Teknoloji Fakültesi

T.C
MARMARA ÜNİVERSİTESİ
TEKNOLOJİ FAKÜLTESİ
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ

VHDL İLE FPGA ÜZERİNDE RISC-V KOMUT SETİ KULLANARAK 32 BİT İŞLEMCİ TASARIMI

LİSANS BİTİRME PROJESİ

SEÇKİN ALBAMYA

PROF. DR. HAYRİYE KORKMAZ

İSTANBUL, 2023

Teknoloji Fakültesi

T.C

MARMARA ÜNİVERSİTESİ

TEKNOLOJİ FAKÜLTESİ

ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ

VHDL İLE FPGA ÜZERİNDE RISC-V KOMUT SETİ KULLANARAK 32 BİT İŞLEMCİ TASARIMI

LİSANS BİTİRME PROJESİ

SEÇKİN ALBAMYA

(170519015)

PROF. DR. HAYRİYE KORKMAZ

İSTANBUL, 2023

ÖNSÖZ

Çocukluğumda başladığım elektronik hobisini yaklaşık 10 yıl sonra profesyonel bir meslek haline getirmenin verdiği gurur içerisindeyim. Önümüzdeki günlerin, ilk günkü heyecan ve mühendislik altyapısıyla daha da çılgın ve heyecan verici projelere yelken açmasını diliyorum.

Bu çalışmada da gündelik yaşamın bir parçası olan bilgisayarların nasıl çalıştığını anlamak ve bu konuda kendi özgün ürünümü ortaya koymayı amaçladım. Bu doğrultuda projemi bitirirken kendi ürünümü ortaya koymanın ve bunu paylaşmanın haklı gururunu yaşıyorum.

Öncelikle çocukluğumdan bu yana beni destekleyen, teknik işlere heveslendiren babama, çeşitli projeler geliştirdiğimiz ve ihtiyaç duyduğumda değerleri fikirlerini paylaşan ağabeyime ve manevi desteğini eksik etmeyen anneme;

Seçtiğim bitirme projesi konusu üzerinde desteklerini esirgemeyen, bilgi ve tecrübeleriyle yol gösteren danışmanım Marmara Üniversitesi Teknoloji Fakültesi Elektrik-Elektronik Mühendisliği öğretim üyesi Sn. Prof. Dr. Hayriye KORKMAZ'a;

Bu tez çalışmasında internet ortamına "Bilgisayar Mimarisi ve Organizasyonu" dersini paylaşarak şahsımı bu konuda heveslendiren ve bilgilendiren TOBB Ekonomi ve Teknoloji Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği öğretim üyesi Sn. Prof. Dr. Oğuz ERGİN'e;

Bu çalışmanın gerçekleştirilmesi için gerekli fiziksel geliştirme ortamı sağladığı için bölümüm Marmara Üniversitesi Teknoloji Fakültesi Elektrik-Elektronik Mühendisliği Bölümü'ne;

Eğitim hayatımız boyunca bizlere destek olan değerli hocalarıma;

teşekkürlerimi en içten dileklerimle sunarım.

Seçkin ALBAMYA

Temmuz, 2023

İÇİNDEKİLER

ÖZET

İşlemci, bir bilgisayarın kalbini oluşturan en temel bileşendir. Bilgisayar için gereken her türlü hesaplamayı yapan, diğer donanımları kontrol eden bir yapıya sahiptir.

Bu çalışmada RISC-V komut setinin RV32I eklentisini destekleyen, 32 bitlik işlemci FPGA üzerinde VHDL donanım tanımlama dili kullanılarak tasarlanmıştır. Çalışmada işlemci tasarımı hakkında güncel yapılar incelenmiş ve bu doğrultuda modüler yapı kullanılarak tasarlanan işlemci, anlaşılması kolay bir mikro mimari çerçevesinde tasarlanmıştır.

Sonuç olarak tamsayı komutlarının tamamını çalıştırabilen, donanımsal olarak sentezlenebilir ve RISC-V geliştirme ortamıyla uyumlu bir işlemci geliştirilmiştir. Geliştirilen bu işlemciye SecoV adı verilmiş, Digilent Nexys A7 FGPA geliştirme kartı üzerine gömülerek Assembly kodları çalıştırılmıştır.

GitHub üzerinden paylaşılarak github.com/seckinalbamya/SecoV adresinden ulaşılabilen çalışmanın, bilgisayar mimarisine ilgi duyan diğer geliştiricilere faydalı olması amaçlanmıştır.

Anahtar kelimeler: RISC-V, FPGA, İşlemci, CPU, VHDL, Açık kaynak

KISALTMALAR

ALU: Aritmethic Logic Unit – aritmetik mantık birimi

CISC: Complex Instruction Set Computer – karmaşık komut setli bilgisayar

FPGA: Field Programmable Gate Array - alanda programlanabilir kapı dizileri

HDL: Hardware Description Language – donanım tanımlama dili

HEX: Hexadecimal – on altılık sayı sistemi

ISA: Instruction Set Architecture - komut seti mimarisi

LED: Light emitting diode – ışık yayan diyot

RAM: Random Access Memory - rastgele erişimli hafıza

RISC: Reduced Instruction Set Computer - indirgenmiş komut setli bilgisayar

ROM: Read Only Memory- sadece yazılabilir hafıza

VHDL: Very High Speed Integrated Circuit Hardware Description Language - yüksek hızlı tümleşik devreler için donanım tanımlama dili

VHSIC: Very High Speed Integrated Circuit – çok hızlı tümleşik devre

ŞEKİLLER TABLOSU

TABLOLAR LİSTESİ

1. GİRİŞ

Bu proje kapsamında işlemcilerin nasıl çalıştığının detaylı şekilde öğrenilmesi amaçlanmıştır. Bu doğrultuda gerçek hayatta çalıştırılabilecek yalın bir işlemci tasarlanarak bilginin pratikleştirilmesi amaçlanmıştır.

RISC-V komut seti destekleyen, FPGA (Field Programmable Gate Array) üzerinde çalıştırılması planlanan bu işlemci tam sayı işlemlerinin tamamını çalıştırabilecek şekilde tasarlanmıştır.

HDL (hardware description language - donanım tasarım dili) ve görsel tasarım araçları kullanılarak tasarlanan işlemci geliştirilirken HDL üzerindeki hakimiyetin arttırılması ve ISA'lar (instruction set architecture - komut seti mimarisi) hakkında tecrübe kazanılması amaçlanmıştır.

İşlemci,yalın bir işlemci altyapısı oluşturularak özel amaçlar beklenen bir işlemci tasarımı için hazır platform oluşturacak şekilde planlanmıştır. Bu platform sayesinde işlemci kolaylıkla özelleştirilebilir. Özel komut ve donanımlar ekleyerek özel birtakım işlemlerin hızlıca yapılması sağlanabilir

Oluşturulan tasarımın dünya ile GitHub üzerinde gerekli belgelendirmeler ile paylaşılarak açık kaynak kodlu bir platform olması, eğitim-öğretime katkı sağlaması ve bu çalışma gibi benzer çalışmalara katkı sağlaması planlanmaktadır.

2. MATERYAL, YÖNTEM VE İŞLEYİŞ

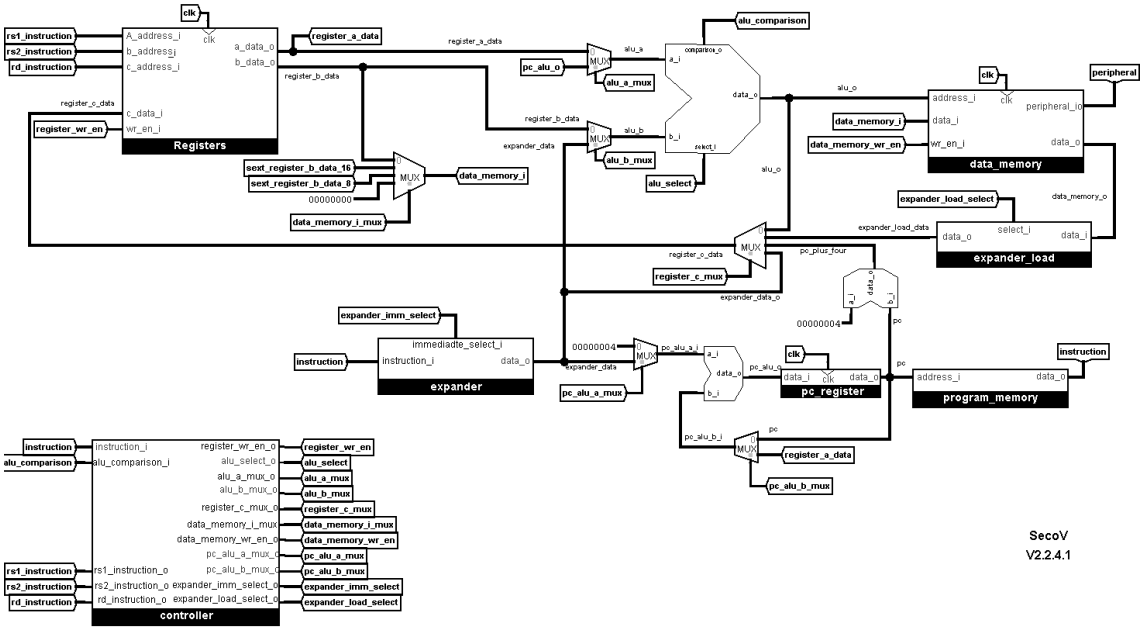
Başlık 2.1 Materyal ve Başlık 2.2 Yöntem altında bu çalışmada kullanılan donanım, yazılım vb. ve kullanılan yöntemler anlatılmıştır. Başlık 2.3 İşleyiş'te ise tasarlanan yapının nasıl çalıştığı görsellerle izah edilmiştir.

2.1 MATERYAL

İşlemci tasarımı, çok sayıda modül ve sinyal içeren bir yapıdır. Bu donanım yapısı tasarlanırken çeşitli teknik ve araçlar kullanılmıştır.

Bu teknik ve araçlar, tasarımın görsel bir şekilde ifadesi için mikro mimari tasarımı, donanımsal olarak tanımlanması için donanım tanımlama dili, gerçek bir geliştirme kartı üzerine sentezlenmesi için sentez aracı ve programcının Assembly kodlarının sayısal dönüşümünü yapan derleyicilerdir.

2.1.1 GÖRSEL TASARIM



Şekil 1 - Tasarlanan işlemcinin mikro mimarisi

Şekil 1'de daha sonra detaylıca anlatılacak olan mikro mimari tasarımı gösterilmiştir. Yapılan tasarım incelendiğinde çok sayıda modül ve modüller arası bağlantı elemanı olan sinyal olduğu görülmektedir. VHDL'de modüller arası bağlantı sinyaller aracılığıyla yapılmaktadır. Bu sinyal yapılarının isimleri ve bağlandıkları pinlerin geliştirme aşamasında geliştiriciye zorluk çıkarttığı, karmaşıklığa yol açtığı çalışma başlarında görülmüştür.

Bu sorunun çözümü için görsel tasarım programı kullanılmasına karar verilmiştir. Bunun için Diagrams.net, Microsoft Visio ve Logisim Evolution programları incelenmiştir. Gerçek bir Logic devre tasarım programı olan Logisim Evolution , ücretsiz olması, çeşitli logic elemanlar bulundurması dolayısıyla tercih edilmiştir.

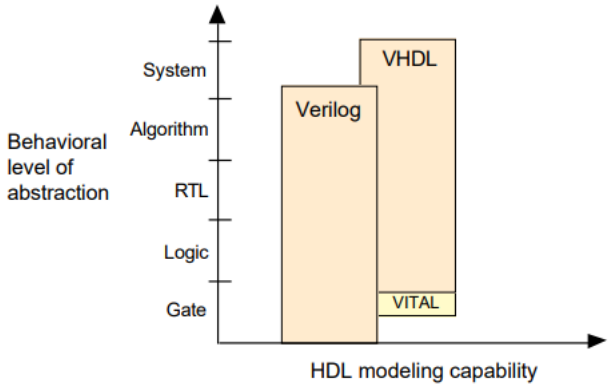
Bu programda işlemcide kullanılan tüm giriş-çıkış pinleri, sinyaller, modül isimleri ve sabit sayılar, yalnızca görsel olarak tasarlanmıştır. Görsel mikro mimari tasarımındaki isimlendirmelere harf ekleri getirilerek giriş-çıkış ve sinyal tipi olmak üzere adlandırmalar yapılmıştır.

Görsel mikro mimari tasarımında kullanılan tüm isimlendirmeler, VHDL modül ve sinyallerinde hiçbir fark olmaksızın kullanılmaktadır. Bu sayede geliştirme aşamasında kullanılmak için hızlı ve kolay bir şema oluşturulmuştur. Ayrıca bu görsel tasarım, bilgisayar mimarisi literatüründe işlemcinin iç yapısını ifade eden önemli bir şemadır.

2.1.2 DONANIM TASARLAMA DİLİ

Bilgisayar mühendisliğinde bir donanım tanımlama dili, elektronik devrelerin ve en yaygın olarak dijital mantık devrelerinin yapısını ve davranışını tanımlamak için kullanılan özel bir bilgisayar dilidir (Wikipedia, 2023).

Dünya çapında donanım tasarımında yaygın olarak VHDL ve Verilog HDL kullanılmaktadır. Her ikisi de çok sayıda yazılım ve araçları tarafından desteklenmektedir. Şekil 2'de VHDL ve Verilog HDL modelleme kapasiteleri yönünden, Tablo 1'de ise diğer özellikleri yönünden incelenmiştir (Smith, 1996) (HardwareBee, 2023).



Şekil 2 - HDL'lerin modelleme kapasiteleri

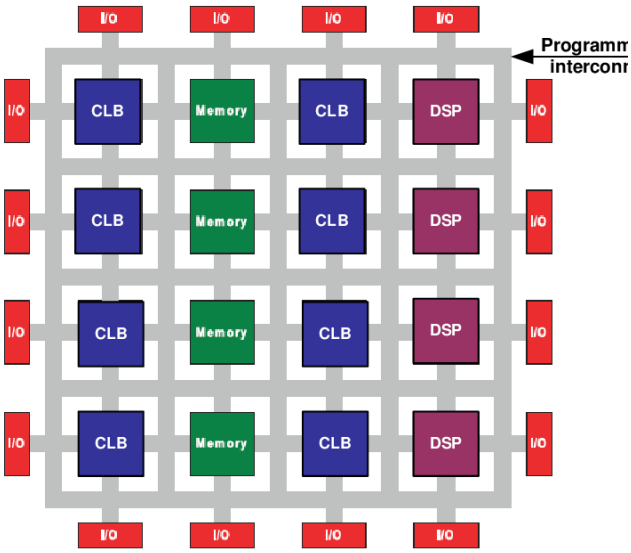
Tablo 1 - Verilog HDL ve VHDL'in karşılaştırması

Verilog HDL	VHDL
ASIC tasarımları için avantajlı	FPGA tasarımları için avantajı
Yazımı kolay	Yazımı zor
Az ayrıntılı	Çok ayrıntılı
Kısmen deterministik yapılı	Çok deterministik yapılı
C dili yapısına benzer	Ada ve Pascal yapılarına benzer

Özellikle Verilog HDL, işlemci mimarisi çalışmalarında daha çok tercih edilen dil olmasına karşın Türkiye ve Avrupa genelinde VHDL daha yaygın şekilde kullanılmaktadır. (VHDLwhiz, 2022) Bu nedenlerden dolayı VHDL, bu çalışmada kullanılacak HDL olarak seçilmiştir.

2.1.3 FPGA, GELİŞTİRME KARTI VE GELİŞTİRME ARACI

FPGA alanda programlanabilir kapı dizileri olarak adlandırılırlar. Çok genel bir tanımlamayla FPGA, dijital tasarımlarda kullandığımız farklı logic kapıların milyonlarcasının tek bir entegre üzerinde toplanmasıdır. Bu logic kapılar istenildiği gibi programlanabilir ve istenilen her türlü dijital tasarım bu elemanlar üzerinde gerçeklenebilir (Aktaş & Örencik, 2011) .



