T.C.
KIRIKKALE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
YÜKSEK LİSANS TEZİ

EĞİTİM AMAÇLI TEMEL BİR MİKROİŞLEMCİ TASARIMI VE GÖMÜLÜ SİSTEM UYGULAMASI

MEHMET BAKACAK

EYLÜL 2021

Bilgisayar Mühendisliği Anabilim Dalında Mehmet BAKACAK tarafından hazırlanan EĞİTİM AMAÇLI TEMEL BİR MİKROİŞLEMCİ TASARIMI VE GÖMÜLÜ SİSTEM UYGULAMASI adlı Yüksek Lisans Tezinin Anabilim Dalı standartlarına uygun olduğunu onaylarım.

Doç. Dr. Atilla ERGÜZEN
Anabilim Dalı Başkanı

Bu tezi okuduğumu ve tezin **Yüksek Lisans Tezi** olarak bütün gereklilikleri yerine getirdiğini onaylarım.

Doç. Dr. H Murat ÜNVER
Danışman

Jüri Üyeleri

Başkan	: Prof. Dr. Necaattin BARIŞÇI	_____
Üye (Danışman)	: Doç. Dr. H. Murat ÜNVER	_____
Üye	: Doç. Dr. Murat LÜY	_____

……/…../…….

Bu tez ile Kırıkkale Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu Yüksek Lisans derecesini onaylamıştır.

Prof. Dr. Recep ÇALIN
Fen Bilimleri Enstitüsü Müdürü

Bu çalışmayı canım kızım Esil ve sevgili eşim Nuray’ a ithaf ediyorum.

ÖZET

EĞİTİM AMAÇLI TEMEL BİR MİKROİŞLEMCİ TASARIMI VE GÖMÜLÜ SİSTEM UYGULAMASI

BAKACAK, Mehmet
Kırıkkale Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı, Yüksek Lisans Tezi
Danışman: Doç. Dr. Halil Murat ÜNVER
Eylül 2021, 134 sayfa

Baskı devre çıkarma ve lehimleme ile devre tasarlamak hem çok zor hem de maliyetlidir. Devre karmaşıklığının artması tasarım metotlarının değişimini de hızlandırmıştır. Hem tasarımcılara esneklik sağlamak, tasarımın süresini ve maliyetini düşürmek hem de tasarımın dijital ortamda uygulanabilir olması için "Donanım Tanımlama Dilleri" (Hardware Data Language-HDL) ortaya çıkmıştır. Donanım tanımlama dili kullanarak, Merkezi İşlem Birimi (Central Processing Unit-CPU) gibi karmaşık devreleri alt birimler halinde yazılımsal olarak tasarlamak ve tek bir Yonga Üzerinde Sistem (System on Chip-SoC) oluşturmak kolay hale gelmiştir. Tasarımları deneyip görmek için de Sahada Programlanabilir Kapı Dizileri (Field Programmable Gate Array-FPGA) gibi tekrar programlanabilir tümleşik devre kartları geliştirilmiştir.

Tasarlanan bir mikroişlemci FPGA kartlarına yüklenebilir ve FPGA kartları üzerinde çalıştırılabilir. Mikroişlemciyi bu şekilde somutlaştırmak, eğitim ortamında mikroişlemcinin yapısının ve çalışmasının daha iyi anlaşılmasını sağlar.

Bu tez çalışmasında kullanılan mikroişlemci, Karmaşık Komut Kümesi Bilgisayar (Complex Instruction Set Computer- CISC) işlemci mimarisi ve Von-Neumann bellek yapısı modelindedir. Tek Çevrim (Single Cycle), 16 bit veri yolu ve 12 bit adres yoluna sahiptir. Her bir komut 16-bit kelime uzunluğuna sahiptir. Hem lojik komutları hem de saklama ve yer değiştirme komutlarını yürütebilmektedir. Bu mikroişlemci, Morris

Mano’ nun 16 Bitlik mikroişlemcisi temel alınarak Quartus uygulaması üzerinde “yüksek hızlı tümleşik devre tanımlama dili” (Very High-Speed Integrated Circuit Hardware Description Language-VHDL) yazılım dili kullanılarak tasarlanmıştır. Tasarlanan mikroişlemci, ModelSim uygulamasında simule edilmiş ve Intel (Altera) firmasının ürettiği DE0 Nano FPGA kartı üzerinde çalıştırılmıştır. Tasarlanan sistem gerçeklenmiş ve doğrulaması yapılmıştır.

Anahtar kelimeler: CPU, Mikroişlemci, CISC, Mikrobilgisayar, FPGA, VHDL, SoC

ABSTRACT

A BASIC MICROPROCESSOR DESIGN AND EMBEDDED SYSTEM APPLICATION FOR EDUCATIONAL PURPOSES

BAKACAK, Mehmet
Kırıkkale University
Graduate School of Natural and Applied Sciences
Department of Computer Engineering, Master's Thesis
Supervisor: Assoc. Prof. Dr. Halil Murat ÜNVER
September 2021, 134 pages

It is very difficult and costly to design circuits with printed circuit boards and soldering. Increasing circuit complexity, it also accelerated the change of design methods. Hardware Data Language (HDL) has emerged to provide flexibility to designers, to reduce the time and cost of design, and to make the design applicable in the digital environment. Using hardware description language, it has become easy to design software in subunits complex circuits such as Central Processing Unit (CPU) and create a single System on Chip (SoC). In order to try and see the designs, reprogrammable integrated circuit boards such as Field Programmable Gate Array (FPGA) have been developed.

A designed microprocessor can be installed on FPGA cards and run on FPGA cards. To embody the microprocessor in this way, it provides a better understanding of the structure and operation of the microprocessor in the educational environment.

The microprocessor used in this thesis, is Complex Instruction Set The Computer (CISC) is in the model of the processor architecture and the Von-Neumann memory structure. It is Single Cycle, It has a 16-bit bus and a 12-bit address bus. Each instruction has a 16-bit word length. It can execute both logic commands as well as storage and relocation commands. Morris Mano's 16-bit microprocessor was designed on the Quartus application using Very High-Speed Integrated Circuit Hardware

Description Language (VHDL) software language. The designed microprocessor is simulated in ModelSim application and it was run on the DE0 Nano FPGA board produced by Intel (Altera). The microprocessor is in the CISC processor architecture, the Von-Neumann memory structure model. The designed system has been implemented and verified.

Key Words: CPU, Microprocessor, CISC, Microcomputer, FPGA, VHDL, SoC

TEŞEKKÜR

Tez çalışmamda desteklerini esirgemeyen danışmanım, Sayın Doç. Dr. Halil Murat ÜNVER'e, yüksek lisans ve diğer eğitim ortamlarında üzerimde emeği olan saygıdeğer hocalarıma, her zaman varlıklarıyla güç veren eşim Nuray DENİZ BAKACAK'a, canım kızım Esil BAKACAK'a, annem Zehra BAKACAK'a ve babam Tevfik BAKACAK'a sonsuz şükranlarımı sunarım.

İÇİNDEKİLER DİZİNİ

Sayfa

ÇİZELGELER DİZİNİ

ÇİZELGE Sayfa

ŞEKİLLER DİZİNİ

ŞEKİL Sayfa

KISALTMALAR DİZİNİ

AB	Adress Bus
AC	Accumulator
ALU	Aritmetic Logic Unit
AR	Address Register
ASIC	Application Specific Integrated Circuit
CISC	Complex Instruction Set Computer
CLBs	Logic Elements or Configurable Logic Blocks
CPLD	Complex Programmable Logic Device
CPU	Central Processing Unit
CU	Control Unit
DB	Data Bus
DR	Data Register
DSP	Digital Signal Processing
FPGA	Field Programmable Gate Array
FF	FlipFlop
HDL	Hardware Data Language
IR	Instruction Register
INPR	Input Register
LUT	Look Up Table
MIB	Merkezi İşlem Birimi (CPU)
MPLL	Multi-purpose PLL
MUX	Multiplexer
OUTR	Output Register
PC	Program Counter
PLD	Programmable Logic Device
PLL	Phase-Locked Loops
RAM	Random Acces Memory
RF	Register File
RISC	Reduced Instruction Set Computer
ROM	Read Only Memory
RTL	Register Transfer Level
SC	Sequence Counter

SoC	System on Chip
SPLD	Simple Programmable Logic Device
SRAM	Static Random Access Memory
VHDL	Very High-Speed Integrated Circuit Hardware Description Language
TR	Temporary Register

1. GİRİŞ

Elektronik devre tasarımlarının karmaşıklığı yeni tasarım metotlarının geliştirilmesini hızlandırmıştır. Elektronik devrelerin tasarlanmasında; kağıt üzerinde tasarlanma, baskı ve lehimleme işlemleri gibi geleneksel yöntemler yerini tasarımı dijital ortamda deneme, düzeltme, uygulama ve sentezleme yapılabilen esnek yöntemlere bırakmıştır [1].

Dijital tasarım ortamları için FPGA'ler geliştirilerek tasarımcılara düşük maliyet ve esneklik sağlanmıştır. FPGA'ler fazla sayıda programlanabilir mantık blokları ve bu bloklar arasında paralel bağlantılardan oluşan sayısal devrelerdir. Bir FPGA yapılandırıldığında, tasarıma özgü bir donanım uygulamasını oluşturacak şekilde bağlanır. FPGA'in yeniden programlanması gerekiyorsa, yapılandırma verileri tekrar tekrar silinebilir ve programlanabilir. Bu, FPGA'in değerli olmasının en önemli nedenidir [2].

FPGA'lerin kullanım alanı oldukça geniştir. Uzay, savunma, havacılık, otomotiv, kablolu/kablosuz iletişim, tv/radyo yayıncılığı, endüstriyel otomasyon ve kontrol sistemleri, bilgisayar, veri depolama, tıbbi elektronik, test/ölçüm gibi birçok endüstri alanında kullanılmaktadır [3].

Teknolojinin hızlı gelişmesi tasarımcıları da daha hızlı sistemler geliştirmeye yöneltmiştir. FPGA'ler, gerçek zamanlı devrelerin yeniden yapılandırılabilir olması gibi avantajları nedeniyle, FPGA tabanlı bir mikroişlemciyi esnek, programlanabilir ve güvenilir kılar. Ayrıca karmaşık elektronik tasarımların prototip hale getirilmesini kolaylaştırırlar [4].

FPGA'ler üzerinde HDL dillerinden (VHDL, Verilog) biri ile kodlama yapılarak kütüphaneler oluşturulabilir, mantık blokları ve ara bağlantılar oluşturulabilir, çeşitli çözümler için özgün tasarımlar modellenebilir. VHDL, öğrenilmesi zor ve karmaşık sistemlerin tasarımına uygun üst düzey bir dildir. Ayrıca bu dil, kullanıcıların karmaşık veri türleri oluşturmasına olanak sağlar. Yeni kütüphane oluşturma ve kütüphane yönetimine VHDL dilinde izin verilmektedir. VHDL dilinin kullanılmasının temel

nedeni; kütüphane ve paket, blok, fonksiyon, prosedür ve bileşen vb. gibi özel bir yapı oluşturulabilmesi, tasarımı daha basit ve kolay yönetilebilir hale getirmesidir. Ayrıca bu yapıların kullanılması taşınabilir ve tekrar kullanılabilir tasarımlar sağlar [2].

FPGA ile birçok bilgisayarın sahip olduğu basit ama eksiksiz bir işlemci, bellek hiyerarşisi ve minimum giriş/çıkış yolları gibi temel bileşenlerden oluşan bir sistem geliştirilebilir. Tasarımcı dijital ortamda gerçek bir test ortamını oluşturabilir ve araştırma ihtiyacına özgü bir işlemci mimarisi modelleyebilir [5].

Bu tez çalışmasının amacı, temel alınan Morris Mano'nun 16 bitlik mikroişlemci modelini, eksiksiz bir şekilde FPGA üzerinde VHDL yazılım dili kullanarak tasarlamak ve gerçeklemektir. Eğitim ortamında da kullanılabilen ve geliştirmeye açık hem yazılımsal hem de donanımsal bir içerik sunmaktır.

Tez çalışmasında; Morris Mano'nun Bilgisayar Sistemi Mimarisi (Üçüncü baskı) kitabının beşinci bölümünde anlatılan işlemci modeli temel alınmıştır. Buradan hareketle "Mano's Micro" isimli örnek bir çalışma da [6] incelenmiştir.

Bu bölümde, mikroişlemciler hakkında giriş bilgiler verilmiş, komut işleme şekline göre mimari yapıları ve bellek organizasyonuna göre mimari yapıları anlatılmış, sonrasında mikroişlemcileri oluşturan temel birimlerden bahsedilmiştir. Daha sonra programlanabilir mantık devreleri hakkında bilgiler verilmiş, bu devrelerden biri olan FPGA'in mimari yapısı ve çalışma şekli anlatılmıştır. FPGA kartları, FPGA kart üreticileri ve ürütecilerin sunduğu yazılımsal ürünler hakkında bilgiler verilmiştir. Devamında donanım tanımlama dillerinden VHDL'in tasarım yapısı, kod yapısı ve VHDL dilinin yazım kuralları genel hatlarıyla açıklanmıştır.

Tezin 2. bölümünde basit yapıda VHDL dili kullanılarak FPGA kartı üzerinde tasarlanan mikroişlemcilerin literatürde yeralan çalışmaları incelenmiştir.

Tezin 3. bölümünde, çalışmada tasarlanan 16-bitlik mikroişlemcinin tasarım yapısı, analiz sentez aşamaları, simülasyon aşaması ve FPGA kartına gömülmesi hakkında detaylı bilgiler materyal ve yöntem olarak verilmiştir.

Tezin 4. bölümünde, tasarlanan 16-bitlik mikroişlemcinin komutlarını test etmek için programlar yazılmış, her bir programın simülasyonları yapılmış ve FPGA kartı üzerinde çalıştırılması gösterilerek elde edilen bulgular ortaya konmuştur.

Tezin 5. bölümünde ise temel alınan mikroişlemci modeli ile referans alınan örnek çalışma tartışılmıştır.

Tezin 6. bölümünde ise bu tez çalışmasından elde edilen sonuç ve önerilere yer verilmiştir.

Tez çalışmasındaki mikroişlemcinin yazılım kodları ise ekler kısmında yeralmaktadır.

1.1. Mikroişlemciler

1.1.1. Mikroişlemcilere Giriş

Mikroişlemciler üzerlerine yüklenen programların yürütülmesini sağlayan ve tüm bileşenleri merkezi yapıda kontrol eden devredir. Mikroişlemciler genellikle komut işleme şekline ve bellek organizasyonlarına göre sınıflandırılmaktadır.

