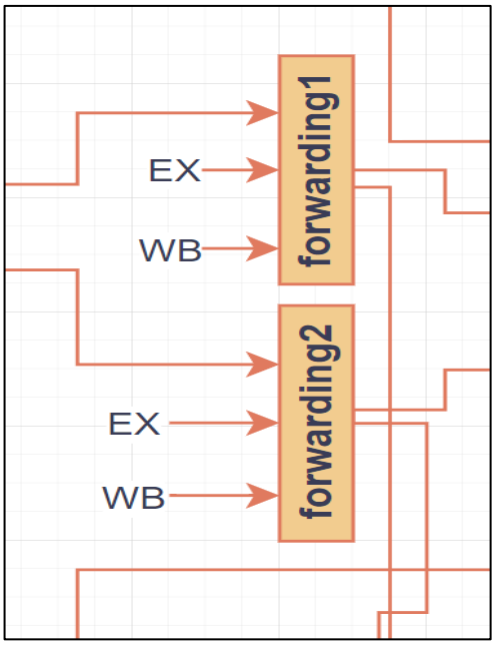
Burada Register File, Control Unit, anlık değerlerin genişletildiği ünite olan Immediate_Generator (IMM) ve dallanmanın sağlandığı ünite olan BRANCH UNIT bulunur. Burada daha çok gelen buyruğun çekirdekte hangi işlemleri yapacağı belirlenir. Buna göre uygun muxların, işlem yapan ünitelerin hangi işlemleri veya çalışıp çalışmayacaklarını belirten sinyallerin verildiği ve yapılacak işlemlerin hangi değerler arasında yapılacağını belirleyen bölgedir.



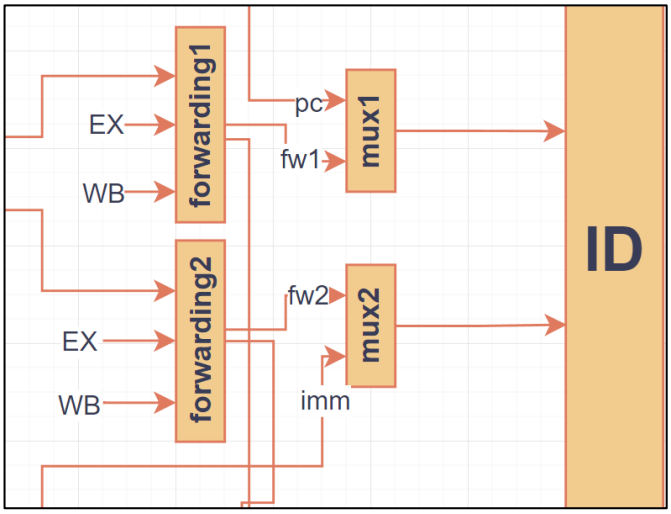
Şekil 4.13. Decode aşamasının genel şematik gösterimi.

Decode aşamasının register çıkışlarının ilk uğradığı muxlar forwarding muxlardır. Bu muxlar boru hattı mimarisinden kaynaklanan güncel verileri daha registera kaydedilmeden kullanılmasını sağlayan muxlardır. Bunlar bir tür geri besleme muxlarıdır.

Şekil 4.14 ve Şekil 4.15'de görüldüğü gibi geri besleme muxları Register File çıkışının direkt olarak bağlanması ile Execute ve Write Back aşamasındaki işlenen değerlerin geri beslemesi ile oluşuyor. Decode aşamasında eğer güncel veri Register File içinde değilse sağ taraftaki güncel aşamalardan yani Execute ve Write Back aşamalarından veri çekiliyor ve buradan diğer muxlara gidiyor.



Şekil 4.14. Forwarding (Geri besleme) muxları.



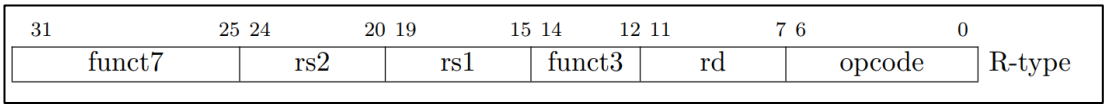
Şekil 4.15. Forwarding işleminden sonraki muxlar.

Bu muxlar Control Unit işleminden gelen sinyaller ile çalışıyor. mux1, Decode aşamasındaki pc değerini yada forwarding1 muxundan gelen değeri seçen muxdur. Mux2 ise forwarding2'den gelen yada anlık değerden (IMM) gelen veriyi seçen mux türüdür. Buradaki işlem örneğin ADD veya ADDI gibi işlemlerde, ADD komutu geldiğinde 2 farklı registerın toplanması ifade eder. Bu durumda mux1 ve mux2'lerde forwarding1 ve forwarding2 muxları seçilmelidir. Diğer yandan ADDI işlemi geldiğinde bir rs1 değeri ile imm değerinin toplanmasını ifade eder. Bu sırada bu

muxlardan gelen forwarding1 ve IMM seçenekleri olan muxların sinyallerinin yollanması gerekir.

Kontrol Ünitesi (Control Unit)

Control Unit yada Kontrol Ünitesi, Decode aşamasında buyruğu çözüp o buyruğun yapacağı işlemin sinyal kodunu üreten birimdir. Control Ünitesi, toplamda, giriş ve çıkış olmak üzere 2 farklı sinyal bulundurur.
Bu sinyaller 32 bitlik buyruk_i giriş sinyali ve 32 bitlik islem_kodu sinyalidir. Giren buyruklar opcode, func3 ve func7 olmak üzere 3 sinyal üzerinden filtrelenmeye başlanır.



Şekil 4.16. R-type buyruk dizilimi.

Yukarıda demek istenen Şekil 4.16'deki buyruk dizilimini örnek alacaksak Control Unit'e gelen buyruğun opcode, funct3, funct7'deki değerlerine göre ayıklanmıştır. Bu değerlerdeki bitlerin sisteme girmesi ve ayıklama işleminin bunlara göre ayarlanması gerekmektedir. Bu şekilde yapılmıştır. Bunun için Verilog'da, ayıklama işlemi daha kolay olsun diye casez yapısı kullanılmıştır.

```
always @(buyruk_i)begin

    casez(buyruk_i)

        LUI:begin
            islem_kodu<=LUI_CODE;
        end
        AUIPC:begin
            islem_kodu<=AUIPC_CODE;
        end
        JAL:begin
            islem_kodu<=JAL_CODE;
        end
        JALR:begin
            islem_kodu<=JALR_CODE;
        end
        ADDI:begin
            islem_kodu<=ADDI_CODE;
        end
```

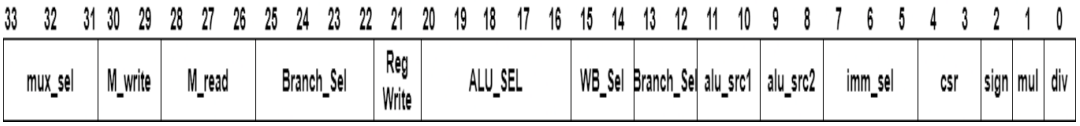
Şekil 4.17. Tasarlanan Control Unit'in decode aşaması.

Casez yapısı ayıklama işlemini kolaylaştırmıştır. İç içe if veya case yapıları ile yazılmak yerine sadece bu şekilde yazılması kolaylık sağlamıştır.

```
localparam BEQ     = 32'b???????_?????_?????_000_?????_1100011;
localparam BNE     = 32'b???????_?????_?????_001_?????_1100011;
localparam BLT     = 32'b???????_?????_?????_100_?????_1100011;
localparam BGE     = 32'b???????_?????_?????_101_?????_1100011;
localparam BLTU    = 32'b???????_?????_?????_110_?????_1100011;
localparam BGEU    = 32'b???????_?????_?????_111_?????_1100011;

localparam DIV     = 32'b0000001_?????_?????_100_?????_0110011;
localparam DIVU    = 32'b0000001_?????_?????_101_?????_0110011;
localparam REM     = 32'b0000001_?????_?????_110_?????_0110011;
localparam REMU    = 32'b0000001_?????_?????_111_?????_0110011;

localparam MUL     = 32'b0000001_?????_?????_000_?????_0110011;
localparam MULH    = 32'b0000001_?????_?????_001_?????_0110011;
localparam MULHSU  = 32'b0000001_?????_?????_010_?????_0110011;
localparam MULHU   = 32'b0000001_?????_?????_011_?????_0110011;
```

Şekil 4.18. Localparam ile tanımlanmış bazı buyruklar.

Şekil 4.18 Casez ile filtrelenecek verilerin bitleri görünecek şekilde ayarlanmıştır. Diğer bitler ise "?" işareti kullanılmıştır. Bu sayede buradaki bitlerin sistemde değişeceği anlamına gelmektedir.

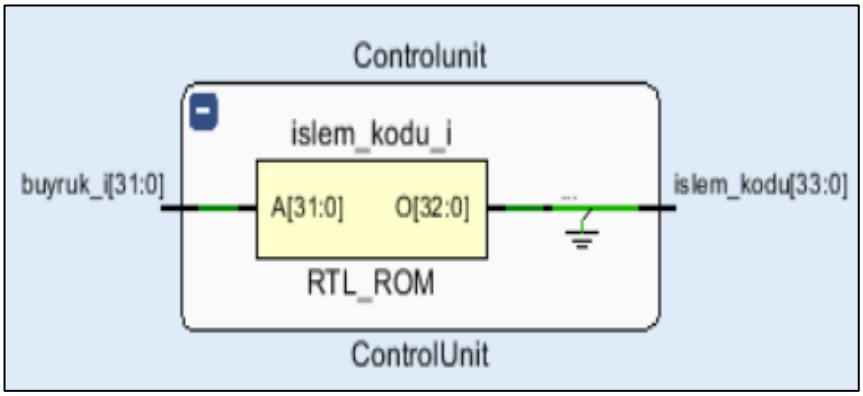


Şekil 4.19. İşlem kodunun yapısı.

Şekil 4.19'de Control Unit'den çıkan işlem kodunun bit sıralaması görülmektedir. Sistemi daha rahat programlamak için tek sinyal içerisine bütün sinyaller eklenmiştir. Sinyallerin anlamları aşağıdaki sıralama ile belirtilmiştir;

- 0'ıncı bit bölme işlemini aktif eden sinyaldir.
- 1'inci bit çarpma işlemini aktif eden sinyaldir.

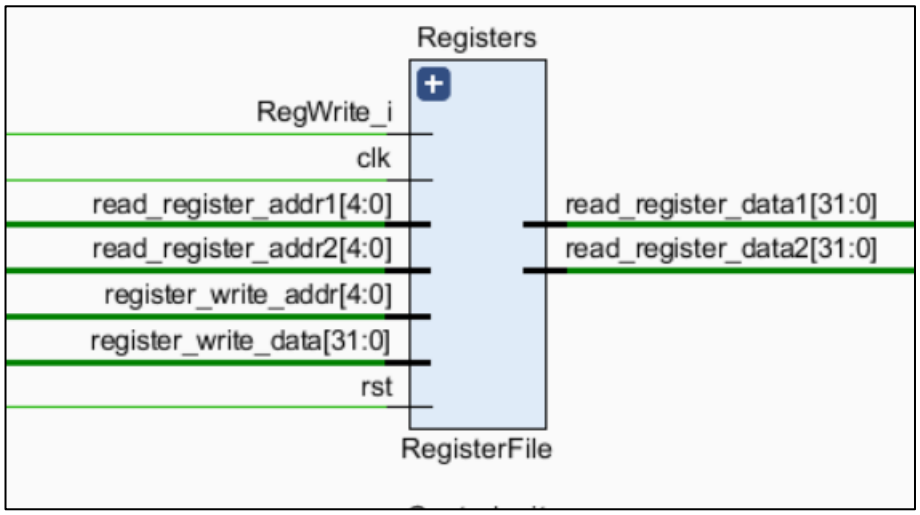
- 2’inci bit çarpma ve bölme işleminde işaretli veya işaretsiz çarpma yapılacağını anlatır.
- 4’üncü ile 3’üncü bit aralığındakiler Execute aşamasındaki muxu kontrol etmek için koyulmuştur.
- 7 ile 5 aralığındaki bitler anlık değerin genişletilme tipini seçen sinyaldir.
- 9 ile 8 ve 11 ile 10’uncu aralıktaki sinyaller mux1 ve mux2 muxlarının seçim sinyalidir.
- 13 ile 12 aralığındaki sinyaller branch olup olmayacağını belirten sinyaldir.
- 15 ile 14 aralığındaki sinyaller geri beslemedeki muxun kontrol sinyalidir.
- 20 ile 16 aralığındaki sinyaller ALU’daki yapılacak aritmetik yada lojik işlemi seçer.
- 21’inci sinyal Write Back aşamasına gelen verinin registera yazılıp yazılmayacağı belirten sinyaldir.
- 25 ile 22 aralığındaki sinyaller branch işleminde koşullu dallanmanın hangi koşula göre yapılacağını seçen sinyallerdir.
- 28 ile 26 aralığındaki sinyaller hafıza hattından okuma işlemi yapılmasını ve hangi okuma işleminin yapılacağını seçen sinyallerdir.
- 30 ile 29 aralığındaki sinyaller hafıza hattında yazma işlemi yapılmasını sağlayan sinyallerdir.



Şekil 4.20. Control Unit’in şematik yapısı.

Register File

Register File, RISC-V çekirdeğinin yapısına göre uygun bir şekilde tasarlanmıştır. RISC-V çekirdek yapısına göre 32 tane 32-bit register bulunması gerekmektedir. Bu registerlar işlem yapılacak verileri kısa süreliğine tutmak amacıyla tasarlanmıştır. Pozitif kenar tetiklemesi ile veriler kaydedilmektedir. Veri çıkışı için herhangi bir clock sinyaline ihtiyaç duymaz, veriler gelen adres ile aynı anda dışarı çıkar. Başlangıç değeri olarak sp registerı yani x2 registerına değer 4000.1000 adres eklenmiştir. Bu register stack point registerıdır ve stack adresini tutar. Bu olay isteğe göre değişebilir. Veya derleyeci ortamında ayarlanabilir.



Şekil 4.21. Register File şematik görüntüsü ve sinyalleri.

Şekil 4.21'da Register File sinyalleri görülmektedir. Bu sinyaller sırasıyla aşağıdaki sıralamaya göre ne oldukları açıklanacaktır;

- Clk sinyali, registera giren clock sinyalidir.
- RegWrite_i sinyali registerın, gelen veriyi kaydedip kaydetmeyeceğini belirten sinyaldir.
- read_register_addr1 ve read_register_addr2 sinyalleri istenilen register verilerinin adreslerini belirtir.
- register_write_addr içeri kaydedilecek verinin adresini belirten sinyaldir.
- register_write_data içeri giren kaydedilecek veriyi belirtir.

- rst sinyali sisteme reset atıldığında 1 olan sinyaldir. Sistemdeki registerların değerlerini sıfırlar.
- read_register_data1 ve read_register_data2, read_register_addr1 ile read_register_addr2 adreslerindeki registerların kaydedilmiş verilerini dışarı çıkartan sinyallerdir.

Anlık Değer Üreticisi

Başka sistemlerde genel olarak anlık değer üreticisi, IMM_GEN veya SIGN_EXTENDER gibi isimlerde bulunur. Bu tarz isimlere benzer şekilde tasarlanmıştır. Direkt olarak decode yapısının içerisinde tasarlanmıştır. Buyrukların içerisindeki imm olan bölümleri alıp 32-bite göre genişletildiği bölümdür. Sistemde I, U, S, B, J olmak üzere 5 farklı genişletme seçeneği vardır.

Branch Unit

Branch Unit Decode modülünün içinde oluşturulmuştur. Dallanma şartlarını yürüten birimdir.

imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU

Şekil 4.22. Dallanma buyrukları.

Şekil 4.22'de görülen dallanma buyruklarının şartı yerine getirip getirmediğini sağlayan birimdir. Eğer şartlar yerine geliyorsa branch_aktif adında bir sinyali 0'dan 1'e çeker bu sinyal, Fetch aşamasına yollanır ve dallanma başlatılır. Bu sırada mux1 ve mux2'den çıkan veriler toplanır. Bu toplanan veriler dallanacak adresi belirtir. Bu dallanacak yeni adres branch_aktif sinyali ile birlikte fetch aşamasına gider ve branch işi başlatılmış olur. Branch aktif olduktan sonra Fetch aşamasında 1 cycle işlem bekletilir. Eğer sisteme stall yani durdurma komutu gelirse branch işlemide o registerda saklanır. Bu sayede branch işlemi olmadan sistem devam etmez.

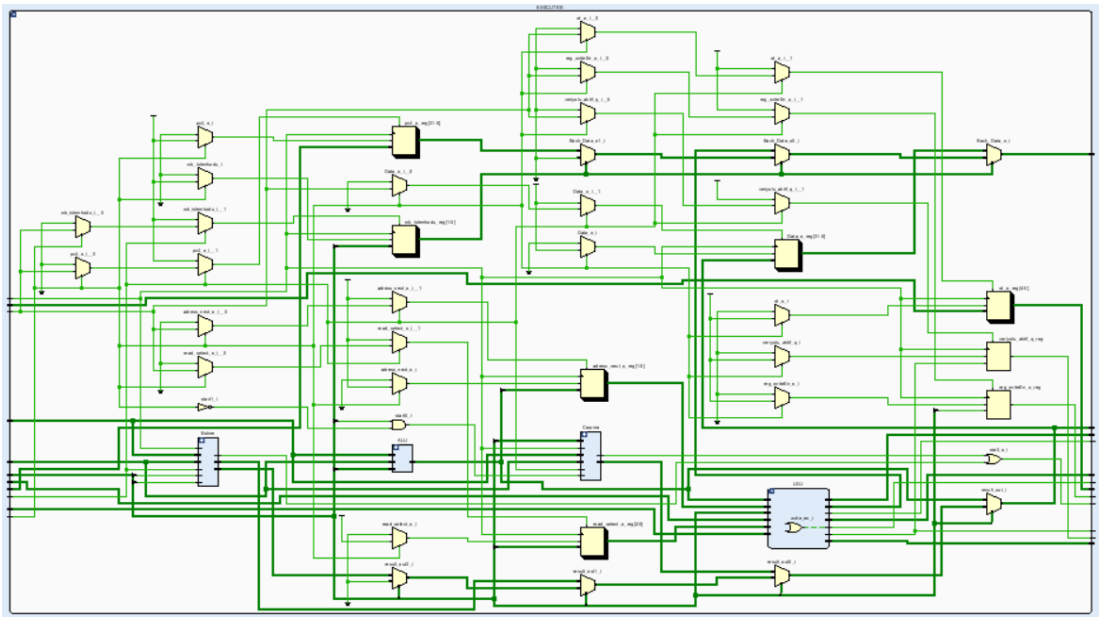
imm[20|10:1|11|19:12]			rd	1101111	JAL
imm[11:0]	rs1	000	rd	1100111	JALR

Şekil 4.23. Koşulsuz dallanma buyrukları.