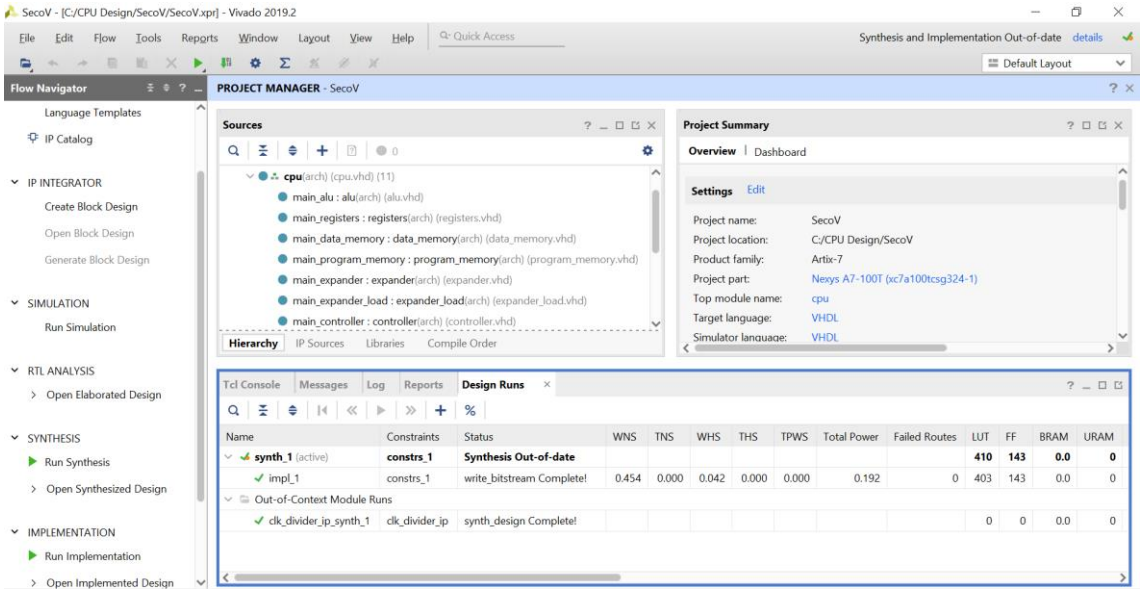
Şekil 3 - FPGA iç yapısı (Mohd, Zainol, & Sohiful, 2015)

Şekil 3'te verilen FPGA iç yapısı, HDL kullanılarak tanımlanan donanımlar ile özelleştirilir. Bu özelleştirme sonucunda istenen donanım tasarımı yapılmış olur.

Donanım dilinin sentezi ve fonksiyonel simülasyonu, FPGA çipinden veya markasından bağımsız olarak tasarlanmaktadır ancak performans analizi ve özelleştirecek konfigürasyon dosyası çipe özel olarak oluşturulmaktadır. Bu özel olması gereken yapının gereksinimlerinden dolayı gerçek hayatta kullanılacak FPGA modelinin desteklediği geliştirme araçları ile tasarım yapılması zorunludur.

Bu zorunluluktan dolayı, bu çalışmada kullanılması planlanan Digilent Nexys A7 geliştirme kartının desteklediği Vivado geliştirme ortamı kullanılmaktadır. Vivado programının sağladığı simülasyon, analiz ve diğer fonksiyonları bu çalışmada önemli bir kolaylık sağlamış ve çalışmanın sonuçlanmasında önemli rol oynamıştır.

Windows 10 üzerinde çalışan Xilinx Vivado 2019.2 HLx Editions sürümü kullanılarak modüller ve testbenchleri tasarlanmış/simüle edilmiştir. Şekil 4'te Vivado 2019.2 HLx Editions'un kullanıcı arayüzü verilmiştir.



Şekil 4 - Vivado 2019.2 HLx Editions kullanıcı arayüzü

2.1.4 KOMUT SETİ

Bir bilgisayardaki en önemli soyutlama katmanının sistem donanım/yazılım arayüzü olduğu söylenebilir (Waterman A. S., 2016). Bu soyutlama katmanı; yazılımdan aldığı talimatlar serisini seçilen işlemci mimarisindeki donanımın anlayacağı bitlere dönüştürerek ilgili donanımı kontrol etmektedir.

İşlemcinin gereksinimlerine göre belirlenen özelleşmiş yazılım fonksiyonları komut olarak adlandırılmaktadır. Yazılan yazılım, derleyiciler tarafından uygun komutlara ve komutlardan da bit seviyesine dönüştürülmektedir. Bit seviyesinde 1 ve 0'lardan oluşan sayı dizileri donanım tarafından çözülerek istenen işlemler yapılmaktadır.

Başlık 3.1 Literatür Taraması'nda daha detaylı şekilde bahsedilecek olan RISC (Reduced Instruction Set Computer) ve CISC (Complex Instruction Set Computer) tabanlı işlemciler bulunmaktadır. Bu farklı mimari yapılar, özgün komut setlerine sahiptir. Tablo 2'de bilgisayar ve mobil aygıtlarda yaygın olarak kullanılan komut setleri kıyaslaması yapılmıştır.

Tablo 2 - Çeşitli komut setlerinin karşılaştırılması (Asanović, 2021)

ISA	Entegre üreticisi	Mimari lisansı	Ticari durumu	Özelleştirilebilirlik	Çekirdeğin fikri mülkiyet durumu
x86	Evet, üç adet satıcı	Hayır	Hayır	Hayır	Tescilli
ARM	Evet, çok sayıda satıcı	Evet, pahalı	Evet, tek satıcı	Hayır (çoğunlukla)	Tescilli
RISC-V	Evet, çok sayıda satıcı	Evet, ücretsiz	Evet, çok sayıda satıcı	Evet	Açık kaynak

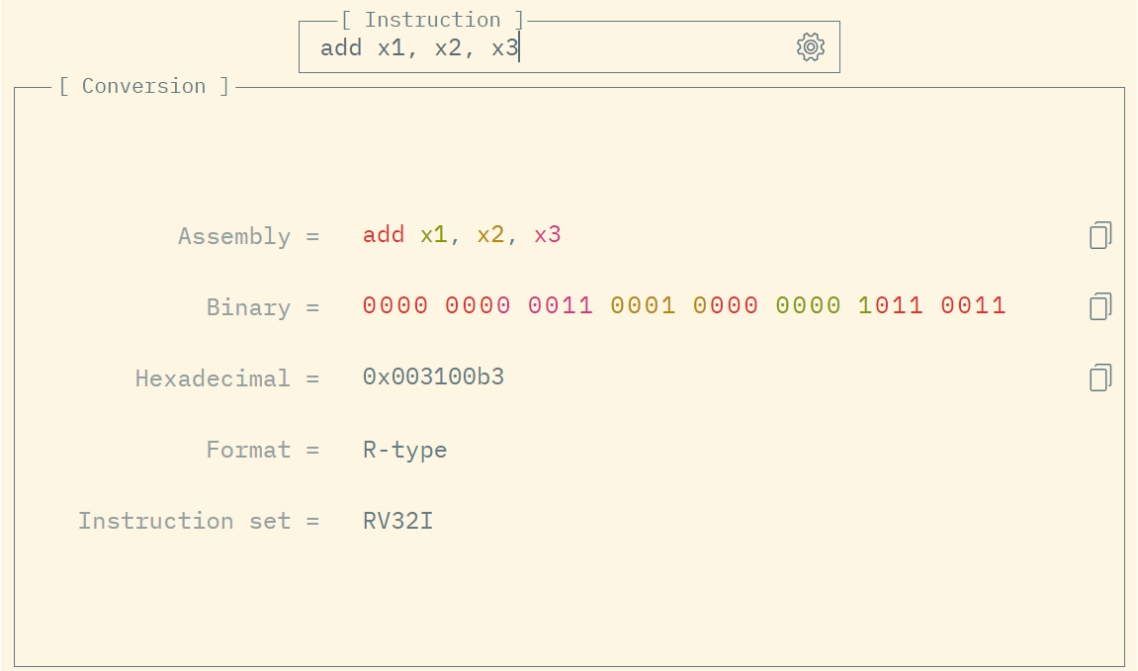
RISC yapısının basit donanım ihtiyacı, açık kaynak kodlu olması, artan popülerliği ve özelleştirilebilir yapısından dolayı RISC-V komut seti kullanılması kararlaştırılmıştır. Ayrıca geliştirme sürecinin tamamlanmasının ardından açık kaynak

kodlu olarak paylaşılacak işlemci için açık kaynak kodlu fikri mülkiyet önemli bir kriterdir.

2.1.5 ASSEMBLY DERLEYİCİ

Assembly dili, komut kodlarının ikili veya hex gösteriminin yerine mnemonik(hatırlatıcı) denilen komut kısaltmalarını kullanır. Bu hatırlatıcılar, kelime olarak assembly programcısının ne işlem yaptığını kısaca belirtir ve görsel açıdan programın okunmasına kolaylık sağlar. Her komut seti için özel olan bu hatırlatıcılar, Assembly derleyiciler tarafından program hafızasına yazılmak üzere sayısal tabana dönüştürülürler (Topaloğlu, 2021).

Bu çalışmada RISC-V hatırlatıcılarını hex tabanına dönüştüren uplab.gitlab.io/rvcodecjs/ adresindeki internet sitesinden faydalanılmıştır. Bu web sitesi ile Assemby dili kullanılarak yazılan test kodları data memory modülüne yazılabilecek formata çevrilmiştir. Şekil 5'te örnek bir buyruğun çevrilmesi gözükmektedir.



Şekil 5 - Çevirimiçi Assembly çevirici

2.2 YÖNTEM

Bu çalışmada FPGA üzerinde çalışacak, tek vuruşluk (single cycle) ve HDL kullanılarak tasarlanması planlanan bir işlemci tasarlanması planlanmaktadır. Bu doğrultuda Başlık 3.1 Materyal'de belirtilen materyaller kullanılarak tasarım sonuçlandırılmıştır.

2.2.1 TASARIM

Tasarlanan işlemci mimarisi, çevrimiçi video platformlarında bulunan, Prof. Dr. Oğuz Ergin'in BİL361 Bilgisayar Mimarisi ve Organizasyonu dersi kapsamında anlatılan RISC-V tabanlı, 10 komut işleyebilen mikro mimari ile başlayıp RISC-V RV32I eklentisinin gereksinimleri doğrultusunda özgün algoritma ve çözümler kullanılarak geliştirilmiştir (Ergin, 2020).

Tasarlanan işlemci çok sayıda modülden oluşmaktadır. Bu modüller ALU(arithmetic logic unit), register (yazmaç), data memory, program memory, expander, expander load, controller, PC register ve PC adder olmak üzere dokuz tanedir. Tasarlanan tüm modüllerin tasarımı ve doğrulaması yapılarak ilgili başlıklarda detaylı şekilde anlatılmıştır.

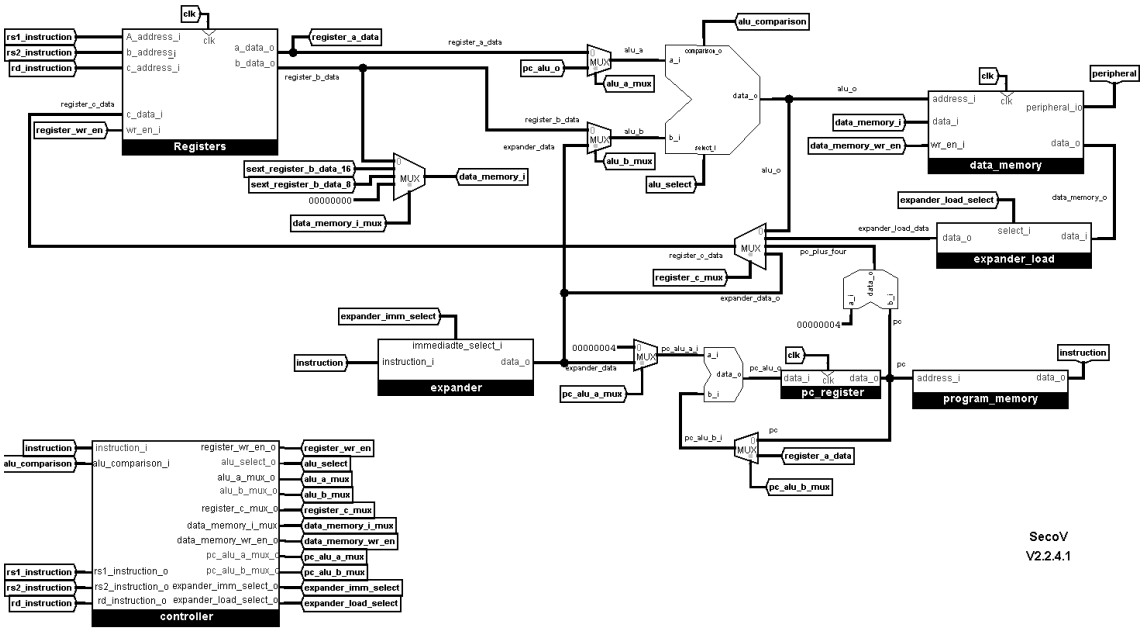
Bilgisayar, sunucu ve telefon gibi yüksek güçlü işlemciye sahip cihazlarda data memory ve program memory sisteme harici olarak entegre edilmektedir. Diğer modüllerden olan ALU, program sayacı, yazmaçlar, expanderler ve controller modülleri çekirdek yapısında yer almaktadır. Bu çalışmada ise data memory, program memory ve çekirdek yapısı bir bütün halinde tasarlanmıştır.

İşlemci bellek yapılarında kullanılan iki yaygın yapıdan birisi olan Harvard mimarisi bu işlemci yapısında kullanılmaktadır. Harvard mimarisine göre data memory ve program memory ayrı ayrı işlemci içerisinde tasarlanarak kullanılmaktadır.

Data memory için ayırılan bellek yapısı içerisinde bazı özel adresler, çevresel sinyal giriş çıkışları için tanımlanmıştır. Bu sayede çevresel girişlerin durumları ve dış dünyaya çıkış yapılması bellek adresinin okunması/yazılması sayesinde sağlanmaktadır.

Tasarımı yapılan işlemci RV32I komut setinin temel fonksiyonlarını çalıştırabilen, çok çekirdekli çalışma ve debug için tasarlanan özel komutları çalıştırmayan bir yapıda tasarlanmıştır. İhtiyaç olmadığı için eklenmeyen bu komutların performans ya da bu işlemcinin tasarlanma amacına yönelik uygulamalara etkisi bulunmamaktadır.

Şekil 6'de ve Ek 1'de (daha büyük şekilde) Tasarlanan işlemcinin mikro mimarisi verilmiştir. Bu mikro mimari şemasında modüller arasındaki tüm bağlantılar, iç denetim sinyalleri, MUX yapıları ve clock sinyalleri VHDL'de kullanılan sinyal ve modül isimleriyle verilmiştir.

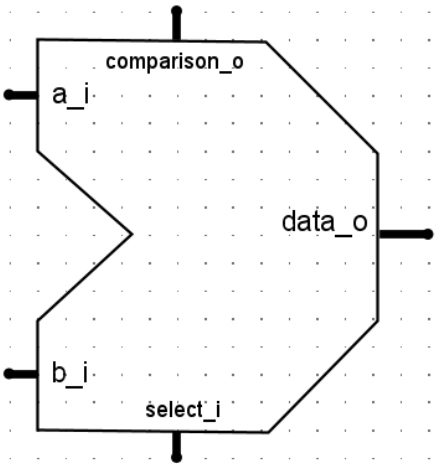


Şekil 6 - Mikro mimari tasarım

Alt başlıklar halinde Şekil 6'da görülen mikro mimari tasarımda verilen modüller paylaşılmıştır.

2.2.1.1 ALU MODÜLÜ

Şekil 7'te verilen ALU işlemcinin kalbidir. Bu modül 32 bit iki veri girişi, 32 bit veri çıkışı, 4 bit işlem seçimi, 3 bit kıyaslama çıkışları olmak üzere 5 porttan oluşmaktadır.



Şekil 7 - ALU modülü

ALU, girişindeki verilere select pininden seçilen işlemi uygulamaktadır. ALU'nun yaptığı işlemler ve çıkış sonuçları Tablo 3'te verilmiştir.

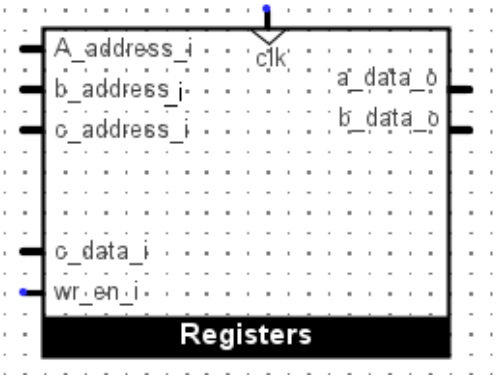
Tablo 3 - ALU işlemleri ve açıklamaları

Komut	Select girişi	Çıkış	Kıyaslama çıkışı	Açıklama
ADD	0000	a+b	X	a ile b'yi toplar.
SUB	0001	a-b	a = b ise '100"	a'dan b'yi çıkartır.
SLL	0010	a sll(b)	X	a'yı b (işaretsiz) kere sola öteler
SLT	0011	X	a<b ise '010'	a ile b'yi kıyaslar
SLTU	0100	X	a(işaretsiz) <b(işaretsiz) ise '001'	a (işaretsiz) ile b (işaretsiz)'yi kıyaslar
XOR	0101	a xor b	X	a ile b'ye xor işlemi uygular.
SRL	0110	a srl(b)	X	a'yı b (işaretsiz) kere sağa öteler
SRA	0111	a sra(b)	X	a'yı b (işaretsiz) kere aritmetik olarak sağa öteler.
OR	1000	a or b	X	a ile b'ye or işlemi uygular.
AND	1001	a and b	X	a ile b'ye and işlemi uygular.