1.1.2. Komut İşleme Şekline Göre Mimari Yapılar

1.1.2.1. CISC

Karmaşık komut setli bilgisayarlar ilk olarak geliştirilmiştir. Değişken komut uzunluğuna sahiptir. Çok sayıda adresleme modu vardır. Az sayıda genel amaçlı kayıtçı desteği bulunur. RISC ile karşılaştırıldığında, gönderilen bir uygulama kodunu temsil etmek için daha az sayıda talimat gerektirir. Buna karşın, karmaşık derleyici gerektirir. CISC komut seti için derlenen uygulama kodu daha az sayıda komutla sonuçlandığından, uygulama ikili dosyalarını bir CISC makinesinde depolamak için daha az belleğe ihtiyaç duyar. Daha az sayıda talimat gerektirmesi, mutlaka bir CISC işlemcisinde çalışan bir uygulamanın, bir RISC işlemcisinde çalışan aynı

uygulamadan daha yüksek performansla sonuçlanması anlamına gelmez. Yükle/Depola ya ek olarak belleğe erişim için başka talimatlar da vardır. Tipik bir CISC komutu (Intel x 86 komutu) ADD AL, BL (AL+BL->AL) şeklindedir. CISC işlemci örnekleri, Intel'in 486'sı, Pentium (tüm sürümler), AMD'nin Krypton'ı ve Athlon [7].

1.1.2.2. RISC

İndirgenmiş komut setli bilgisayarlardır. Sabit komut uzunluğuna, birkaç adresleme moduna sahiptir. Çok sayıda genel amaçlı kayıt desteği vardır. Gönderilen bir uygulama kodunu temsil etmek için CISC'e kıyasla daha fazla sayıda talimat gerektirir. Daha az karmaşık derleyici gerektirir. RISC komut seti için derlenen uygulama kodu daha fazla sayıda komutla sonuçlandığından (CISC ile karşılaştırıldığında), bir RISC makinesinde uygulama ikili dosyalarını depolamak için daha fazla belleğe ihtiyaç duyarlar. Daha fazla sayıda talimat içermesi, bir RISC işlemci üzerinde çalışan bir uygulamanın, bir CISC işlemci üzerinde çalışan aynı uygulamadan daha düşük performansla sonuçlanacağı anlamına gelmez. Yükleme/Depolama belleğe erişebilenlerdir. Tipik bir RISC komutu ADD rs1, rs2, rd (rs1+rs2 -> rd) şeklindedir. RISC işlemci örnekleri; SUN's UltraSparc, MIPS's MIPS32, MIPS64, ARM'S ARM11, Motorola's PowerPC vb [7].

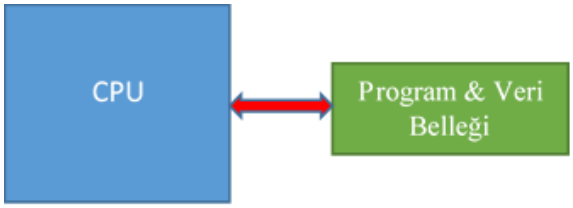
1.1.3. Bellek Organizasyonuna Göre Mimari Yapılar

Her iki mimariyi birbirinden ayıran en önemli özellik veri ve kodların tutulduğu yerlerdir. Bu durum aynı zamanda mikroişlemcinin çalışma hızını etkiler.

1.1.3.1. Von Neumann Mimarisi

Şekil 1.1.'de de görüldüğü üzere Von Neumann mimari yapısına [8] göre program komutları ve veri aynı bellekte tutulur. Önce komut getirilir, daha sonra veri alınıp

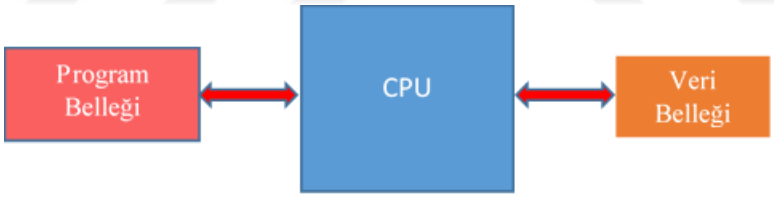
işlenir. Bu mimaride gecikmeler yaşanmaktadır. Fakat hızlı ön bellekler (cache memory) ve bant genişliği artırılarak gecikmeler azaltılabilir.



Şekil 1.1. Von Neumann mimarisi

1.1.3.2. Harvard Mimarisi

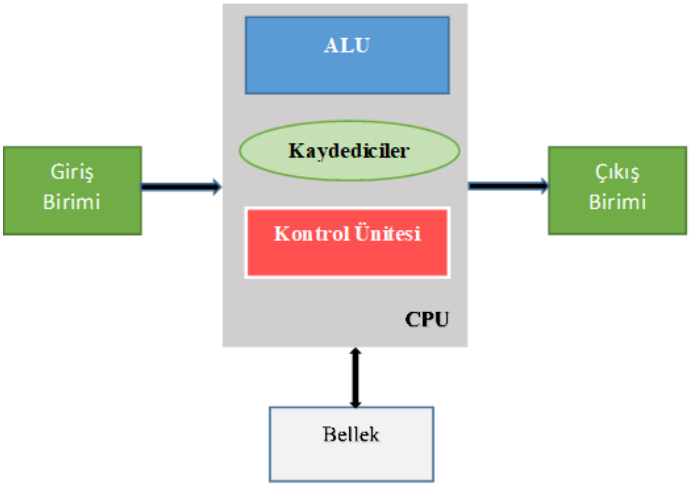
Şekil 1.2.'de görüldüğü üzere Harvard mimarisinde [8] program komutları ve veriler farklı belleklerde tutulur. Adres ve veri yolları farklıdır. Aynı anda komut ve operand erişimi yapılabilir. Çalışması Von Neumann'a göre daha hızlıdır. PIC ve Sayısal Sinyal İşlemcileri (DSP) kullanılır.



Şekil 1.2. Harvard mimarisi

1.1.4. Mikroişlemcinin Temel Birimleri

Temel bir mikroişlemci yapısı ve çevre birimleri Şekil 1.3.'deki gibidir.



Şekil 1.3. Mikroişlemcinin temel birimleri

1.1.4.1. Kontrol Birimi

Kontrol birimi, talimatları okur ve diğer bileşenleri çalıştırmak için gerekli dijital sinyalleri üretir. Örneğin, iki sayıyı birbirine ekleme talimatı, kontrol biriminin toplama modülünü etkinleştirmesine neden olur.

1.1.4.2. Aritmetik ve Mantık Birimi (ALU)

ALU, aritmetik işlemleri gerçekleştiren mikroişlemcinin bir parçasıdır. ALU'lar tipik olarak toplama, çıkarma, bölme, çarpma ve iki sayının (and, or, nor, not, vb.) mantıksal işlemleri gerçekleştirebilir.

1.1.4.3. Kaydediciler (Registers)

En genel anlamı donanım kaydedicisidir. Bilgi bitlerini depolamak için kullanılır. Kaydedicilerin tüm bitlerine okuma ve yazma işlemleri yürütülür. Amaçlarına göre farklı isimlerde tanımlanırlar.

1.1.4.4. Giriş/Çıkış Birimi (I/O)

İşlemcinin, bilgisayar sisteminin geri kalanıyla iletişim kurabilmesi gerekir. Bu iletişim, I/O bağlantı noktaları aracılığıyla gerçekleşir. I/O bağlantı noktaları, RAM ve ayrıca bir bilgisayarın diğer çevre birimleri ile arabirim oluşturur.

1.1.4.5. Ön Bellek (Cache Memory)

Üretilen çoğu CPU'da önbellek yoktur. Önbellek, çip üzerinde bulunan ancak kayıt olarak kabul edilmeyen bellektir. Harici belleği okumak çok yavaştır (işlemcinin hızına kıyasla) ve yerel bir önbelleği okumak çok daha hızlı olduğu için önbellek kullanılır. Modern işlemcilerde önbellek çipin toplam alanının %50'sini veya daha fazlasını kaplayabilir [9].

1.2. Programlanabilir Mantık Devreleri (PLD)

Programlanabilir mantık devreleri (Programmable Logic Devices-PLDs) gelişim sırasına göre, basit programlanabilir mantık devresi (Simple Programmable Logic Device-SPLD), karmaşık programlanabilir mantık devresi (Complex Programmable Logic Device–CPLD) ve FPGA şeklindedir.

SPLD, PLDs arasında en basit karmaşık yapıya sahip cihazlardır. 4 farklı yapıda tanımlanabilir. PLA (Programmable Logic Array) esnek ama yavaş, PAL (Programmable Array Logic) PLA'dan daha az esnek daha hızlı ve daha ucuz, GLA (Generic Array Logic) daha kapasiteli fakat karmaşık, PROM (Programmable Read-Only Memory) ise programlanabilir fakat yavaş cihazlardır.

CPLD, PLDs arasında en karmaşık yapılı cihazlardır. SPLD'lere göre kapasitesi fazla ve çok hızlıdır. Çok fazla sayıda kapı içerebilir. Tekrar programlanabilir fakat içerisindeki dizi yapıları sabit olduğu için kullanımı esnek değildir.

FPGA, PLDs hibrit modelidir ve çok büyük kapasiteye sahiptir. Üzerinde fazla sayıda mantık kapısı ve bağlantı pini bulundurur [10]. Çizelge 1.1.'de PLD'ler karşılaştırmalı olarak verilmiştir [11].

Çizelge 1.1. PLD'lerin karşılaştırılması

	SPLD	CPLD	FPGA
Tekrar Programlanabilirliği	Silindikten sonra tekrar programlanabilir	Tekrar programlanabilir	SRAM tabanlı olanlar tekrar programlanabilir
Bağımsız Programlanabilme	Devre dışında	Devre içinde	Devre içinde
Ebat	küçük	orta	Orta/büyük
İçerdiği eşdeğer kapı sayısı	-	900-20000	10000+
Pin Sayısı (adet)	16-28	44-300	50+
Mantık Bloğu (adet)	8-24	32-500	5000+
Flip-Flop Sayısı (adet)	8-24	32-500	5000+
Yapı Türü	EPROM, EEPROM	EPROM, EEPROM, Flash	Flash, EEPROM
Program, Enerji Kesilirse	Silinmez, kalıcıdır	Silinmez, kalıcıdır	SRAM tabanlı olanlarda silinir, kalıcı değildir.

1.3. FPGA

Sahada programlanabilir kapı dizileri olarak ifade edilir. Sahada programlanabilir olması FPGA'in yeniden programlanabilir olmasını, kapı dizileri olması ise maskeli kapı dizileri özelliğinde olan uygulamaya özel tümleşik devrelere (Application Specific Integrated Circuit –ASIC) benzerliğinden ötürüdür [12].

ASIC'ler belli bir uygulamaya özel olarak üretilmişlerdir. Uzun sürede, maliyetli, zor tasarım ve test aşamasına ihtiyacı vardır. Eğer tasarımda hata varsa düzeltmek için tekrar uzun bir süreye ve maliyete ihtiyaç vardır. Hızlı çalışırlar. Tasarım ve testi başarı sağlandığında fazla sayıda çip üretimi için maliyet oldukça düşüktür. Bir kez başta tasarım yapılır ve bu tasarıma göre çip üretilir. Tekrar programlanamazlar [13]. Bu durum, tasarım esnekliği ve tekrar programlanabilme özelliğine sahip FPGA'lerin geliştirilmesini sağlamıştır.

FPGA iki farklı türde üretilirler. Sabit rastgele erişimli bellek (Static Random Access Memory-SRAM) yapısında olan ve bir kez programlanabilen türleri vardır. Tür farklılığı nedeniyle firmalar, FPGA üretiminde yeni yapılar ortaya çıkarmıştır. Bunlar SRAM tabanlı, EEPROM tabanlı, Flash tabanlı ve Anti-Sigorta tabanlı olarak FPGA üretilmişlerdir. Çizelge 1.2.'de FPGA üreticilerinin kullandığı yapılar ve Çizelge 1.3. de yapılandırma teknolojileri yer almaktadır [11].

Çizelge 1.2. FPGA üreticilerinin kullandıkları yapılar

FPGA Üreticisi	**Kullanılan Yapı**
Xilinx	SRAM
Altera	SRAM, Flash
QuickLogic	Anti-Sigorta
Actel	Anti-Sigorta
Croospoint	Anti-Sigorta
Lattice	SRAM, Flash

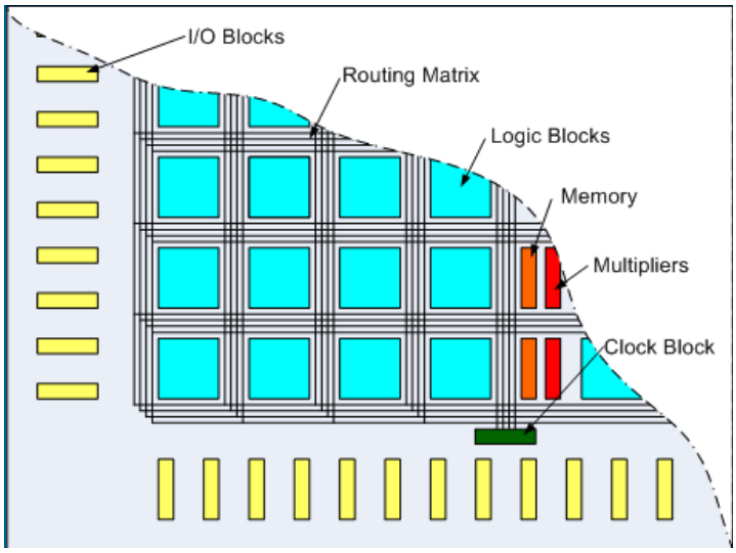
Çizelge 1.3. FPGA yapılandırma teknolojileri

Teknoloji Tabanı	Özellikleri
SRAM	Kalıcı hafızası yoktur. Cihazı başlatmadan önce tekrar programlamak gerekir. Hızlı bir şekilde yeniden programlanabilir. Cihaz devre içinde yeniden yapılandırılabilir.
Anti-Fuse	Konfigürasyon, istenen işlevselliği uygulamak için dahili sigortaların "yakılması" ile ayarlanır. Bir kez programlanabilirler.
EPROM	Yapılandırması EPROM'lara benzer. Yapılandırma kalıcıdır. Cihaz devre dışı yapılandırılmalıdır.
EEPROM	Yapılandırması EEPROM'lara benzer. Yapılandırma kalıcıdır. Cihaz devre dışı yapılandırılmalıdır. Tekrar devre dışında yapılandırılabilir.
Flash	EEPROM'u benzer. Daha hızlı silinebilir ve tekrar programlanabilir.

İlk FPGA, XC2064 isimli, Ross Freeman tarafından Xilinx firması adına 1985 yılında üretilmiştir. FPGA'lerin konfigürasyonu PROM'dan yüklenir. Kapılar kullanıcı tarafından ilişkilendirilebilen ayarlanabilir mantık devreleri (Logic Blocks/Elements or Configurable Logic Blocks-CLBs) içerir [14].

1.3.1. FPGA İç Yapıları

Yaygın olarak SRAM tabanlı FPGA kartları kullanıldığı için yapı olarak bu türde olanlar incelenecektir. Genel itibariyle çok büyük farklılıklar yoktur. Şekil 1.4.'de FPGA'in genel yapısı [15] yer almaktadır.