JAL ve JALR koşulsuz dallanma operatörleridir. Bu buyruklar geldiği sırada direkt dallanma başlar Branch Unit koşula bağlı kalmadan direkt olarak aktif sinyali gönderir ve sistem istenilen adrese dallanır.

4.2.1.3. Execute

Execute aşaması işlemcinin gelen verileri hafızasal, aritmetik ve lojik işlemlere tabi tuttuğu yerdir. İşlemci çekirdeği genel olarak RV32IM olduğı için M eklentisine sahip olduğundan dolayı çarpma ve bölme işlemi yapabilir.



Şekil 4.24. Execute aşamasının şematik gösterimi.

Genel Yapısında ALU (Arithmetic Logic Unit), MUL (Çarpma Birimi), DIV (Bölme Birimi), LSU (Load-Store Unit) bulunmaktadır. ALU, MUL ve DIV çıkışı muxa bağlanmıştır. Muxun görevi Write Back aşamasına hangi biriminden çıkan değerin gideceğini belirtir. ALU çıkışı mux dışında LSU'ya bağlıdır. Bunun sebebi ALU çıkışının store veya load komutlarını çalıştırırken adres görevi görmesinden

kaynaklanmaktadır. ALU, burada store ve load komutları için adres belirtmek amacıyla kullanılan modüldür. Normalde branch sistemi kullanılırken ALU branch adresi tanımlamak içinde kullanılabilir. Tasarlanan işlemcide branch işlemi Decode aşamasına alındığı için burada ALU branch adresini hesaplamamaktadır.

ALU

ALU tasarlanan işlemcinin, aritmetik mantık birimidir. Burada aritmetik ve lojik işlemleri yapılır. RISC-V el kitabında belirlenen buyruk setlerindeki işlemler bu yerde tanımlanmıştır.

0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

Şekil 4.25. ALU'ya tanımlı işlemler [20].

Şekil 4.25'teki görülen buyruklar ALU içinde yapılan işlemlerdir. Bu işlemler sırasıyla şunlardır:

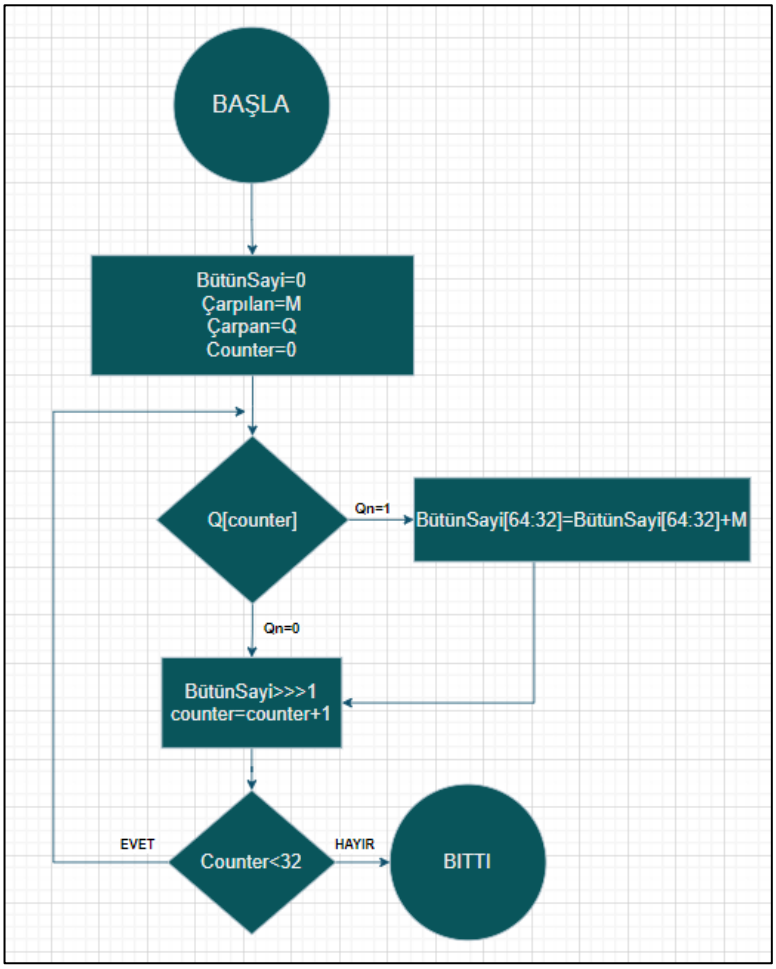
- ADD işlemcideki toplama işlemini yapan buyruktur.
- SUB işlemcideki çıkartma işlemini yapan buyruktur.
- SLL işlemcideki lojik sola kaydırma işlemini yapan buyruktur.
- SLT işlemcideki set less than anlamına geliyor. Belirlenen 2 registerdan biri diğerinden küçükse belirlenen rd registerına 1 set eder.
- SLTU, SLT buyruğu ile aynıdır sadece karşılaştırılacak registerlar unsigned olarak alınır.
- XOR, 2 register arasında "xor" işlemi yapan buyruktur.
- SRL, işlemcideki lojik sağ kaydırma işlemini yapan buyruktur.
- SRA işlemcideki aritmetik sağ kaydırma işlemini yapan buyruktur.

- OR işlemcideki 2 register arasında “veya” işlemini yapan buyruktur.
- AND işlemcideki 2 register arasında “ve” işlemini yapan buyruktur.

Yukarıda verilen bütün aritmetik ve lojik işlemler tasarlanan, işlemciye tanımlanmıştır. Toplama ve çıkartma için özel bir algoritma kullanılmamıştır.

MUL (Çarpma Modülü)

İşlemci için tasarlanmış çarpma işlemidir. M eklentisine göre tasarlanmış bir modüldür. Unsigned Binary Multiplication Algorithm kullanılmıştır. Negatif kenar tetiklemesi ile çalışıyor. Unisgned Binary Multiplication algoritması finite state machine ile yapılmıştır.



Şekil 4.26. Unsigned Binary Multiplication algoritması.

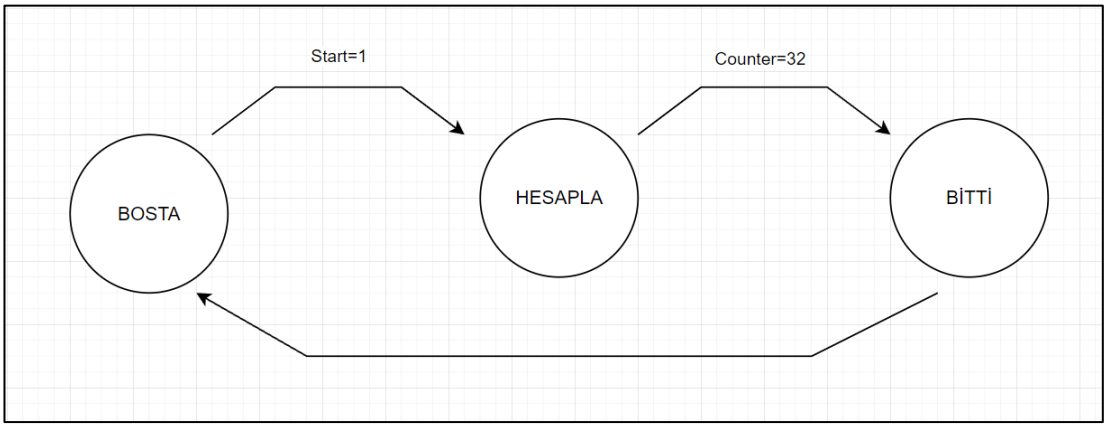
Şekil 4.26'te işlemcide kullanılan algoritma görülmektedir. Unsigned algoritması olduğu için gelen işlem_kodun'daki ALU koduna göre yapılacak işlemler seçilmektedir.

RV32M Standard Extension						
0000001	rs2	rs1	000	rd	0110011	MUL
0000001	rs2	rs1	001	rd	0110011	MULH
0000001	rs2	rs1	010	rd	0110011	MULHSU
0000001	rs2	rs1	011	rd	0110011	MULHU

Şekil 4.27. Çarpma ve bölme buyrukları [20].

Şekil 4.27'deki çarpma buyrukları kullanılmıştır. Çarpma buyruklarının özellikleri aşağıdaki gibidir;

- MUL buyruğu signed 2 tane registerın çarpımının ilk 32 bitini veren buyruktur.
- MULH buyruğu signed 2 tane registerın çarpımının son 32 bitini veren buyruktur.
- MULHU buyruğu unsigned 2 tane registerın çarpımının son 32 bitini veren buyruktur.
- MULHSU buyruğu signed rs1 ile unsigned rs2 değerinin çarpımının son 32 bitini veren buyruktur.



Şekil 4.28. Çarpma işleminin Finite State Machine şeması.

Şekil 4.28'da çarpma işleminin finite state machine şeması görülmektedir. BOSTA, HESAPLA ve BITTI olmak üzere 3 farklı durumdan oluşmaktadır. BOSTA kısmı

çarpmanın boşta beklediği kısmıdır. Bu kısımda dışarıdan start sinyali gelesiye kadar bekler. Eğer dışardan start sinyali geldiğinde ve dışarıdan gelen ALU_Kod sinyaline göre şekil 4.27'deki çarpma işlemlerinden biri belirlenir. Belirlenen çarpma işlemine göre işlem signed ise modüle giren verilerin 32. bitleri, 1-bitlik özel registerlara kaydedilir. Bunun sebebi çarpma algoritmasının unsigned algoritma olmasından kaynaklanmaktadır. Q ve M registerlarına çarpanlar kaydedilir. butun_sayı registerı ve counter 0'a çekilir. butun_sayi registerı 65 bitlik bir sayıdır bunun sebebi çarpma işleminden çıkan son 32 biti içermesi. Sondaki 65. bit ise algoritmada carry olarak belirtilir ve hesaplamada kullanılır. Genel sistem basitçe şu şekilde çalışmaktadır. İşlemciden bir tane 5 bitlik ALU_Kod ve 1 bitlik start biti gelir. Durum BOSTA iken HESAPLA durumuna geçer. Her clock darbesinde counter 1 artar. Bu sırada Q'nun counter değerinde olan biti 1 ise butun_sayi'nın 64 ile 32 aralığındaki bitleri M çarpanı ile toplanır. Daha sonra 1 sağ kaydırılır. Q'nun counter değerinde olan biti 0 ise butun_sayi değer sadece 1 defa sağ kaydırılır. Bu işlem counter 32 olasıya kadar devam eder. Sonra sistem BITTI durumuna geçer ve eğer işlem signed ise kaydedilen 32. bitler xor'lanır. Bu sayede çıkan sonucun negatif veya pozitif olduğu anlaşılır. Çıkan değer bu duruma göre ters çevrilir. Eğer çıkan sonuç unsigned isteniyorsa, 32. bitler umursanmadan sonuç dışarıya verilir. Bir clock darbesi sonrası tekrar sistem BOSTA konumuna geçer. Stall sinyali 0'a çekilir sistem devam eder.

DIV (Bölme Modülü)

Tasarlanan işlemci çekirdeğinin bölme modülüdür. Non-Restoring Division algoritması ile tasarlanmıştır.

```
module divider #(parameter BIT=32) (
        input                clk,
        input                reset,
        input [BIT-1:0]      dividend,
        input [BIT-1:0]      divisor,
        input                sign,
        input                start,
        output               busy,
        output reg[BIT-1:0] quotient,
        output reg[BIT-1:0] remainder
);
```

Şekil 4.29. Bölme modülünün giriş çıkışları.

Şekil 4.29’de çarpma modülünün giriş çıkışları görülmektedir. Bu giriş ve çıkışların açıklaması aşağıdaki gibidir:

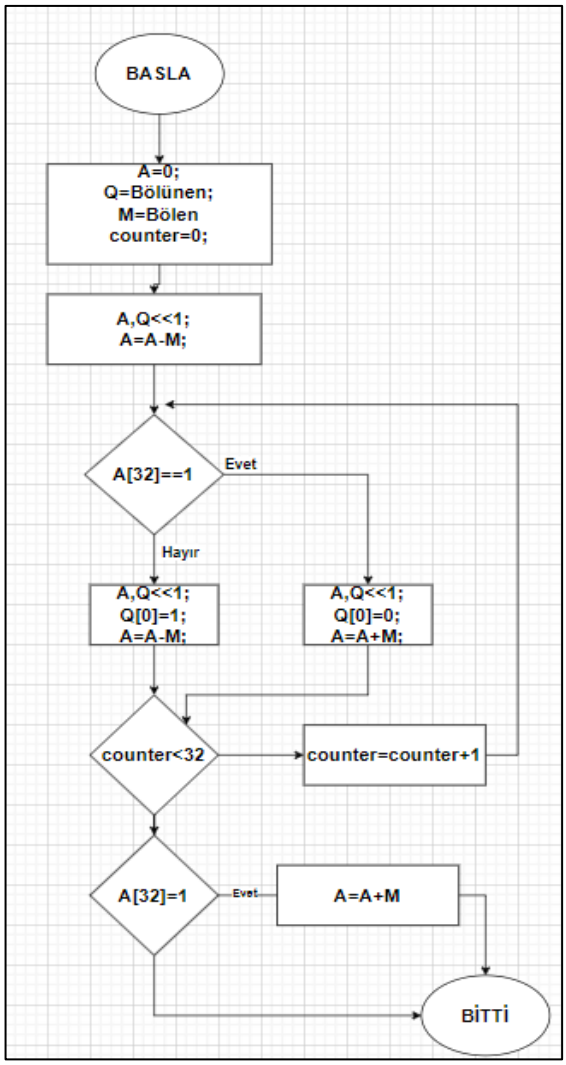
- clk girişi sisteme giren clock darbesini ifade eder.
- reset girişi dışarıdan gelen reset sinyalini ifade eder.
- dividend ifadesi 32-bit bölünen sayının ifadesidir.
- divisor ifadesi 32-bit bölen sayının ifadesidir.
- sign girişi, sistemin işaretli mi yada işaretsiz mi bölme yapacacağını belli eder.
- start işlemi, sistemin bölmeyi başlatacağını ifade eder.
- busy sinyali, tasarladığımız çekirdeği durduran sinyal komutudur.
- quotinent çıkışı, bölme işleminden çıkan bölüm sayısıdır.
- remainder çıkışı, bölümden kalan sayıyı ifade eder.

Çizelge 4.2. Bölme işlemi buyrukları.

0000001	rs2	rs1	100	rd	0110011	DIV
0000001	rs2	rs1	101	rd	0110011	DIVU
0000001	rs2	rs1	110	rd	0110011	REM
0000001	rs2	rs1	111	rd	0110011	REMU

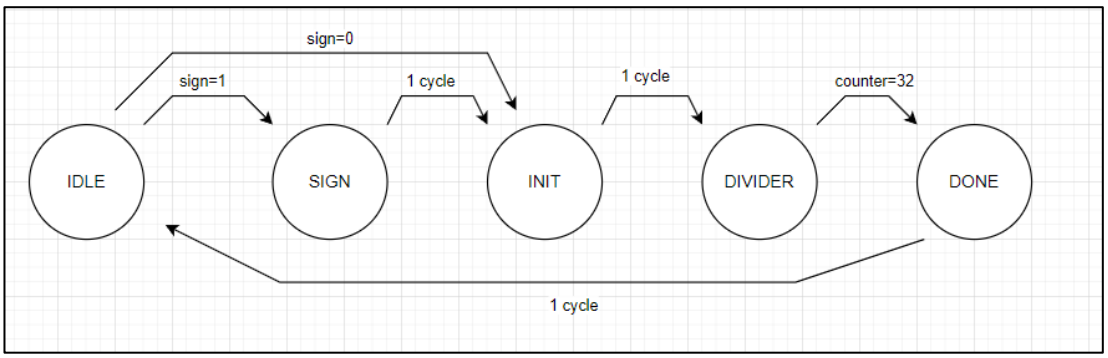
Yukarıda verilen Çizelge 4.2’de bölme buyrukları görülmektedir. Bu buyruklar tasarlanan bölme modülünü tetiklemektedir. Çizelge 4.1’teki buyrukların açıklamaları aşağıda gibi sıralanmıştır:

- DIV, 32-bit signed bölme işlemi yapmaktadır.
- DIVU, 32-bit unsigned bölme işlemi yapmaktadır.
- REM, 32-bit signed bölme işleminden kalanı vermektedir.
- REMU, 32-bit unsigned bölme işleminden kalanı vermektedir.



Şekil 4.30. Non-Restoring Division algoritması.

Şekil 4.30 Tasarımda kullanılan Non-Restoring Division Algoritması görülmektedir. Bu algoritmada unsigned işlem yapılıp ilk bitleri tutularak işlem yapılmaktadır.



Şekil 4.31. Bölme işleminin finite state machine şematiği.