Tek vuruşluk işlemcide aynı clock sinyalinde tüm işlemlerin bitebilmesi için concurrent olarak, bir diğer değişle eşzamanlı olarak, tasarlanan ALU, clock vuruşundan bağımsız olarak çalışacak şekilde tasarlanmıştır.

2.2.1.2 REGİSTER MODÜLÜ

Şekil 8'de verilen Register modülü işlemcinin çekirdeğinde yer alan hızlı şekilde veri okuyup yazabildiği depolama birimidir. Bu modül üç adres girişi, iki veri çıkışı, bir veri girişi, clock ve yazma denetimi girişi olmak üzere sekiz pinden oluşmaktadır.



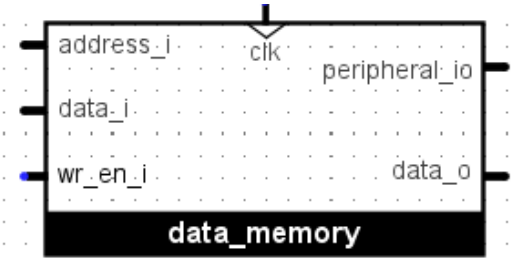
Şekil 8 - Register modülü

Register modülü, aynı clock vuruşunda seçilen hedef yazmacına veri yazarken aynı zamanda veri çıkışlarından verilerin okunabildiği yapıda bir bellek modülüdür. İşlemcinin içerisinde, iç işleyişin merkezinde konumlandırılarak kullanılmaktadır.

RISC-V, standart olarak 32 adet yazmaç kullanmaktadır (Waterman, Asanovic, & SiFive Inc., 2017). Register içerisine 31 genel amaçlı yazmaç ve bir adet tüm bitleri 0 olarak set edilmiş x0 yazmacı, işlemcinin bit genişliği olan 32 bit olarak tanımlanmıştır. Her yükselen kenar clock vuruşuyla yazmaç bilgileri ve çıktıları güncellenmektedir.

2.2.1.3 DATA MEMORY MODÜLÜ

Şekil 9'da verilen data memory işlemcinin yüksek kapasiteli veri saklama ihtiyacı duyduğu zaman kullandığı depolama birimidir. RAM olarak da bilinmektedir. Bu modül, 32 bit genişliğinde adres girişi, veri girişi, veri çıkışı; 1 bit genişliğinde clock, yazma denetimi girişi ve özelleştirilebilir çevresel birim bağlantısı olmak üzere altı pinden oluşmaktadır.

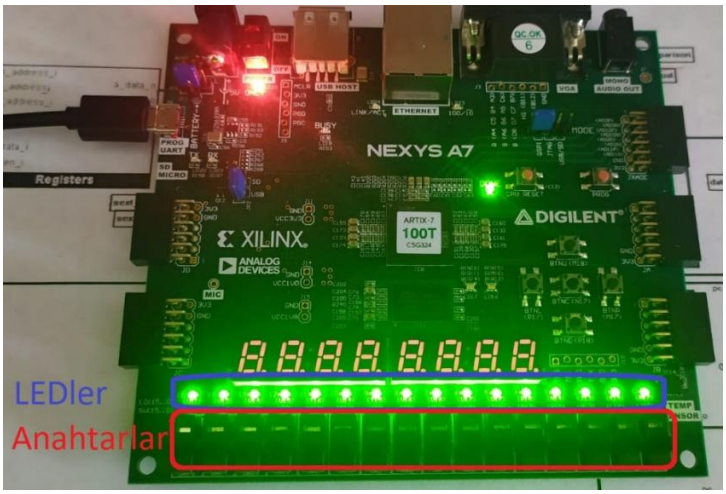


Şekil 9 - Data memory modülü

Data memory, store komutlarıyla, adres pininden seçilen hedef adresine veri yazarken, load komutlarıyla, adres pininden seçilen hedef adresinin okunabildiği yapıda bir bellek modülüdür. İşlemcinin çekirdeğinin dışında konumlandırılarak kullanılmaktadır.

Data memory içerisinde 32 bit genişliğinde memory_size parametresi ile tasarımcı tarafından belirlenen sayıda verilerin saklanabileceği depolama alanı tanımlanmıştır. Hafıza boyutu, sonradan kolaylıkla değiştirilebilecek şekilde generic parametre olarak tanımlanarak kolaylıkla özelleştirilebilir bir yapı sağlanmıştır. Her yükselen kenar clock sinyali gelmesiyle Data memory, içerisinde saklanan bilgileri ve çıktıları güncellenmektedir.

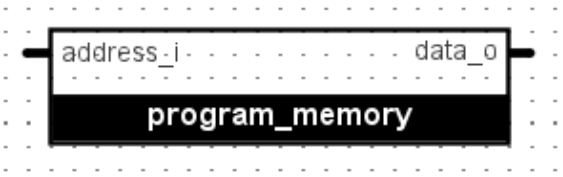
Data memory üzerindeki son iki adresin saklanan veriye (RAM[memory_size-1] ve RAM[memory_size]) test yazılımında kullanılmak üzere Nexys A7 geliştirme kartı üzerindeki çeşitli giriş/çıkış aygıtları bağlanmıştır. Şekil 10'da görülen kart üzerindeki 16 adet anahtar ve 16 adet LED bağlanarak dış dünyadaki çeşitli girişlerin alınması ve sağlanmış, LEDler ile görsel çıktı sonuçları elde edilmiştir.



Şekil 10 - Nexys A7 üzerinde çevresel donanım olarak kullanılan LED ve anahtarlar

2.2.1.4 PROGRAM MEMORY MODÜLÜ

Şekil 11'de verilen program memory işlemcinin çalıştıracağı komutları içerisinde barındırdığı depolama birimidir. Clock sinalinden bağımsız olarak çalışan bu modül, 32 bit genişliğinde adres girişi ve veri çıkışı olmak üzere iki pinden oluşmaktadır.

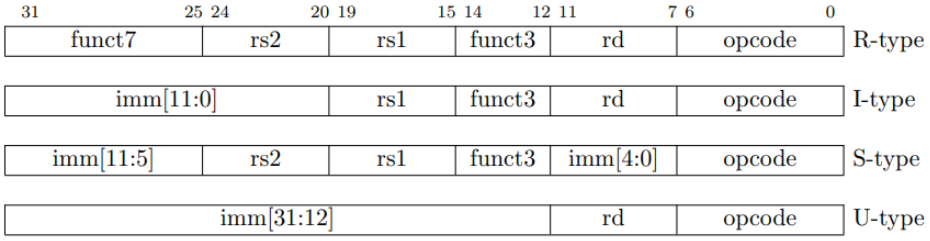


Şekil 11 - Program memory modülü

Program memorynin kapasitesi, VHDL'de generic parametre olarak tanımlanarak hafıza boyutunun kolaylıkla istenilen şekilde değiştirilebilmesi sağlanmıştır.

2.2.1.5 EXPANDER MODÜLÜ

Komut setinde bazı komutlar içerisinde yazmaç adresleri, bazıları ise sabit bir değer (anlık değer) içermektedir. Bu anlık değerlerin bit genişlikleri komut türüne göre değişmektedir. Şekil 12'de farklı komut türlerine göre anlık değerlerinin nasıl konumlandırıldığı gözükmektedir.



Şekil 12 - Farklı türdeki komut yapısı (Patterson & Hennessy, 2017)

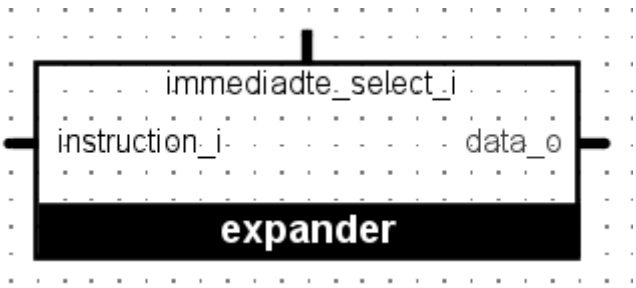
Şekil 12'de de görüldüğü üzere anlık değerlerin uzunlukları 32 bitten az olmak üzere çeşitli uzunluktadır. İşlemci içerisinde anlık değerlerin kullanıldığı Register, ALU ve PC adder modülleri 32 bit veri arayüzleri ile çalışmaktadır. Bu değerlerin işlemci içerisinde kullanılabilmesi için anlık değeri değişmeyecek şekilde 32 bite genişletilmesi gerekmektedir.

Çeşitli komut tiplerine göre hangi uzunlukta genişletileceği, Controller modülü tarafından doğru select sinyali seçilerek yapılmalıdır. Tablo 4'te select girişine göre nasıl genişletme işleminin yapıldığı verilmiştir.

Tablo 4 - Expander modülü işleyişi

Komut türü	select_i	Anlık bit genişliği	Genişletme sonucu(x anlık, X genişletme bitleri)
I	000	12	XXXXXXXXXXXXXXXXXXXXxxxxxxxxxxxx
U	001	20	XXXXXXXXXXXXxxxxxxxxxxxxxxxxxxxx
S	010	12	XXXXXXXXXXXXXXXXXXXXxxxxxxxxxxxx
B	011	12	XXXXXXXXXXXXXXXXXXXXxxxxxxxxxxx0
UJ	100	20	XXXXXXXXXXXxxxxxxxxxxxxxxxxxxxx0

Anlatılan bit uzunluğu uyumsuzluğunu gidermek için kullanılan Şekil 13'teki Expander modülü 32 bit komut girişi, veri çıkışı ve 3 bit select girişi olmak üzere üç pinden oluşmaktadır.



Şekil 13 - Expander modülü

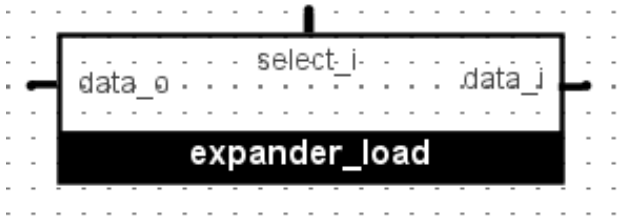
2.2.1.6 EXPANDER LOAD MODÜLÜ

RISC-V komut setinde load komutları LB, LH, LW, LBU ve LHU olmak üzere 5 adettir. Bu komutlar ile data memoryden veri alınarak yazmaçlara veri yazılmaktadır. İlgili veri alınırken veriler kısmi veya tamamen alınabilmektedir. Tablo 5'te ilgili komutlara göre verinin nasıl yazmaca taşınacağı gösterilmiştir.

Tablo 5 - Expander load modülü işleyişi

Komut	Açıklama	Select	Data çıkışı (X=registerden alınan veri)
LB	LSB 8 bit veriyi işaretle genişleterek registere aktarır.	000	xxxxxxxxxxxxxxxxxxxxxxxxXXXXXXXX
LH	LSB 16 bit veriyi işaretle genişleterek registere aktarır.	001	xxxxxxxxxxxxxxxxXXXXXXXXXXXXXXXX
LW	Tüm veriyi registere aktarır.	010	XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
LBU	LSB 8 bit veriyi işaretsiz genişleterek registere aktarır.	011	000000000000000000000000XXXXXXXX
LHU	LSB 16 bit veriyi işaretsiz genişleterek registere aktarır.	100	0000000000000000XXXXXXXXXXXXXXXX

Data memoryden alınan veriyi bölmek ve genişletmek için kullanılan Şekil 14'teki Expander load modülü 32 bit veri girişi, veri çıkışı ve 3 bit select girişi olmak üzere üç pinden oluşmaktadır.

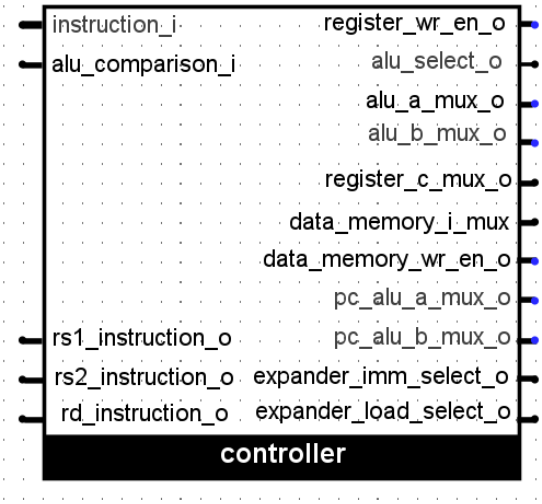


Şekil 14 - Expander Load modülü

2.2.1.7 CONTROLLER MODÜLÜ

İşlemci mikro mimarisi incelendiğinde birçok MUX, yazma kontrolü, fonksiyon seçim uçları gibi denetim sinyallerinin bulunduğu görülmektedir. Bu sinyaller doğru şekilde sürülmedikleri taktirde işlemciden anlamlı sonuç çıktısı beklenemez.

Denetim sinyallerinin kontrolü için kullanılan Şekil 15'teki Controller modülü çeşitli bit uzunluklarında komut girişi, ALU kıyaslama girişi, yazmaç adresi çıkışları ve denetim sinyalleri olmak üzere on altı pinden oluşmaktadır.



Şekil 15 - Controller modülü

Clock sinyalinden bağımsız, gelen komut değişimine göre çalışacak şekilde ayarlanan Controller, önce gelen komutu opcodelarına göre sınıflandırarak belirli komut tiplerine göre ayırmaktadır. Tablo 6'da tasnif için kullanılan çeşitli komut türleri ve opcode değerleri verilmiştir.

Tablo 6 - Controller komut tipi sınıflandırması

Komut tipi	Opcode
R	0110011
I Load	0000011
I	0010011
S	0100011
B	1100011
U	1101111 veya 0010111
U Jump	1100111
System	1110011
Problem	Diğer

Tablo 6'ya göre yapılan sınıflandırma sonrasında komut tipi, opcode, funct3 ve funct7 değerlerine göre çözülerek komut tespit edilmektedir. Komutlar elde edilirken hangi değerlerin kullanılarak çözüldüğü ve yapılan işlemler Tablo 7'de verilmiştir.

Tablo 7 - Komut çözümleri (X: önemsiz, &: ve işlemi, |: veya işlemi)

	Opcode	Komut tipi	funct3	funct7	Koşul
LUI	0110111	I	X	X	opcode & komut tipi
AUIPC	0010111	I	X	X	opcode & komut tipi
JAL	X	U Jump	X	X	komut tipi
JALR	X	I Jump	X	X	komut tipi
BEQ	X	B	000	X	komut tipi & funct3
BNE	X	B	001	X	komut tipi & funct3
BLT	X	B	100	X	komut tipi & funct3
BGE	X	B	101	X	komut tipi & funct3
BLTU	X	B	110	X	komut tipi & funct3
BGEU	X	B	111	X	komut tipi & funct3
LB	X	I Load	000	X	komut tipi & funct3
LH	X	I Load	001	X	komut tipi & funct3
LW	X	I Load	010	X	komut tipi & funct3
LBU	X	I Load	100	X	komut tipi & funct3
LHU	X	I Load	101	X	komut tipi & funct3
SB	X	S	000	X	komut tipi & funct3
SH	X	S	001	X	komut tipi & funct3
SW	X	S	010	X	komut tipi & funct3
ADDI	X	I	000	X	komut tipi & funct3
SLTI	X	I	010	X	komut tipi & funct3
SLTIU	X	I	011	X	komut tipi & funct3
XORI	X	I	100	X	komut tipi & funct3
ORI	X	I	110	X	komut tipi & funct3
ANDI	X	I	111	X	komut tipi & funct3
SLLI	X	I	001	X	komut tipi & funct3
SRLI	X	I	101	0000000	komut tipi & funct3 & funct7
SRAI	X	I	101	0100000	komut tipi & funct3 & funct7
ADD	X	R	000	0000000	komut tipi & funct3 & funct8
SUB	X	R	000	0100000	komut tipi & funct3 & funct9
SLL	X	R	001	0000000	komut tipi & funct3 & funct10
SLT	X	R	010	0000000	komut tipi & funct3 & funct11

	Opcode	Komut tipi	funct3	funct7	Koşul
SLTU	X	R	011	0000000	komut tipi & funct3 & funct12
XOR	X	R	100	0000000	komut tipi & funct3 & funct13
SRL	X	R	101	0000000	komut tipi & funct3 & funct14
SRA	X	R	101	0100000	komut tipi & funct3 & funct15
OR	X	R	110	0000000	komut tipi & funct3 & funct16
AND	X	R	111	0000000	komut tipi & funct3 & funct17
PROBLEM	DİĞER	DİĞER	DİĞER	DİĞER	opcode | komut tipi | funct3 | funct7

Komutlar ve komut tipi çözüldükten sonra, denetim sinyalleri oluşturulabilir. Ek 2'de controller modülünün denetim sinyallerinin nasıl oluşturulduğu verilmiştir.