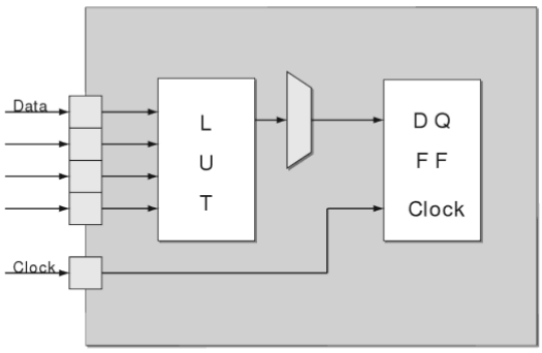


Şekil 1.4. Temel FPGA yapısı

1.3.1.1. Mantık Blokları (Logic Blocks/Configurable Logic Blocks)

FPGA mantık blokları üreticilerde "mantık blok", "makro hücresi", "mantık birimi" gibi isimler verilse de, genelde aynı amaca hizmet eder. Mimariler farklı olabilir. Ayarlanabildiği için CLB de denir.

FPGA'ler mimari açıdan Look Up tablosu(Look Up Table-LUT) ve çoklayıcı (Multiplexer-MUX) yapıda mantık bloğu içerir. Yaygın olarak LUT tipinin kullanıldığı görülür. Şekil 1.5.'de mantık bloğunun içyapısı görülmektedir [11]. Mantık bloğu, n bit data girişine sahip Look Up tablosu, LUT ve D tipi Flip Flop içerir. LUT bellek görevi üstlenir. FPGA'in kapasitesini belirler.

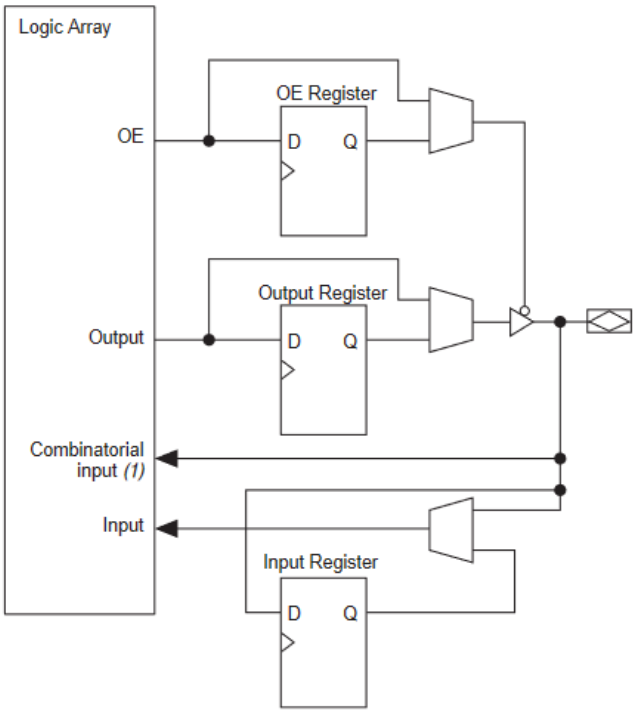


Şekil 1.5. Mantık bloğu

1.3.1.2. I/O Blokları (I/O Blocks)

Mantık bloklarının diğer birimlerle iletişimini sağlayan giriş/çıkış yapılardır. Şekil 1.4.'de görüldüğü gibi FPGA'in en dış kenarlarında yer alırlar.

Tek I/O blok yapısında, kaydediciler, MUX devreleri, kontrol ve saat sinyalleri yer alır. Şekil 1.6.'da Intel FPGA Cyclone serisi I/O blok içyapısı görülür [16].



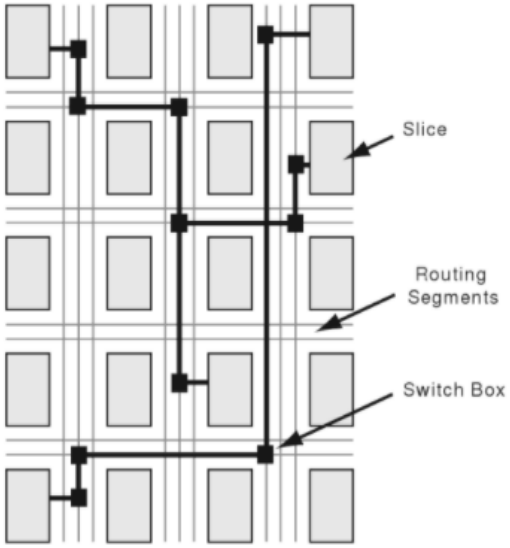
Şekil 1.6. Intel (Altera) FPGA Cyclone I/O yapısı

Bir I/O bloğu sadece giriş veya sadece çıkış bloğu olarak programlandırılabilir. FPGA tasarlandıktan sonra en son olarak I/O blokları ayarlanır.

1.3.1.3. Matris Anahtarlamalı Yönlendirme Yolları

Dikey ve yatay düzlemde mantık blokları arasındaki bağlantıyı kuran yapılardır. Tüm CLB'lerin etrafında olacak şekilde FPGA içerisinde bulunur. FPGA üreticisine göre sayıları değişebilir. Bağlantı noktalarında yer alan programlanabilir anahtarlar

sayesinde, FPGA'e yüklenen programa göre bağlantı yönlendirmeleri yapılır. Şekil 1.7.'de sinyal yönlendirme yolları görülmektedir [11].



Şekil 1.7. FPGA sinyal yönlendirme yolları

1.3.1.4. FPGA Üzerinde Bulunan Diğer Blokları

Üreticiler çeşitli ihtiyaçlara göre FPGA kartlarına ilave donanım eklemeleri yapmıştır. Üreticiden üreticiye hatta modelden modele farklılıklar söz konusudur. En yaygın kullanılanlar aşağıdaki gibidir.

- **Saat Kaynağı (Clock Resources)**

FPGA yerel ve sistem saatlerini işlemek, yönetmek ve ayarlamak için birincil FPGA öğesi saat bloğudur. Saat ayarı iki farklı teknolojiye dayalı olarak gerçekleştirilebilir; PLL ve DLL. PLL'ler, voltaj kontrollü bir osilatöre ayar yaparak istenen saat fazını veya frekans çıkışını üretir. DLL'ler, istenen saat fazını veya frekansını üretmek için FPGA'nın içindeki kalibre edilmiş kademeli gecikme hattı devresinden gelen sinyallere erişir. DLL'ler dijital devrelerdir [11].

- **Bellek Blokları**

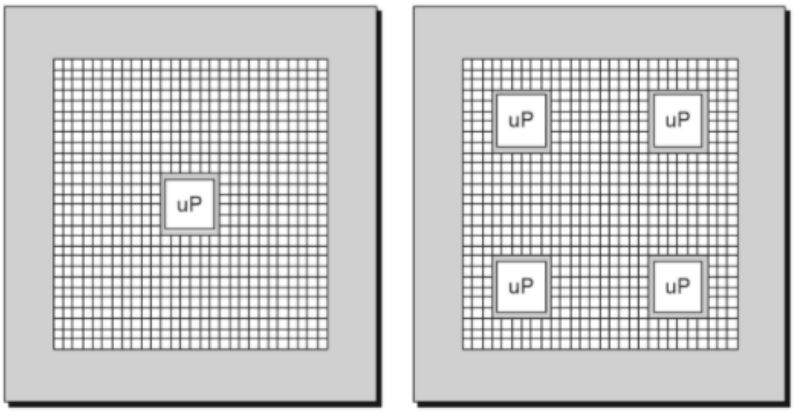
Bellek kaynakları, birçok gelişmiş FPGA uygulaması için kritik öneme sahiptir. FPGA'larda dağıtılmış ve blok bellek olmak üzere iki ana bellek türü vardır. Dağıtılmış bellek, LUT öğelerinin SRAM bellek bloklarının uygulamaları olduğu gerçeğinden yararlanır. Blok bellek, FPGA içinde ayrılmış SRAM bellek bloklarının uygulanmasıdır [11].

- **Sayısal Sinyal İşleme (Digital Signal Processing-DSP) Blokları**

Sayısal sinyal işleme fonksiyonu içeren karmaşık devreler tasarlamak için bazı üreticiler, FPGA kartında blok belleklerle birlikte DSP blokları da eklemişlerdir. Örnek olarak Intel FPGA Cyclone IV EP4CGX150 serisi kartlarında 360 adet DSP blok bulundurur [17].

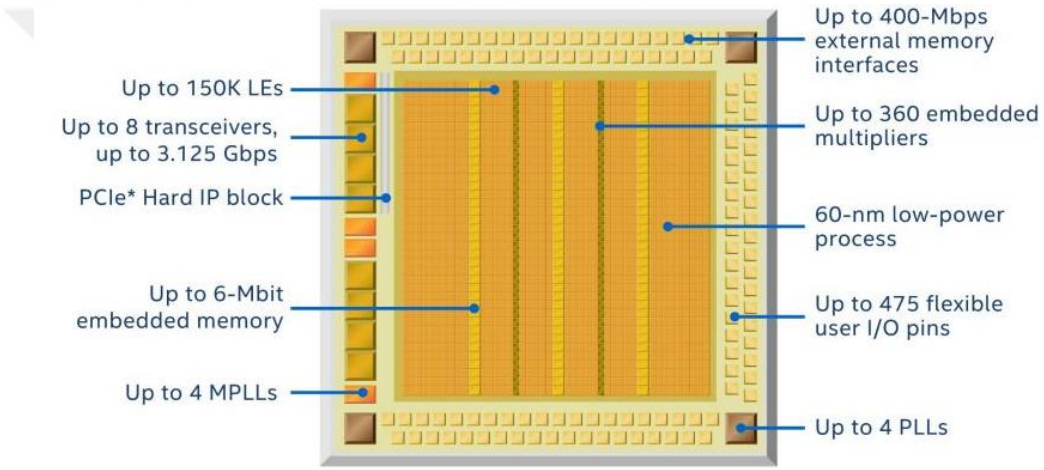
- **Dahili İşlemci Blokları**

Tasarımların çoğunda işlemciye ihtiyaç vardır. FPGA güç ve kaynak tüketiminden tasarruf için üreticiler ekleyebilir. Bu tasarımcı için avantajlı bir durumdur. Şekil 1.8.'de sol tarafta tek işlemcili FPGA, sağ taraftaki resimde ise dört çekirdek işlemcili FPGA görülmektedir [18].



Şekil 1.8. Tek işlemcili ve iki işlemcili FPGA

Altera (Intel) firmasının ürettiği Cyclone IV GX serisi bir FPGA kartının blok diyagramı Şekil 1.9.'da görülmektedir. Bu kart üzerinde, mantık kapıları, I/O elemanları ve faz kilitli döngüler (Phase-Locked Loops-PLLs) ile çevrilidir. Hem GX hem de E serisi FPGA kartlarda, şemanın her bir köşesinde bulunan dört genel amaçlı PLL'ye sahiptir. Cyclone IV GX FPGA'da şemanın üst, alt ve sağ taraflarında I/O elemanları bulunurken Cyclone IV E FPGA'da şemanın dört tarafında I/O'lar bulunur. Cyclone IV GX FPGA şemanın sol tarafında, dört alıcı-vericiden oluşan iki dörtlü halinde sekiz adede kadar alıcı-verici bulunur. Her bir alıcı-verici dörtlüsünün üst ve alt kısmında, alıcı-vericinin veya FPGA yapısının kullanabileceği çok amaçlı bir PLL (Multi-purpose PLL-MPLL) bulunur [16].



Şekil 1.9. Altera (Intel) firmasının ürettiği Cyclone IV GX serisi FPGA blok diyagramı

1.3.2. FPGA Kartları

Xilinx, Intel(Altera), Actel, Lattice ve diğer FPGA üreticileri arasında Xilinx ve Intel(Altera) Pazar payının büyük bir bölümünü elinde bulundurur. Bunun en büyük sebebi ürettikleri FPGA geliştirme kartlarına yine kendilerinin sunduğu uygulamalarla tasarım sentez desteğidir. Diğer kart üreticilerinin tasarım için kendi uygulamaları yoktur. Tasarım programlama için üçüncü parti yazılımlara ihtiyaçları vardır. Bu sebeple Xilinx ve Intel(Altera) firmalarının geliştirilmiş FPGA kartları incelenecektir. Ürünlerin teknolojik olarak üzerinde barındırdıklarını görebilme açısından gelişmiş kartları tanıtılmıştır. Ayrıca ürünlerin seçiminde kıyaslama söz konusu değildir.

Tavsiye niteliği taşımamaktadır. Başlangıç seviyesi için açıklanan kartların daha alt modelleri tercih edilmesi hem maliyet hem de öğrenme açısında daha uygun olacaktır.

1.3.2.1. Xilinx

- **Xilinx FPGA Kartı**

Spartan-6 XC6SLX45 serisi ürünün genel yapısı içerisinde, 43661 adet mantık bloğu, 2,088Kb bellek, 58 adet DSP bloğuna sahiptir. Bunlara ilave çok sayıda I/O bağlantıları, bellekler vs. üzerinde barındırmaktadır. Şekil 1.10.'da ürün resmi yer almaktadır [19].

Şekil 1.10. Xilinx Spartan 6 XC6SLX45 geliştirme kartı

- **Xilinx Tasarım Programları**

Bünyesinde tasarımcıların ihtiyacına yönelik FPGA, DSP, ISim gibi çeşitli uygulamaları vardır [20].

– **Xilinx ISE Design Suite:** Tasarım ve sentezleme programıdır. Ücretsiz versiyonu da vardır.

– **Xilinx Platform Studio (XPS) :** Tasarımcılar kendi konfigürasyonuna göre yazılımsal işlemci sistemi oluşturabilmeleri ve bunları kullanabilmeleri için geliştirilmiştir. XPS uygulamasında oluşturulan bir işlemci ISE uygulamasında kullanılabilmektedir.

– **MicroBlaze:** Xilinx'in orta sınıf cihazlar için ürettiği 32 bitlik yazılımsal(soft) işlemcisidir. Kendi FPGA kartlarında kullanılabilir.

– **ISE Simulator (ISim):** Xilinx'in ISE ortamında tasarlanan projelerin simülasyonunun yapılmasını sağlayan bir uygulamasıdır.

– **ChipScope:** Tasarımlardaki veri yollarının izlenmesi, analiz edilmesini sağlayan çeşitli ek fonksiyonları olan uygulamasıdır.

1.3.2.2. Intel (Altera)

➢ **Intel FPGA Kartı**

Cyclone 10 GX serisi bu ürünün genel yapısı içerisinde, 220000 adet mantık bloğu, 11,740Kb bellek ve 192 adet DSP bloğuna sahiptir. Bunlara ilave çok sayıda I/O bağlantıları, bellekler vs. üzerinde barındırmaktadır [21]. Şekil 1.11.'de ürün [22] resmi yer almaktadır.

Şekil 1.11. Intel FPGA Cyclone 10 GX geliştirme kartı

➢ **Intel (Altera) Uygulamaları**

Firma kendi FPGA kartlarının tasarımı ve sentezi için sunduğu yazılımlar aşağıdaki gibidir [23].