Şekil 4.31'deki şematiğin mantığında finite state machine kullanılmıştır. Sistem boşta beklerken IDLE aşamasında beklemektedir. Bölme buyruğu geldiğinde işlem_kodu'ndaki işareti aktif eder buda sistemin start bitine denk gelerek sistemi başlatır. Eğer yapılacak işlem signed bir işlem ise 32. Bitleri 1 bitlik registerlara kaydedilir ve busy register 1'e çekilerek çekirdeğin durması sağlanır. Eğer unsigned ise 32. bitler kaydedilmeden sisteme devam edilir. Finite state machine durumu IDLE'dan SIGN veya INIT aşamasına geçer. Eğer bölme işlemi işaretli ise SIGN durumunda negative işaretli olanlar 2'ye göre tümleyeni alınır pozitife çekilirler. Unsigned bir bölme ise SIGN durumu atlayarak direkt olarak INIT durumuna geçerler. INIT durumunda sistem algoritmadaki A, Q'yu bir sola kaydırır ve A=A-M işlemini başlatır. Bu olay 1 döngü sürer ve DIVIDER durumuna geçer. DIVIDER durumunda counter_reg denilen bir register 31. hite eşit olduğunda bu döngüden çıkar. Bu döngü sırasında Şekil 4.29'uncuda görülen işlemleri yapılmaya başlanır. A registerının en anlamlı bitine bakılarak 2 farklı işlemden biri seçilir ve döngü counter_reg 31'e eşit olasıya kadar devam eder. Durum DONE'a geçer. Burada işaretli ve sonucun eksi çıkması gerekiyorsa ama bölen 0 ise quotient çıkışından -1 çıkışı verilmektedir. Ama bölen 0 değilse ve quotinent var olan sonucun negatifi olarak çıkar. İşlem negatifse ve divisor yani bölen 0'sa remainder yani kalan dividend olarak dışarı çıkacaktır. Bölen yani divisor 0 değilse çıkan remainder sonucunun negatifi dışarı verilecektir. Bu işlemler unsigned olarak yapılacaksa ve divisior 0 ise signed işlemindekinin aynı sonucu verilecektir. Divisior 0 değilse unsigned olduğu için quotinent tersi alınmadan çıkacaktır. Remainder divisor 0 ise dividend çıktısı verilecektir. Eğer değilse herhangi bir tersi alınmadan remainderın kendi çıktısı dışarı verilecektir.

Memory (LSU)

İşlemcideki veri önbelleği ile haberleşen birimdir. Burada belleğe yazılması gereken veya çekilmesi gereken verilerin tetiklendiği yerdir. Adından dolayı hafıza kısmı sanılabilir. Ama birçok farklı sistemde LSU olarak tanımlanmaktadır. LSU tetikleyen 8 farklı buyruk vardır. Bu buyruklar geldiğinde LSU mux görevi görmeye başlar ve istenilen verileri veri önbelleğine iletir. Bu buyruklar şunlardır;

- LB, hafızadan byte çekilmesini sağlar.

- LH, hafızadan halfword çekilmesini sağlar.
- LHU, hafızadan unsigned halfword çekilmesini sağlar.
- LBU, hafızadan unsigned byte çekilmesini sağlar.
- LW, hafızadan word çekilmesini sağlar. Sistem 32-bit olduğu için WORD 32-bit olur. Bu yerel bir boyut olduğu için böyledir.
- SB, hafızaya byte yazdırır.
- SH, hafızaya half-word yazdırır.
- SW, hafızaya word yazdırır.

```
module MEMORY(

    input       [31:0]    adress_i          ,
    input       [31:0]    rs2_data_i        ,
    input       [30:0]    islem_kodu        ,


    input       [31:0]    mem_data_i        ,
    output                veriyolu_aktif    ,
    output      [31:0]    mem_adress_o      ,
    output      [31:0]    mem_write_data_o  ,
    output                mem_read_en       ,
    output                mem_write_en      ,
    output     [3:0]      wstrb             ,
    output     [31:0]     mem_read_data     ,
    input      [1:0]      adress_next       ,
    input      [2:0]      read_select_next


);
```

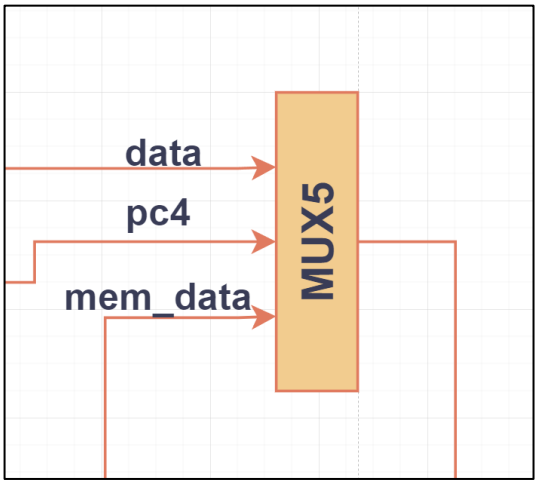
Şekil 4.32. MEMORY modülünün giriş çıkış sinyaller.

Şekil 4.32'da MEMORY modülünün giriş çıkış pinleri görülmektedir. Bu pinler işlemciden gelen sinyalleri veri önbelleğine doğru bir şekilde girmesini sağlamaktadır. MEMORY modülü burada muxlar kullanarak sistemi en doğru şekilde verileri veri önbelleğinden çekirdeğe, çekirdektende önbelleğe doğru taşımaktadır. adressi_i input girişi ALU bağlıdır. Bu giriş ALU'dan gelen adresi belirtir. Bu adres değeri kaydedilecek veya çekilecek verilerin adresidir. rs2_data_i, rs2 registerına bağlı bir giriştir. Bu kaydedilecek veridir. islem_kodu hangi buyruğun isteği üzerine muxları ayarlayan sinyalleri yollar. veriyolu_aktif, eğer son 16 biti 2000,2001,2002 temsil

ediyorsa veri önbelleği ile haberleşmeyi bırakıp Wishbone Bus aracılığıyla çevre birimleri ile haberleşmeye başlatmak için kullanılır. MEMORY modülü sadece veri önbelleği ile haberleşmez. Wishbone Bus protokolü ile sistemde bulunan çevre birimleri ile de haberleşmektedir. Bu sinyaller sırasıyla mem_data_i, mem_adress_o, mem_write_data_o, mem_write_en, mem_read_en, mem_read_data'dır ve memory kısmı ile haberleşen kısımdır. Bu kısımlar gelen veriyi veya yollanan verileri MEMORY'nin yazma veya okuma kısmının aktif olması ile ilgili kısımlardır. wstrb komutuda veri önbelleği ile ilgili haberleşme kısmıdır. Burda write strobe anlamına gelir. 4 bitlik bir sinyaldir. Her biti hafızadaki 32 bitlik bir satırın byte'nı temsil eder. Hangi byte'na veri yazılacaksa 4 bitten yazılacak byte kısmına denk gelen biti 1'e çekilir. Eğer word yazılacaksa bütün bitleri 1'e çekilir. En son sinyal read_select_o sinyali hafızadan çekirdeğe gelecek verinin bir clock geçikmeli geleceği için MEMORY aşamasındada load işlemi bir sonraki aşamaya kayacağı için muxlarda istenilen read komutu olmayabilir. Bunda dolayı gerideki read işlemini 1 cycle bekleterek read_select_next işlemine sokuyoruz.

4.2.1.4. Write Back

Write Back aşaması ekstra bir modül olarak tasarlanmamıştır. Execute modülünün içerisinde tanımlanmıştır. Bu aşamada elde edilen veriler geri yazdırılmak üzere Register File gider. Bu aşamada bir tane mux bulunur.



Şekil 4.33. Write Back aşamasındaki mux.

Şekil 4.33'de en son Register File'a doğru geri dönmekte olan verinin seçileceği mux görülmektedir. Burada aritmetik veya lojik işlemlerden çıkan verilerin son aşama olan Write Back aşamasındaki muxa gidip oradan işlem kodu ile hangisinin seçileceğini belirleriz. ALU, MUL, DIV gibi matematiksel işlemlerden çıkan veriler data adındaki sinyal ile muxa girmektedir. Program sayacının 4 fazlası pc4 olarak muxa giriş yapmaktadır. LSU modülünden çıkan veri mem_data adı ile muxa giriş yapmaktadır. Burada reg_writeEn_o sinyalide Register File'ın yazma özelliğini aktif etmek için vardır ve 5 bitlik rd registerıda kayıt edilecek verinin adresini belli eder. Bütün bu aşamada bu sinyaller Register File'a geri beslenir.

4.2.1.5. Denetim Birimi

Denetim birimi, işlemci çekirdeğindeki tehlikeleri kaldırmak amacıyla kurulmuş bir modüldür. Load-Use ve veri tehlikelerini ortadan kaldırır. Veri tehlikelerini kaldırmak için forwarding muxlarını kontrol eden bir kısım vardır. Burada daha önce Execute ve Write Back aşamasında olan güncel verilerin geri beslenmesi gerekip gerekmediğini kontrol eder. Load-Use tehlikesi ise lw gibi çekme komutları geldiğinde ve bir arkasından çekilen güncel verinin kullanılması gerektiği bir buyruk geldiğinde geri besleme kısmı Execute aşamasında güncel veri yerine güncel verinin adresini yollayacaktır. Bunun sebebi veri çekerken çekilen verinin hesabını ALU'nun yapmasından kaynaklanmaktadır. Denetim birimi arka buyruğu 1 tur bekletir. 1 cycle sonra aralarında 1 boşluk oluşur. Bu sayede işlemcideki hata ortadan kaldırılmıştır.

4.2.2. Buyruk Önbelleği (Instruction Cache)

Boyutu 4 Kb olan bir önbellek tasarlanmıştır. Önbellek yöntemi olarakda Direct Mapped Cache yöntemi kullanılmıştır.

Buyruk önbelleğinde "Tag Ram" ve "Onay Ram" olmak üzere 2 farklı ram vardır. Onay_Ram 1 bit 1024 tane, Tag_Ram ise 22 bit 1024 tanedir. Bu ramler gelen verinin hit olup olmadığını anlamak için kurulmuş sistemlerdir.

Çizelge 4.3. Önbellek adres bit özellikleri

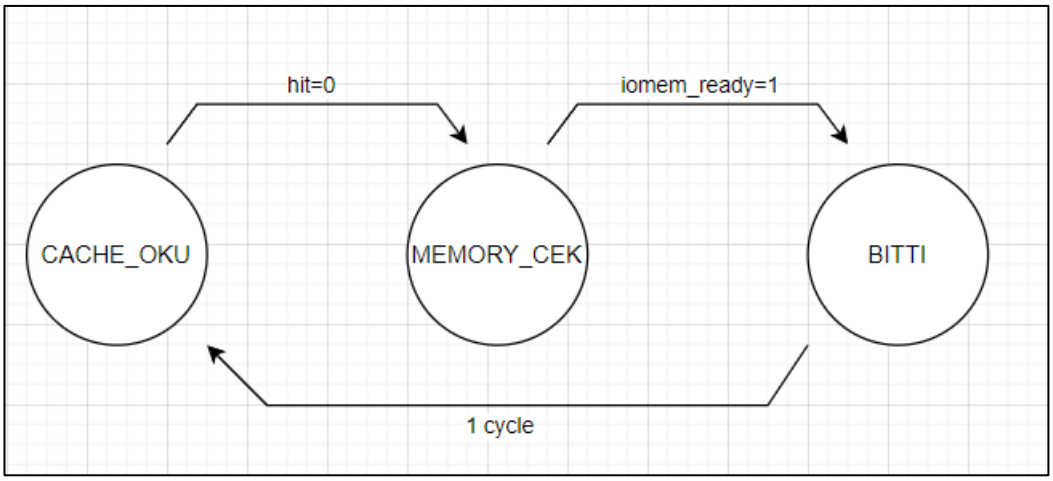
TAG	INDEX	OFFSET
31 30 29 28 ……13 12	11 10 9 8 7 6 5 4 3 2	1 0

Adresteki Tag ve Index kavramı, Çizelge 4.3'ye göre çalışır. Offset bölümü kullanılmamaktadır. Bunun sebebi sıkıştırılmış buyrukların bulunmamasıdır ve hizasız hafıza birimine sahip olmamasıdır. Sıkışıtırılmış buyruklar olmadığı için dallanmaların hiçbiri 32 bit'lik satırların ara değerlerine dallanamaz. Bundan dolayı offset bitlerine ihtiyaç duymamıza gerek yoktur. Fetch aşamasında gelecek adres değerleri hep 4 ve 4'ün katı olacağı için de ihtiyacımız yoktur.

```
module icache (

        input                   clk_i,
        input                   reset_i,

        input       [31:0]      adress_i,
        output                  instruction_kabul_o,
        output reg              icache_busy,
        output      [31:0]      instruction_o,

        output reg              iomem_valid   ,
        input                   iomem_ready   ,
        output reg [3:0]        iomem_wstrb   ,
        output reg [31:0]       iomem_addr    ,
        output reg [31:0]       iomem_wdata   ,
        input      [31:0]       iomem_rdata
);
```

Şekil 4.34. Buyruk önbelleğinin giriş ve çıkışları.

Şekil 4.34'de görüldüğü gibi tasarlanan buyruk önbelleğinin giriş ve çıkışları görülmektedir. address_i'den instruction_o'ya olan kısma kadarki sinyaller çekirdek ile haberleşen kısımları temsil eder. Başında iomem olan bütün sinyaller ise ana hafıza birimi ile haberleşecek kısımdır.



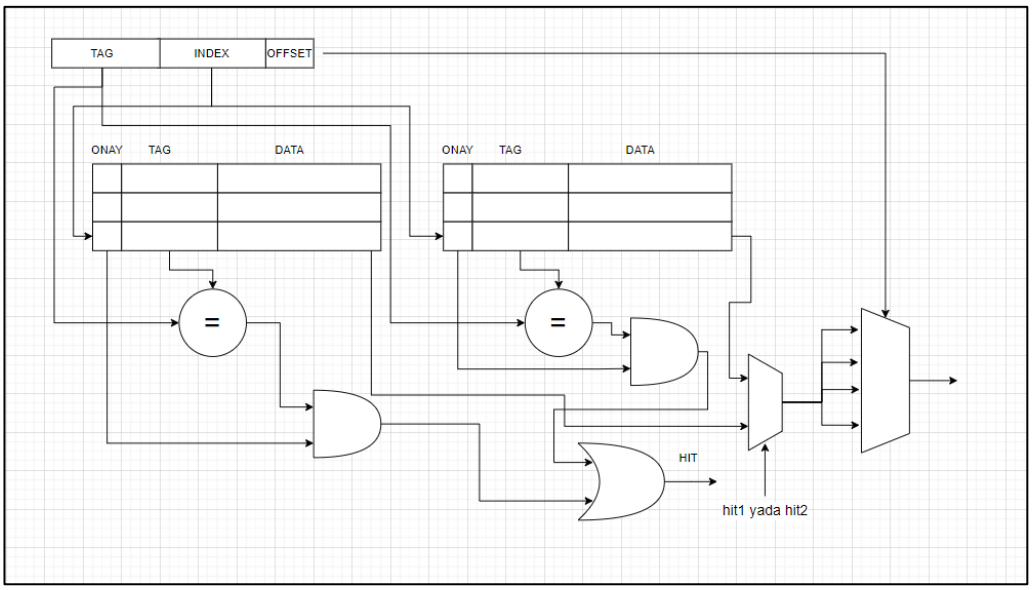
Şekil 4.35. Buyruk öbelleğinin FSM diyagramı.

CACHE_OKU durumunda çekirdekten önbellek sistemine giren adresin, tag raminden ve onay raminden veriler çekilir onay ramindeki değer 1 ise ve tag ram ile gelen adresdeki 31'den 10'uncu bite kadar olan adresler eşleşiyorsa hit 1 verilir. Eşleşmiyorsa hit 0 verilir. Hit 0 olduğunda durum CACHE_OKU'dan MEMORY_CEK durumuna geçer. Burada iomem_valid sinyali 1'e çekilir. Çekirdekten gelen adres iomem_adress eşitlenir. wstrb veri çekilirken 0 olmak zorundadır. Bu sırada iomem_ready sinyali beklenir. Ana hafızadan veri geldiğinde ana hafıza iomem_ready sinyalini yollar. Önbelleğe iomem_ready sinyali geldiğinde yine bu aşamada write_en register aktif edilir. Gelen veri okunan_registerına kaydedilir. Bu register doğrudan önbellek hafızasının data_i'sine bağlıdır. Bu sırada ram adındaki hafıza birimine 32-bitlik ana hafızadan gelen veri kaydedilir. Etiket için ise tag adındaki hafıza birimine çekirdekten gelen 32 bitlik adresin 31'den 10'a kadar olan adres kaydedilir. Çekirdekten gelen adresin bulunduğu konumdaki onay bitide 1'e çekilir. Durum bu sıradan sonra BITTI işlemine geçerek bütün write_en gibi daha önceden ayarlanmış bütün sinyaller 0'a çekilir. Bu sırada yeni gelen veriler kaydedildiği için hit 1'e çekilmiş olur. BITTI durumundan 1 döngü sonrası CACHE_OKU gelinir ve gelen veri çekirdeğe yollanır. Tasarlanan buyruk kümesi bu şekilde çalışmaktadır.

4.2.3. Veri Önbelleği (Data Cache)

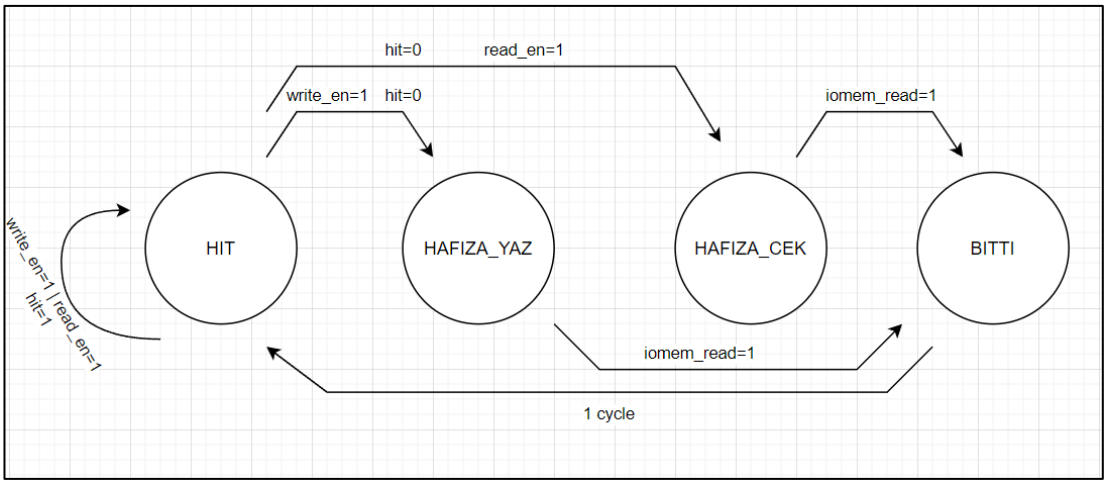
Veri önbelleği 2 Way Set Associative Cache yapısına göre tasarlanmıştır. 2 tane 1 KB ram belleğine sahiptir.