Ek 2 incelendiğinde, komut tiplerine göre denetim sinyallerinin nasıl belirlenmekte olduğu, kaynak ve hedef yazmacı adreslerinin doğrudan komuttan, bazı denetim sinyallerinin ise ALU kıyaslama çıkışından geldiği görülmektedir.

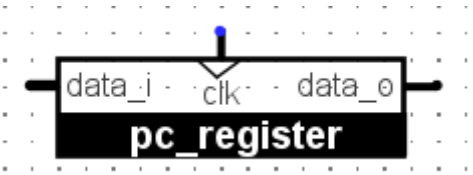
2.2.1.8 PC REGİSTER MODÜLÜ

İşlemci üzerindeki tüm işlemler clock vuruşuna göre başlayıp bitmektedir. Yeni clock vuruşuyla birlikte gelen yeni komut, çeşitli birimlerde yürütülerek işlem tamamlanmaktadır.

Komutlar, kalıcı hafıza içerisinde kendilerine ait adreslerde saklanır. İşlenecek olan komut adresi ise Program Counter (PC) adı verilen özel bir yazmaçta saklanır. İşlemcinin çalışmasını devam ettirebilmesi için yeni clock vuruşunun algılanmasıyla birlikte program sayacının güncellenmesi gerekmektedir.

Yapılan tasarımda PC güncelleme işleminin yapılması için PC register modülü oluşturulmuştur. Bu modül, her yeni clock vuruşu sinyali gelmesiyle birlikte girişindeki veriyi çıkışa aktarmaktadır. Bu sayede hem yeni komut adreslerinin clock vuruşuna bağlı olarak getirilmesini hem de saklanmasını sağlar.

Şekil 16'daki PC counter modülü 32 bit veri girişi, veri çıkışı ve clock girişi olmak üzere üç pinden oluşmaktadır.

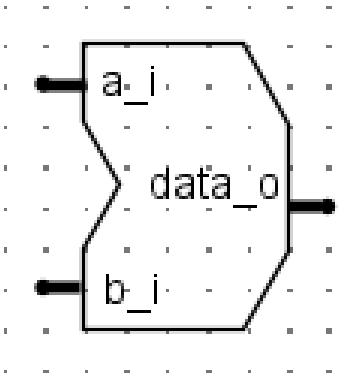


Şekil 16 - PC Register modülü

2.2.1.9 PC ADDER MODÜLÜ

PC, bir sonraki komutun alınabilmesi için güncellenmelidir. Ayrıca bazı dallanma komutları bir sonraki komut adresine (PC+4) ihtiyaç duyulmaktadır. Bu işlem için PC değeri ile işlem yapacak 32 bitlik basit bir toplayıcı birimine ihtiyaç duyulmuştur. Bu işlem için eşzamanlı çalışan, PC adder adında basit bir toplayıcı modül tasarlanmıştır.

Şekil 17'deki PC adder modülü iki adet 32 bit veri girişi ve bir adet veri çıkışı girişi olmak üzere üç pinden oluşmaktadır.

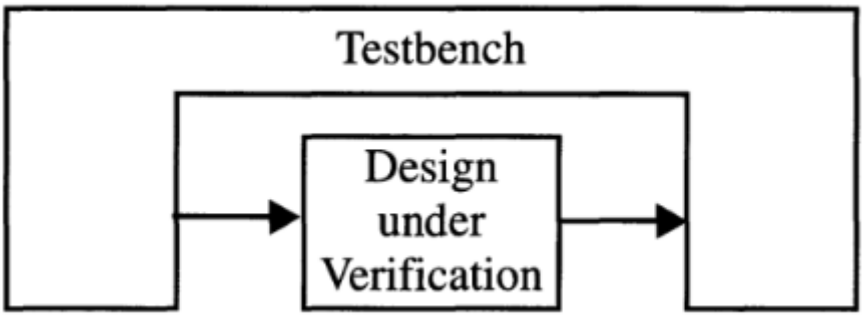


Şekil 17 - PC Adder modülü

2.2.2 DOĞRULAMA

Donanım tasarım dili kullanılarak yapılan tasarımlarda testbench ve doğrulama önemli bir aşamadır. Yapılan tasarımın doğru davranış sergileyip sergilemediği testbench adında bir test düzeneği hazırlanarak kontrol edilmelidir.

Testbenchte yazılımında, tasarlanan tasarıma istenen girişler verilerek çıkış değerleri izlenmektedir. Şekil 18'de testbench ve tasarım ilişkisi görsel olarak ifade edilmiştir. (Bergeron, 2000)



Şekil 18 - Testbench ve tasarım ilişkisi

Çıkış değerleri geliştirme ortamının simülasyon sekmesinde bulunan görsel bir arayüz üzerinden çıkış dalga biçimi (waveform) gözlenerek elde edilmektedir.

Tasarlanan modül, testbench kodunda alt eleman (component) olarak çağırılır ve modülün işleyişini test edecek çeşitli girdiler ,özellikle sınır koşullar, verilir. Çıkış koşulları, giriş değerlerine göre waveform üzerinde gözlenerek davranışsal doğrulama yapılabilir.

2.2.2.1 CLOCK SENTEZLENMESİ

Tasarlanan tüm modüller ve işleyişi gerçek hayatta uygulamaya geçirilmeden Vivado 2019.2 yazılımındaki simülasyon aracı kullanılarak doğrulama yapılmıştır. Bu simülasyon aşamasında elde edilen waveform ile her clock vuruşunda tasarım üzerindeki

tüm bit değerleri gözlemlenerek geliştirme aşamasındaki hatalar giderilmiş ve bu sayede doğrulama sağlıklı bir şekilde yapılmıştır.

Gerçek dünyadaki geliştirme kartı üzerinde sabit frekanslı clock sinyali bulunmaktadır. Bu clock sinyalinin yükselen kenarında program sayacı, yazmaç ve data memory verileri güncellenerek kullanılır. Bu çalışmada geliştirme kartı üzerindeki clock tüm tasarım için referans olacak şekilde kullanılmaktadır.

Simülasyon ortamında yukarıda anlatılan gerçek clock sinyali bulunmaz. Clock sinyalini referans olarak kullanan tasarımda bu sorunun aşılması için testbench yazılımlarının tamamına clock üreteci eklenmiştir. Gerçek devrede kullanılan 50 MHz clock değeri, testbench modüllerine eklenerek clock sinyali sentezlenmiştir. Şekil 19'da clock sinyalinin nasıl elde edildiği ve Şekil 20'de elde edilen clock sinyalinin waveform çıktısı görülmektedir.

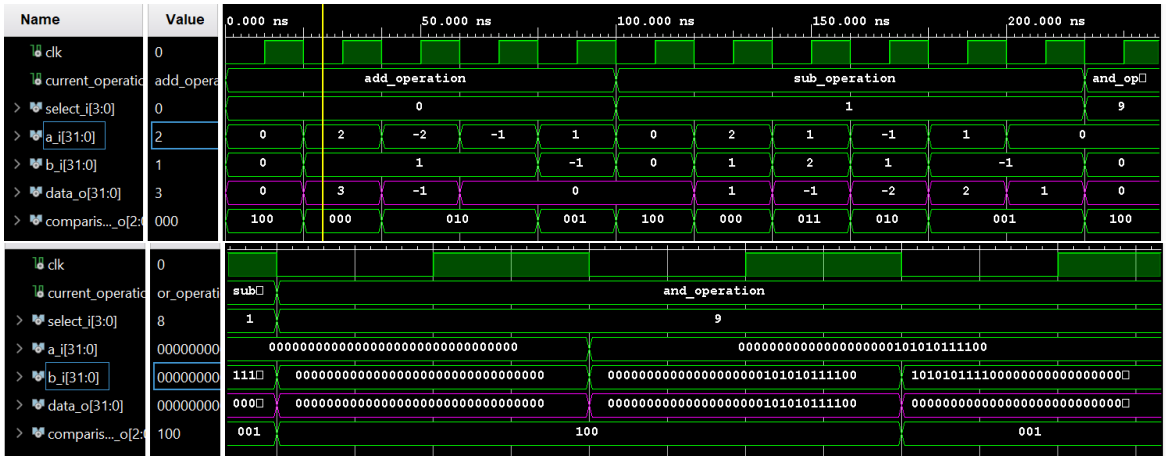
```
clock_process: process
    begin
        clk <= '0';
        wait for clock_period/2;
        clk <= '1';
        wait for clock_period/2;
    end process;
```

Şekil 19 - Simülasyonda kullanılan clock üreteci

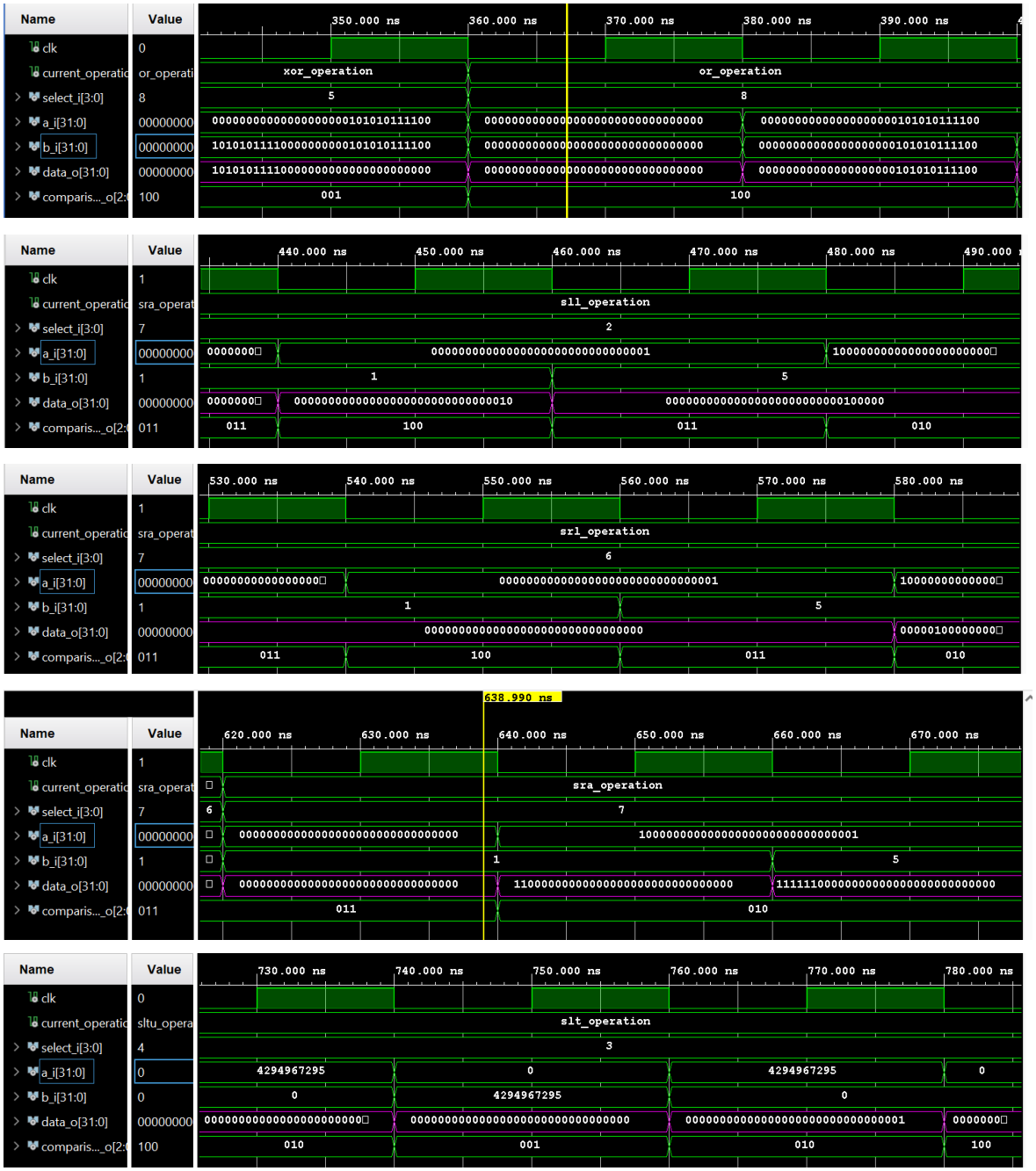
Şekil 20 - Clock üretecinin testbench çıktıları

2.2.2.2 ALU MODÜLÜ DOĞRULAMASI

Modül, ana işlemci yapısına entegre edilmeden önce , çalışmasını test etmek için, çeşitli girdiler verilerek davranışı gözlemlenmiş ve alınan sonuç Şekil 21 ve Şekil 22'de verilmiştir.



Şekil 21 - ALU testbench çıktıları

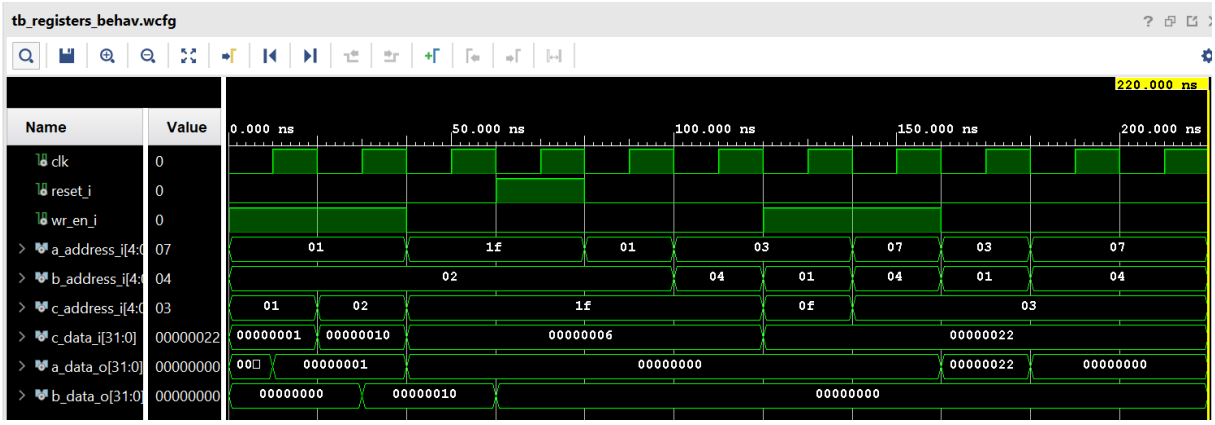


Şekil 22 - ALU testbench çıktıları

Şekil 21'de verilen testbench girdileri seçilirken özellikle, overflow, işaretli ve işaretsiz tamsayılar ile işlem gibi modülün işleyişinde kritik olan ekstrem koşullarda nasıl davrandığı izlenmiştir.

2.2.2.3 REGİSTER MODÜLÜ DOĞRULAMASI

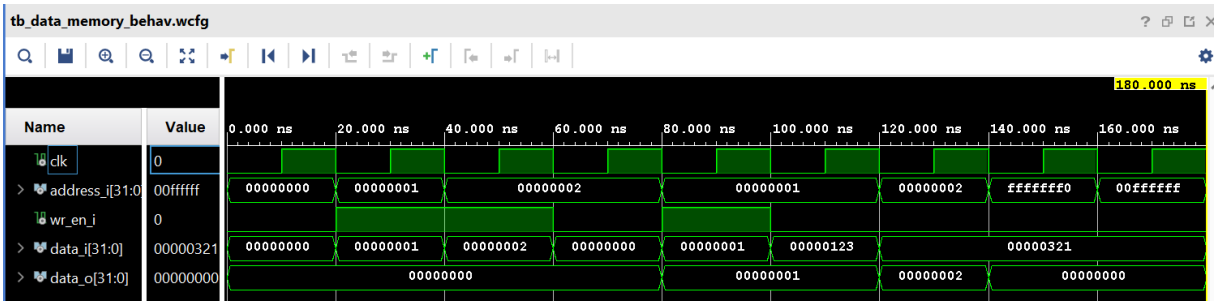
Şekil 23'te tasarlanan register modülü için yapılan testbench waveformu verilmiştir. Çeşitli girdilere karşılık modülün davranışı incelenerek çalışması doğrulanmıştır.



Şekil 23 - Register testchbench çıktıları

2.2.2.4 DATA MEMORY MODÜLÜ DOĞRULAMASI

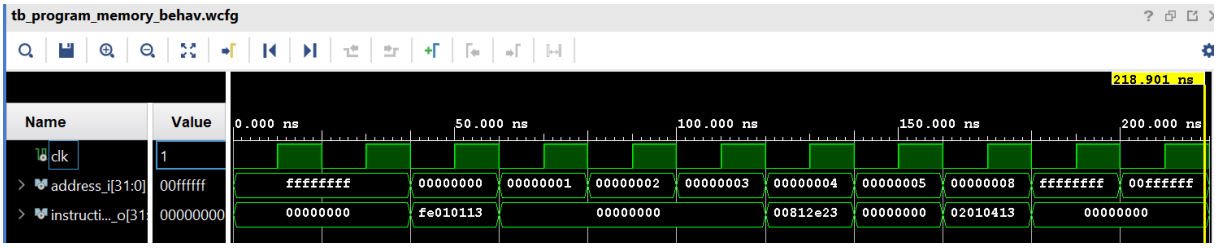
Şekil 24'te tasarlanan data memory modülü için yapılan testbench waveformu verilmiştir. Çeşitli girdilere karşılık modülün davranışı incelenerek çalışması doğrulanmıştır.