– **Quartus:** Tasarım ve sentezleme programıdır. Ücretsiz versiyonu da vardır.

– **Nios:** C veya C++ dili kullanılarak mikroişlemci programlanabilmektedir.

– **ModelSim:** FPGA tasarımlarının simülasyonunun yapılmasını sağlayan bir uygulamadır. Ücretsiz versiyonu da vardır.

– **DSP Builder:** Sayısal sinyal işleme tasarımlarında, tasarım süreçlerini kısaltmak için kullanılan hazır DSP paket uygulamasıdır.

1.3.3. FPGA Tasarım Yöntemleri

Genelde şematik tasarım programlama veya HDL programlama ile tasarlanır. Şematik tasarımda, kütüphaneden seçilen devre elemanları tasarımcı tarafından tek tek birleştirilir ve ayarları yapılır. Şematik tasarım küçük projeler için kolay görünebilir fakat üzerinde yapılacak değişiklik için çok fazla zaman harcanır. Aynı zamanda kullanılacak hazır şema kütüphanesi üretici ile sınırlıdır. HDL ile yapılan tasarımlarda ise sadece kodu değiştirmek yeterlidir, aynı zamanda tasarımcı üreticiden bağımsız olur. Yani ihtiyacına göre özgün tasarımlar geliştirebilir. Bu yüzden karmaşık devre tasarımlarında HDL dilleri daha çok kullanılır [18].

HDL dilleri, 1990'lı yıllardan itibaren kullanımı hızla popülerleşmeye başlamıştır. HDL geliştirme çalışmaları, 1977'lerde ISP ve KARL dilleri ile başlamıştır. Sonrasında ilk modern HDL dili 1984 yılında Verilog adıyla geliştirilmiş ardından, VHDL ise 1980'lerde çalışmalara başlanmış, 1987 yılında standart hale getirilmiştir. VHDL ile Verilog HDL dilleri birbirlerinden farklı yapıdadır. VHDL, ADA programlama dillerini temel alan bir yapıdır. C++ diline benzer. Verilog HDL'yi öğrenmesi ve yazması daha kolaydır. C programlama diline yakın bir yapıdadır. Fakat VHDL'e göre karmaşık devre tasarım ve sentezleme sırasında çok fazla kapı kullanılır. Aynı zamanda VHDL gibi esnek değildir. VHDL, ilk olarak Amerikan Savunma Bakanlığı ihtiyacına yönelik olarak geliştirilmiştir [24]. Günümüzde SoC tasarımlarında yaygın olarak VHDL dili kullanılmaktadır.

1.4. VHDL

VHDL, çok yüksek hızlı tümleşik devrelerin donanım tasarımında kullanılan bir programlama dilidir. Elektrik & Elektronik Mühendisleri Enstitüsü (IEEE) tarafından 1987 yılında "Std 1076-1987" olarak standartlaştırılmış olan bu dil, son olarak IEEE standartları tarafından "Std 1076-2019" sürümü ile tekrar düzenlenmiştir [25]. Bu

bölümde anlatılacaklar VHDL dilinin temel yapısı hakkında fikir vermek açısından hazırlanmıştır.

1.4.1. VHDL Tasarım Yapısı

VHDL'de sayısal devre tasarımı dört farklı yapıdan oluşur. Bunlar varlık (entity), mimari (architecture), paket (package) ve konfigürasyon(configuration).

– **Entity:** Tasarımın kendisi ile dış birimler arasındaki yapıyı temsil eder. Burada I/O portları tanımlanır.

– **Architecture:** Tanımlanan modülün fonksiyonunun belirlendiği yapıdır.

– **Configuration:** Tasarımın alt bileşenlerinin bir araya gelerek oluşturacağı tasarımın belirlendiği yapıdır.

– **Packed:** Oluşturulan yapıların başka tasarımlarda da kullanılması amacıyla bildirildiği ve gruplaştırıldığı yapıdır [26].

VHDL dilinde "veya" kapısını oluşturan örnek bir yapı Çizelge 1.4.'de gösterilmiştir.

Çizelge 1.4. VHDL dilinde veya kapısının mimari yapısı

Packed	library ieee; use ieee.std_logic_1164.all;
Entity	Entity veya_kapi is port(X: in bit, Y: in bit, Z: out bit); End veya_kapi;
Architecture	architecture mimari of veya_kapi is begin Z <= X or Y; End mimari;
Configuration	Configuration Veya_Konfig of veya_kapi is for davranis end for; end configuration Veya_Konfig

1.4.2. VHDL Kod Yapısı

1.4.2.1. Veri Nesneleri (Data Objects)

VHDL veri nesneleri aşağıdaki gibi tanımlanmaktadır [27].

– **Sinyaller(Signals) :** Sinyal veri nesneleri aşağıdaki gibi tanımlanır.

SIGNAL sinyal_ismi :tip_ismi[:=değer] ;

Örnek: SIGNAL address : STD_LOGIC_VECTOR(7 downto 0) ;

– **Sabitler (Constants):** Sabitlere, değer başlangıçta atanır ve sonradan değiştirilemezler.

CONSTANT sabit_ismi:tip_ismi[:=değer] ;

Örnek: CONSTANT Elde :BOOLEAN:=TRUE

– **Değişkenler(Variables):** Değişkenler geçici bilgi saklar.

VARIABLE değişken_ismi:tip_ismi[:=değer] ;

Örnek: VARIABLE index : INTEGER range 0 to 9 := 5 ;

1.4.2.2. Operatörler

Operatörler yedi ayrı grupta sınıflandırılabilir. Çizelge 1.5.'de gösterilmiştir [27].

Çizelge 1.5. VHDL operatörleri

Operatör sınıfı	Operatör adları
İkili mantık	and or nand xor xnor
İlişkisel	= /= < <= > >=
Yer değiştirme	sll srl sla sra rol ror
Ekleme	+ - and &(birleştirme)
Tekli işaret	+ -
Çarpma	* / mod rem
Diğer	not abs **

1.4.3. VHDL Yazım Kuralları

- Harfe duyarlı değildir. Örneğin: Sin <= X or Y; ya da sIn <= x oR y;
- Satır sonuna noktalı virgül(;) işareti konulur.
- Sayfada Kodlar arası boşluklar duyarlı değildir.

Örneğin: sP <= Rn_x and Rn_y; ya da sP <=rn_x AND rn_y;

- Açıklama yapmak için satır başına "--"(iki adet tire işareti) vermek yeterlidir.

  -- Bu program Mehmet BAKACAK tarafından yazılmıştır.

  AS_kay <= AS_kay; -- Komut aktarım yapmaktadır [28].

2. LİTERATÜR ÇALIŞMASI

Literatürde FPGA ile tasarlanan çeşitli mikroişlemci yapıları mevcuttur. “Simulation and FPGA Implementation of a Simple Computer” isimli çalışmada, 9 adet farklı bitlerden yazmaçlar, 1024x16-bitlik bellek ve ALU başta olmak üzere çeşitli çevre birimlerinden oluşan 16-bitlik bir mikroişlemci Xilinx Spartan 3 FPGA kartı üzerinde oluşturulmuştur. Ortalama bağlantı gecikmesi 6.357 ns, maksimum pin gecikmesi 12.054 ns, maksimum çalışma frekansı 73.465 MHz ulaşılmıştır [29].

“FPGA Implementation of an 8-bit Simple Processor” isimli çalışmada, VHDL dili kullanılarak tasarlanan FPGA tabanlı 8 bitlik mikroişlemci de, temel olarak 4 adet 4 bitlik yazmaç, 8 bitlik kelimelerden oluşan 16 kelimelik bir bellek, bir kontrol birimi ve bir aritmetik mantık biriminden (ALU) oluşmaktadır. Xilinx Spartan 3 xc3s200FT256 FPGA kartı üzerinde uygulanmış, minimum 10.486 ns periyotla maksimum 95.364 MHz frekansa ulaşılmıştır [4].

“MIB-16 FPGA Based Design and Implementation of a 16-Bit Microprocessor for Educational Use” isimli çalışmada, VHDL dili kullanılarak FPGA tabanlı 16 bitlik bir mikroişlemci tasarlanmıştır. Bu işlemci de 16 adet genel amaçlı yazmaç, program sayacı (Program Counter-PC), 3 bitlik durum yazmacı ve diğer alt birimler ile 16 bit kelime kapasiteli harici bir bellekten oluşmaktadır. Xilinx Spartan-3 FPGA kartı üzerinde tasarlanan mikroişlemcinin performansı aritmetik ve mantık işlemleri için 3 MHz, yükleme ve depolama işlemleri için 1.5 MHz, hızlı yükleme ve depolama işlemleri için 2.3 MHz olarak belirtilmiştir [30].

“FPGA Kullanarak 16 Bitlik Mikroişlemci Tasarımı” isimli tez çalışmasında tasarlanan mikroişlemci, 16 bitlik veri yolu, akümülatör, 8 adet genel amaçlı yazmaç olmak üzere farklı yapılarda diğer yazmaçlar, 8 KB bellekten oluşmaktadır. Bazı yapıları hazır modüller kullanılmış ihtiyaca göre yeniden uyarlanarak kullanılmıştır. Xilinx Spartan 3E FPGA kartı üzerinde VHDL dili ile oluşturulan işlemci, 65.95 MHz maksimum çalışma frekansına sahiptir. [31].

“Hardware Design and Simulation of a Microprocessor” çalışmasında, 8 bitlik bir CPU tasarlanmıştır. Tasarım ve simulasyon için Logisim isimli uygulama kullanılmıştır. Donanım tasarımında rehber olması hedeflenmiştir [32].

“VHDL ile Mikroişlemci Tasarımı ve Eğitimde Uygulanabilirliği” isimli çalışmada, VHDL dili kullanılarak 8 adet işlem kapasitesine sahip 16 bitlik ALU tasarımı anlatılmıştır [33].

“8 Bit Instructional Processor using FPGA” isimli çalışmada, 8085 mikroişlemci referans alınarak 8 bitlik bir işlemci tasarlanmıştır. Aritmetik ve mantıksal komutlar, veri aktarımı ve dallanma komutları test edilmiştir. Xilinx ISE uygulaması ile tasarım ve sentez işlemlerini yapmıştır. Xilinx Spartan 6 FPGA kartı üzerinde de sonuçlarını başarılı bir şekilde gözlemlemiştir. Bir sonraki aşama olarak 16 ve 32 bit şeklinde genişletme hedeflemiştir [34].

“A Case Study: Design of 16 bit Arithmetic and Logical unit using Xillinx 14.7 and Implementation on FPGA Board” isimli çalışmada, çalışmacı Basys 3 Artix 7 FPGA kartı üzerinde VHDL dilini kullanarak 16 bitlik ALU tasarlamış ve komut kümesini ve hızını artırmayı hedeflemiştir [35].

“An Approach for Teaching Processor Design” isimli çalışmada, Ruse Üniversitesi öğrencilerinin basit bir işlemci için ALU tasarımını VHDL dilinde tasarlaması, üzerinde değişiklikler yaparak kendilerini geliştirmeleri ve FPGA kartı üzerinde tasarımlarını çalıştırabilmeleri hedeflenmiştir [36].

“To Design 16 bit Synchronous Microprocessor using VHDL on FPGA” isimli çalışmada, 16 bitlik senkron bir işlemciyi VHDL dilinde tasarlanması anlatılmıştır. ALU, ROM, RAM, Kontrol ünitesinin tasarımlarından ve simülasyonlarından kesitler sunulmuştur. Çalışmada hedefledikleri FPGA kartına yükleme işlemini donanım sorunları nedeniyle yapamamışlardır [37].

3. MATERYAL ve YÖNTEM

Morris Mano'nun Bilgisayar Sistemi Mimarisi (üçüncü baskı) kitabının beşinci bölümünde anlatıldığı işlemci modeli temel alınmıştır. Mikroişlemci, CISC işlemci mimarisi, Von-Neumann bellek yapısı modelindedir. Mikroişlemcide bütün komutlar tek saat çevriminde (single cycle) ve saat darbesinin yükselen kenarında işlem görür. 16 bit veri yolu ve 12 bit adres yoluna sahiptir. Sekiz adet kaydedici bulunur. Her bir komut 16 bit kelime uzunluğuna sahiptir. Mikroişlemci, tanımlanan lojik komutlar ile saklama ve yer değiştirme komutlarını yürütebilmektedir. Mikroişlemcinin yürüteceği işlemler için 16 bit veri yolu, 12 bit adres yoluna sahip 2^{12} =4096 word x 16 bit kapasiteli RAM eklenmiştir. Tasarım için Intel (Altera) FPGA kartları için geliştirme platformu olan Quartus II 10.1 sp1 Standard Edition kullanılmıştır. Çalışmaların test edilmesi ve simülasyon çalışmaları için ise Modelsim Altera 6.6d (Quartus II 10.1 sp1) Starter Edition kullanılmıştır. Tasarlanan sistemin gerçeklemesi ve doğrulaması yapılmıştır.

3.1. 16 Bitlik Mikroişlemcinin Tasarım Yapısı

3.1.1. Kaydediciler (Registers)

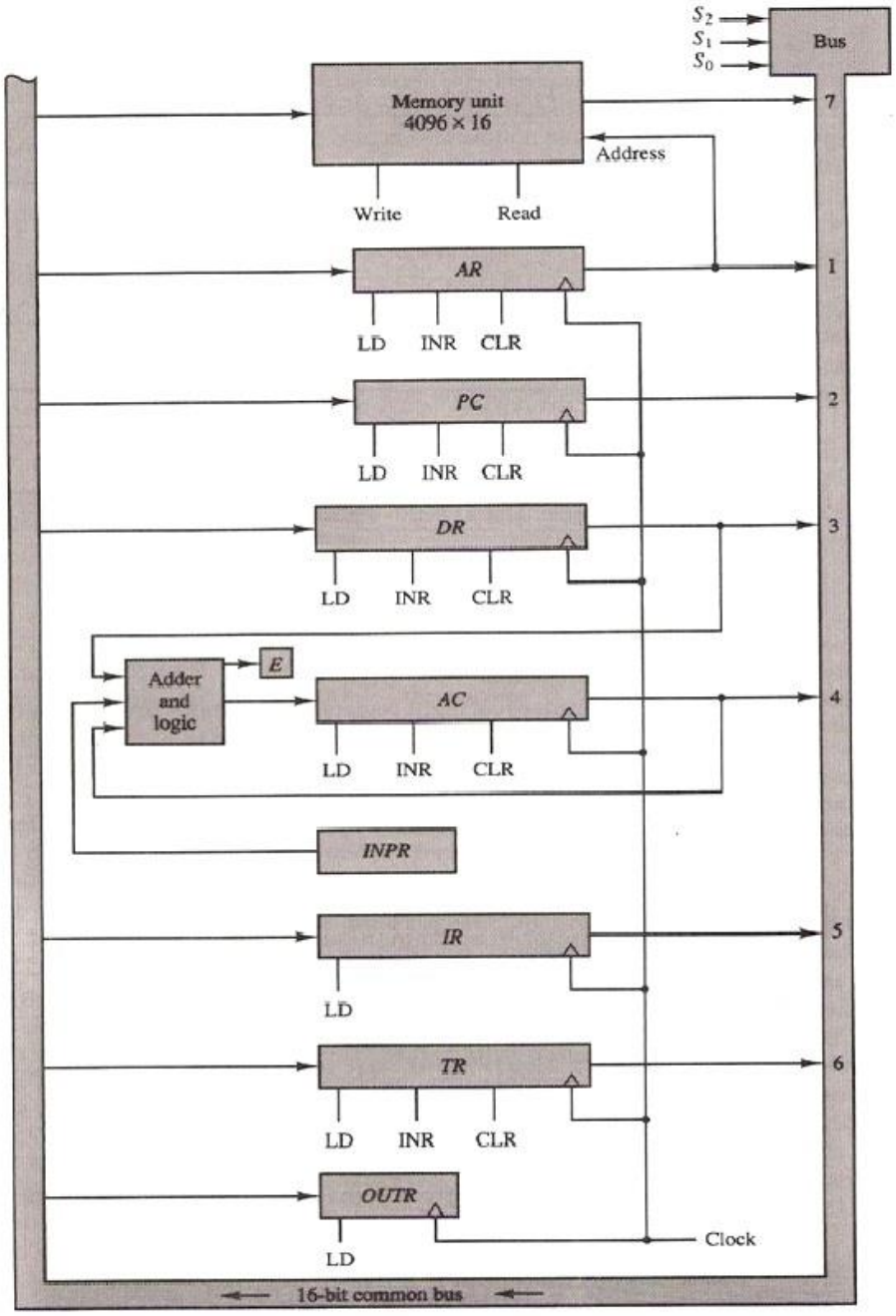
Mikroişlemcide toplam 8 adet kaydedici bulunmaktadır. Çizelge 3.1.'de tanımlı kaydediciler açıklamalarıyla birlikte gösterilmiştir [38].