Veri önbelleği 2 yollu olduğu için 2 farklı ram yapısına sahiptir. 2 farklı ramler 8-bitlik olmak üzere kendi içlerinde 4 farklı ramlere ayrılmıştır. Bunun sebebi satırlar arasına byte veya half word yazabilmek içindir.



Şekil 4.36. Tasarlanan Two Way Set Associative yapısı.

Şekil 4.36'da tasarlanan Two Way Set Associative yapısı görülmektedir. Bu yapının Direct Mapped Cache yapısına göre 2 farklı rami bulunmaktadır. Bunun sebebi Direct Mapped Cache aynı indexe ama farklı 2 taga sahip verilerin bellekte silinip tekrar üzerine yazılmasıydı. Two Way Set Associative'de ise bu olay 2 farklı ram yapısına sahip olduğundan dolayı Direct Mapped Cache'e göre farklılık gösteriyor. Belirli bir adres geldiğinde, ana hafızadan veri önbelleğin hafızasına veri yazdıktan sonra tekrar aynı indexe sahip ama tagleri farklı başka bir adres geldiğinde bu veri önbellekteki ilk hafızanın üstüne yazılmıyor. 2. hafızanın üstüne yazılıyor bu sayede önbellekte sürekli sil, kaydet, çek gibi işlemler peş peşe gelmeyerek sistemi yavaşlatmıyor.



Şekil 4.37. Veri önbelliğinin Finite State Machine yapısı.

Tasarlanan finite state machine 4 durumdan oluşmaktadır. Bunları HIT, HAFIZA_YAZ, HAFIZA_CEK, BITTI'dir. HIT aşamasında sistem çekirdekten yazma veya okuma işleminin gelmesini bekler. Bu işlem sırasında herhangi bir okuma veya yazma işlemi geldiğinde dışarıdan gelen adresin tagi önbellek içerisinde kaydedilen tag ile kıyaslanır. Ama bu sistemi Direct Mapped Cache'den ayıran özellik, sistemde tag0 ve tag1 diye 2 farklı tag yapısının bulunmasıdır. Bunun sebebi her tagın farklı hafıza deposunu belirtmesidir. Dışarıdan gelen adres, tag0 veya tag1'den biri ile uyuşuyorsa ve onay0 veya onay1'den gelen adresteki değeri 1 ise sisteme hit verilir. tag0 ve onay0 uyuşursa yada tag1 ve onay1 uyuşursa hit verilir. Ram0 ve ram1 ile gösterilen hafızalardan hangisi hit almışsa dışarıya yani çekirdeğe adresin index kısmındaki veri verilir. Yada aynı şekilde yazma işlemi varsa ve hit verilmişse indexin o kısmına çekirdekten gelen veri yazılır. Eğer hit olan verinin üstüne çekirdekten başka bir veri yazılırsa, boyutu 1-bit olan dirty hafızasınada gelen index adresinin belirlediği bölge 1'e çekilir. Bu sayede güncellenmiş adresi belirtmiş oluruz. Dirty biti ana hafızadan önbelleğe çekilen verinin değiştirilip değiştirilmediğini anlatır. Eğer gelen adres bölgesinde dirty biti 1 ise o adreste modifiye edilmiş bir veri vardır ve ana hafızada güncellenir. Eğer önbellekteki her 2 hafıza birimi için sorgu yapılır, tagler uyuşmaz veya onay biti 0 ise hit verilmez ve durum bir sonraki aşamaya geçer. Eğer yazma işleminde hit verilmezse HAFIZA_YAZ durumuna geçer. Eğer veri çekme işlemi var ve hit verilmezse HAFIZA_CEK durumuna geçer. Bu 2 durumda da stall sinyali 1'e çekilir ve çekirdek durdurulur. HAFIZA_YAZ durumunda iomem_valid biti 1'e çekilir ve aynı anda iomem_addr değerine çekirdekten gelen address,

iomem_wstrb değerine çekirdekten gelen yazma komutu, iomem_wdata'ya ise çekirdekten gelen kaydedilecek data aktarılır. Bu sırada çekirdekten iomem_ready sinyali beklenir. Eğer iomem_ready sinyali 1 olursa yazma işlemi başarılı olmuş olur. iomem_valid ve diğer sinyaller 0'a çekilir. Bir döngü sonrası durum BITTI aşamasına geçer. Bu aşamada stall 0'a çekilir ve çekirdeğin durdurma işlemine son verilir. Eğer HIT aşamasından HAFIZA_CEK aşamasına geçilirse ilk başta LRU değerlerinin kıyaslanmasına bakılır. LRU 2 bitlik 256 genişliğinde veri hafızasıdır. LRU (Least Recently Used) verinin kullanımını takip eder. Tag ve Dirty gibi adres yapısına sahip 2 bitlik veri tutar. 2 farklı ram yapısında hangi verinin daha fazla kullanıldığını yada daha az kullanıldığını tespit etmek için tasarlanılmıştır. Durum HIT aşamasında hit0 1'e yükseltildiğinde ram0 verisi kullanılacağı için lru1 1 ile toplanarak kendisine eklenir. Tam tersi olarak hit1 'e yükseldiğinde bu sefer lru0 1 ile toplanarak kendisine eklenir. Bunun anlamıda lru0'ın daha az kullanıldığı anlamına gelir. LRU değeri ne kadar büyükse veri o kadar az kullanıldığı anlamına gelir. Daha önce bahsedilen HAFIZA_CEK aşamasına tekrar geri dönecek olursak. Bu aşamada lrulardan lru0 veya lru1 arasında kontrol yapılır hangisi az kullanıldığı öğrenilmesi için. Yeni çekilecek olan veri az kullanılan ram hafızası bölgesinin üstüne yazılacaktır. Bu sayede yeni çekilen verinin daha fazla kullanılacağı düşünüleceği için sistem bu şekilde işlemektedir. Eğer az kullanılan veri çekirdek tarafından modifiye edilip dirty bitide 1 ise bu sefer yeni veri çekilmeden az kullanılan modifiye edilmiş veri ana hafızaya yazılarak güncellenir. Tasarlanan veri önbelleği bu şekilde çalışmaktadır.

4.2.4. Interconnect (Dahili Bağlantı)

Ana hafıza hattının sadece tek bir veri yolu girişi ve çıkışı olduğu için buyruk önbelleği ve veri önbelleği gibi 2 farklı önbellek ile haberleşebilmesi için interconnect adında bir mux. sistemi kuruldu. Başlangıç olarak bütün mux.lar BUYRUK modunda çalışır. Sisteme veri önbelleği istek attığında buyruk işlemi bitene kadar VERI moduna geçmez. Buyruk işlemi bittiğinde sistem veri moduna geçerek bütün mux.lar verinin sinyallerini anabelleğe iletir.

```
module interconnect(
        input           clk,
        //-----------buyruk onbellegi--------------
        input  [31:0]    buyruk_addr_i  ,
        input            buyruk_valid_i ,
        output [31:0]    buyruk_data_o  ,
        output           buyruk_ready_o ,

        //-----------------------------------------
        //----------veri onbellgi--------------------
        input  [31:0]   veri_addr_i     ,
        input  [3:0]    veri_wstrb_i    ,
        input           veri_valid_i    ,
        output          veri_ready_o    ,
        output [31:0]   veri_read_data  ,
        input  [31:0]   veri_write_data ,

        //-----------------------------------------

        output          iomem_valid_o  ,
        input           iomem_ready_i  ,
        output [3:0]    iomem_wstrb_o  ,
        output [31:0]   iomem_addr_o   ,
        output [31:0]   iomem_wdata_o  ,
        input  [31:0]   iomem_rdata_i

);
```

Şekil 4.38. İnterconnect'in Verilog giriş çıkış görüntüsü.

4.2.5. Sbox

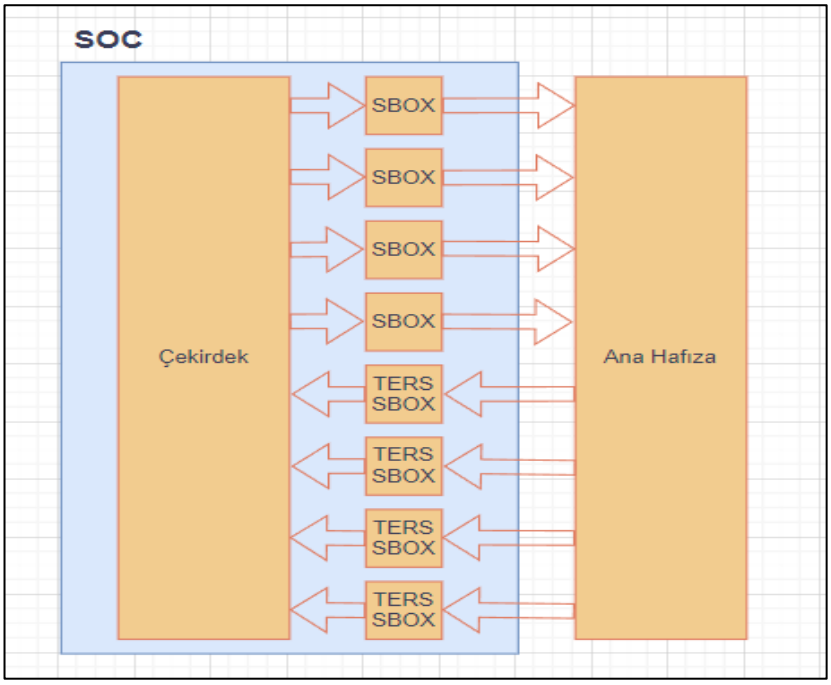
Ana hafızaya veri yollanırken veya veri çekilirken verinin çözülüp şifrelenmesi gerekir. Onun için alttaki S-box sistemi Teknofest pdf'inden alınmış ve çekirdeğe uygunlanmıştır [44].

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	1	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	4	c7	23	c3	18	96	5	9a	7	12	80	e2	eb	27	b2	75
40	9	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	0	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	2	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	6	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	8
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	3	f6	0e	61	35	57	b9	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Şekil 4.39. Sbox.

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
10	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
20	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
30	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
40	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
50	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
60	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
70	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
80	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
90	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a0	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b0	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c0	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d0	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e0	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f0	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Şekil 4.40. Ters Sbox.

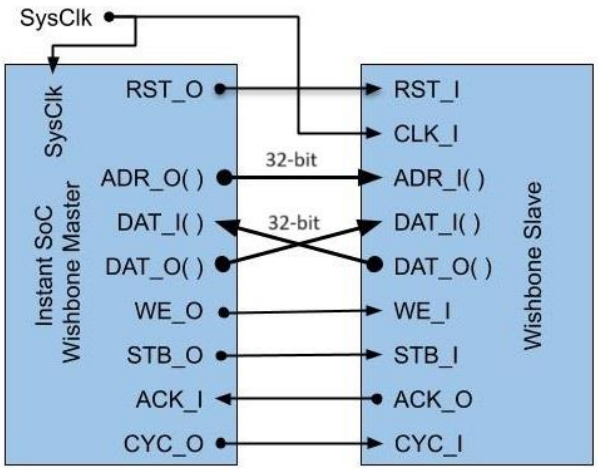


Şekil 4.41. SBOX bağlantısı.

Şekil 4.41'da görüldüğü gibi bağlanmıştır. Her S-box 8 bitlik işlem yapıyor. Bundan dolayı şifreleyip çözen toplamda 8 tane S-box sistemde bulunmaktadır.

4.2.6. Wishbone Bus (Wishbone Veriyolu)

Çevre birimleri ile haberleşmesi için Wishbone Bus kullanılmıştır. Bir tane Wishbone Bus master ve UART, SPI ve PWM için slave modülleri kullanılmıştır. Wishbone bus modüller arası iletişimi sağlayan standartlaştırılmış bir protokoldür.



Şekil 4.42. Wishbone sinyal bağlantısı gösterimi.

Şekil 4.44'deki referans olarak gösterilen sinyal bağlantısı kullanılmıştır.

4.2.7. Çevre Birimleri

Tasarlanan işlemciye UART, SPI ve PWM olmak üzere 3 farklı çevre birimi eklenmiştir. Bu çevre birimleri referansta belirtilen github sayfasından alınmıştır. [46] Tasarladığımız çekirdeğe uygun bir şekilde eklenmiştir. Bu çevre birimleri WISHBONE bus yapısı çekirdek ile haberleştirilmesi sağlanmıştır. Her çevre biriminin wishbone busa sahip slave modülü vardır. Çevre birimlerinin bellek haritası TEKNOFEST yarışmasına göre belirlenmiştir.

4.2.7.1. UART

İşlemciye RX ve TX'e sahip UART modülü eklenmiştir. UART modülü çekirdek ile haberleşebilmesi için WISHBONE BUS kullanılmıştır.

Çizelge 4.4. UART bellek haritası.

Adres	İsim	Açıklama
0x20000000	uart_ctrl	Kontrol registerı
0x20000004	uart_status	Durum registerı
0x20000008	uart_rdata	Veri okuma registerı
0x2000000c	uart_wdata	Veri yazma registerı

İşlemcinin UART bellek haritası Çizelge 4.4'deki gibidir. uart_crtl, uart_status, uart_rdata, uart_wdata 4 farklı register yapısına sahiptir. Bu register yapılarının bazılarıda sadece okuma yapılıyorken diğerlerinde hem okuma veya hem yazma yapılabilmektedir.

UART Kontrol Kayıtçısı

Uart kontrol yazmacı sistemin tx ve rx özelliğini aktif eden ve baudrate ayarının yapıldığı yerdir. uart_ctrl 32 bite sahip bir yazmaçtır. Belli bit aralıkları bu özellikleri aktif veya pasif etmesini sağlamaktadır.

Çizelge 4.5. uart_ctrl'nin bitleri.

uart_ctrl			
Bitler	**Tip**	**İsim**	**Açıklama**
0	Okuma/Yazma	tx_en	Tx'i etkinleştirir.
1	Okuma/Yazma	rx_en	Rx'I etkinleştirir.
15:2			Boş Alan
31:16	Okuma/Yazma	baud_div	Baudrate girilir.

Çizelge 4.5'de belirtilen bitlerin özellikleri sırasıyla aşağıdaki gibidir;

- 0'ıncı bit aktif olursa tx_en aktif olur ve sistemde uart_tx modülü veri yollamaya başlar. Fifo boş değilse fifo içindeki verileri sırayla yollamaya başlar.
- 1'inci bit aktif olursa rx_en aktif olmuş olur. uart_rx modülü dışarıdan veri dinlemeye başlar.
- 15:2'inci bit aralığı boş alandır herhangi bir işlevi yoktur.
- 31 ile 16'ıncı aralık sistemi hangi baudrate aralığında çalışacağını belirten aralıktır.

$$f_{buad} = \frac{f_{clk}}{baud_{div} + 1} \tag{4.1}$$

Yukarıdaki formüle göre baudrate hesabı yapılmakta assembly ile uart programlanacaksa bu formüle göre baudrate değeri hesaplanması gerekmektedir.

UART Durum Kayıtçısı

Uart Durum Registeri çekirdekten sadece okuma yapılmasını sağlayan registerdir. RX ve TX modüllerinin fifolarının boş olup olmadığını sorgular. Çekirdek 0x20000004 olan adrese okuma isteği attığında bu registerdan uart fifosunun durumu hakkında sorgu verileri çekirdeğe yollanır.

Çizelge 4.6. UART durum registerları.

uart_status			
Bitler	**Tip**	**İsim**	**Açıklama**
0	Sadece Okuma	tx_full	Tx buffer dolu
1	Sadece Okuma	tx_empty	Tx buffer boş
2	Sadece Okuma	tx_full	Rx buffer dolu
3	Sadece Okuma	tx_empty	Rx buffer boş

UART Veri Okuma Kayıtçısı

Dışarıdan veri geldiğinde bu veriler fifoda birikir bu verilerin çekirdek tarafından çekilmesini sağlayan registerdır. Uart_rx'in fifosu ile doğrudan bağlantısı vardır. Çekirdek 0x20000008 adresine çekme isteği attığında bu registerdan veri okuma işlemi yapılmaktadır.

Çizelge 4.7. UART veri okuma registerı.

uart_rdata			
Bitler	**Tip**	**İsim**	**Açıklama**
31:0	Sadece Okuma	rdata	Okunacak veri

UART Veri Yazma Kayıtçısı

Çekirdekten UART tx fifosuna veri yazılmasını sağlamak amacıyla kullanılan registerdır. 0x2000000c adresinde bulunur bu address geldiğinde çekirdekten gelen veri bu registera yazılıp daha sonra bu registerdan uart_tx fifosuna kaydedilir.

Çizelge 4.8. Uart veri yazma register'ı.

uart_wdata			
Bitler	**Tip**	**İsim**	**Açıklama**
31:0	Sadece Yazma	wdata	Yazılacak veri

4.2.7.2. SPI

İşlemci çekirdeğinin farklı haberleşme protokollerine sahip olması için eklenmiştir. 5 farklı registera sahiptir. Çizelge 4.9'da belirtildiği gibi bir bellek haritası vardır.

Çizelge 4.9. SPI bellek haritası.

Adres	İsim	Açıklama
0x20010000	spi_ctrl	Kontrol registerı
0x20010004	spi_status	Durum registerı
0x20010008	spi_rdata	Seri alma registerı
0x2001000c	spi_wdata	Seri iletim registerı
0x20010010	spi_cmd	Komut registerı

Çizelge 4.9'de görüldüğü gibi belirli adres değerlerinde belli registerlara ulaşıldığı görülmektedir. Bu registerlar SPI protokolü ile alakalı işlemler yapacağımız sırada yardımcı olacak olan register adresleridir. Tasarlanan işlemcimize eklenen SPI'ın birçok farklı modu ve özelliği vardır. Bunlar sırasıyla altbaşlık şeklinde açıklanacaktır.