Şekil 24 - Data memory testbench çıktıları

2.2.2.5 PROGRAM MEMORY MODÜLÜ DOĞRULAMASI

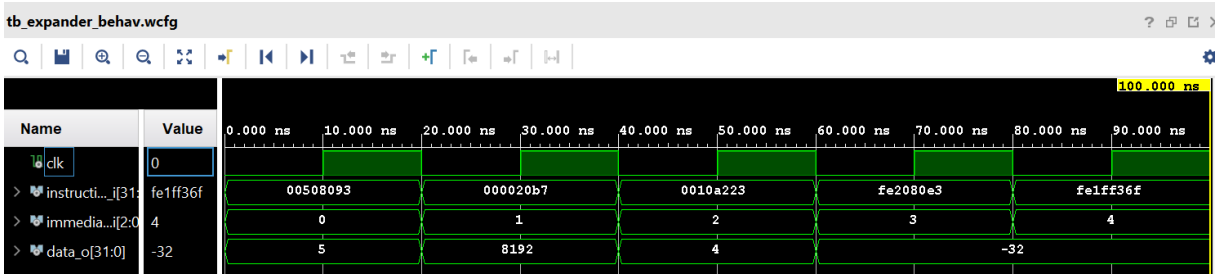
Şekil 25'te tasarlanan program memory modülü için yapılan testbench waveformu verilmiştir. Çeşitli girdilere karşılık modülün davranışı incelenerek çalışması doğrulanmıştır.



Şekil 25 - Program memory testbench çıktıları

2.2.2.6 EXPANDER MODÜLÜ DOĞRULAMASI

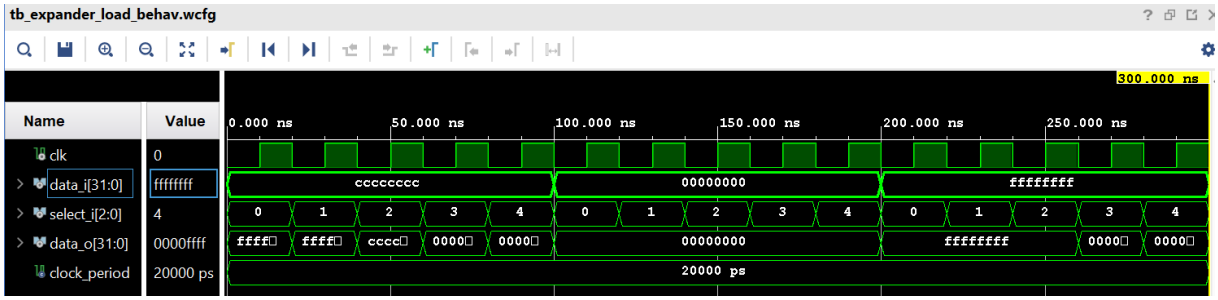
Şekil 26'da tasarlanan expander modülü için yapılan testbench waveformu verilmiştir. Çeşitli girdilere karşılık modülün davranışı incelenerek çalışması doğrulanmıştır.



Şekil 26 - Expander testbench çıktıları

2.2.2.7 EXPANDER LOAD MODÜLÜ DOĞRULAMASI

Şekil 27'de tasarlanan expander load modülü için yapılan testbench waveformu verilmiştir. Çeşitli girdilere karşılık modülün davranışı incelenerek çalışması doğrulanmıştır.



Şekil 27 - Expander load testbench çıktıları

2.2.2.8 CONTROLLER MODÜLÜ DOĞRULAMASI

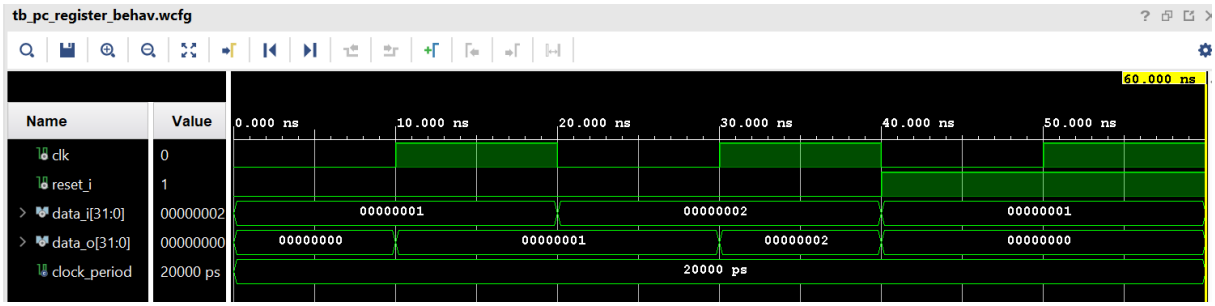
Şekil 28 ve Şekil 29'da tasarlanan controller modülü için yapılan testbench waveformu verilmiştir. Çeşitli girdilere karşılık modülün davranışı incelenerek çalışması doğrulanmıştır.

Şekil 28 - Controller testbench çıktısı

Şekil 29 - Controller testbench çıktısı

2.2.2.9 PC REGISTER MODÜLÜ DOĞRULAMASI

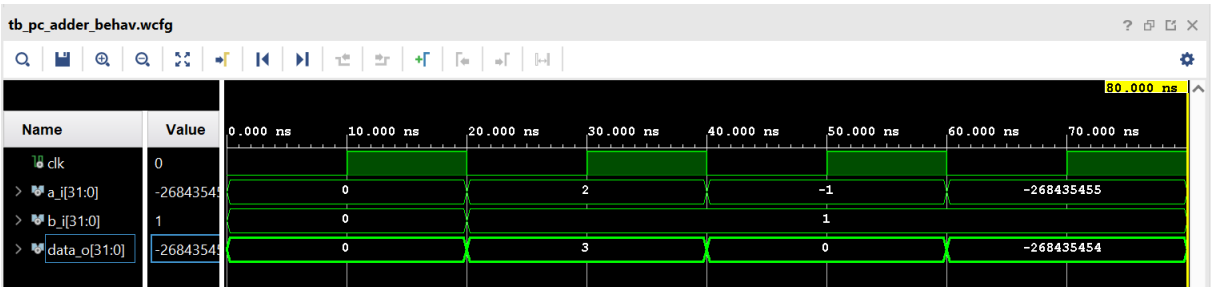
Şekil 30'da tasarlanan PC register modülü için yapılan testbench waveformu verilmiştir. Çeşitli girdilere karşılık modülün davranışı incelenerek çalışması doğrulanmıştır.



Şekil 30 - PC register modülü testbench çıktıları

2.2.2.10 PC ADDER MODÜLÜ DOĞRULAMASI

Şekil 31'de tasarlanan PC adder modülü için yapılan testbench waveformu verilmiştir. Çeşitli girdilere karşılık modülün davranışı incelenerek çalışması doğrulanmıştır.



Şekil 31 - PC adder modülü testbench çıktıları

2.2.2.11 CLK DİVİDER IP MODÜLÜ DOĞRULAMASI

Nexys A7 FPGA geliştirme kartı 100MHz bir clock frekansına sahiptir. Bu clock frekansında, basitlik amaçlanarak tasarlanan işlemcinin kritik zaman gereksinimleri karşılanamamaktadır. Bu nedenden dolayı çalışma frekansı 50MHz'e düşürülerek zaman gereksinimleri sağlanmıştır. Şekil 32'de 50MHz için Vivado 2019.2'nin "Report Timing Summary" özelliği kullanılarak alınan zaman değerleri verilmiştir.

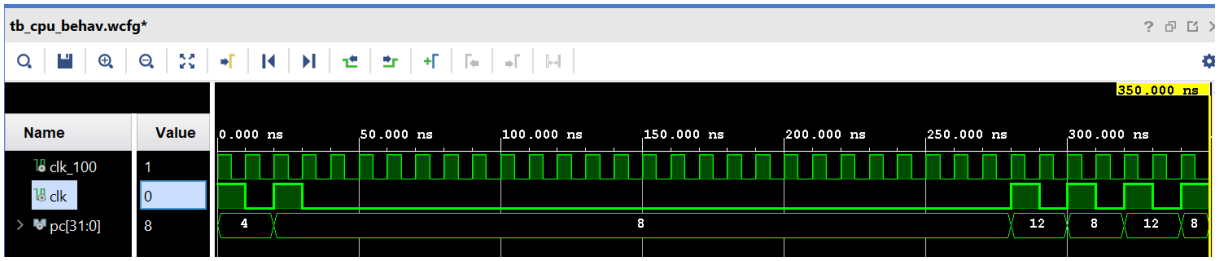
Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	6,430 ns	Worst Hold Slack (WHS):	0,046 ns	Worst Pulse Width Slack (WPWS):	3,000 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	258	Total Number of Endpoints:	258	Total Number of Endpoints:	148

All user specified timing constraints are met.

Şekil 32 - Tasarımın zaman raporu çıktısı

Yukarıda anlatılan nedenden dolayı clock frekansının azaltılması gerekmektedir. Bu doğrultuda Vivado 2019.2 Clocking Wizard versiyon 6.0 (Rev. 4) kullanılarak clock frekansı ikiye bölünmüştür. Hazır bir modülden faydalanıldığından yalnızca doğrulaması yapılan bu modülün Şekil 33'te 100 MHz giriş frekansının senkronizasyondan sonra bölerek çıkışa aktardığı gözükmektedir.



Şekil 33 - CLK divider IP modülü testbench çıktıları

Bu senkronizasyonun ilk çalışmada gerçekleştiği ve işleyişe engel olmadığı program sayacı izlenerek görülmektedir.

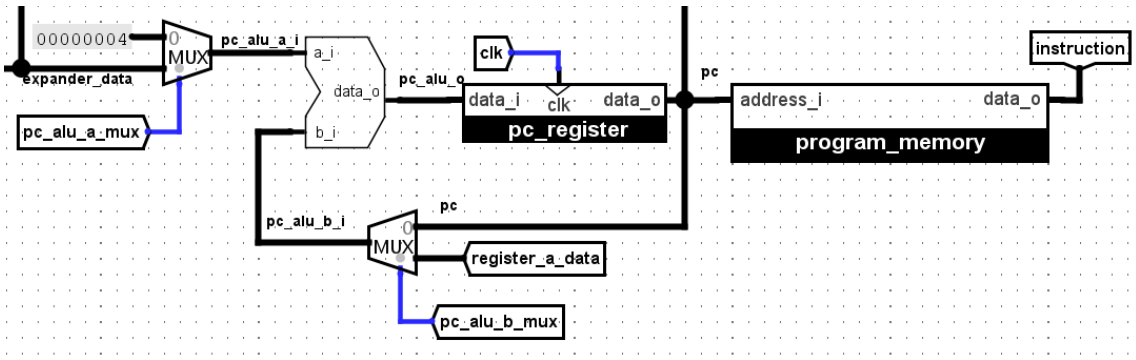
2.3 İŞLEYİŞ

RISC işlemciler çalışma döngüsünde 5 adet işleme sahiptir. Bu işlemler fetch (getir), decode (çöz), execute (çalıştır), memory access (belleğe eriş), writeback (geri yaz) işlemleridir (Wikipedia, 2023).

2.3.1 GETİR AŞAMASI

Getirme aşamasında program sayacı birimi tarafından, her yeni clock sinyalinde, program belleğine o an çalıştırılması istenen komutun adres bilgisi oluşturulmaktadır. Oluşturulan adres, program hafızasında ilgili komutu işaret etmektedir. Program memory modülü ilgili adresteki komutu çıkışına ileterek getirme aşamasını sonlandırlır.

Her yeni getirme aşamasında, program sayacının içeriği RISC standartı olan +4 değeri veya dallanma komutundan gelen veriye göre arttırılmaktadır. Böylece bellekten alınacak bir sonraki veri baytı işaret edilmektedir (Morris, 1985). Şekil 34'te tasarlanan işlemcideki getir işlemi için tasarlanan modül verilmiştir.

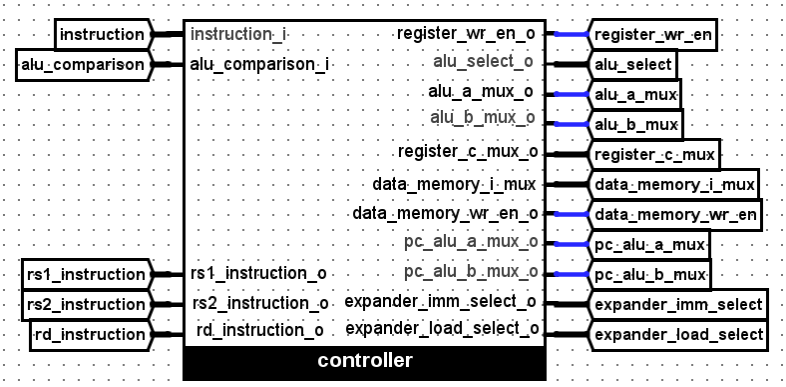


Şekil 34 - Getir işlemi için kullanılan yapı

2.3.2 ÇÖZ AŞAMASI

Komutlar getir aşamasından sonra, opcode ve diğer tanımlayıcı bitlerine göre çözülmeli ve iç denetim sinyalleri oluşturulmalıdır.

Bu tasarımda RISC-V standartlarına göre controller modülü çözme işlemi yapılmakta ve sonrasında iç denetim sinyalleri sentezlenmektedir. Şekil 35'te controller modülüne komutun girdisi ve denetim sinyallerinin çıkışı görülmektedir.



Şekil 35 - Çözme aşaması için kullanılan yapı

2.3.3 ÇALIŞTIR AŞAMASI

Bu aşamada komut tarafından belirtilen işlem gerçekleştirilmektedir (Eren, 2021). Çalıştır aşaması Başlık 2.2.1.1'de verilen ALU modülü üzerinde gerçekleştirilmektedir.

Komutlar çözüldükten sonra sentezlenen denetim sinyalleri ve iç yapıdaki MUX'lar kullanılarak ilgili veriler doğru yerlere iletilmektedir.

2.3.4 BELLEĞE ERİŞ AŞAMASI

Belleğe erişim gerektiren komut geldiğinde istenen adresin sentezlenmesinin ardından adresin belleğe iletilerek verinin getirilmesi işlemidir.

Tasarlanan işlemcide data memory modülü tarafından gerçekleştirilmekte olup adres bilgisi için ALU ve expander modülü kullanılmaktadır.

2.3.5 GERİ YAZ AŞAMASI

Sonucun geri yazılmasını gerektiren komut yapılarında kullanılan işlemdir.

Register modülü üzerinden, ALU, data memory, program sayacı veya expander modüllerinden gelen ilgili verinin hedef yazmacına yazılmasıyla gerçekleştirilmektedir.

Getir, çöz, çalıştır, belleğe eriş ve geri yaz işlemleri tasarlanan işlemci tek vuruşluk bir yapıya sahip olduğundan aynı anda gerçekleşmektedir. Pipeline (boru hattı) tekniği kullanan işlemcilerde bu aşamalar ardışıl işlemler olarak gerçekleştirilmektedirler.

2.3.6 KOMUT TİPLERİNE GÖRE İŞLEYİŞ

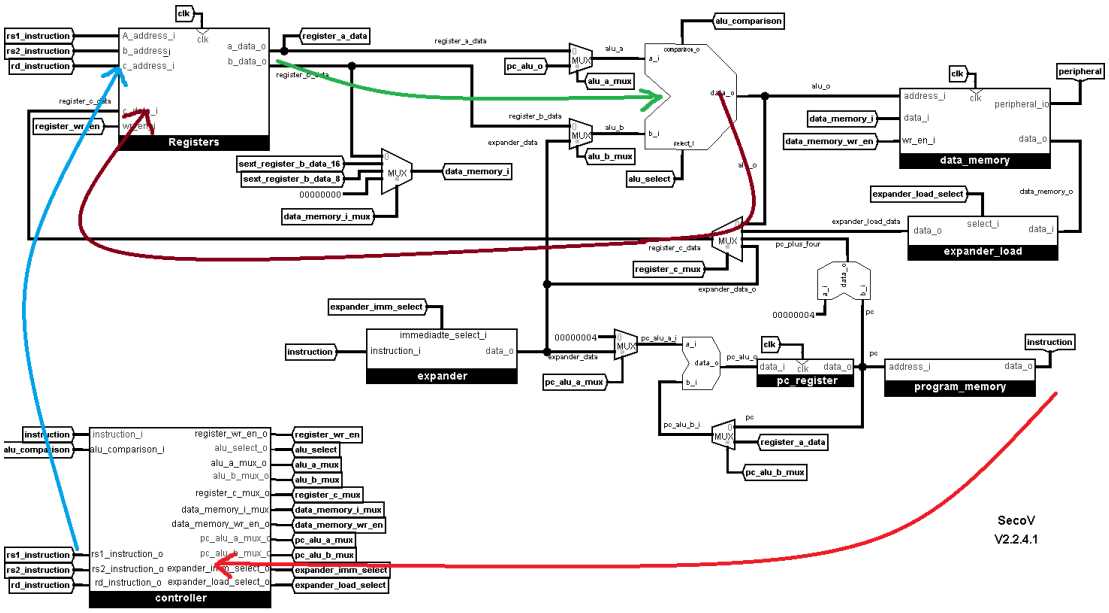
İşlemcinin iç yapısının işleyişi genel olarak Başlık 2.2.1.5 Expander Modülü bölümünde verilen Şekil 13'teki komut tiplerine göre sınıflandırılabilir.