Çizelge 3.1. Kaydediciler ve görevleri

Sembolü	Bit Sayısı	Register Adı	Görevi
DR	16	Data register	Veri kaydedicisi. Bu kaydedici, ALU işlemi için verileri tutar. Kaydedici çıkışı, D0 veri giriş hattındaki ALU'ya doğrudan bağlanır.
AR	12	Address register	Adres kaydedicisi. Bu kaydedici hafıza adresini tutar. Kaydedici çıkışı doğrudan belleğin adres hattına bağlıdır.
AC	16	Accumulator	Akümülatör. Bu kaydedicinin çıkışı direkt olarak ALU D1 veri giriş hattına bağlıdır.
IR	16	Instruction register	Talimat kaydedicisi. Bu kaydedici mevcut talimatı tutar
PC	12	Program counter	Program sayıcı. Yalnızca en az anlamlı 12 bit anlamlıdır.
TR	16	Temporary register	Geçici kaydedici. Geçici verileri tutar.
INPR	16	Input register	Giriş karakterini tutar. Kapasitesi artırılmıştır.
OUTR	16	Output register	Çıkış karakterini tutar. Kapasitesi artırılmıştır.

3.1.2. Veri Yolu Sistemi

AC hariç kaydedicilerin girişleri veri yoluna bağlanır. AC, girişini ALU'dan alır. Bu kaydedicilerin her biri LD, INR ve CLR kontrol hatlarına bağlıdır. LD, verileri kaydediciye yükleyecek, INR kaydedicisi artıracak ve CLR kaydediciyi temizleyecektir. Şekil 3.1.'de mikroişlemcinin genel veri yolu yapısı gösterilmiştir [38].

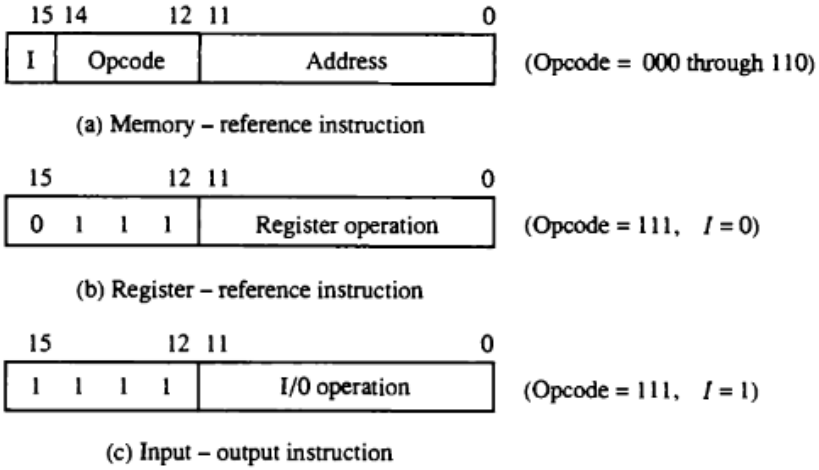


Şekil 3.1. Genel veri yolu yapısı

3.1.3. Komut Adresleme Türleri

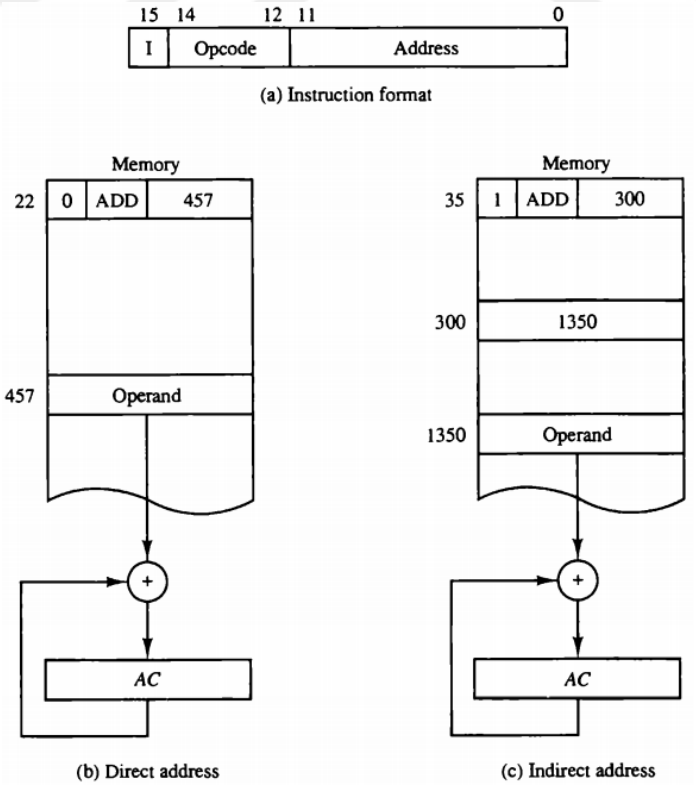
Komut setleri 6 bellek adresli komuttan oluşur. Şekil 3.2.'de [38] görüldüğü gibi işlem kodu 4 bitten oluşur. Kalan 12 bit ise ya belleği adreslemek ya da diğer (bellek dışı) talimatları kodlamak için kullanılır. En anlamlı bit, doğrudan veya dolaylı adresleme

türünü belirler. Eğer en anlamlı bit 0 ise direk adresleme (direct address), 1 ise dolaylı adresleme (inderect address) yapar. Diğer 3 bit komutu belirler.



Şekil 3.2. Komut adresleme türleri

Direk ve dolaylı adreslemenin bellekte yerleşimi Şekil 3.3'de [38] görüldüğü gibidir.



Şekil 3.3. Komut adresleme türü bellek yerleşimi

3.1.4. Komut Seti (Instruction Set)

Tasarlanan mikroişlemcinin tüm komut seti Çizelge 3.2.'de [38] listelenmiştir.

Çizelge 3.2. Komut seti

Symbol	Hexadecimal code $I = 0$	Hexadecimal code $I = 1$	Description
AND	0xxx	8xxx	AND memory word to *AC*
ADD	1xxx	9xxx	Add memory word to *AC*
LDA	2xxx	Axxx	Load memory word to *AC*
STA	3xxx	Bxxx	Store content of *AC* in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear *AC*
CLE	7400		Clear *E*
CMA	7200		Complement *AC*
CME	7100		Complement *E*
CIR	7080		Circulate right *AC* and *E*
CIL	7040		Circulate left *AC* and *E*
INC	7020		Increment *AC*
SPA	7010		Skip next instruction if *AC* positive
SNA	7008		Skip next instruction if *AC* negative
SZA	7004		Skip next instruction if *AC* zero
SZE	7002		Skip next instruction if *E* is 0
HLT	7001		Halt computer
INP	F800		Input character to *AC*
OUT	F400		Output character from *AC*
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

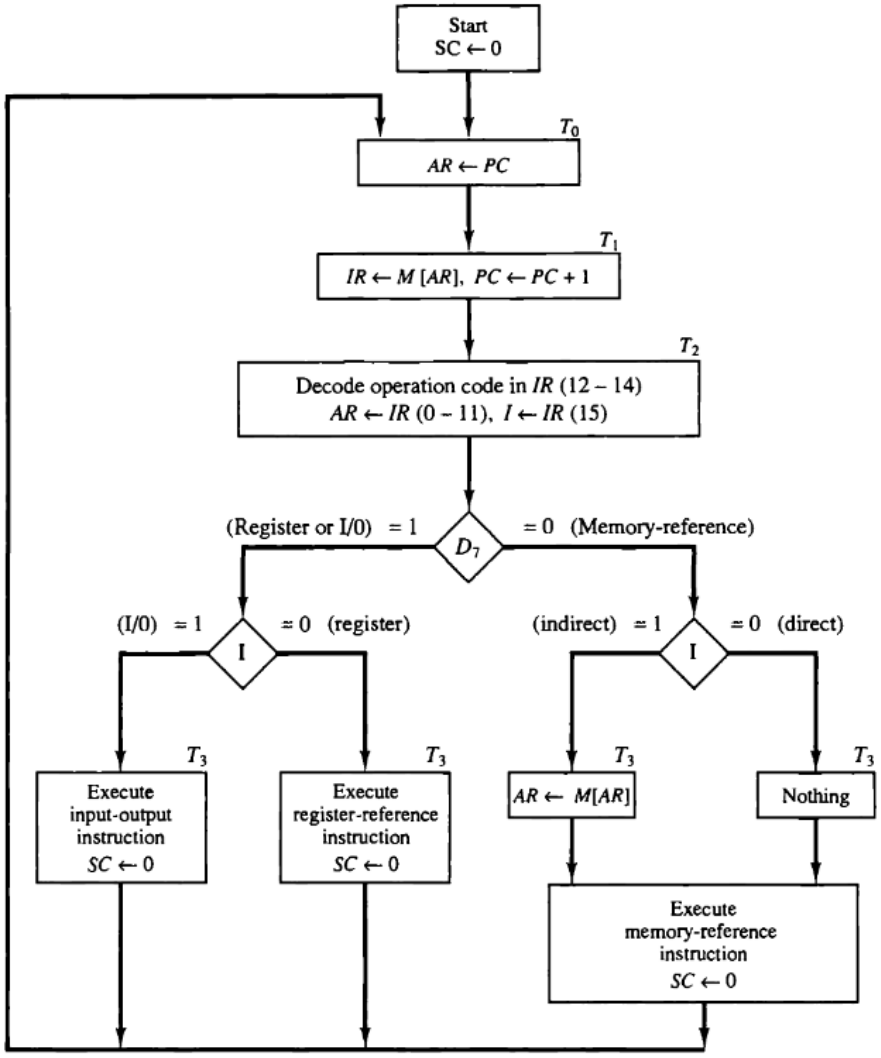
Mikroişlemcinin kontrol fonksiyonlarının tanımlarını içeren liste Çizelge 3.3.'de [38] gösterilmiştir.

Çizelge 3.3. Tasarlanan mikroişlemcinin fonksiyon tanımları ve mikroişlemler

Fetch	$R'T_0$:	$AR \leftarrow PC$
	$R'T_1$:	$IR \leftarrow M[AR]$, $PC \leftarrow PC + 1$
Decode	$R'T_2$:	$D_0, \ldots, D_7 \leftarrow$ Decode $IR(12\text{–}14)$, $AR \leftarrow IR(0\text{–}11)$, $I \leftarrow IR(15)$
Indirect	D_7IT_3:	$AR \leftarrow M[AR]$
Interrupt:		
$T_0'T_1'T_2'(IEN)(FGI + FGO)$:		$R \leftarrow 1$
	RT_0:	$AR \leftarrow 0$, $TR \leftarrow PC$
	RT_1:	$M[AR] \leftarrow TR$, $PC \leftarrow 0$
	RT_2:	$PC \leftarrow PC + 1$, $IEN \leftarrow 0$, $R \leftarrow 0$, $SC \leftarrow 0$
Memory-reference:		
AND	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \wedge DR$, $SC \leftarrow 0$
ADD	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$AC \leftarrow AC + DR$, $E \leftarrow C_{out}$, $SC \leftarrow 0$
LDA	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$AC \leftarrow DR$, $SC \leftarrow 0$
STA	D_3T_4:	$M[AR] \leftarrow AC$, $SC \leftarrow 0$
BUN	D_4T_4:	$PC \leftarrow AR$, $SC \leftarrow 0$
BSA	D_5T_4:	$M[AR] \leftarrow PC$, $AR \leftarrow AR + 1$
	D_5T_5:	$PC \leftarrow AR$, $SC \leftarrow 0$
ISZ	D_6T_4:	$DR \leftarrow M[AR]$
	D_6T_5:	$DR \leftarrow DR + 1$
	D_6T_6:	$M[AR] \leftarrow DR$, if $(DR = 0)$ then $(PC \leftarrow PC + 1)$, $SC \leftarrow 0$
Register-reference:		
	$D_7I'T_3 = r$ (common to all register-reference instructions)	
	$IR(i) = B_i$ $(i = 0, 1, 2, \ldots, 11)$	
	r:	$SC \leftarrow 0$
CLA	rB_{11}:	$AC \leftarrow 0$
CLE	rB_{10}:	$E \leftarrow 0$
CMA	rB_9:	$AC \leftarrow \overline{AC}$
CME	rB_8:	$E \leftarrow \overline{E}$
CIR	rB_7:	$AC \leftarrow$ shr AC, $AC(15) \leftarrow E$, $E \leftarrow AC(0)$
CIL	rB_6:	$AC \leftarrow$ shl AC, $AC(0) \leftarrow E$, $E \leftarrow AC(15)$
INC	rB_5:	$AC \leftarrow AC + 1$
SPA	rB_4:	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$
SNA	rB_3:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$
SZA	rB_2:	If $(AC = 0)$ then $PC \leftarrow PC + 1$
SZE	rB_1:	If $(E = 0)$ then $(PC \leftarrow PC + 1)$
HLT	rB_0:	$S \leftarrow 0$
Input-output:		
	$D_7IT_3 = p$ (common to all input–output instructions)	
	$IR(i) = B_i$ $(i = 6, 7, 8, 9, 10, 11)$	
	p:	$SC \leftarrow 0$
INP	pB_{11}:	$AC(0\text{–}7) \leftarrow INPR$, $FGI \leftarrow 0$
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{–}7)$, $FGO \leftarrow 0$
SKI	pB_9:	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$
SKO	pB_8:	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$
ION	pB_7:	$IEN \leftarrow 1$
IOF	pB_6:	$IEN \leftarrow 0$