SPI Kontrol Registerı

SPI kontrol registerının SPI'ı açıp kapama veya çalışma modlarını aktif edip kapatma gibi işlemleri barından registerlardır. Her biti farklı özelliği aktif eder. Wishbone bus ile registerlara veri yazılmaktadır.

Çizelge 4.10. SPI kontrol registerı.

spi_ctrl			
Bitler	**Tip**	**Isim**	**Açıklama**
0	Okuma/Yazma	spi_en	SPI etkinleştirme
1	Okuma/Yazma	spi_rst	SPI sıfırlama
2	Okuma/Yazma	cpha	Seri saat fazı
3	Okuma/Yazma	cpol	Seri saat polaritesi
15:4			Boş Alan
31:16	Okuma/Yazma	sck_div	Sck_kontrol

Çizelge 4.10'da SPI kontrol registerın bitlerinin özellikleri görülmektedir.

SPI Statü Registerı

SPI statü register, sadece okuma özelliğine sahip register türüdür. Bütün bitleri SPI'ın içinde olan modüllerinin fifosunun dolu olup olmadığını anlamak için vardır. Çizelge 4.11'da, bütün bitlerinin özellikleri yazmaktadır.

Çizelge 4.11. SPI statü registerları.

Spi_status			
Bitler	**Tip**	**İsim**	**Açıklama**
0	Sadece Okuma	Mosi_full	Mosi buffer dolu
1	Sadece Okuma	Mosi_empty	Mosi buffer boş
2	Sadece Okuma	Miso_full	Miso buffer dolu
3	Sadece Okuma	Miso_empty	Miso buffer boş
4	Sadece Okuma	Cmd_full	Cmd buffer dolu
5	Sadece Okuma	Cmd_emtpy	Cmd buffer boş

SPI Okuma Registerı

SPI okuma registerı dışarıdan okunan ve daha sonra fifoya kaydedilen verileri çekirdeğe yollamak için olan registerdır. Çekirdek 0x20010008'inci adresden veri çekme isteği yolladığında bu register aracılığı ile veriler çekirdeğe iletilmektedir. Çizelge 4.12'da registerın özellikleri yazmaktadır.

Çizelge 4.12. SPI okuma registerı.

spi_rdata			
Bitler	**Tip**	**İsim**	**Açıklama**
31:0	Sadece Okuma	rdata	Okunan veri

SPI Yazma Registerı

SPI yazma register, MOSI ile yollanacak verileri ilk başta fifoya kaydetmek için çekirdek arasında iletişimi sağlayan registerdır. Çekirdek 0x2001000c adresine yazma

komutu yolladığında ilk başta bu registera kadedilir daha sonra fifoya yazılır. Çizelge 4.13'da registerın özellikleri yazmaktadır.

Çizelge 4.13. SPI yazma registerı.

spi_wdata			
Bitler	**Tip**	**İsim**	**Açıklama**
31:0	Sadece Yazma	wdata	Yazılacak veri

SPI Komut Registerı

SPI komut registerında çift yönlü çalışma kontrol edilir. Bu registera gelen komuta göre veri transferleri gerçekleşmektedir.

Length bitleri alışveriş yapılacak veri boyutunu byte cinsinde 1 fazlası olacak şekilde belirtir. Daha iyi haberleşme sağlanması için veri transferinden sonra çip seçimi ayarlanır. direction biti ise veri transferinin yönünü belirtir. 0 ise boş döngü, 1 ise miso ve 2 ise mosi olacak şekildedir [46].

Çizelge 4.14. SPI komut registerı.

spi_cmd			
Bitler	**Tip**	**İsim**	**Açıklama**
8:0	Okuma / Yazma	length	Transfer byte miktar
9	Okuma / Yazma	cs_active	Transferden sonra çip seçme aktif
13:12	Okuma / Yazma	direction	Transfer yönü

4.2.7.3. PWM

İşlemciden pwm siyali alabilmek için pwm modülü eklenmiştir. Wishbone bus protokolüne bağlı olaran registerların aşağıdaki Çizelge 4.15'de adresleri görülmektedir [46].

Çizelge 4.15. PWM bellek haritası.

Adres	İsim	Açıklama
0x20020000	pwm_control	PWM kontrol registerı
0x20020008	pwm_period	PWM periodunu belirler
0x20020010	pwm_threshold	PWM eşik değeridir
0x20020020	pwm_step	PWM adım değeridir.

PWM Kontrol Registerı

PWM kontrol register pwm aktif etmek veya kapatmak için koyuların bir registerdır. İlk bitinin 1 veya 0 olmasına göre pwm açılıp kapanır.

Çizelge 4.16. PWM kontrol register.

pwm_control	
Bitler	**Açıklama**
1	Pwm_aktif

PWM Sinyal Ayarlama Registerları

Buradaki registerlar PWM sinyalinin yükselen kenar alçalan kenar oranlarını ayalarmak için konulmuş registerlardır. Aşağıdaki çizelgelerde bu registerların adlarını ve özelliklerini görülmektedir.

Çizelge 4.17. PWM periyot register.

Pwm_period	
Bitler	**Açıklama**
31:0	PWM sinyalinin periodunu belirler

Çizelge 4.18. PWM eşik değeri register.

Pwm_treshold	
Bitler	**Açıklama**
31:0	PWM sinyalinin eşik değerini belirler.

Çizelge 4.19. PWM adım register.

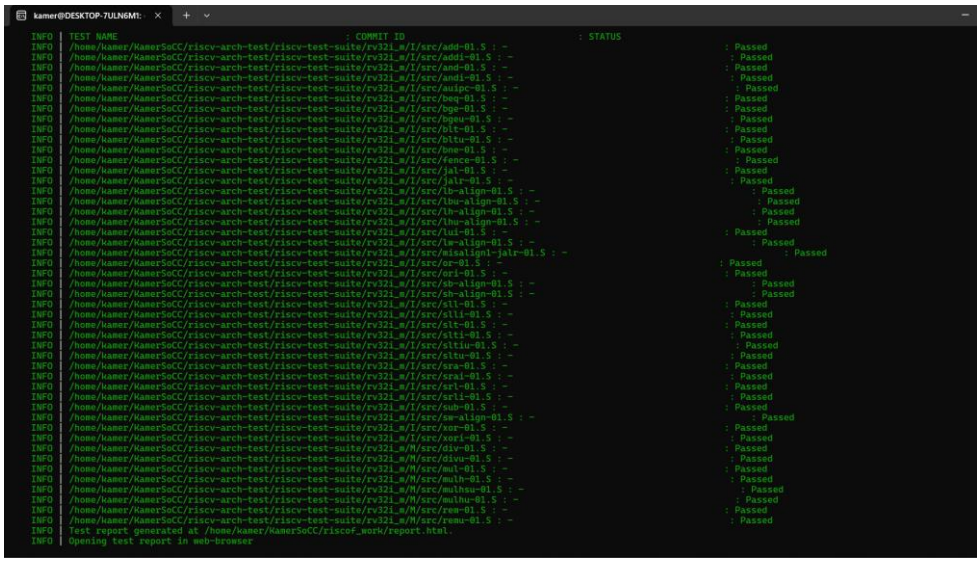
Pwm_step	
Bitler	**Açıklama**
31:0	PWM sinyalinin adım sayısını belirler.

BÖLÜM 5

DENEYSEL SONUÇLAR

Bu bölümde tasarlanan işlemcinin Vivado simülasyonu gibi çeşitli yollarla ve kodlarla hatalarının ayıklanmasından ve testlerinden bahsedilecektir. Çalışamalar genellikle, Vivado simülasyonu yada Zybo Z7-20 FPGA kartı ile test edilmiştir.

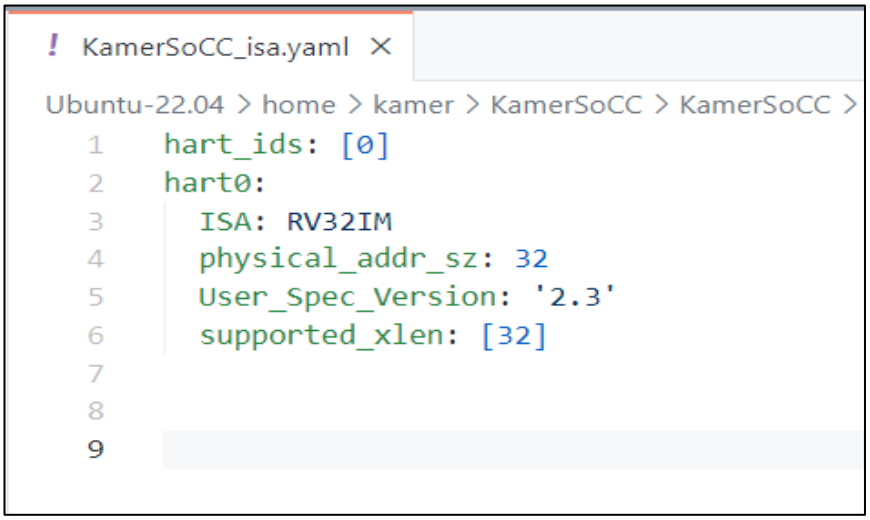
5.1. RISCOF TESTİ



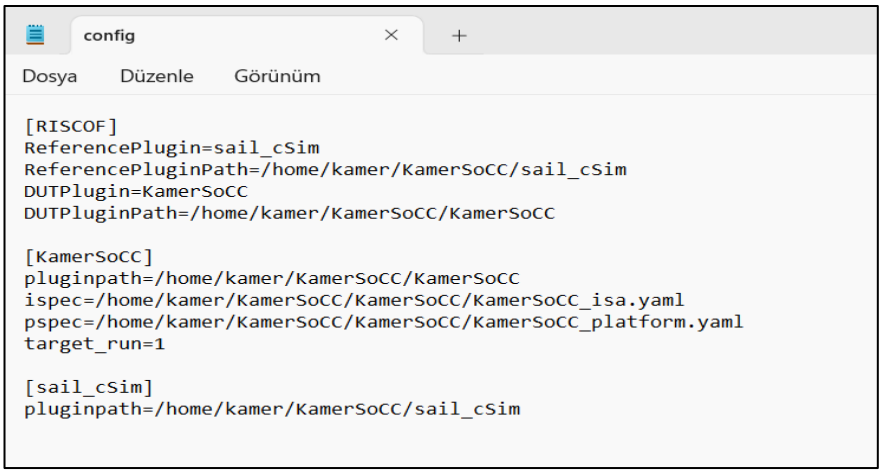
Şekil 5.1. RISCOF sonucu.

Ubuntu ortamına RISCOF kurulmuştur. RISCOF ortamını kurma aşamasında referanstaki çekirdeğin dosyaları incelenmiştir [47]. Ubuntu içine KamerSoCC diye dizin açılıp uygun komutlarla sistem o dizine kurulmuştur. İşlemcinin çekirdeği, oluşturduğumuz dizine önbelleksiz olarak aktarılmıştır. Teknofest yarışmasındaki wrapper'a çekirdekten gelen kayıt verilerini dışarı signature uzantılı dosyası şeklinde yazması için kod eklenmiştir. Wrapper'a bağlı işlemci çekirdeğini çalıştırmak için sisteme iverilog yazılımı kurulmuştur. Sistemin çekirdeğinin RISCOF'dan gelen test komutlarını derlemesi için Makefile dosyası oluşturulmuştur.

YAML dosyaları RV32IM olacak şekilde ayarlanmıştır. YAML dosyaları içinde CSR'ler ile alakalı ayarlar varsa silinmiştir.



Şekil 5.2. RISCOF ISA YAML dosyası.

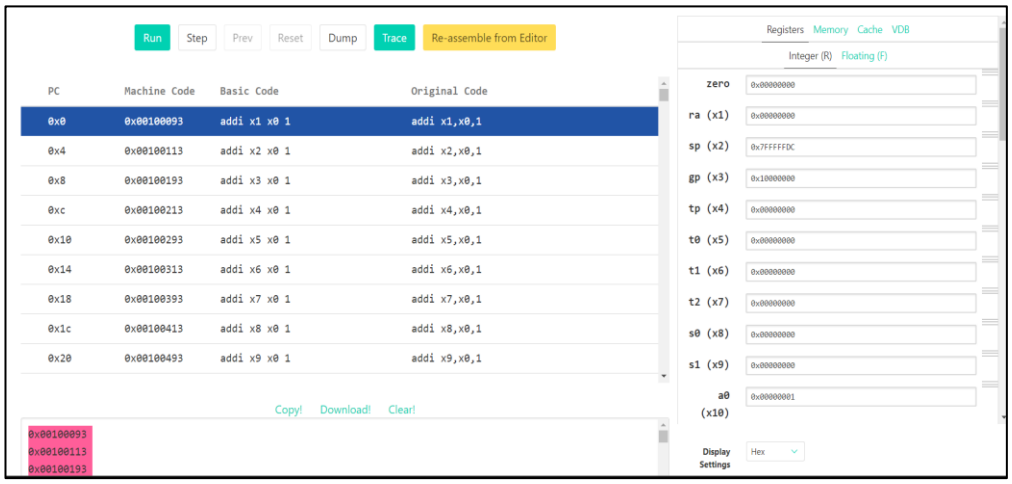


Şekil 5.3. RISCOF Config dosyası.

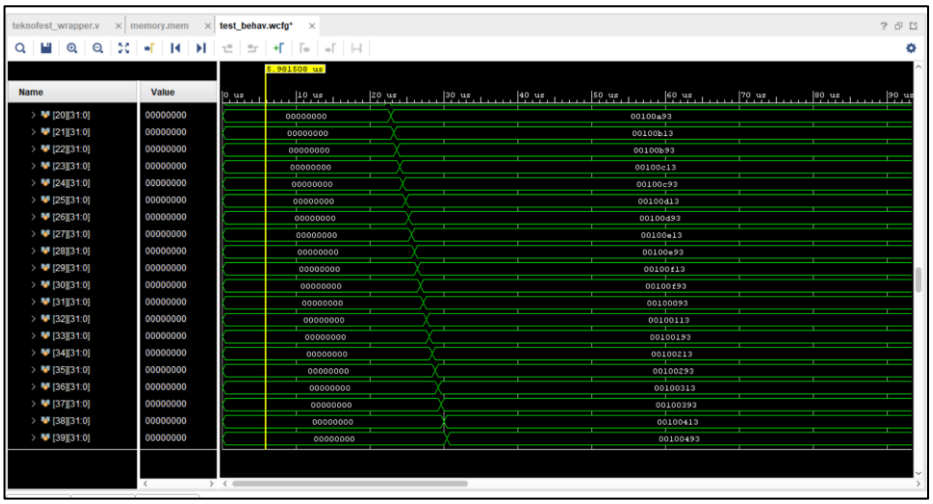
Şekil 5.3'e göre Config dosyası ayarlanmış kendi çekirdeğimizin yolu ve referans işlemcinin yolu yazılmıştır. Python dosyası ile bütün işlemin çalıştırılma sıralaması ayarlandığında sistem başlatılmıştır. Şekil 5.1'de görüldüğü gibi RISCOF testi sonucunda bütün buyruklar testi geçmeyi başarmıştır. Bu test sonucunda işlemcilerde belli hazardlar varsa RISCOF içinde bu hatalar anlaşılmayabilir. Bu test sonrası buyruklarda problem görülmemiştir.

5.2. VİVADO SİMÜLASYONLARI

Bu aşamada Vivado aracının simülasyon kısmı kullanılarak işlemcinin testleri gerçekleştirilmeye çalışılmıştır. Karşılaştırılmalar genellikle Venus ile yapılmıştır. C gibi diller kullanılacağı zaman karşılaştırılma işlemleri SPIKE adında UBUNTU sisteminde çalışan simülasyon programı ile yapılmıştır.

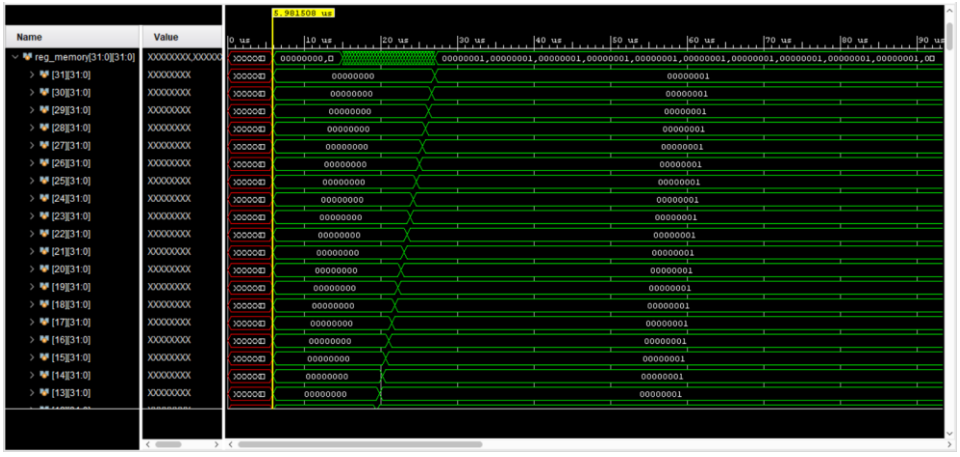


5.4. Venus ile önbellektest kodu.

- 

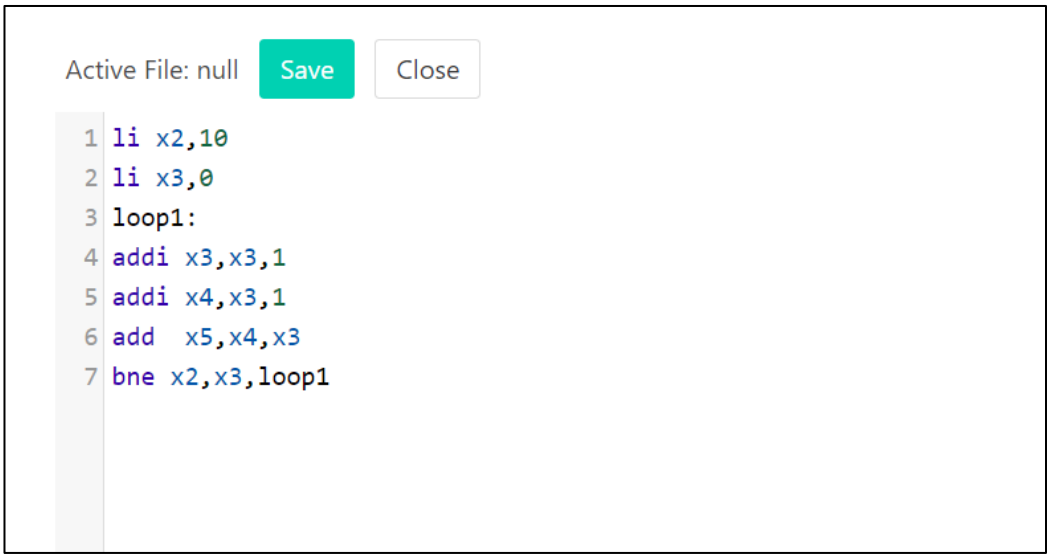
5.5. Vivado buyruk önbelleği simülasyonu.