İlgili komut tipleri için gerekli denetim sinyalleri Ek 2'deki tabloda verilmiştir. Bu tablodan yola çıkarak R tipi, I tipi, I Load tipi, S tipi, B tipi, U tipi ve J tipi olmak üzere toplam 7 farklı yapıda incelenmiş olup verilerin nasıl ve nereden geldiği mikro mimari üzerinde anlatılmıştır.

2.3.6.1 R TİPİ KOMUTLARIN İŞLEYİŞİ

İki farklı yazmaç verisi üzerinde aritmetik veya mantıksal işlem yapmak için kullanılmaktadır. İki farklı yazmaçtan alınan veri ALU'ya iletilmektedir. Sonrasında ALU'nun çıktısı hedef yazmacına iletilerek sonuç ilgili yazmaca yazılmaktadır.

Şekil 36'da ilgili komutun program memory modülünden controller modülüne iletilmesi kırmızı, üzerinden işlem yapılacak yazmaç adresleri mavi, ilgili yazmaçların veri çıktıları yeşil ve ALU'dan yazmaçlara geri yazılacak veri yolu kahverengi olarak gösterilmiştir.

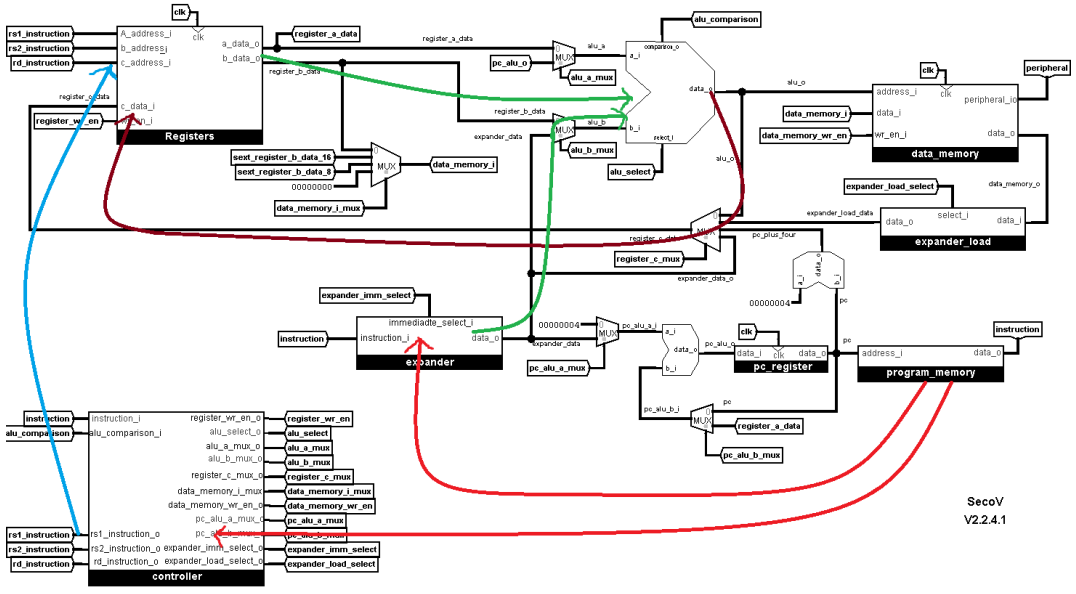


Şekil 36 - R tipi komutların işleyişi

2.3.6.2 I TİPİ KOMUTLARIN İŞLEYİŞİ

Bir adet yazmaç ve bir adet sabit (anlık) veri üzerinde aritmetik veya mantıksal işlem yapmak için kullanılmaktadır. Bir adedi yazmaçtan, diğeri direkt olarak komuttan alınarak uygun formata çevirilen veri ALU'ya iletilmektedir. Sonrasında ALU'nun çıktısı hedef yazmacına iletilerek sonuç ilgili yazmaca yazılmaktadır.

Şekil 37'de ilgili komutun program memory modülünden controller modülüne ve expander modülüne iletilmesi kırmızı, üzerinden işlem yapılacak yazmaç adresleri mavi, ilgili yazmaç ve anlık değerin veri çıktıları yeşil ve ALU'dan yazmaçlara geri yazılacak veri yolu kahverengi olarak gösterilmiştir.

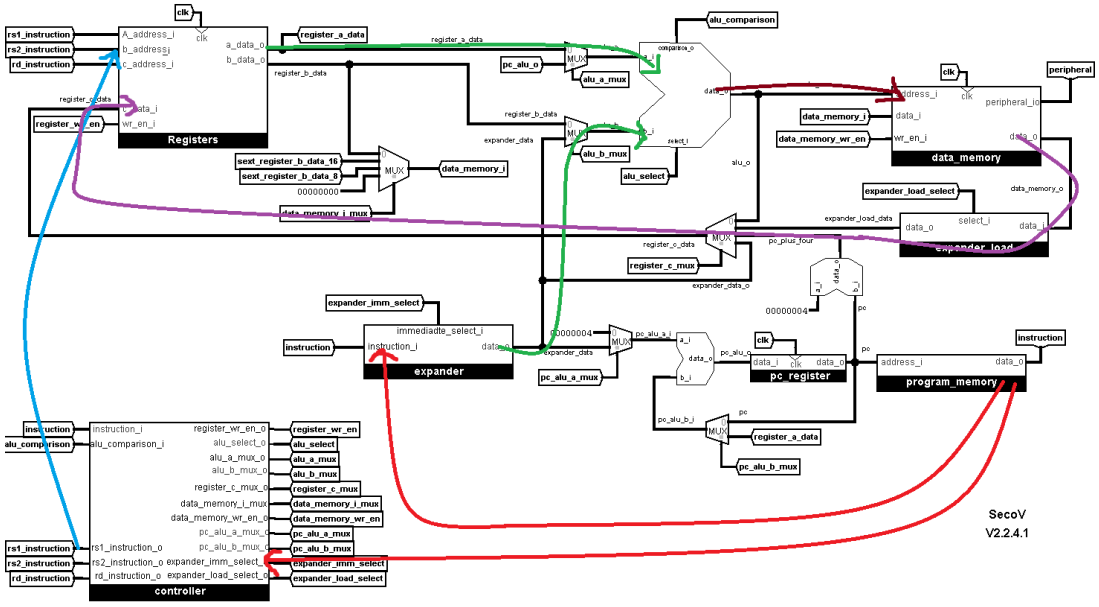


Şekil 37 - I tipi komutların işleyişi

2.3.6.3 I LOAD TİPİ KOMUTLARIN İŞLEYİŞİ

Hedef yazmacına kalıcı hafızada (data memory) saklanan veriyi yazmak için kullanılmaktadır. Data memorynin adres girişine yazmaçtaki sabit bir değer ile komut içerisinde verilen anlık değer toplanılarak elde edilen değer iletilmektedir. Bu adres değerinde saklanan veri, hedef yazmacına yazılmaktadır.

Şekil 38'de ilgili komutun program memory modülünden controller modülüne ve expander modülüne iletilmesi kırmızı, üzerinden işlem yapılacak yazmaç adresi mavi, adres değeri için kullanılacak yazmaç değeri ve anlık değerin veri çıktıları yeşil ve data memoryden yazmaca geri yazılacak veri yolu mor olarak gösterilmiştir.

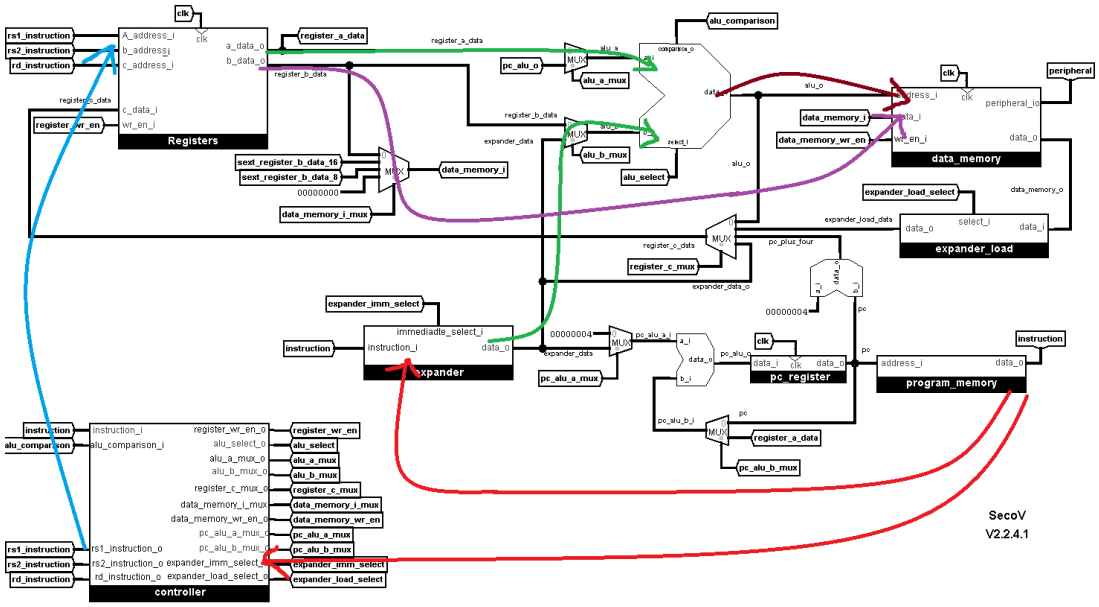


Şekil 38 - I Load tipi komutların işleyişi

2.3.6.4 S TİPİ KOMUTLARIN İŞLEYİŞİ

Yazmaçtaki veriyi kalıcı hafızaya ileten komutlar için kullanılmaktadır. Yazılan veri dışındaki bir yazmaç değerinde saklanan bir değere komutta verilmiş bir anlık değer eklenerek elde edilen adresteki kalıcı hafızaya veri yazılmaktadır.

Şekil 39'da ilgili komutun program memory modülünden controller modülüne ve expander modülüne iletilmesi kırmızı, üzerinden işlem yapılacak yazmaç adresleri mavi, ilgili yazmaç ve anlık değerin veri çıktıları yeşil ve ALU'dan data memory modülüne aktarılan adres çıktısı kahverengi, yazılmak istenen veriyi taşıyan yazmaç veri yolu mor olarak gösterilmiştir.

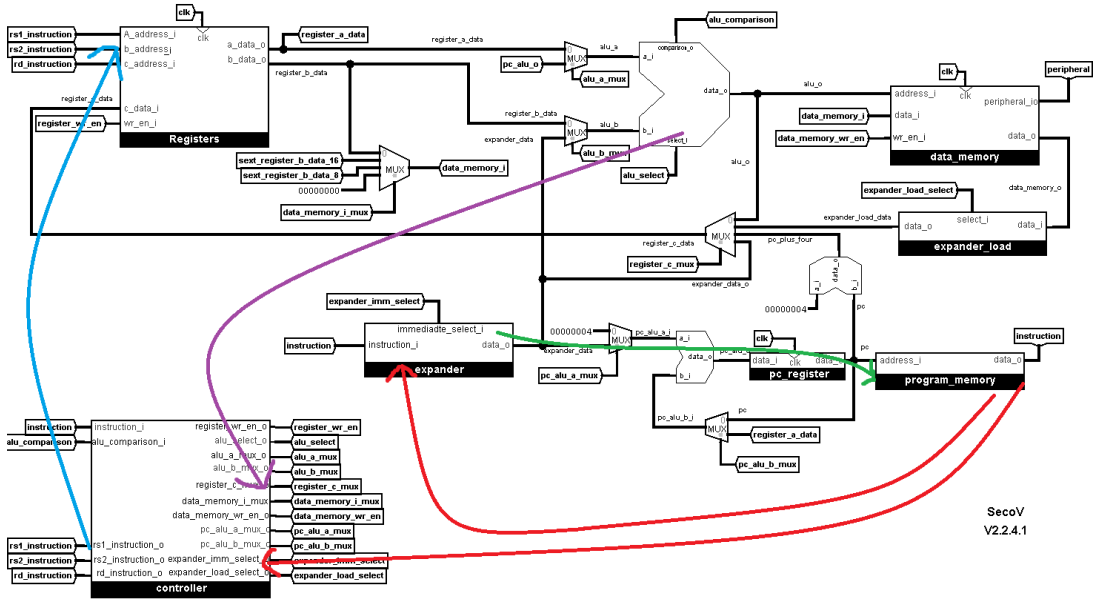


Şekil 39 - S tipi komutların işleyişi

2.3.6.5 B TİPİ KOMUTLARIN İŞLEYİŞİ

Programcı tarafından belirlenen koşullar sağlandığında yazılımın uygun yere dallanmasını sağlamaktadır. İki farklı yazmaçtaki veriyi kıyaslayarak istenilen şartın gerçeklenip gerçeklenmediğini kontrol etmektedir. Şart sağlandığında program sayacına gelen komutta direkt olarak verilen anlık değerini ekleyerek programın akışına yön verir.

Şekil 40'da ilgili komutun program memory modülünden controller modülüne ve expander modülüne iletilmesi kırmızı, kıyaslanacak yazmaç adresleri mavi, program sayacına eklenecek anlık değerin veri çıktıları yeşil ve ALU'dan controller modülüne aktarılan kıyaslama veri yolu mor olarak gösterilmiştir.

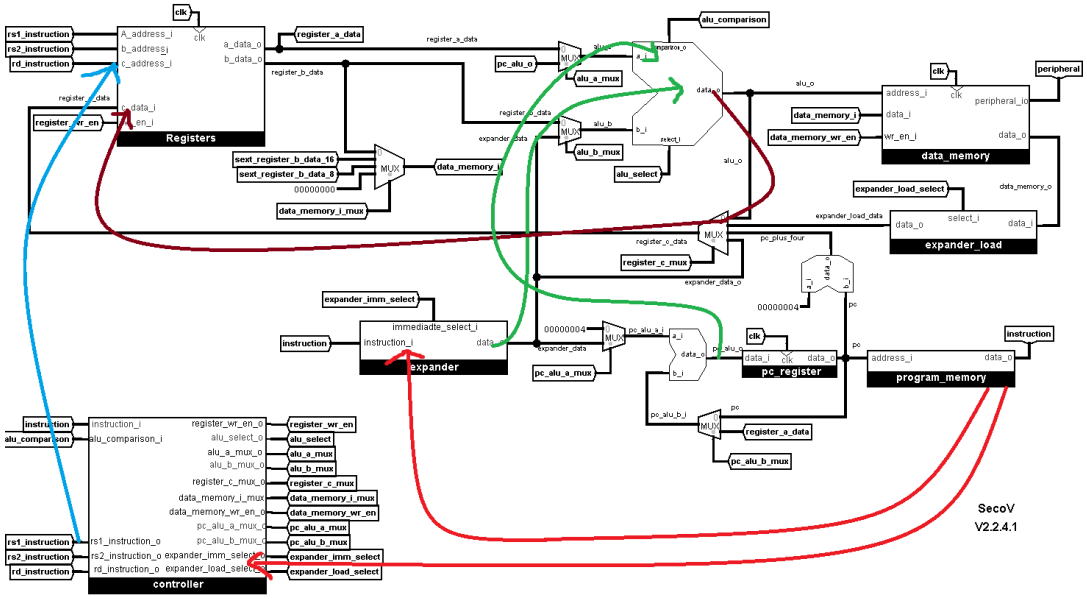


Şekil 40 - B tipi komut işleyişi

2.3.6.6 U TİPİ KOMUTLARIN İŞLEYİŞİ

İstenen yazmaçlar üzerindeki 20 bitlik veriyi değiştirmek için kullanılır. 32 bitlik bir yazmaç üzerindeki verinin değiştirilebilmesi için 20 bitlik işlemler U tipi, 12 bitlik işlemler I Load tipi komutlarla yapılır. Direkt olarak komuttan uygun formata çevrilen veya program sayacından alınan veri hedef yazmacına yazılmaktadır.

Şekil 41'de ilgili komutun program memory modülünden controller modülüne ve expander modülüne iletilmesi kırmızı, üzerinden işlem yapılacak yazmaç adresi mavi, program sayacı ve anlık değerin veri çıktıları yeşil ve ALU'dan yazmaca geri yazılacak veri yolu kahverengi olarak gösterilmiştir.