3.1.5. Talimat Türünü Belirleme

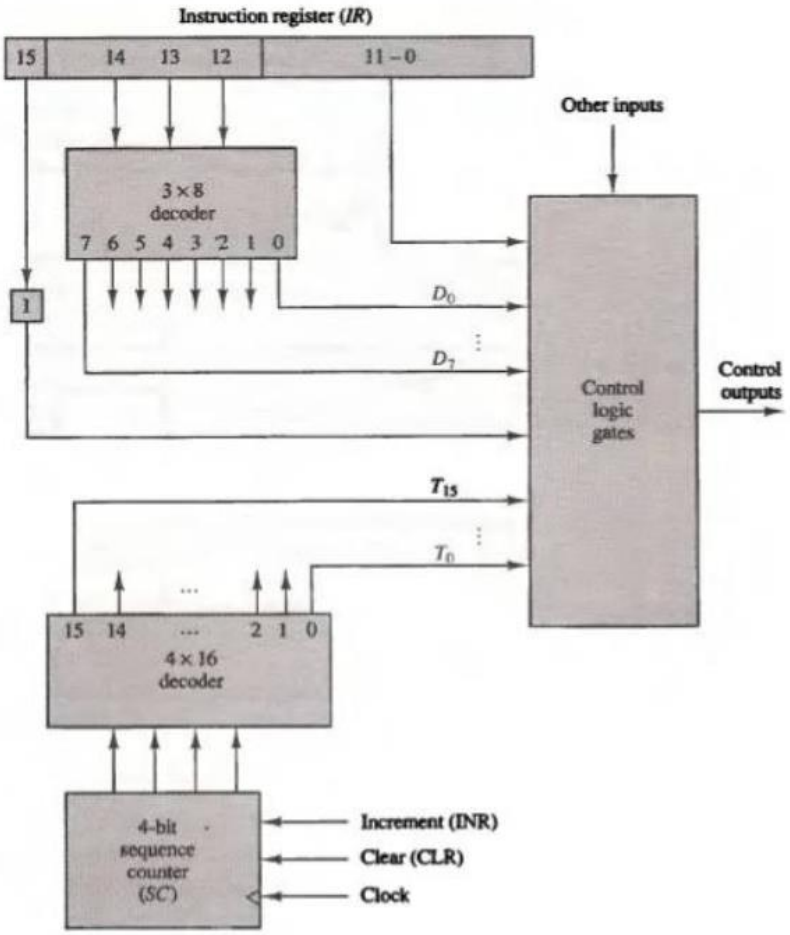
Kod çözmeden sonra aktif olan zamanlama sinyali T3'tür. T3 süresi boyunca, kontrol birimi hafızadan okunan talimatın tipini belirler. Şekil 3.3. komut döngüsü için bir başlangıç konfigürasyonu sunar ve kodun çözülmesinden sonra kontrolün talimat tipini nasıl belirlediğini gösterir. Temel mikroişlemcide mevcut olan üç olası komut türü Şekil 3.4.'de belirtilmiştir. İşlem kodu ikili 111'e eşitse, kod çözücü çıkışı D_7 1'e eşittir [38].



Şekil 3.4. Talimat çevrimi akış şeması (mikroişlemci operasyonu)

3.1.6. Zamanlama ve Kontrol

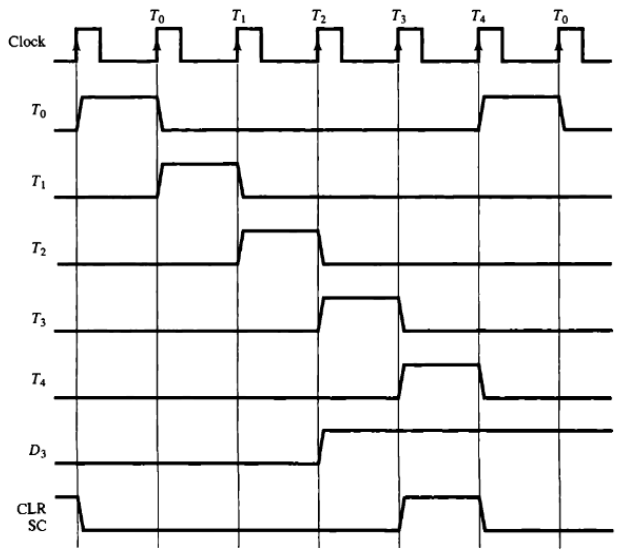
Mikroişlemcideki tüm kaydedicilerin zamanlaması bir ana saat üreteci tarafından kontrol edilir. Saat darbeleri sistemdeki tüm kaydedicilere ve flip-floplara uygulanır. Kontrol sinyallerinin blok diyagramı Şekil 3.5.'de gösterilmiştir. İki kod çözücü (decoder), bir sıra sayacı (Sequence Counter-SC) ve bir dizi kontrol mantık kapısından oluşur. SC, eşzamanlı olarak artırılabilir veya temizlenebilir. Çoğu zaman SC, 4x16 kod çözücüden gelen zamanlama sinyallerinin sırasını sağlamak için artırılır. Arada bir, sayaç 0'a temizlenir ve bir sonraki aktif zamanlama sinyalinin T0 olmasına neden olur.



Şekil 3.5. Mikroişlemcinin kontrol ünitesi

Örnek olarak, T_0, T_1, T_2, T_3 ve T_4 zamanlama sinyallerini sırayla sağlamak için SC'nin artırıldığı durumu düşünün. T_4 zamanında, eğer kod çözücü çıkışı D_3 aktifse SC, 0 olarak temizlenir. Bu, D_3T_4: SC←0 ifadesiyle sembolik olarak ifade edilir.

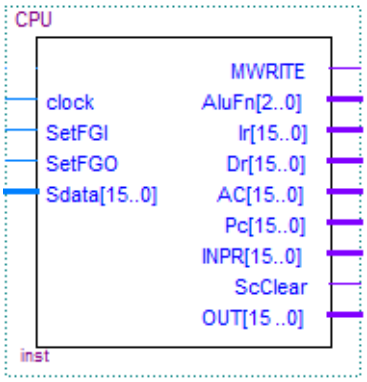
Şek. 3.6.'da kontrol sinyallerinin, zaman ilişkisini gösterir. Sıra sayacı SC, saatin pozitif geçişine yanıt verir. Başlangıçta SC'nin CLR girişi aktiftir. Saatin ilk pozitif geçişi SC'yi 0'a siler, bu da kod çözücüden t_0 zamanlama sinyalini etkinleştirir. T0, bir kez saat döngüsü sırasında aktiftir. SC, CLR girişi aktif olmadığı sürece her pozitif saat geçişinde artırılır. SC temizlenmemişse, zamanlama sinyalleri T_5, T_6 ile T_{15}'e kadar devam edecek ve T_0'a geri dönecektir [38].



Şekil 3.6. Zamanlama ve kontrol

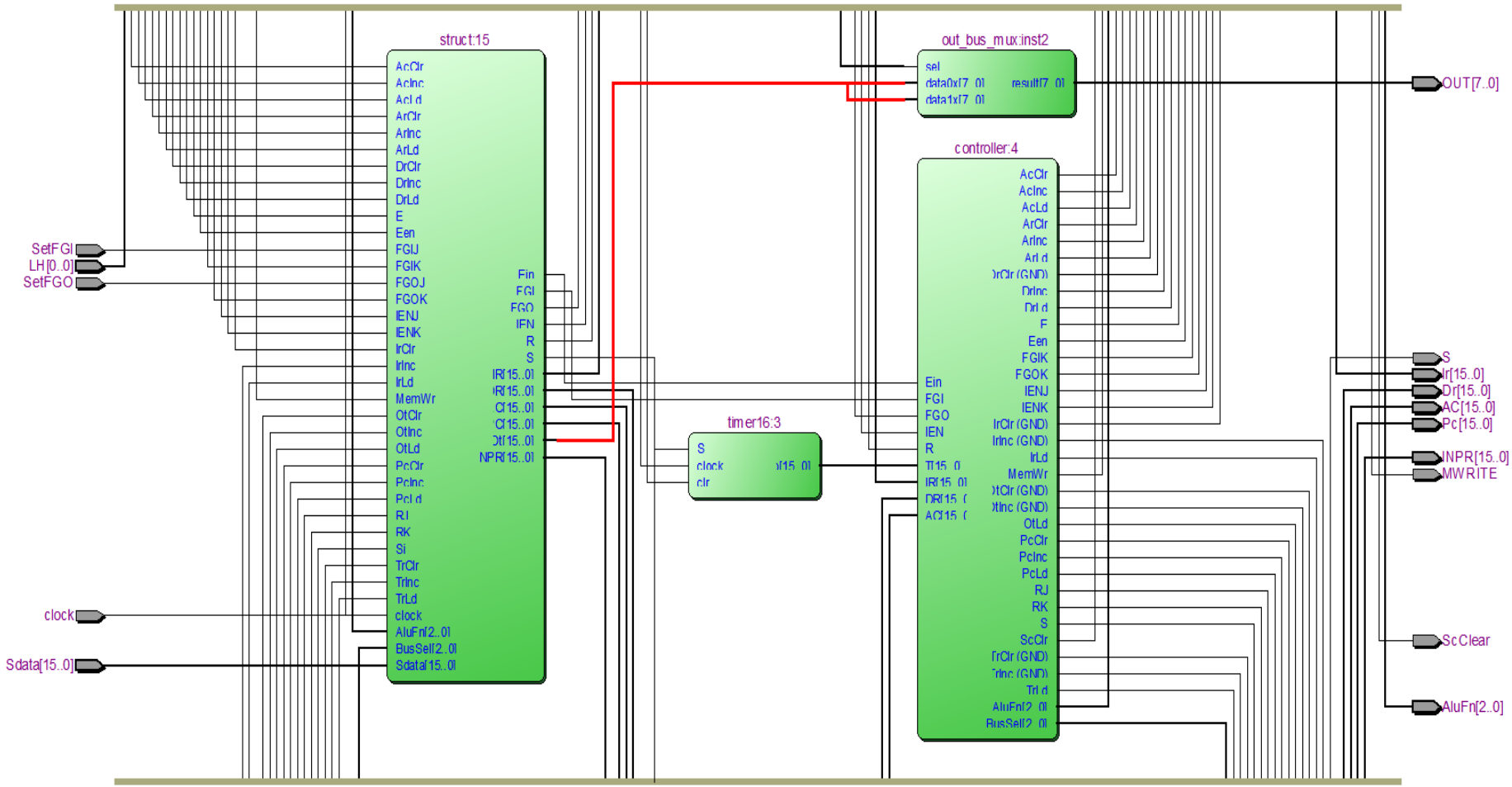
3.1.7. CPU Mimari Tasarımı

Tasarımı yapılan 16 bitlik mikroişlemci, dört adet giriş portu ve 9 adet çıkış portuna sahiptir. Şekil 3.7.'de bu portlar görülmektedir.



Şekil 3.7. 16-Bitlik mikroişlemcinin üst mimari görünümü

Mikroişlemci yine kendi içinde üç alt birimden oluşur. Bunlar zamanlayıcı birimi (timer), kontrol birimi (controller) ve kaydedicilerin, ALU ve hafızanın bulunduğu tanımlama (structure) biriminden oluşur. Şekil 3.8.'de tasarım gösterilmiştir. Bu alt birimlerde kendi içinde çeşitli alt birimlerden meydana gelmektedir.



Şekil 3.8. 16-Bitlik mikroişlemcinin alt mimari görünümü

3.1.8. Tasarlanan Mikroişlemcinin Bileşenleri

VHDL dilinde paket tanımlamaları tasarımlarda büyük kolaylık sağlamaktadır. Bunları bir pakette tanımlamak ve ardından paketi tüm VHDL dosyalarına dahil etmek sistem tasarlamayı daha fonksiyonel yapmaktadır. Tanımları, veri tiplerini ve çeşitli birimleri kontrol etmek için üç paket tanımlanmıştır. Bunlar "defines.vhd" (Bkz. Ek 5. Defines Yapısının VHDL Kodları), "datatypes.vhd" (Bkz. Ek 23. Datatypes Yapısının VHDL Kodları) ve "devices.vhd" (Bkz. Ek 7. Devices Yapısının VHDL Kodları) şeklinde tanımlanmıştır.

3.1.8.1. Kod Çözücü

İlk önce birkaç temel birimler tanımlanır. Mano'nun kitabının 2. Bölümü [38] mantıksal kapı tasarımları detaylı bir şekilde anlatılmaktadır. İlk önce, etkinleştirme hattı olan veya olmayan 1x2 kod çözücüler tanımlanmıştır. Bu oluşturulan 1x2'lik kod çözücüleri Mano'nun kitabında anlatıldığı gibi daha büyük kod çözücüler oluşturmak için kullanılmıştır. Bu kod çözücüler tasarımda "dec1x2 (etkinleştirilmeyen kod çözücü)" (Bkz. Ek 17. Dec1x2 Biriminin VHDL Kodları), "dec1x2e (etkinleştirilmiş kod çözücü)" (Bkz Ek. 18. Dec1x2e Biriminin VHDL Kodları), "dec2x4" (Bkz Ek. 19. Dec2x4 Biriminin VHDL Kodları), "dec2x4e" (Bkz Ek. 20. Dec2x4e Biriminin VHDL Kodları), "dec3x8" (Bkz Ek. 21. Dec3x8 Biriminin VHDL Kodları) ve "dec4x16" (Bkz Ek. 22. Dec4x16 Biriminin VHDL Kodları) şeklinde tanımlanmıştır.

3.1.8.2. Flip Flops (FF)

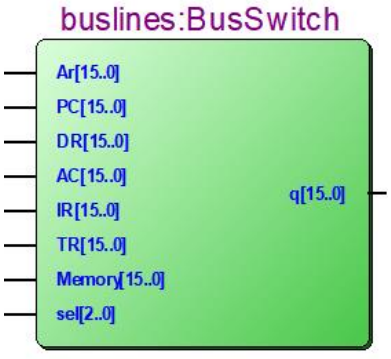
Tasarımda register ve diğer işlemler için geçici hafıza birimleri olarak D-F (Bkz. Ek 24. DFFlop Biriminin VHDL Kodları) F ve JK-FF (Bkz. Ek 28. JKFFlop Biriminin VHDL Kodları) tanımlanmıştır.