Buyruk önbelleğinin, bütün sıralı bir şekilde yazılmış ADDI komutlarını doğru bir biçimde çekip önbelleğin kendi bellek hafızasına doğru bir şekilde yazıp yazmadığını kontrol edilmiştir. Sistemin doğru çalıştığı görülmüştür.



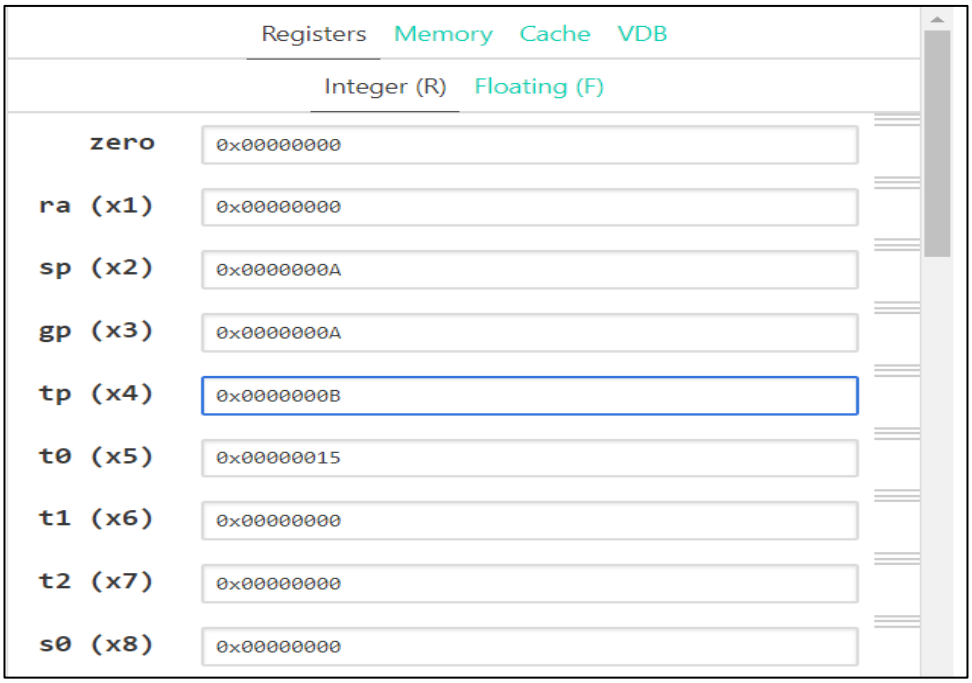
5.6 Vivado Register File kontrolü.

Aynı buyruk kümesi ile register file içinede bakılmış olup 1 sayısının doğru yazılıp yazılmadığı kontrol edilmiştir. Register file içinde bir problem bulunmamıştır.



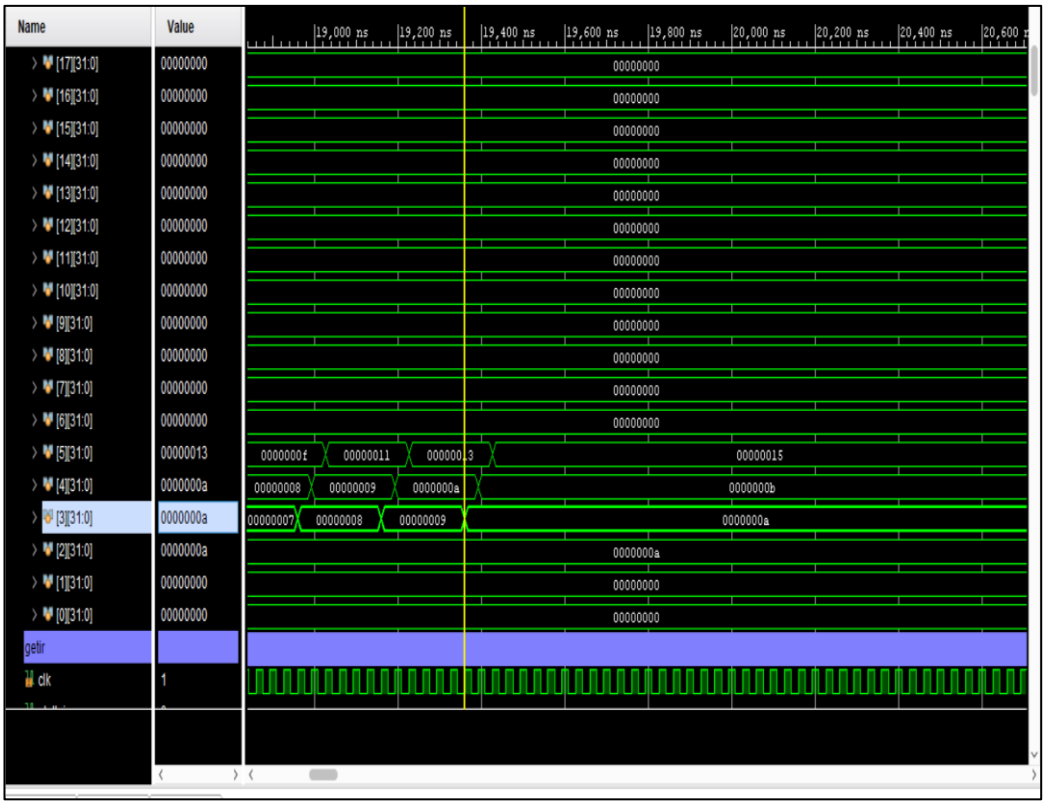
Şekil 5.7. Venus veri tehlikesi için yazılmış kod.

Burada Şekil 5.7'de görülen kod, veri tehlikesi için özel yazılmış bir koddur. Buradaki olay daha verilerin boru hattı aşamasında geri yazılmadan tekrar başka buyruk tarafından kullanılmaya çalışıldığında, denetim_birimin'de programlanmış geri besleme biriminin doğru çalışıp çalışmadığı kontrol edilmek istenmiştir.



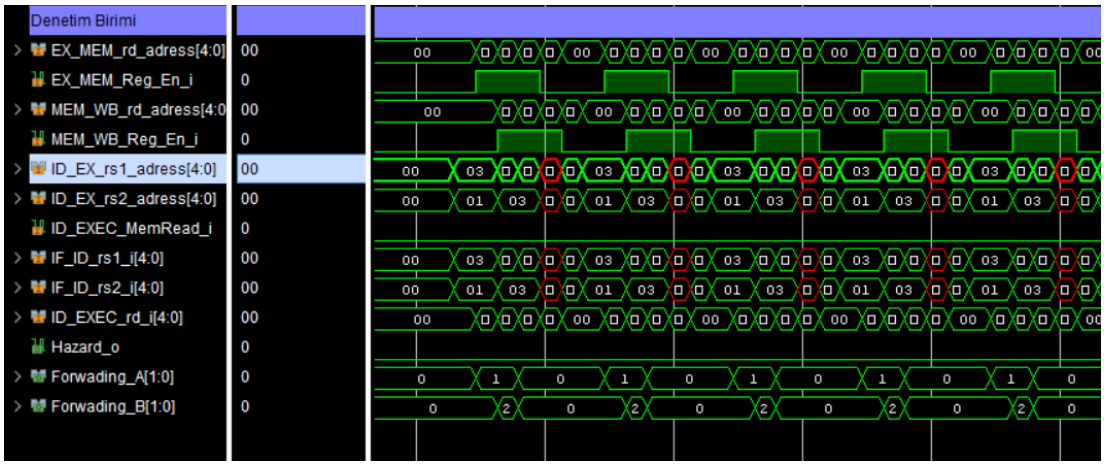
Şekil 5.8. Veri tehlikesi için yazılmış kodun Venus'teki testi.

Şekil 5.8'daki referans işlemcide görüldüğü gibi register sonuçları elde edilmiştir.



Şekil 5.9. Vivado Register File sonucu.

Şekil 5.9'de veri tehlikesinin oluşturabileceği hataların engellenmesinin başarılı olduğu görülmüştür. Tasarlanmış denetim_birimi'ndeki geri besleme modülünün doğru çalıştığı anlaşılmıştır.



Şekil 5.10. Denetim Birimi.

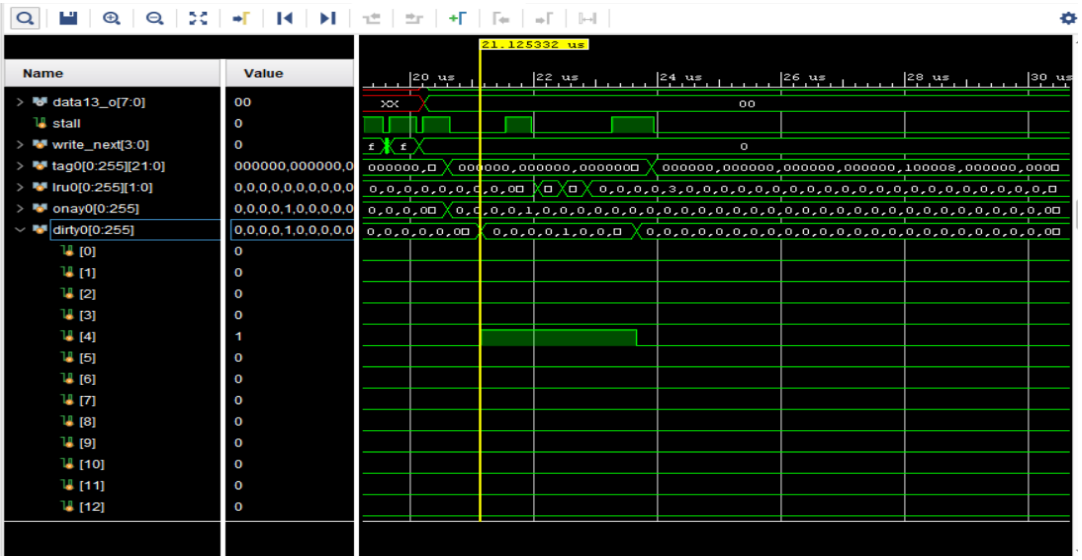
Şekil 5.10'de denetim_birimi'nin geri besleme yapılması gereken verilerin geri besleme mux.larının ayarlandığı görülmektedir. Forwarding_A ve Forwarding_B mux.larının 0'dan 1'e ve 0'dan 2'ye döndükleri görülmektedir. Bu Decode aşamasındaki forwarding mux.larını etkileyecek sinyallerdir. Sistem düzgün çalışmaktadır.

Active File: null Save Close

```
## bu veriler ilk başta hit olmadığı için Ana hafızaya yazılacak
li x2,0x40000010
li x3,0x40001010
li x4,0x40002010
li x5,1
li x6,2
li x7,3
sw x5,0(x2)
sw x6,0(x3)
sw x7,0(x4)
## ana hafızadan indexleri aynı olduğu için
## önbelleğin 2 yolunada kaydedilmesi sağlanacak.
lw x8,0(x2)
lw x8,0(x2)
sw x7,0(x2)
lw x9,0(x3)
lw x9,0(x3)
lw x9,0(x3)
lw x9,0(x3)
lw x9,0(x4)
## Bu yere kadar 1 yoldaki veri çekirdek tarafından değiştirilerek dirty biti aktif edilecek ve 2 yoldaki
## veri daha fazla kullanılarak dirty biti olan veri az kullanılmış olacak ve ana hafızaya yazıdırılıp
## dirty biti düşünürülmesi sağlanacaktır
```

Şekil 5.11 Veri önbelleği test kodları.

Şekil 5.11'daki yazılan test kodlarında, veri önbelleğinin 2 yoluna da veri yazılması amaçlanmıştır. Sistem şu şekilde çalışmaktadır. 1. yola veri çekilecek ve o veri kullanılarak 2. yolun LRU'su arttırılacak, daha sonra yeni çekilecek aynı indexe sahip verinin 2. yola kaydedilmesi sağlanmaktadır. Daha sonra 2. yola yeni veri kaydedildikten sonra 1. yolun aynı index adresindeki verinin değeri değiştirilerek kirli biti aktif hale getirilmiştir. Bu sayede yeni veri çekileceği zaman önbellekteki güncellenmiş verinin ana hafızaya atılıp yeni gelecek verinin üzerine yazılması sağlanmıştır.



Şekil 5.12 Veri önbelleğinin dirty0 biti.

Şekil 5.13. lru0'ın artması.

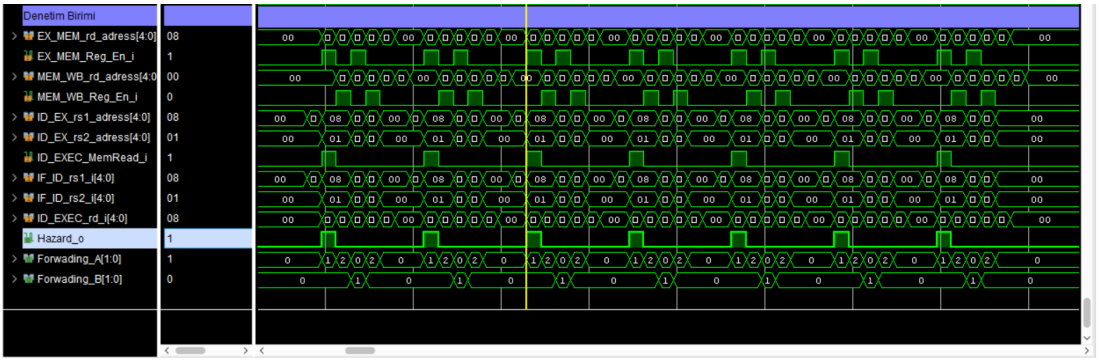
Şekil 4.12 ve Şekil 4.13'e göre 1. yolda veri kullanılmadığı 2. yoldaki veri kullanıldığı için ilk yolun LRU'su artmakta ve ilk yolun veri değeri önceden değiştirildiği için dirty0 1'e çekilmektedir. Daha sonra aynı indekse sahip ama etiketi farklı bir veri geldiğinde 1. yolun LRU değeri daha fazla olduğu için yeni gelen veri ilk yoldaki veri ile değişmesi gerekmektedir. Güncel veri ilk başta ana hafızaya atılır. dirty0 biti Şekil 4.10'da olduğu gibi 0'a çekilir. Yeni veri 1. yola kaydedilir.

Load-Use adında başka bir veri tehlikesi vardır. Execute aşamasında ALU, MUL ve DIV dışında sonuç veren MEMORY adında bir modül vardır. Bu modül veri önbelleğinden gelen verileri çeker ve bir sonraki aşamaya aktarır. Fakat ALU, MUL ve DIV ile aynı mux.a bağlı değildir. Ortak mux.un çıkışı ise geri beslemeye bağlıdır. Eğer EXECUTE aşamasında LW buyruğu varsa ve bu sırada LW buyruğundan çekilen veri bir sonraki buyruğun ihtiyacı olan veriyse geri besleme sırasında çekilen veri geri beslenemez onun yerine çekilen verinin adresi Decode aşamasına geri beslenir buda bir hata sebebidir. Bunu düzeltmek için boru hattı mimarisine iki buyruk arasında bir boşluk yapmak gerek.

```
1
2  li x31,10
3  li x2,0x40000010
4  li x3,0x40001010
5  li x4,0x40002010
6  li x5,1
7  sw x5,0(x2)
8
9  loop:
10 lw x8,0(x2)
11 addi x8,x8,1
12 sw x8,0(x2)
13 bne x8,x31,loop
14
15
16
```

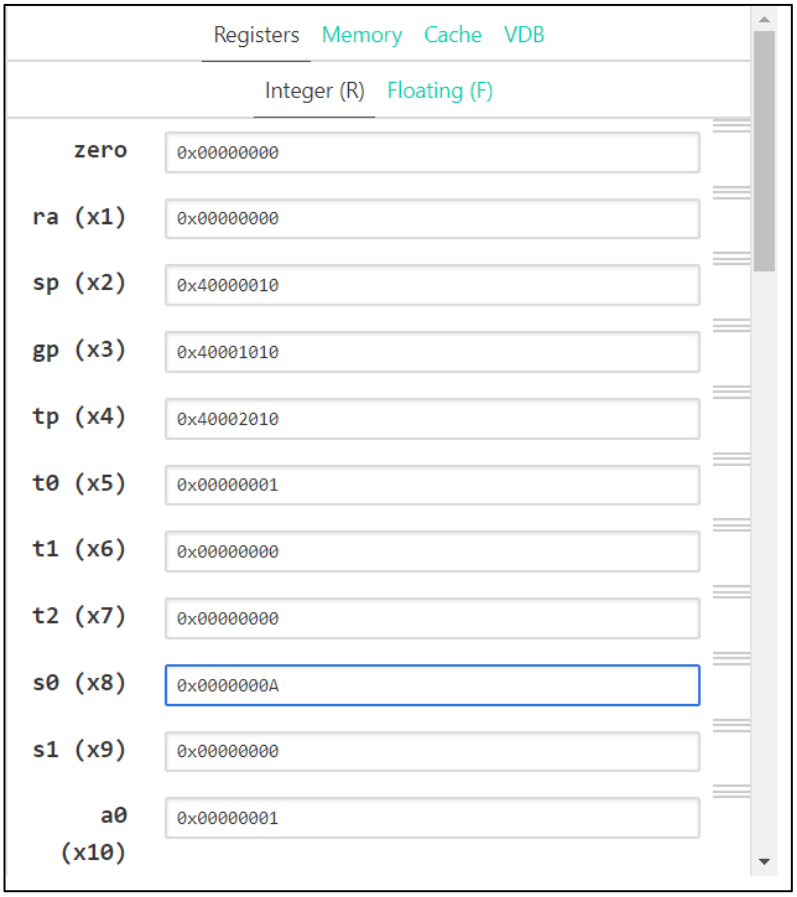
Şekil 5.14. Load-Use tehlikesi için yazılmış kod.