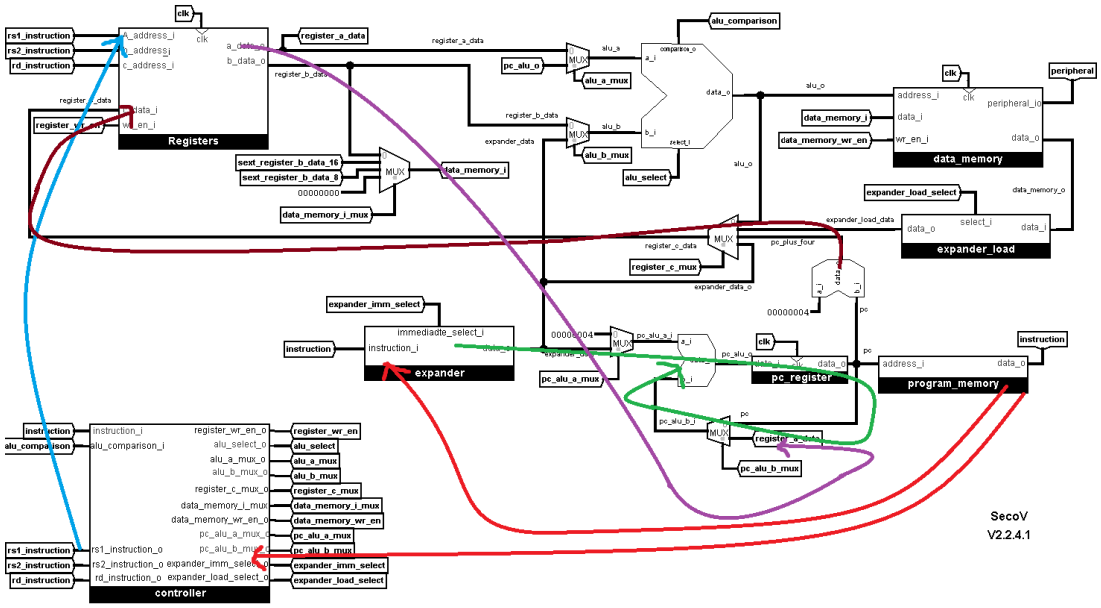


Şekil 41 - U tipi komutların işleyişi

2.3.6.7 J TİPİ KOMUTLARIN İŞLEYİŞİ

Program sayacını değiştirmek için kullanılır. Direkt olarak komuttan uygun formata çevrilen veya seçilen kaynak yazmacıyla toplanan alınan veri program sayacı yazmacına yazılmaktadır. Program sayacının eski değeri ise başka bir yazmaca istenildiğinde geri dönülebilmesi için saklanmaktadır.

Şekil 42'de ilgili komutun program memory modülünden controller modülüne ve expander modülüne iletilmesi kırmızı, PC'nin yazılacağı hedef yazmaç adresi mavi, program sayacına yazılacak anlık değer yeşil ve ALU'dan yazmaca geri yazılacak veri yolu kahverengi olarak gösterilmiştir.



Şekil 42 - J tipi komutların işleyişi

3. BULGULAR

Başlık 3.1 Litaratür Taraması, Başlık 3.2 Bulgular ve Başlık 3.3 Projenin Gerçek Hayatta Uygulaması'da çalışmanın başka çalışmalarla kıyaslanması, çalışmanın teorik ve pratik sonuçları verilmiştir.

3.1 LİTERATÜR TARAMASI

Çalışmada tasarlanan işlemci RISC mantığı kullanmaktadır. RISC, basit komutlar ve donanımları ile tasarlanmış bir yapıdır. RISC, yaygın olarak kullanılan diğer yapı olan CISC'e göre oldukça yalın komutlar ve donanımlara sahip bir yapıdır. RISC, basit komutlar, basit donanım ve beraberinde güç tasarrufu avantajları sağlamaktadır. Tablo 8'de avantaj ve dezavantajlarıyla CISC ve RISC yapılarının kıyaslaması verilmiştir.

Tablo 8 - CISC ve RISC mimarilerinin kıyaslaması (Microcontrollerslab, 2023)

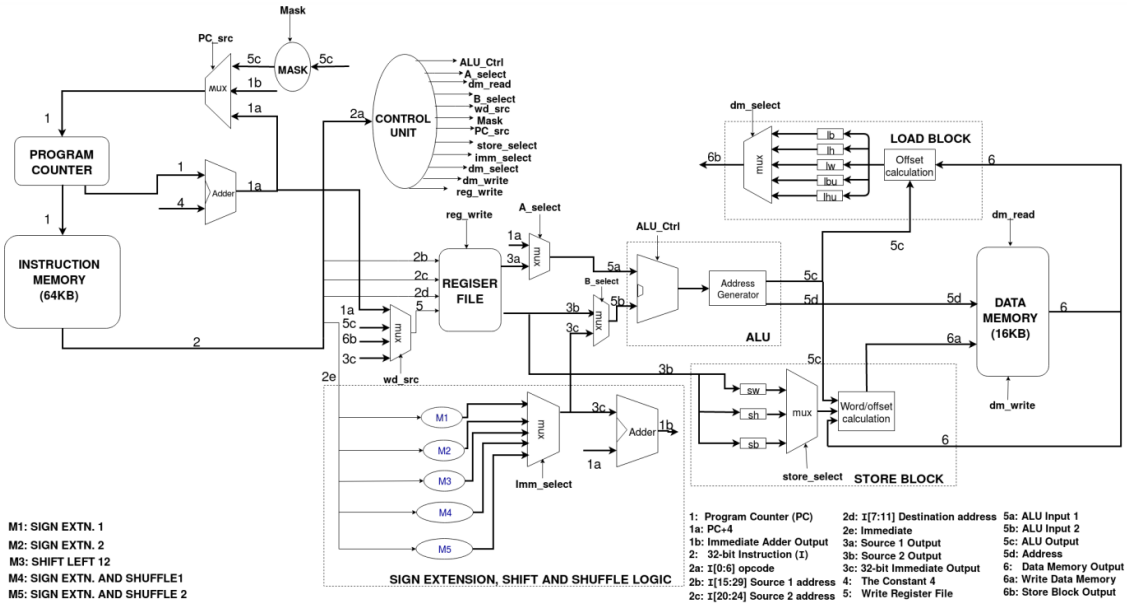
CISC	RISC
Avantajlar	**Avantajlar**
Programlaması basit	Basit komutlar
Az derleyici yükü	Basit donanım
Az register ihtiyacı	Sabit uzunluklu komutlar
Az kod boyutu	Komutların kullandığı saat vuruşu sabit
Dezavantajlar	Düşük güç tüketimi
Karmaşık komutlar	**Dezavantajlar**
Karmaşık donanım	Programlaması karmaşık
Değişken uzunluklu komutlar	Çok derleyici yükü
Komutların kullandığı saat vuruşu değişken	Çok register ihtiyacı
Yüksek güç tüketimi	Çok kod boyutu

Bilgisayar mimarisi üzerine yapılan çalışmaların büyük çoğunluğu Verilog HDL ile yazılmıştır. Yaygın çalışmaların aksine, modülerlik kabiliyeti yüksek, ülkemiz sanayii ve savunma sanayisinde daha yaygın olarak kullanılan VHDL kullanılarak modüller tamamen özgün şekilde tasarlanmış olup başlık 2. Materyal ve Yöntem altında detaylı şekilde işleyişleri açıklanmıştır.

Literatür taraması yapıldığında, benzer çalışmaların ağırlıklı olarak RISC mimarilerinden MIPS ile yapıldığı görülmüştür. MIPS mimarisi 1980'li yıllara uzanan oldukça eski bir geçmişe sahiptir (Wikipedia, 2023).

Yapılan çalışmada 2015'te sunulan RISC-V mimarisi kullanılmıştır. (Patterson & Hennessy, 2017) Oldukça modern ve geçmiş RISC mimarilerinin eksiklerine yönelik akılcı çözümler barındıran RISC-V, açık kaynak kodlu ve tamamen ücretsiz bir komut seti olduğu için günümüzde oldukça popülerdir. Bu popülerliğin getirdiği avantaj olan geliştirme ortamları sayesinde donanım tasarımı yapmak ve bunu gerçek sistemlere adapte etmek oldukça kolay olmaktadır.

Şekil 43'te, Indian Institute of Technology Patna'da yapılan bir çalışma verilmiştir (Dennis, Priyam, Virk, Agrawal, & Sharma, 2017). Verilog HDL ile tasarlanan bu çalışma, tasarlanan işlemci ile kıyaslandığında aynı gereksinimler için benzer donanım çözümleri ve yaklaşımlarına rastlanılmıştır.



Şekil 43 - Benzer işlemci örneği (Dennis, Priyam, Virk, Agrawal, & Sharma, 2017)

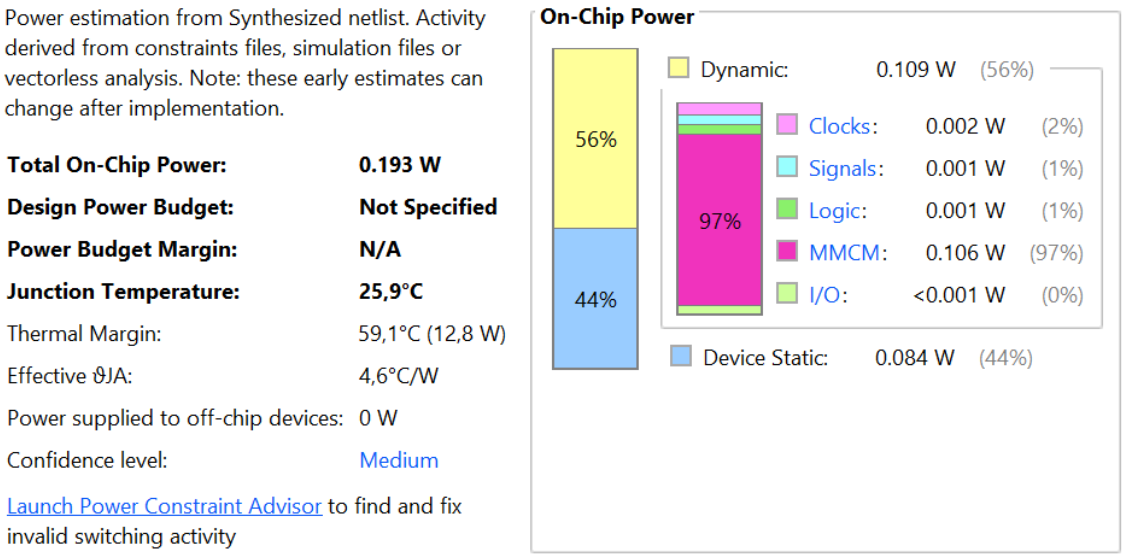
3.2 BULGULAR

Elde edilen sonuçta RV32I komut seti ile, tüm tamsayı işlemleri yapabilen, aynı zamanda neredeyse diğer tüm komut seti eklentilerini (atomik işlemler hariç) taklit edebilem bir işlemci yapısı elde edilmiştir (Waterman, Asanovic, & SiFive Inc., 2017).

Tasarlanan işlemcinin olabildiğince yalın tasarlanması amaçlanmış olup bu doğrultuda sentez sonucunda kullanılan FPGA kaynakları ve güç tüketimi verisi Şekil 44 ve Şekil 45'te verilmiştir.

Name	Slice LUTs (63400)	Slice Registers (126800)	Bonded IOB (210)	BUFGCTRL (32)	MMCME2_ADV (6)
N cpu	363	142	17	2	1
main_alu (alu)	1	0	0	0	0
main_clk_divider (clk_divider_ip)	0	0	0	2	1
main_data_memory (data_memory)	0	16	0	0	0
main_pc (pc_register)	331	30	0	0	0
main_pc_adder (pc_adder)	0	0	0	0	0
main_registers (registers)	31	96	0	0	0

Şekil 44 - Sentez sonrasında kullanılan FPGA kaynakları



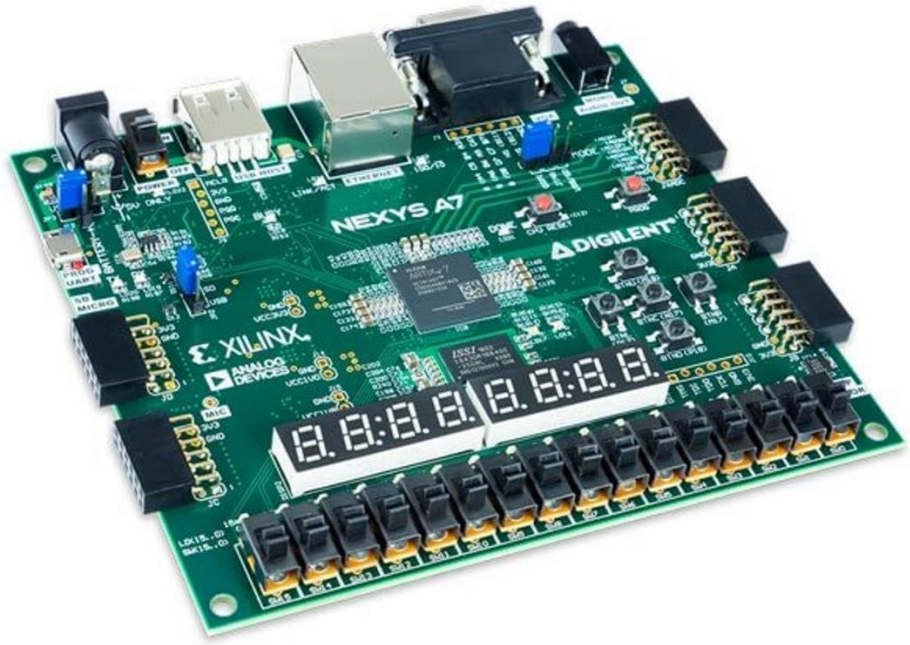
Şekil 45 - Tasarlanan işlemcinin güç tüketim analizi

İşlemcinin çekirdek tasarımının performansını test etmek için oldukça az sayıda bellek modülleri çalışmaya dahil edilmesi uygun görülmüştür. Bu doğrultuda Şekil 44 ve Şekil 45'teki sonuçlar, 2 KiB data memory ve 256 B program memory ile elde edilmiştir.

Sonuçlar incelendiğinde tasarlanan işlemci, Indian Institute of Technology Patna'da yapılan ve internet ortamındaki benzer çalışmalarla kıyaslandığında oldukça düşük kaynak kullanımı ancak oldukça fazla güç tüketimi görülmüştür.

3.3 PROJENİN GERÇEK HAYATTA UYGULAMASI

Tasarlanan işlemcinin gerçek hayat üzerinde çalıştırılması için Şekil 46'da verilen Digilent firmasının Nexys A7 geliştirme kartı kullanılmıştır.



Şekil 46 - Digilent Nexys A7 geliştirme kartı

Üzerinde sıcaklık, mikrofon, ivme ölçer, VGA çıkışı, çeşitli butonlar, anahtarlar ve LEDler bulunduran bu geliştirme kartı, çalışma esnasında önemli ölçüde kolaylık sağlamıştır.

Başlık 2.2.1.3'te de anlatıldığı üzere çevresel giriş/çıkış sinyallerinin data memory adreslerinden bir tanesine tanımlanması sayesinde bellekteki verilerin okunması/değiştirilmesi ile dış dünya ile etkileşim sağlanmaktadır.

Bu yapı kullanılarak Şekil 47'de verilen Assembly komutları kullanılarak gerçek hayatta 1 Hz frekansında yanıp sönen bir led uygulaması yapılmıştır.

```
--test 17 flip flop (real)

lui x2, 0x00bed      x"00bec137"
addi x2, x2, 0xc20   x"c2010113"
addi x1, x1, 1       x"00108093"
blt x1, x2, -4       x"fe20cee3"
xori x3, x3, -1      x"fff1c193"
sw x3, 510(x0)       x"1e302f23"
and x1, x1, x0       x"0000f0b3"
beq x0, x0, -20      x"fe0006e3"
```

Şekil 47 - LED uygulaması Assembly kodları

Şekil 47'de verilen kodda LEDlerin belirli bir süre geçtikten sonra çalışma durumunun evrilmesi sağlanmıştır. Bu işlem için bir sayaç, yazmaçların bir kıyaslama yazmacı ile kıyaslanması sonucunda gerçekleşmektedir.

Öncelikle kıyaslama için kullanılacak olan X2 yazmacına 1. ve 2. Satırlardaki LUI VE ADDI komutları kullanılarak değeri 125,000,000 olan bir eşik değeri kaydedilmektedir. Bu eşik değer çalışma frekansına bağlı olarak ledlerin yanıp sönme frekansını belirlemektedir.

3. satırda kullanılan ADDI komutu ile her çalıştırıldığında kıyaslama için kullanılacak diğer yazmacın değeri 1 arttırılmaktadır. Bu yazmaç, ADDI komutunun çalışma sayısını saymaktadır. Sabit periyotlu clock sinyal kaynağına bağlı olarak çalışan işlemcide clock sinyali sayısı zaman sayacı olarak kullanılmaktadır.