3.1.8.3. Kaydediciler (Registers)

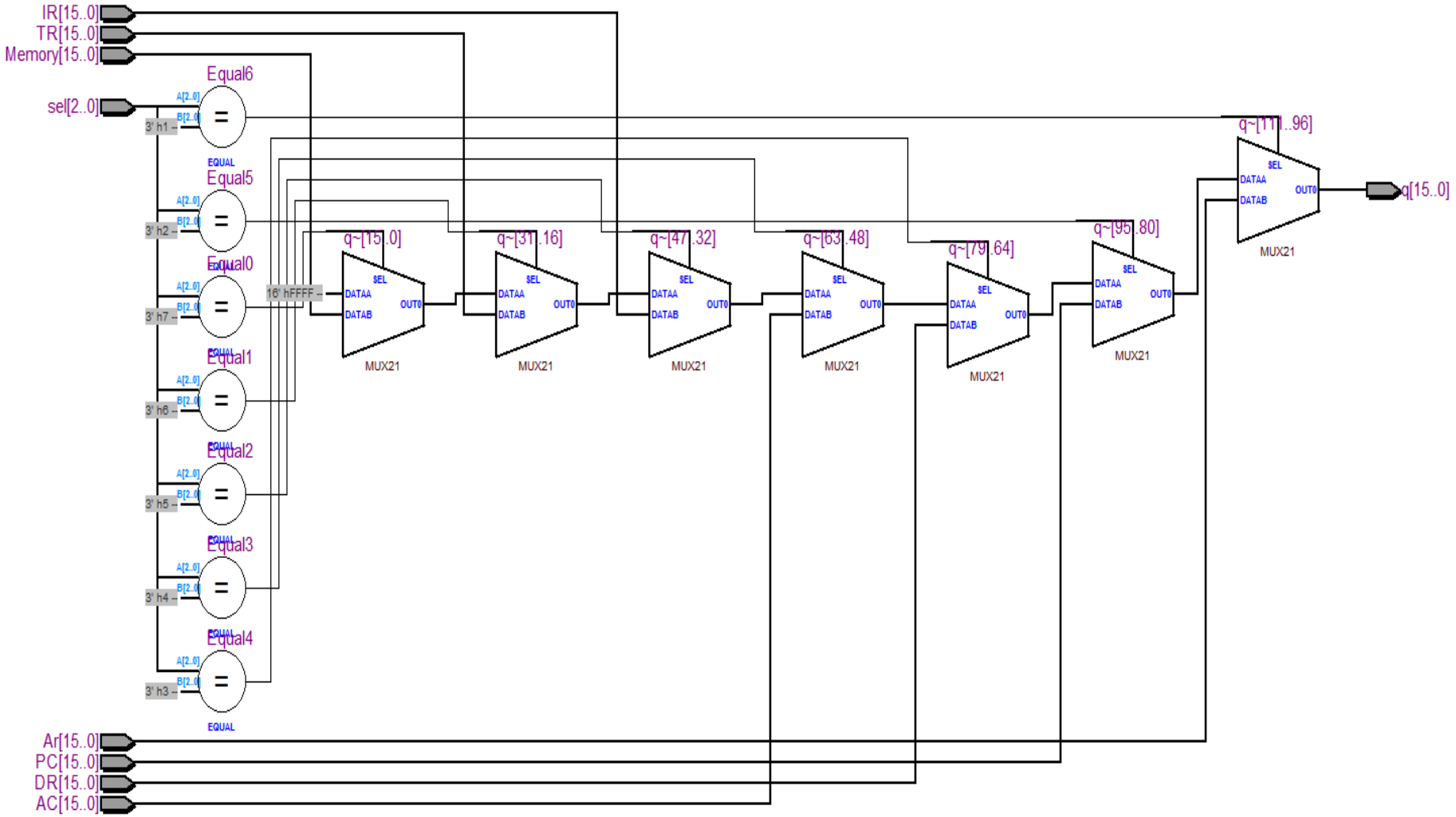
İlk önce JK FF kullanarak 1 bitlik bir kaydedici tanımlanmıştır. Reg1 (Bkz. Ek 10. Reg1 Biriminin VHDL Kodları) isminde ilk kaydedici VHDL dilinde oluşturulduktan sonra, artık n-bit bir kaydedici tanımlamak için VHDL'nin iki önemli özelliğini kullanılır. Bunlardan biri genel bildirimdir. Bu, tasarımı herhangi bir n için genelleştirmemizi sağlar. İkinci olarak, birbirine bağlı n adet 1 bitlik yazmaç dizisini oluşturmak için bir for döngüsü kullanılır. Ayrıca, n için varsayılan bir değer atanabilir. Bu şekilde tasarımda ihtiyaç olan regn (Bkz. Ek 13. Regn Biriminin VHDL Kodları), reg16 (Bkz. Ek 12. Reg16 Biriminin VHDL Kodları) ve reg12 (Bkz. Ek 11. Reg12 Biriminin VHDL Kodları) bitlik kaydediciler kolay bir şekilde tanımlanmıştır. Tasarımda 12 bitlik kaydedici ve 16 bitlik kaydediciler olarak tanımlanmışlardır. Bu diğer bağlantılarla senkronizasyonu kolaylaştırır ve karışıklığı önlemektedir. Çünkü kullanılmayan bitler 0 olacak, ir etkisi de olmayacaktır. Fakat tasarımda karmaşıklık olmayacaktır.

3.1.8.4. Veri Yolu Yapısı

CPU çoklu veri yolu kullanır. Bu, herhangi bir zamanda veri yolunun bir veri kaynağına sahip olduğu anlamına gelir. Üç bitlik bir seçme hattı kaynağı seçer. Olası kaynakların her biri için seçim kodlarını sabit kodlamak yerine, diğer kaynak dosyalarla koordinasyon için "defines.vhd" (Bkz. Ek 16. Buslines Biriminin VHDL kodları) paketinde tanımlanan sabitler kullanılır. Bus tasarımı dış mimari tasarımı Şekil 3.9.'da, iç mimari tasarım yapısı ise 3.10.'da yer almaktadır.



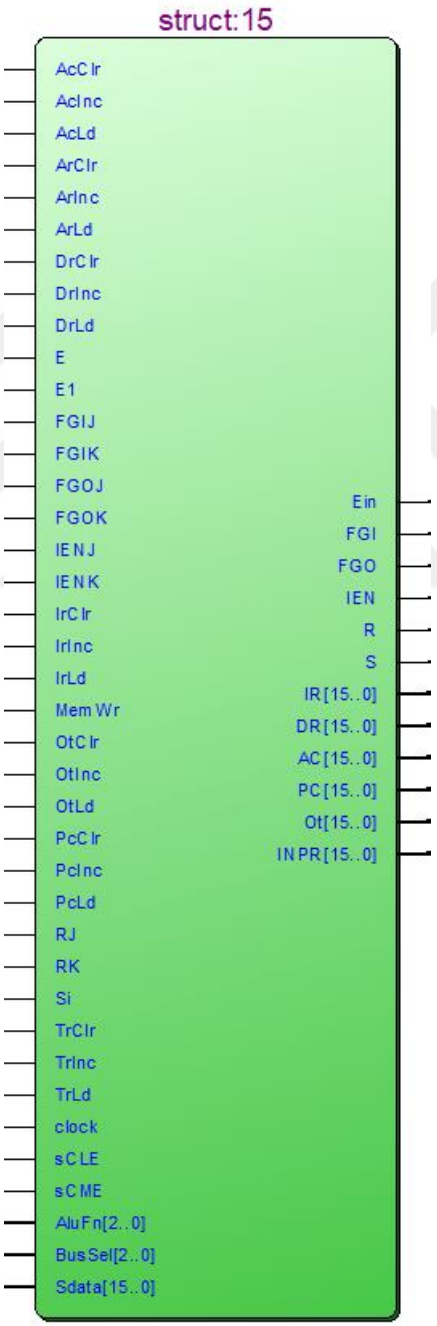
Şekil 3.9. BusSwitch dış mimari tasarımı



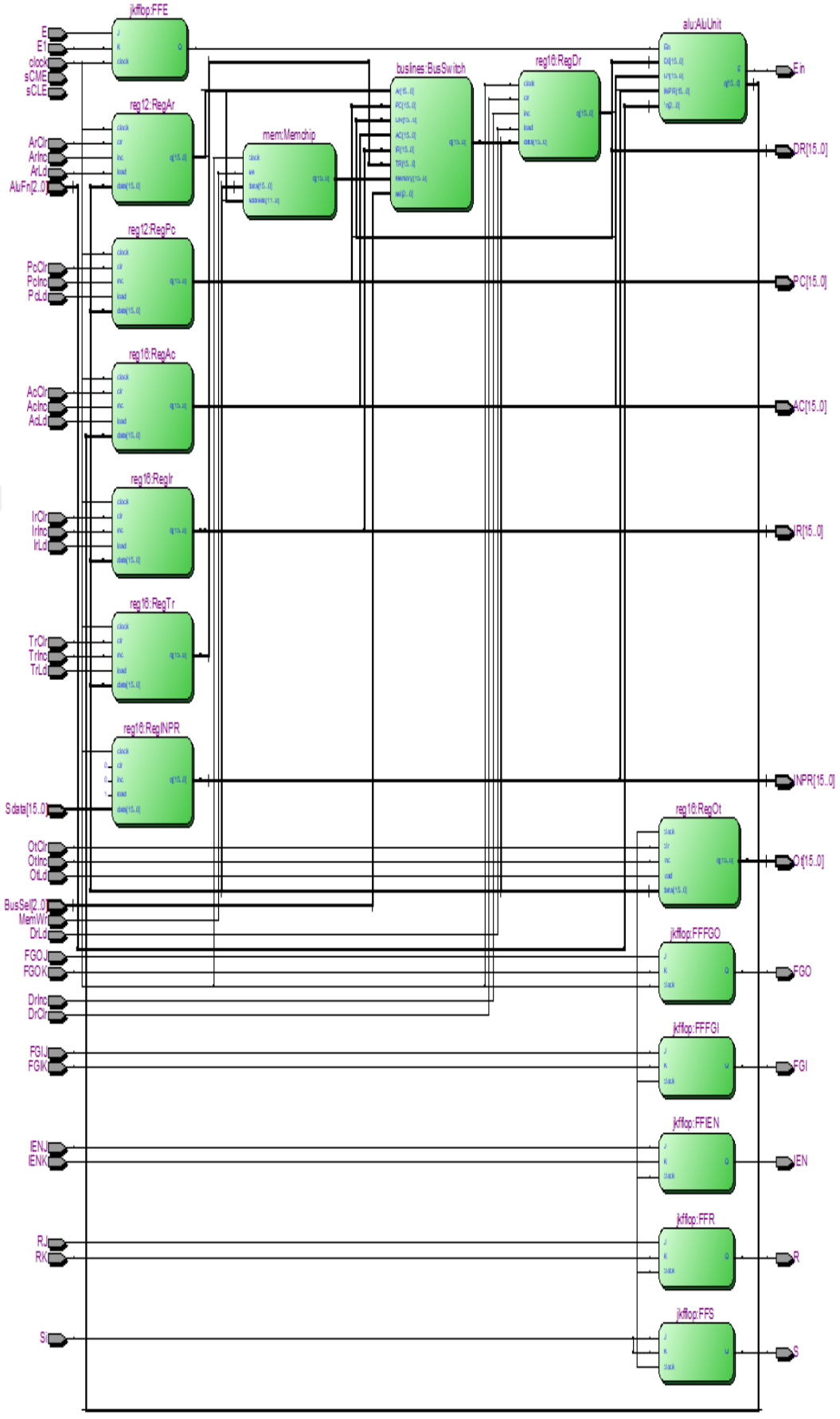
Şekil 3.10. BusSwitch iç mimari tasarımı

3.1.8.5. Tanımlama Ünitesi (Structure)

Tasarımda “struct.vhd” (Bkz. Ek 2. Structure Yapısının VHDL Kodları) şeklinde tanımlanmıştır. İçerisinde kaydediciler, mem çipi, veri yolu anahtarlama için bus switch, FF’lar ve ALU yer almaktadır. Şekil 3.11.’de mimarinin dış yapısı, Şekil 3.12.’de de içyapısı gösterilmiştir.



Şekil 3.11. Structure dış mimari tasarımı

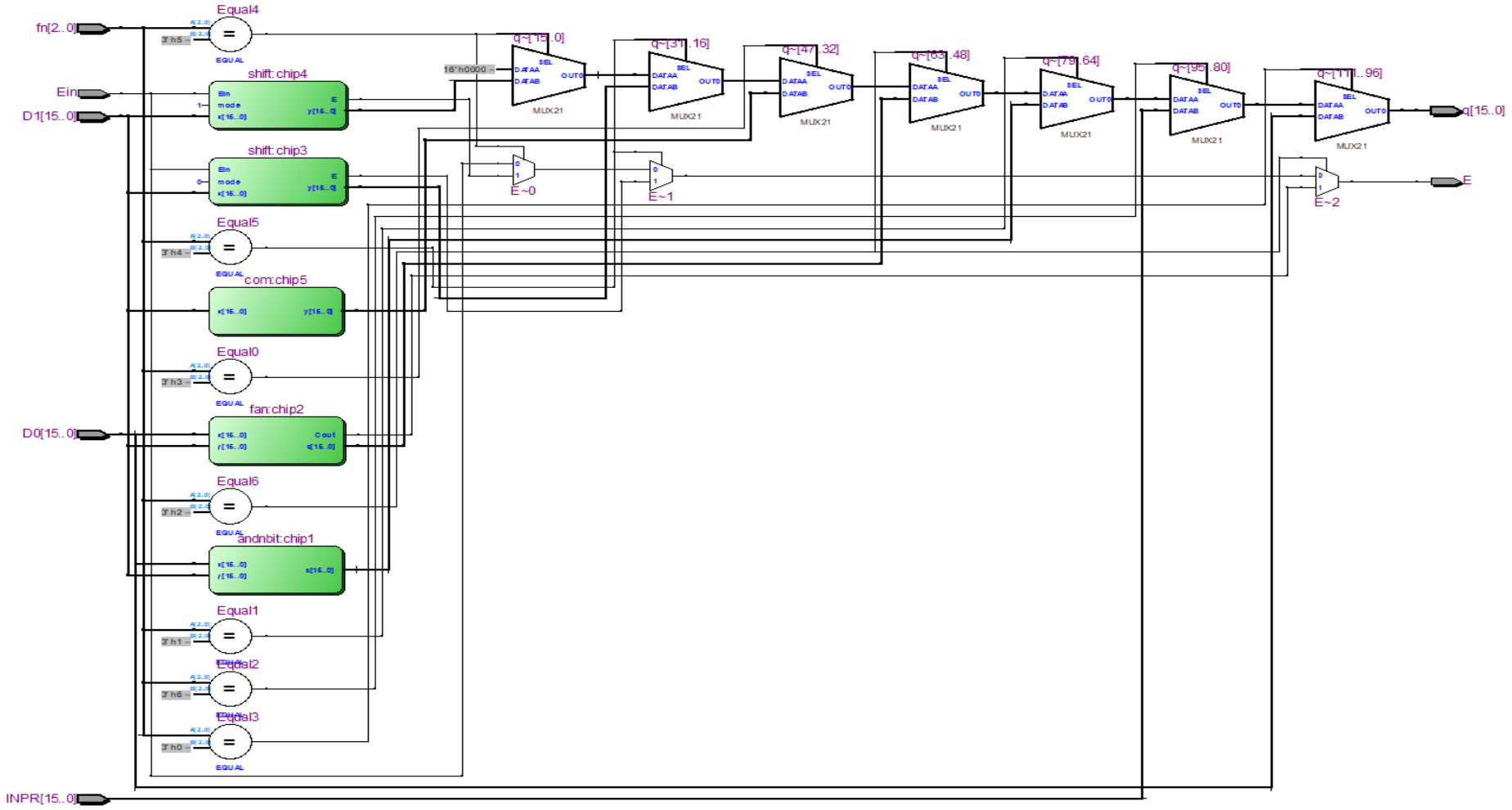


Şekil 3.12. Structure iç mimari tasarımı

3.1.8.6. Aritmetik Mantık Birimi (ALU)

Tasarımda "alu.vhd" (Bkz. Ek 1. ALU Yapısının VHDL Kodları) şeklinde tanımlanmıştır. ALU biriminin dış yapısı Şekil 3.13.'de görülmektedir. 5 adet giriş, 2 adet çıkış portuna sahiptir. Şekil 3.14.'de de ALU'nun iç mimari tasarımı yer almaktadır.