Tasarlanan denetim_biriminin bu veri tehlikesini engelleyip engellemediğini görmek için yukarıdaki gibi kod yazılmıştır.

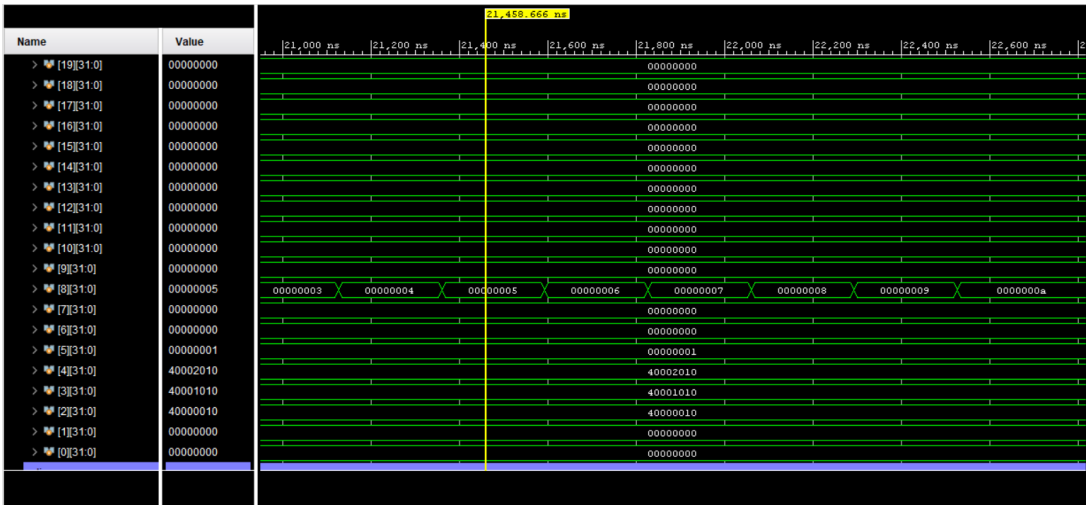


Şekil 5.15. Vivado simülasyonunda Hazard_o çıkışı.

Şekil 5.15'deki sonuca göre Hazard_o sinyali istenilen değerlerde yükselmektedir. Herhangi bir problem görülmemektedir.



Şekil 5.16. Venus Register File simülasyon çıktısı.



Şekil 5.17. Vivado Register File simülasyonu.

İşlemcimiz Vivado simülasyonundaki değerler ile referans işlemcideki registerların değerlerinin birbiriyle uyuştuğu gözlemlenmiştir. Herhangi bir problem bulunmamıştır.

Ana hafızaya kod atarken SBOX olduğundan dolayı ilk başta verileri programladığımız python kodu ile şifreliyoruz.

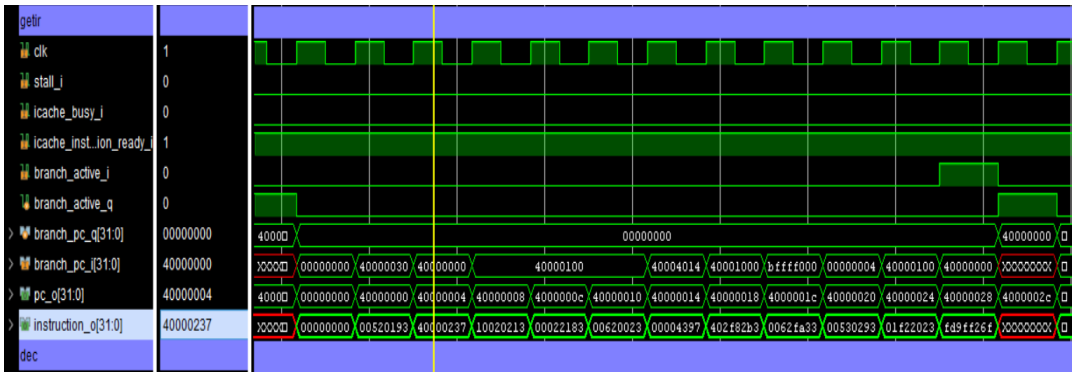
```
00520193
40000237
10020213
00022183
00620023
00004397
402F82B3
0062FA33
00530293
01F22023
FD9FF26F
```

Şekil 5.18 Şifrelenilecek hexidecimal sayılar.

```
63007CDC
0963779A
CA77777D
6377FDEC
63AA6326
63631A88
0915136D
63AA2DC3
63ED77DC
7C89B726
54DB89A8
```

Şekil 5.19. Şifrelenmiş hexidecimal sayılar.

Şekil 5.18'daki hexadecimal buyruklar, programladığımız python kodu ile şifrelenerek Şekil 5.19'deki gibi şifrelenmiş hali görülmektedir. Bu şifrelenmiş hexadecimal kodlar, Vivado simülasyonunda ana hafızaya yüklenerek çekirdeğin Fetch aşamasındaki hali takip edilmiştir.

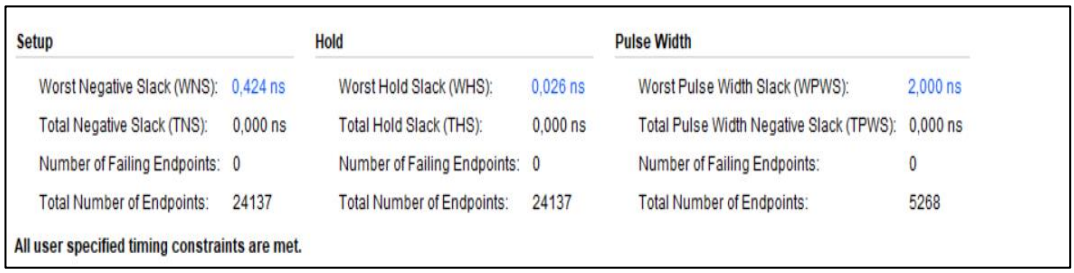


Şekil 5.20. FETCH aşaması Vivado simülasyonu.

Şekil 5.20'de görüldüğü gibi şifrelenmiş hexidecimal sayıların FETCH aşamasına çözülerek geldiği görülmüştür. SBOX sisteminde herhangi bir problem görülmemiştir.

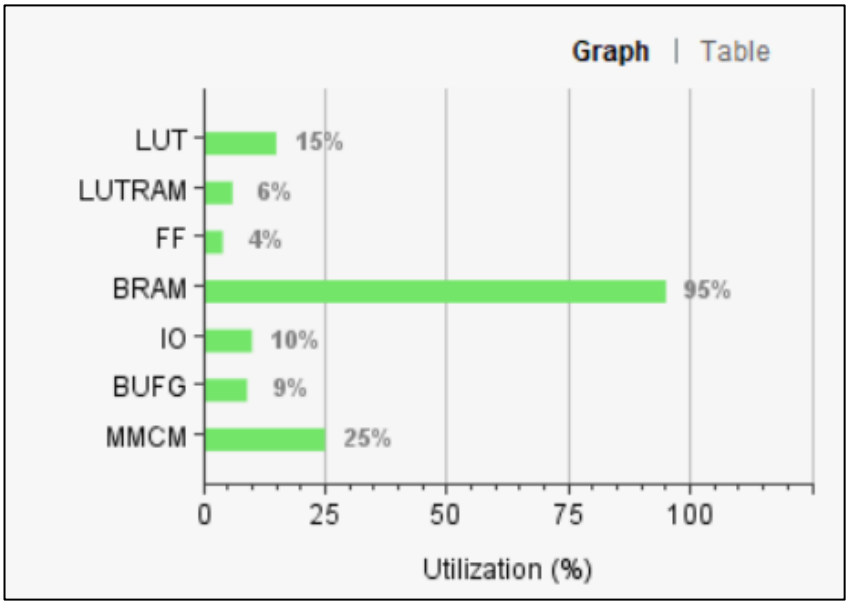
5.3 VİVADO SENTEZ VE İMPLEMENTASYON SONUÇLARI

Tasarlanan işlemcinin clock girişi yerine Clock Wizard eklenmiştir. Sistemin hızı 125 Mhz'den 40 Mhz'e düşürülmüştür. Bunun sebebi 40 MHz'den yüksek bir frekans denersek Timing Fail hatası almamızdır. Sistemin sentez ve implementasyonu 40 MHz ile başlatılmıştır.



Şekil 5.21. Timing sonuçları.

Şekil 5.21'da görüldüğü gibi 40 MHz'de herhangi bir timing problem ile karşılaşılmamıştır.



Şekil 5.22. Zybo-Z7-20 kartından harcanan kaynakların yüzdelik oranı.

Resource	Utilization	Available	Utilization ...
LUT	7918	53200	14.88
LUTRAM	984	17400	5.66
FF	3995	106400	3.75
BRAM	133	140	95.00
IO	12	125	9.60
BUFG	3	32	9.38
MMCM	1	4	25.00

Şekil 5.23. Zybo-Z7-20 kartından harcanan kaynaklar.

5.4. FPGA KARTINDA YAPILAN TESTLER

Tasarladığımız işlemcimiz, FPGA kartında test yapılabilmesi için bitstream yapılmış olup karta aktarılmıştır. Bu sırada işlemcinin dışardan UART ile buyruk alabilmesi için bir Teknofest yarışmasında kullanılan wrapper alınmıştır. GCC kullanılırken ise Teknofest baremetal ve CoreMark dosyasındaki kodlar incelenmiştir. Wrapper, işlemciyi FPGA kartına gömüldüğü halde wrapperın içindeki özel UART sayesinde hafıza hattına buyruk yükleyebilmektedir [43].

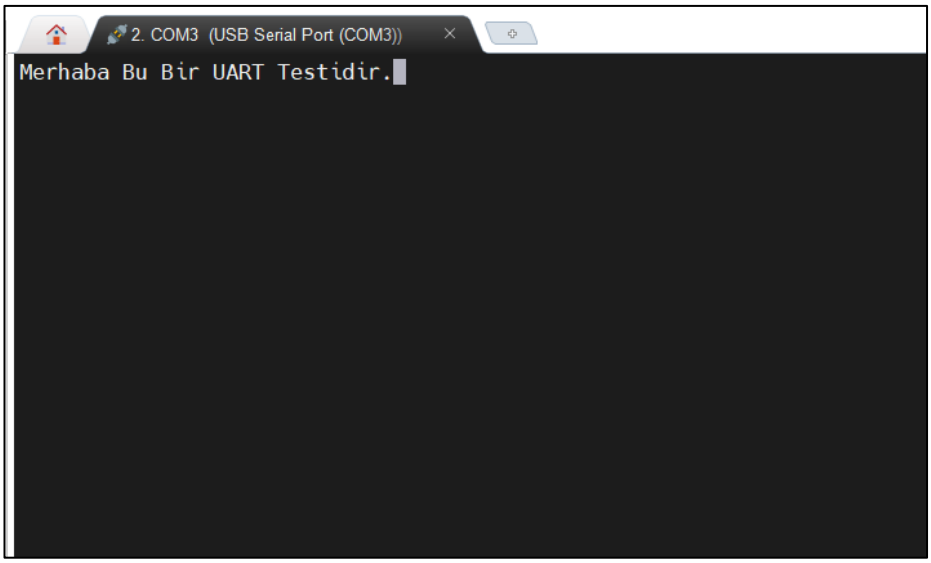
5.4.1. UART Testi

Şekil 2.24'de görüldüğü gibi tasarlanan işlemciye C ile UART test kodu yazılmış ve Zybo Z7-20 kartı ile uygulanmıştır.

test1 > C main1.c

```
1
2   #include "uart.h"
3
4
5
6   int main(){
7
8     init_uart();
9     ee_printf("Merhaba Bu Bir UART Testidir.");
10
11  }
12
13
14
```

Şekil 5.24. Basit UART test kodu.



Şekil 5.25. Uart Terminal Sonucu.

Şekil 5.25'te görüldüğü gibi UART testi başarıyla sonuçlanmıştır. init_uart() fonksiyonu uart sistemini otomatik ayarlamaktadır ve baudrate değerini 115200 olarak tutmaktadır. Herhangi bir problem ile karşılaşılmamıştır.

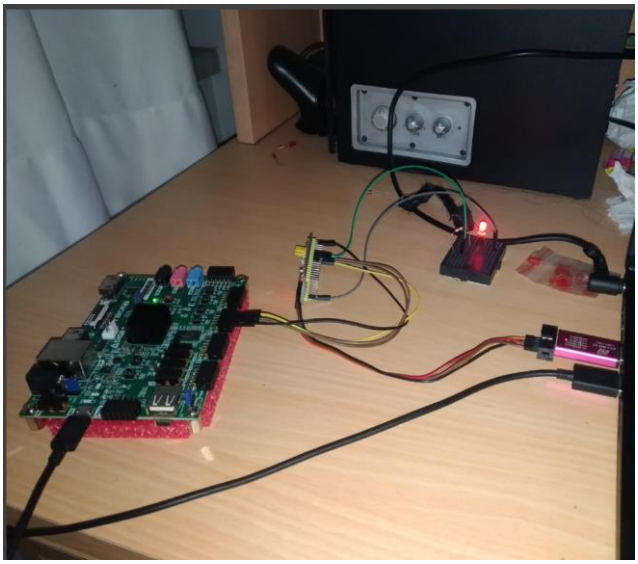
5.4.2. SPI Testi

UART'a benzer bir şekilde Şekil 2.26'da görüldüğü gibi tasarlanan işlemciye C ile SPI test kodu yazılmış ve Zybo Z7-20 kartı ile uygulanmıştır.

main1.c

```c
#include "printf.h"
#include "spi_driver.h"


int main(){

init_uart();
    SPI_WDATA = 0x4B;
    SPI_CTRL = 0x000F0001;
    SPI_CMD = 0x00002000;
    while(!((SPI_STATUS>>1)%2));

    while(1);
}
```

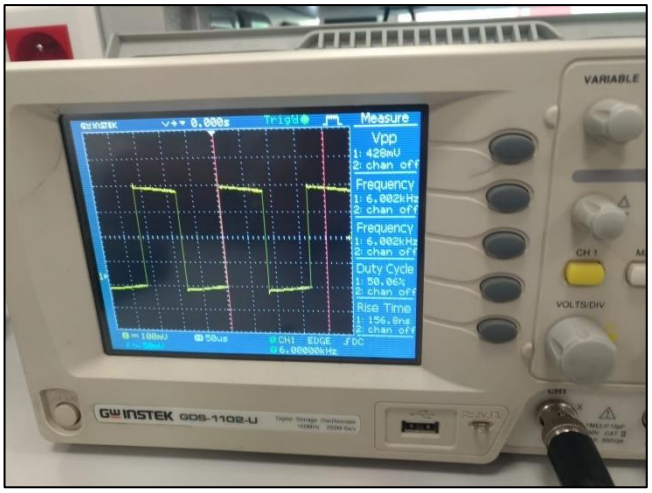
Şekil 5.26. SPI test kodları.



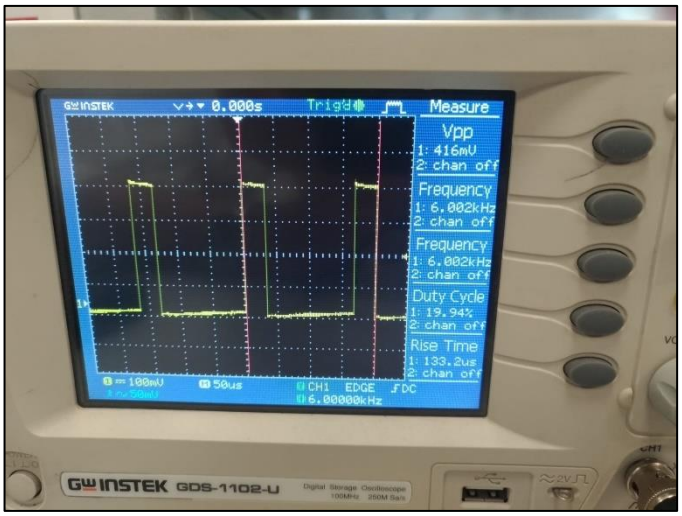
Şekil 5.27. SPI testinin FPGA ortamında gerçekleştirilmesi.

SPI testinde STM32 mikrodenetleyicisi kullanılmıştır. Bu işlemde kendi tasarladığımız işlemcimiz ile STM32 arasında sinyal göndererek led yakma işlemi gerçekleştirilmesi amaçlanmıştır. Şekil 5.26'deki test kodlarını deneyerek, Şekil 5.27'daki gibi ortam hazırlanmış ve test edilmiştir. Test sonucunda ledin yandığı görülmüştür. SPI testi başarılı bir şekilde gerçekleştirilmiştir.

5.4.3. PWM Testi



Şekil 5.28. %50 Duty Cycle PWM sinyali.