4. satırda kullanılan BLT komutu ile X1 yazmacının X2 yazmacından küçük olması durumunda program sayacı değiştirilerek bir önceki komut olan 3. Satırdaki ADDI komutu tekrar çalıştırılmaktadır. X1 yazmacının X2 yazmacına eşit veya büyük olması durumunda ise şart doğru olmayacağından yazılım bir sonraki satırı işlemeye geçmektedir.

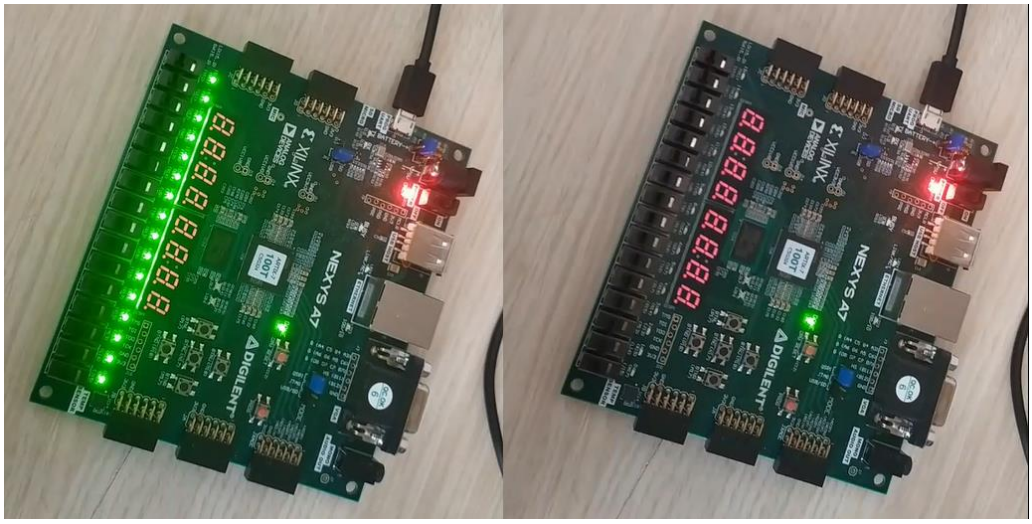
5. satırda kullanılan XORI komutu, bir sonraki satırda çıkışa yönlendirilecek olan X3 yazmacının tüm bitlerini terslemektedir. Bu sayede ledlerin yanıyorsa sönmeleri, sönüyorsa yanmaları sağlanmaktadır.

6. satırda kullanılan SW komutu, çıkış pinlerine tanımlanmış data memory adresi olan 510. adresteki veriye, çıkış için değeri güncellenen X3 yazmacını yazmaktadır.

7. satırda kullanılan AND komutu ile kıyaslama için kullanılan X1 yazmacı sıfırlanarak döngü baştan başlatılmaktadır.

8. satırda kullanılan BEQ komutu ile yazılım hiçbir koşul aranmaksızın 3. satırdaki ADDI komutuna dönecek şekilde tekrar program sayacını değiştirmektedir.

Yukarıda verilen işlemler çalıştırıldığında Şekil 48'de görüldüğü üzere ledlerin yanıp söneceği basit bir uygulama işlemci üzerinde çalıştırılmıştır.



Şekil 48 - Led uygulamasının çalışması

4. SONUÇLAR

Mikro mimari tasarımının yalınlığı geliştirme aşamasındaki ilk öncelik olarak belirlenmiştir. Bu yalınlık bazı karmaşık işlem gerektiren komutların çeşitli modüllerde davranışsal olarak tanımlanması ile sağlanmıştır.

Yalınlığı ön planda olması istenen tasarımdaki bir diğer tercih modüler mimaridir. Modüler mimarinin kullanılması sonucu benzer örneklere göre okuması, geliştirmesi ve özelleştirmesi kolay bir tasarım elde edilmiştir.

Tasarlanan işlemcinin kritik zaman gereksinimlerini sağlayamaması nedeniyle yavaşlatılması gerektiği Başlık 2.2.2.11'de daha detaylı olarak anlatılmıştı. Bu gereksinimlerin sağlanmasının bir yönetimi pipeline tekniği kullanılmasıdır. Bunun sonucunda işlemci, daha yüksek hızlarda ve performansta çalışabilir (Ramamoorthy & Li, 1977).

Ancak pipeline, işlemci performansını arttıran bir yapıyken donanım karmaşıklığını arttırır. Yapılan çalışmadaki öncelik basitlik ve işlemci mimarisinin anlaşılması olduğundan daha basit bir yapı olan tek vuruşluk bir işlemci tasarlanmıştır. Yüksek performans beklentisi ile tasarlanmayan bu çalışmada yalınlık olarak geri dönen bu kararın kritik zaman gereksinimleri ve performansa etkisi direkt görülmüştür.

Tasarım sonrasında düşük kaynak kullanımına karşılık yüksek güç tüketimi istenmeyen bir durumdur. Karmaşık bir yapı oluşturulurken dikkat edilmesi gereken bir konu olan güç tüketiminin çeşitli yöntemlerle tasarımın en başından itibaren göz önüne alınması gerektiği, elde edilen performans parametreleri sonrası anlaşılmıştır.

KAYNAKÇA

Aktaş, M., & Örencik, B. (2011). *FPGA Mimarisi. (A*racılığıyla Başa, B., (2014). FPGA Yapıları ile Dijital Osiloskop Gerçeklemesi)

Asanović, K. (2021, Aralık). Keynote: State of the Union.

Bergeron, J. (2000). *Writing Testbenches: Functional Verification of HDL Models.*

Dennis, D. K., Priyam, A., Virk, S. S., Agrawal, S., & Sharma, T. (2017). Single Cycle RISC-V Micro Architecture Processor and its FPGA Prototype. *2017 7th International Symposium on Embedded Computing and System Design (ISED).* Durgapur, India.

Eren, İ. (2021). *Five stages of RISC pipeline*. Level Up Coding: https://levelup.gitconnected.com/five-stages-of-risc-pipeline-aad0c3eb1233 adresinden alındı

Ergin, O. (2020). *YouTube*. Şubat 2023 tarihinde https://youtu.be/T5VYmIereuY adresinden alındı

HardwareBee. (2023). HardwareBee: https://hardwarebee.com/verilog-vs-vhdl-what-to-choose/ adresinden alındı

Microcontrollerslab. (2023). *Difference between RISC and CISC*. Microcontrollerslab: https://microcontrollerslab.com/difference-between-risc-and-cisc/ adresinden alındı

Mohd, N., Zainol, M., & Sohiful, A. (2015). *Field Programmable Gate Array (FPGA): From Conventional to Modern Architectures.*

Morris, N. M. (1985). The CPU and its Fetch-Execute Cycle. N. M. Morris içinde, *Microelectronic and Microprocessor-based Systems.*

Patterson, D. A., & Hennessy, J. L. (2017). *Computer Organization and Design RISC-V Edition.* https://en.wikipedia.org/wiki/MIPS_architecture adresinden alındı

Ramamoorthy, C. V., & Li, H. F. (1977, March). Pipeline Architecture. *Computing Surveys Vol. 9 No. 1*.

Smith, D. J. (1996). VHDL & Verilog Compared & Contrasted - Plus Modeled Example Written in.

Topaloğlu, N. (2021). *Mikroişlemciler Ve Assembly Dili.*

VHDLwhiz. (2022, Aralık). VHDLwhiz: https://vhdlwhiz.com/should-i-learn-vhdl-if-verilog-is-becoming-more-popular/ adresinden alındı

Waterman, A. S. (2016). *Design of the RISC-V Instruction Set Architecture.* içinde

Waterman, A., Asanovic, K., & SiFive Inc. (2017). The RISC-V Instruction Set Manual. Berkeley: CS Division, EECS Department, University of California.

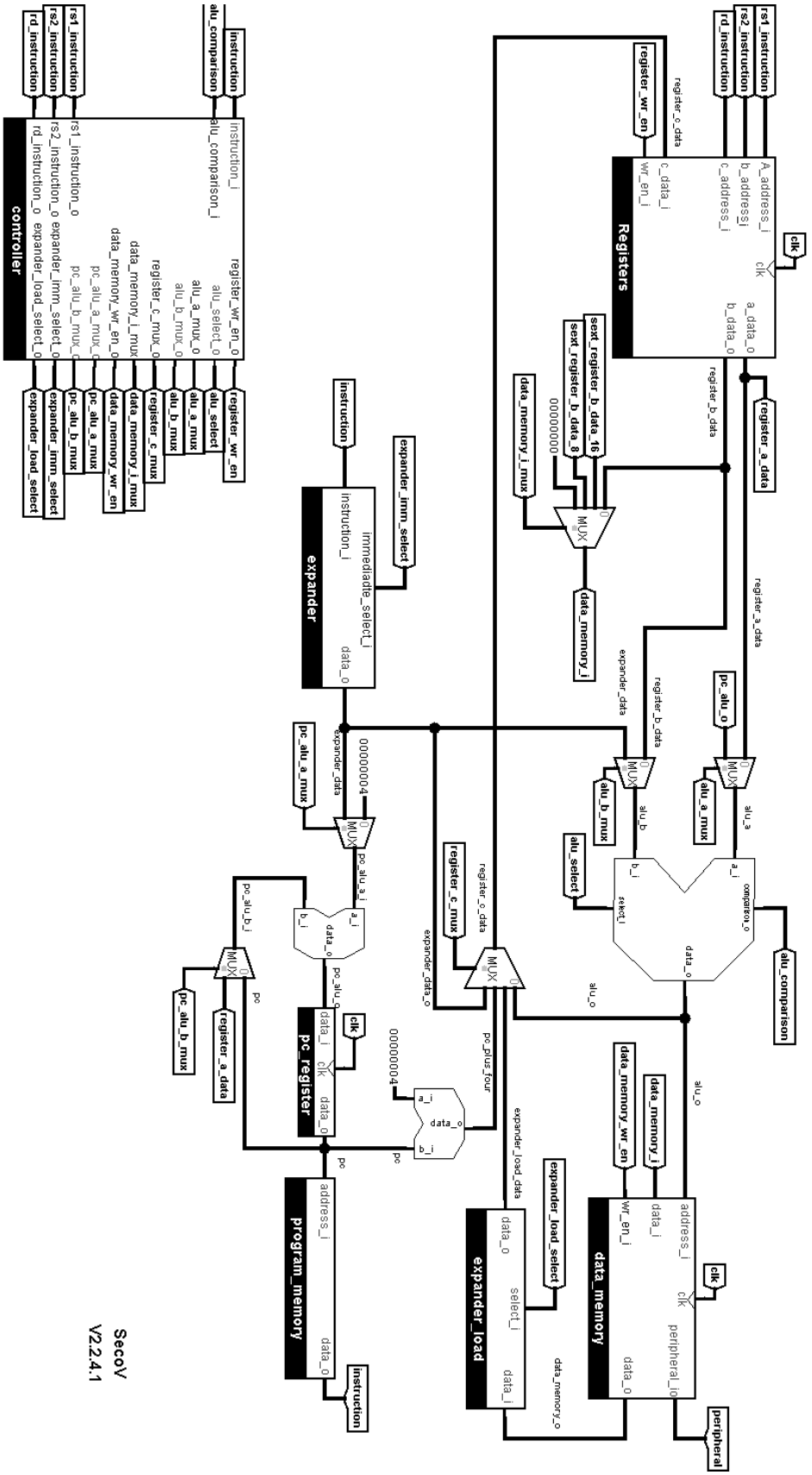
Wikipedia. (2023). *Classic RISC pipeline*. Wikipedia: https://en.wikipedia.org/wiki/Classic_RISC_pipeline adresinden alındı

Wikipedia. (2023, Haziran). *Hardware description language*. Wikipedia: https://en.wikipedia.org/wiki/Hardware_description_language adresinden alındı

Wikipedia. (2023). *MIPS Architecture*. Wikipedia: https://en.wikipedia.org/wiki/MIPS_architecture adresinden alındı

EKLER

Ek 1 – İşlemcinin mikro mimari tasarımı



Ek 2 - Denetim sinyali değerleri

Komut tipi	Denetim sinyali	Sinyal değeri	Komut	Komut tipi	Denetim sinyali	Sinyal değeri	Komut
R	rs1_instruction	instruction[19:15]		I load	rs1_instruction	instruction[19:15]	
	rs2_instruction	instruction[24:20]			rs2_instruction	X	
	rd_instruction	instruction[11:7]			rd_instruction	instruction[11:7]	
	register_wr_en	1			register_wr_en	1	
	alu_select	0000	ADD		alu_select	0000	
		0001	SUB		alu_a_mux	0	
		0010	SLL		alu_b_mux	1	
		0011	SLT		register_c_mux	01	
		0100	SLTU		data_memory_i_mux	X	
		0101	XOR		data_memory_wr_en	0	
		0110	SRL		pc_alu_a_mux	0	
		0111	SRA		pc_alu_b_mux	0	
		1000	OR		expander_imm_select	000	
		1001	AND		expander_load_select	000	LB
	alu_a_mux	0				001	LH
	alu_b_mux	0				010	LW
	register_c_mux	00				011	LBU
	data_memory_i_mux	X				100	LHU
	data_memory_wr_en	0					
	pc_alu_a_mux	0					
	pc_alu_b_mux	0					
	expander_imm_select	X					
	expander_load_select	X					

Komut tipi	Denetim sinyali	Sinyal değeri	Komut	Komut tipi	Denetim sinyali	Sinyal değeri	Komut
I	rs1_instruction	instruction[19:15]		S	rs1_instruction	instruction[19:15]	
	rs2_instruction	X			rs2_instruction	instruction[24:20]	
	rd_instruction	instruction[11:7]			rd_instruction	X	
	register_wr_en	1			register_wr_en	0	
	alu_select	0000	ADDI		alu_select	0000	
		0011	SLTI		alu_a_mux	0	
		0100	SLTIU		alu_b_mux	1	
		0101	XORI		register_c_mux	X	
		1000	ORI		data_memory_i_mux	00	SB
		1001	ANDI			01	SH
		0010	SLLI			10	SW
		0110	SRLI		data_memory_wr_en	1	
		0111	SRAI		pc_alu_a_mux	0	
	alu_a_mux	0			pc_alu_b_mux	0	
	alu_b_mux	1			expander_imm_select	010	
	register_c_mux	00			expander_load_select	X	
	data_memory_i_mux	X					
	data_memory_wr_en	0					
	pc_alu_a_mux	0					
	pc_alu_b_mux	0					
	expander_imm_select	000					
	expander_load_select	X					

Ek 2 - Denetim sinyali değerleri (devamı)

Komut tipi	Denetim sinyali	Sinyal değeri	Komut	Komut tipi	Denetim sinyali	Sinyal değeri	Komut
B	rs1_instruction	instruction[19:15]		U	rs1_instruction	X	
	rs2_instruction	instruction[24:20]			rs2_instruction	X	
	rd_instruction	X			rd_instruction	instruction[11:7]	
	register_wr_en	0			register_wr_en	1	
	alu_select	0001			alu_select	X	
	alu_a_mux	0			alu_a_mux	0	LUI
	alu_b_mux	0				1	AUIPC
	register_c_mux	X			alu_b_mux	1	
	data_memory_i_mux	X			register_c_mux	11	LUI
	data_memory_wr_en	0				00	AUIPC
	pc_alu_a_mux	alu_comparison[2]	BEQ		data_memory_i_mux	X	
		not alu_comparison[2]	BNE		data_memory_wr_en	1	
		alu_comparison[1]	BLT		pc_alu_a_mux	0	
		not alu_comparison[1]	BGE		pc_alu_b_mux	0	
		alu_comparison[0]	ALU		expander_imm_select	001	
		not alu_comparison[0]	BGEU		expander_load_select	X	
	pc_alu_b_mux	0					
	expander_imm_select	011					
	expander_load_select	X					

Komut tipi	Denetim sinyali	Sinyal değeri	Komut	Komut tipi	Denetim sinyali	Sinyal değeri	Komut
UJ	rs1_instruction	X		IJ	rs1_instruction	instruction[19:15]	
	rs2_instruction	X			rs2_instruction	X	
	rd_instruction	instruction[11:7]			rd_instruction	instruction[11:7]	
	register_wr_en	1			register_wr_en	1	
	alu_select	X			alu_select	X	
	alu_a_mux	X			alu_a_mux	X	
	alu_b_mux	X			alu_b_mux	X	
	register_c_mux	10			register_c_mux	10	
	data_memory_i_mux	X			data_memory_i_mux	X	
	data_memory_wr_en	0			data_memory_wr_en	0	
	pc_alu_a_mux	1			pc_alu_a_mux	1	
	pc_alu_b_mux	0			pc_alu_b_mux	1	
	expander_imm_select	100			expander_imm_select	000	
	expander_load_select	X			expander_load_select	X	