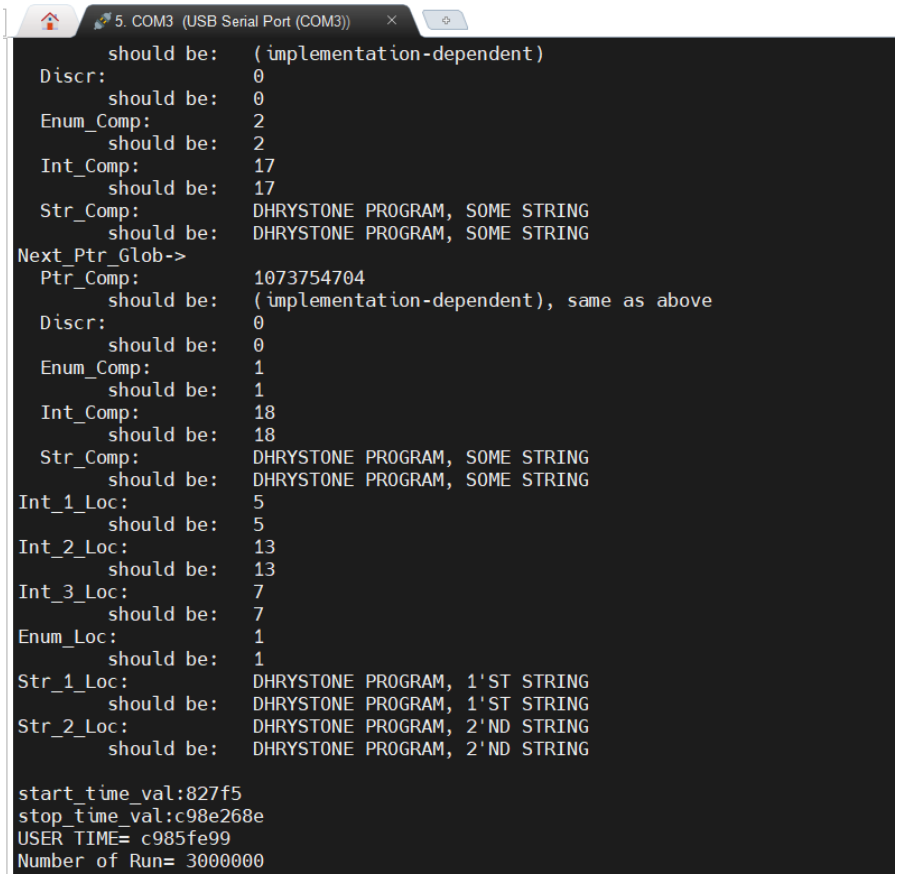
Şekil 5.29. %20 Duty Cycle PWM sinyali.

2 farklı kod ile test edilmiş. İşlemcimize eklenmiş olan pwm modülü ile 2 farklı sinyal elde edilmeye çalışılmıştır. Şekil 5.28'de görüldüğü gibi osiloskopta görünen değer %50 duty cycle bir pwm sinyalidir. Şekil 5.29'de görüldüğü gibi başka bir test uygulanıp %20 duty cycle olan bir sinyal elde edilmesi, başarı ile sonuçlanmıştır.

5.4.4. Dhrystone Benchmark Testi

Tasarladığımız işlemcimize Dhrystone Benchmark testi yapılmıştır. Number_of_runs değeri 3 000 000 girilmiştir. İşlemci 40 MHz ayarlanmış ve test başlatılmıştır.

Dhrystone testi sonucunda stop_time_val ve start_time_val çıkartarak aradaki süre farkını bulmamamız gerekmektedir. Aradaki fark 3381001881 çıkmaktadır. Terminalde bahsedilen USER_TIME bu sonucu göstermektedir. Sistemin çalışıp bittiği yere kadar geçen cycle sayısıdır. İşlemcimiz 40Mhz'dir ve 3 000 000 number of run ayarı yapılarak sistem başlatılmış olup çıkan sonuç hexadecimal olarak şekil 5.30'da görülmektedir. Çıkan USER_TIME sonucuna göre aşağıdaki işlemler yapılmıştır.



Şekil 5.30. Dhrystone Benchmark sonucu.

$$Dhrystone\ Per\ Second = \frac{HZ * Numberof Runs}{usertime} \tag{5.1}$$

5.1'deki formül uygulandığı sırada çıkan sonuç 35492,4 sayısı çıkmaktadır.

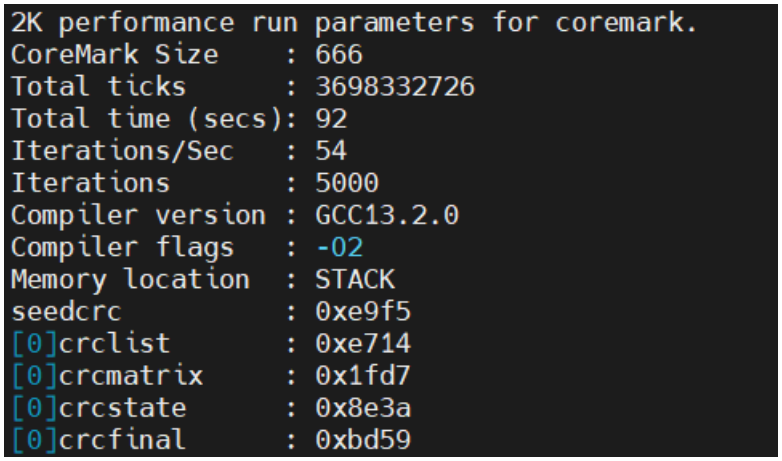
$$DMIPS = \frac{Dhrystone\ Per\ Second}{1757} \tag{5.2}$$

$$\frac{DMIPS}{MHz} = \frac{DMIPS}{MHz} \tag{5.3}$$

Bu formüller uygulandığında işlemcimizin sonucu 0,5050 DMIPS/MHz çıkmaktadır.

5.4.5. CoreMark Benchmark Testi

Sisteme CoreMark testide yapılmıştır. Zybo Z7-20 kartında test edilmiştir. 40 Mhz olan işlemcimiz 5000 iterasyon ayarlanarak test başlatılmıştır.



Şekil 5.31. CoreMark Benchmark sonucu.

CoreMark sonucu Iteration/Sec değerinin Mhz bölümü sonucu 1,35 CoreMark/Mhz sonucu çıkmıştır.

BÖLÜM 6

SONUÇLAR

Bu çalışmada FPGA tabanlı RISC-V mimarisine sahip 32-bit kripto işlemci tasarlanmıştır. Çekirdeğin doğru çalışdığını test etmek için bazı simülasyon programlarından faydalanılmıştır. UART, SPI ve PWM gibi çevre birimlerinin testi hem simülasyon ortamında hemde Zybo Z7-20 FPGA kartı ile test edilmiştir. FPGA tabanlı RISC-V mimarisine sahip 32-bit kripto işlemci tasarım uygulamasının sonucunda;

- Tasarlanan işlemcinin Zybo Z7-20 FPGA kartında 40 Mhz'de çalışmaktadır.
- Tasarlanan işlemcinin buyrukları RISCOF ortamında test edilmiştir. Bütün buyruklar başarılı bir şekilde testi geçmiştir.
- UART testi yapılmıştır. Başarıyla tamamlanmıştır.
- Tasarladığımız işlemci ile STM32 mikrodenetleyici kartı arasında SPI testi yapılmıştır. SPI testinde herhangi problem görülmemiştir. STM32, işlemciden gelen 0x4B hexadecimal sayısını alarak ledi yakmayı başarmıştır.
- PWM testi için osiloskop kullanılmıştır. PWM modülündeki eşik değerleri ayarlanarak 6KHz'lik bir sinyal başarılı bir şekilde üretilmiştir.
- Dhrystone benchmark testi yapılmıştır. FPGA 40Mhz'de, ana hafıza işlemlerinin 2 clock gecikmeli bir şekilde çalışması ve numbers_of_run değerinin 3000 000 olarak ayarlayarak dhrystone testi başlatılmıştır. Bunun sonucunda dhrystone testinin başlangıcından bitimine kadar toplamda 3381001881 tane clock sayısı geçmiştir. Uygun hesaplamalar yapıldığında tasarlanan işlemcinin 0,0505 DMIPS/MHz sonucuna ulaşılmıştır.
- Coremark benchmark testi yapılmıştır. 5000 iterasyonda 1,35 CoreMark/Mhz'lik bir sonuç elde edilmiştir.

Bu proje tamamlandığında görülmüştürki; çarpma algoritmasının döngü sayısının yüksek olması ve önbellek boyutlarının az olmasından kaynaklanan sebeplerle işlemci hızının düştüğü görülmüştür.

KAYNAKLAR

1. İnternet: İstanbul Teknik Üniversitesi "A Brief History of Computer Technology.", **https://web.itu.edu.tr/~gerzeli/History.htm**

2. İnternet: IEEE Life Members, "ENIAC: The World's First Computer", **https://life.ieee.org/eniac-the-worlds-first-computer/** (2023).

3. İnternet: IEEE Spectrum, "Chip Hall of Fame: Intel 4004 Microprocessor", **https://spectrum.ieee.org/chip-hall-of-fame-intel-4004-microprocessor** (2024).

4. İnternet: TechTarget, "What Is RISC (Reduced Instruction Set Computer)?", **https://www.techtarget.com/whatis/definition/RISC-reduced-instruction-set-computer** (2024).

5. İnternet: CHM, "RISC: Is Simpler Better?", **https://www.computerhistory.org/revolution/digital-logic/12/286** (2024)

6. İnternet: Wevolver, "RISC-V Architecture: A Comprehensive Guide to the Open-Source ISA." **https://www.wevolver.com/article/risc-v-architecture** (2024).

7. İnternet: Tutorialspoint, "What is computer architecture?", **https://www.tutorialspoint.com/what-is-computer-architecture** (2024).

8. İnternet: University of Sunderland, "What is computer architecture?", **https://online.sunderland.ac.uk/what-is-computer-architecture/** (2024);

9. İnternet: EM360, "What is Computer Architecture? Definition, Types, Components", **https://em360tech.com/tech-article/computer-architecture** (2024).

10. İnternet: GeeksforGeeks, "Computer Organization | Von Neumann architecture", **https://www.geeksforgeeks.org/computer-organization-von-neumann-architecture/**

11. İnternet: Wikipedia, "von Neumann architecture", **https://en.wikipedia.org/wiki/Von_Neumann_architecture** (2024).

12. İnternet: GeeksforGeeks, "Harvard Architecture", **https://www.geeksforgeeks.org/harvard-architecture/** (2024).

13. İnternet: Vikipedia, "Harvard mimarisi", **https://tr.wikipedia.org/wiki/Harvard_mimarisi**

14. İnternet: Ofcskn, “CISC Ve RISC İşlemci Mimarileri Nedir? RISC Ve CISC Farkları Nelerdir?”, **https://ofcskn.com/tr/cisc-ve-risc-nedir**

15. İnternet: Javatpoint ,“Difference Between RISC and CISC - javatpoint.”, **https://www.javatpoint.com/risc-vs-cisc**

16. İnternet: GeeksforGeeks, “RISC and CISC in Computer Organization” **https://www.geeksforgeeks.org/computer-organization-risc-and-cisc/**

17. İnternet: Binary Terms, “What is CISC Processors? Architecture, Examples, Instruction Set, Advantages & Disadvanatges”, **https://binaryterms.com/cisc-processors.html**

18. İnternet: Synopsys, “What is RISC-V? – How Does it Work?”, **https://www.synopsys.com/glossary/what-is-risc-v.html**

19. İnternet: WikiChip, “Standard Extensions - RISC-V -.”, **https://en.wikichip.org/wiki/risc-v/standard_extensions**

20. Waterman, A. and Asanovi´c, K. A., “The RISC-V Instruction Set Manual Volume I: User-Level ISA Document Version 2.2 Editors,”, **CS Division, EECS Department, University of California, Berkeley**, California, (2017)

21. Harish, B., Sivani, K. and Rukmini, M. S. S., “Design and performance comparison among various types of adder topologies,” ***Proceedings of the 3rd International Conference on Computing Methodologies and Communication,*** Erode, 725–730 (2019)

22. Wibowo, F. W., “Comparison of Multiplication Algorithms Based on FPGA,” ***2018 2nd Borneo International Conference on Applied Mathematics and Engineering***, Balikpapan, 326–331 (2018).

23. Raveendran, A., Patil, V. B., Selvakumar, D. and Desalphine, V., “A RISC-V instruction set processor-micro-Architecture design and analysis,” ***2016 International Conference on VLSI Systems, Architectures, Technology and Applications***, Bengaluru, 725-730 (2016).

24. Birari, A., Birla, P., Varghese, K. and Bharadwaj, A., “A RISC-V ISA Compatible Processor IP,” ***2020 24th International Symposium on VLSI Design and Test, VDAT***, Bhubaneswar, India, *1-6 (2020).*

25. Poli, L., Saha, S., Zhai, X. and McDonald-Maier, K. D., “Design and Implementation of a RISC V Processor on FPGA,” ***Proceedings - 2021 17th International Conference on Mobility, Sensing and Networking***, Exeter ,161–166, (2021).

26. Bora, S. and Paily, R., "Design and Implementation of Adaptive Binary Divider for Fixed-Point and Floating-Point Numbers," ***Circuits Syst Signal Process***, Assam, 1131–1145, (2022)

27. Saussereau, J., Jego, C., Leroux, C. and Begueret, J. B., "Design and Implementation of a RISC-V core with a Flexible Pipeline for Design Space Exploration," ***ICECS 2023 - 2023 30th IEEE International Conference on Electronics, Circuits and Systems: Technosapiens for Saving Humanity***, Istanbul, 1-5 (2023).

28. Jose, T. and Shankar, D., "System Model Evaluation of RISC-V Cores for improved performance and fault tolerance," ***Proceedings - 2023 IEEE Space Computing Conference, SCC 2023***, Pasadena, CA, USA, 86–91, (2023).

29. Saif, M. H. B., Sadad, N. U. and Mondal, M. N. I., "FPGA Implementation of Educational RISC- V Processor Suitable for Embedded Applications," ***3rd International Conference on Electrical, Computer and Communication Engineering, ECCE 2023***, Chittagong, 1-5 (2023)

30. He, J. and Ko, S. B., "High-performance RISC-V processor with improved dispatch and commit schemes," ***2020 International Conference on Electronics, Information, and Communication, ICEIC 2020***, Barcelona,1-4, (2020).

31. Ata, S. O. ve Özkök, Y. İ., "RISC-V Mimarisi Yazılım Ekosisteminin Değerlendirilmesi", İstanbul.

32. Tozlu, Y. S. ve Yılmaz, Y., "DESIGN AND IMPLEMENTATION OF A 32-BIT RISC-V CORE", Lisans Bitirme Projesi, **İstanbul Teknik Üniversitesi Elektrik Elektronik Fakültesi**, İstanbul (2021).

33. ÖZTEKİN, H., "Bilgisayar mimarisi simülatörü tasarımı.", Yüksek Lisans Tezi, **Sakarya Üniversitesi Fen Bilimleri Enstitüsü,** Sakarya (2009).

34. Ramezani, S., "BİLGİSAYAR ARİTMETİK ALGORİTMALARININ FPGA ÜZERİNDE GERÇEKLENMESİ", Lisans Bitirme Projesi, **İstanbul Teknik Üniversitesi Elektrik Elektronik Fakültesi**, İstanbul (2011).

35. Baysal, K., "ŞİFRELEME İŞLEMLERİ İÇİN FPGA İLE YÜKSEK KAPASİTELİ ÇARPMA DEVRESİ TASARIMI", Yüksek Lisans Tezi, **Trakya Üniversitesi Fen Bilimleri Enstitüsü**, Edirne (2015).

36. Kumari, K. L. V. R., Rani, M. A., and Balaji, N., "FPGA implementation of memory design and testing," ***Proceedings - 7th IEEE International Advanced Computing Conference, IACC 2017***, Hyderabad, 552–555 (2017).

37. Omran, S. S. and Amory, I. A., "Design of two dimensional reconfigurable cache memory using FPGA," ***International Conference on Electronic Devices, Systems, and Applications***, United Arab Emirates, 1-8 (2017).

38. Awedh, M. and Mueen, A., "Design and FPGA Implementation of UART Using Microprogrammed Controller.", ***Sch. J. Eng. Tech***, 3(6):600-608 (2015).

39. Deng, D., Chen, S. and Joós, G., "FPGA implementation of PWM pattern generators," ***Canadian Conference on Electrical and Computer Engineering***, 225–230, Toronto, (2001).

40. Gazirro, M., Junior, J. M. A., Junior, O. H. A., Cavallari, M. R. and Carmo, J. P., "Design and Evaluation of Open-Source Soft-Core Processors", ***Electronics,*** 13 (4): 781 (2024).

41. Y. Chang, Y. Liu, C.Peng, J. Guo and Y. Zhao, "Design of a Configurable Five-Stage Pipeline Processor Core Based on RV32IM", ***Electronics***, 13(1):120 (2024).

42. Toker, O., "A High-Level Synthesis Approach for a RISC-V RV32I-Based System on Chip and Its FPGA Implementation", ***Engineering Proceedings***, 58(1):72, (2023).

43. İnternet: Github, "TEKNOFEST_2023_Cip_Tasarim_Yarismasi", **https://github.com/TUTEL-TUBITAK/TEKNOFEST_2023_Cip_Tasarim_Yarismasi** (2023).

44. İnternet: Docplayer, "ÇİP TASARIM YARIŞMASI ŞARTNAMESİ" **https://**docplayer**.biz.tr/226586067-Cip-tasarim-yarismasi-sartnamesi.html** (2022).

45. İnternet: Github, "KASIRGA-KIZIL/tekno-kizil: KASIRGA - KIZIL Takımı Teknofest 2023 Çip Tasarımı - KIZIL İşlemci Projesi." ,**https://github.com/KASIRGA-KIZIL/tekno-kizil** (2023).

46. İnternet: Teknofest, "ÇİP TASARIM YARIŞMASI ŞARTNAMESİ", **https://cdn.teknofest.org/media/upload/userFormUpload/%C3%87ip_Tasar%C4%B1m_Yar%C4%B1%C5%9Fmas%C4%B1_2024_Genel_Sartname_seNy5.pdf** (2023).

47. İnternet: Github "stnolting/neorv32-riscof: Port of RISCOF to check the NEORV32 for RISC-V ISA compatibility.", **https://github.com/stnolting/neorv32-riscof**

ÖZGEÇMİŞ

Kamer KIRALİ, Kozlu Anadolu Lisesi'nden mezun oldu. 2014 yılında Karabük Üniversitesi Mekatronik Mühendisliği Bölümü'nde öğrenime başlamıştır. 2018 yılında Karabük Üniversitesi'nden mezun olmuştur. 2021 yılında Karabük Üniversitesi Lisansüstü Eğitim Enstitüsü Mekatronik Mühendisliği Anabilim Dalı'nda yüksek lisans programına başladı ve eğitimine devam etmektedir.