Şekil 3.13. ALU biriminin dış yapısı



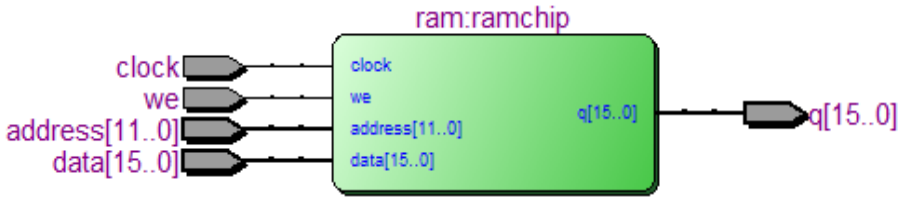
Şekil 3.14. ALU iç mimari yapısı

3.1.8.7. ALU Fonksiyon Tanımlamaları

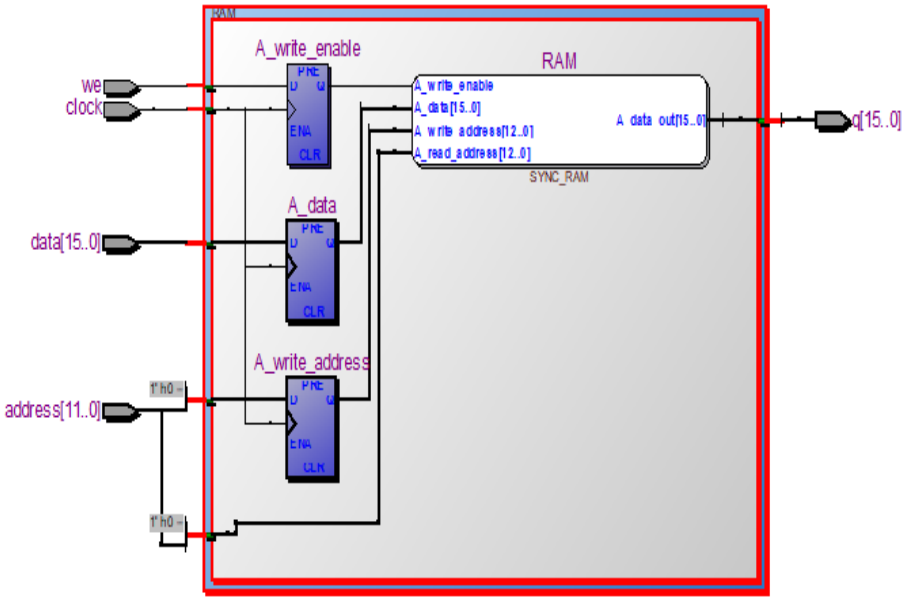
ALU birimi içerisinde, fonksiyon tanımlamaları yapılmıştır. Bir bit yarım toplayıcı (One bit half adder) "hadder.vhd" (Bkz. Ek 27. Hadder Biriminin VHDL Kodları), bir bit tam toplayıcı (one bit full adder) "fa1" (Bkz. Ek 25. Fa1 Biriminin VHDL Kodları), n bit tam toplayıcı (n bit full adder) "fan.vhd" (Bkz. Ek 26. Fan Birimi VHDL Kodları) olarak tanımlamaları yapılmıştır. Mantıksal fonksiyonlar için, bir bitlik ve kapısı (one bit and gate) "and1bit.vhd" (Bkz. Ek 8. and1bit Biriminin VHDL Kodları), iki adet n bitlik ve kapısı işlemi için "andnbit.vhd" (Bkz Ek 9. Andnbit Biriminin VHDL Kodları) olarak tanımlamaları yapılmıştır. Yer değiştirme işlemleri için ise sağa ve sola kaydırma "shift.vhd" (Bkz. Ek 31. Shift Biriminin VHDL Kodları) olarak tanımlanmıştır.

3.1.8.8. Bellek Birimi (RAM)

Tasarımda "ram.vhd" (Bkz. Ek 30. Ram Biriminin VHDL Kodları) olarak tanımlanmıştır. 16 bit data, 12 bir adres girişli, yazma sinyali (write enable) ve clock bağlantısı vardır. Write enable 0 ise okuma, 1 ise belleğe yazma işlemi yapmaktadır. Von-Neumann mimari tasarımına sahiptir. 4096 word genişliğinde 16 bitlik bir RAM yapısındadır. RAM birimine yazılan program kodu ve veriler işletilir. Bu konu simülasyon bölümünde örneklerle açıklanacaktır. Ram tasarımının dış mimarisi 3.15.'de, iç mimarisi ise 3.16.'de gösterilmektedir.



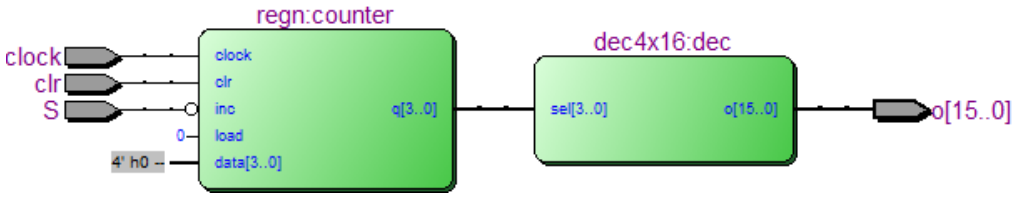
Şekil 3.15. Ram birimi dış mimari tasarımı



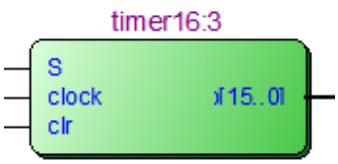
Şekil 3.16. Ram birimi iç mimari tasarımı

3.1.8.9. Zamanlayıcı (Timer)

Tasarımda zamanlayıcı "timer16.vhd" (Bkz. Ek 4. Timer16 Yapısının VHDL Kodları) olarak tanımlanmıştır. 4 bitlik bir kaydedici ve 4x16 kod çözücü alt birimlerinden oluşmaktadır. Önce bir n-bitlik sayıcı tanımlanmıştır, sonra bu sayıcı bir kod çözücüye bağlanmıştır. Şekil 3.17.'de timer iç mimari yapısı gösterilmiştir. Zamanlayıcı dış mimarisi Şekil 3.18.'de gösterilmiştir.



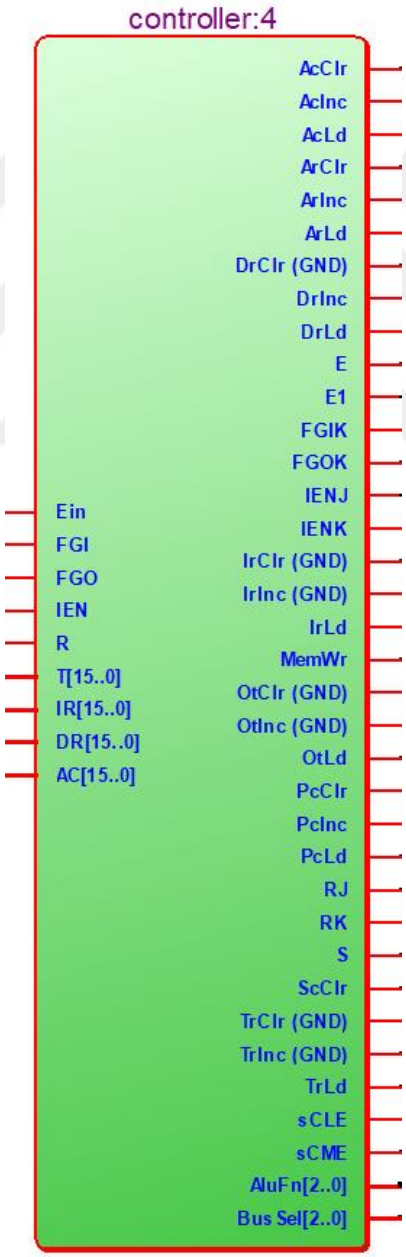
Şekil 3.17. Timer16 iç mimarisi



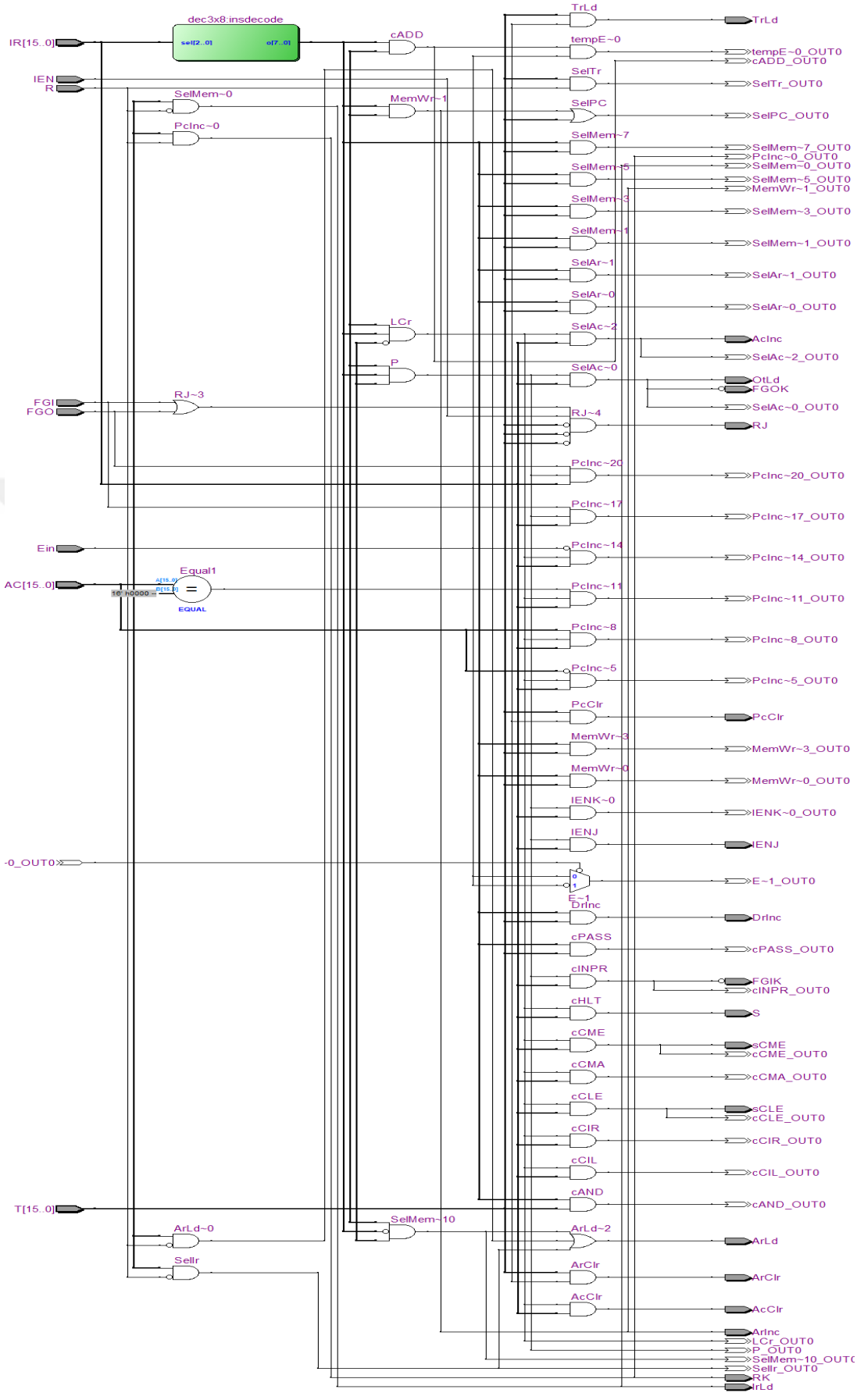
Şekil 3.18. Timer16 dış mimarisi

3.1.8.10. Kontrol Ünitesi (Controller)

Kontrol ünitesi mikroişlemcinin kalbi gibi düşünülebilir. Tasarımda kontrol ünitesi "controller.vhd" (Bkz. Ek 3. Kontrol Ünitesinin VHDL Kodları) olarak tanımlanmıştır. IR saat sırasına ve diğer kaydedicilerin durumuna göre uygun kontrol sinyali üretir. Şekil 3.19.'de kontrol ünitesi dış mimarisi, Şekil 3.20.'de de kontrol ünitesi iç mimarisi gösterilmiştir. Mano'nun 3.baskı kitabının 5. Bölümünde [38] tarif ettiği yapıda ve tüm sinyalleri üretecek şekilde tanımlanmıştır.



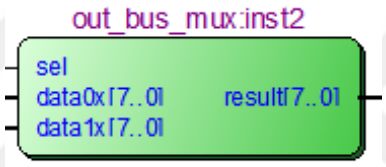
Şekil 3.19. Kontrol ünitesi dış mimari yapısı



Şekil 3.20. Kontrol ünitesi iç mimari yapısı

3.1.8.11. Çıkış Portu Modülü (Out BUS MUX)

Tasarımda çıkış portunu 8‘er bit olarak FPGA kartı çıkışında göstermek için “out_bus_mux.vhd” (Bkz. Ek 6. OUT BUS MUX Yapısının VHDL Kodları) tanımlanmıştır. Bu alt birim, mikroişlemcinin 16 bitlik çıkış portundan elde edilen değeri 8’er bit şeklinde iki seviye şeklinde göstermek için tanımlanmıştır. Şekil 3.21.’de mimari görünümü yer almaktadır. Bunun yapılmasının sebebi gömülü sistem uygulamasında kullanılacak Intel (Altera) DE0 Nano FPGA kartının 8 adet led çıkış birimine sahip olmasıdır. Sonuçlar bu led üzerinde ilk 8 bit 0-7.bitleri gösterecek, FPGA kartı üzerindeki butona basınca diğer 8 bit 8-15.bitler gösterilecek şekilde tanımlaması yapılmıştır.



Şekil 3.21. Out bus mux mimari yapısı

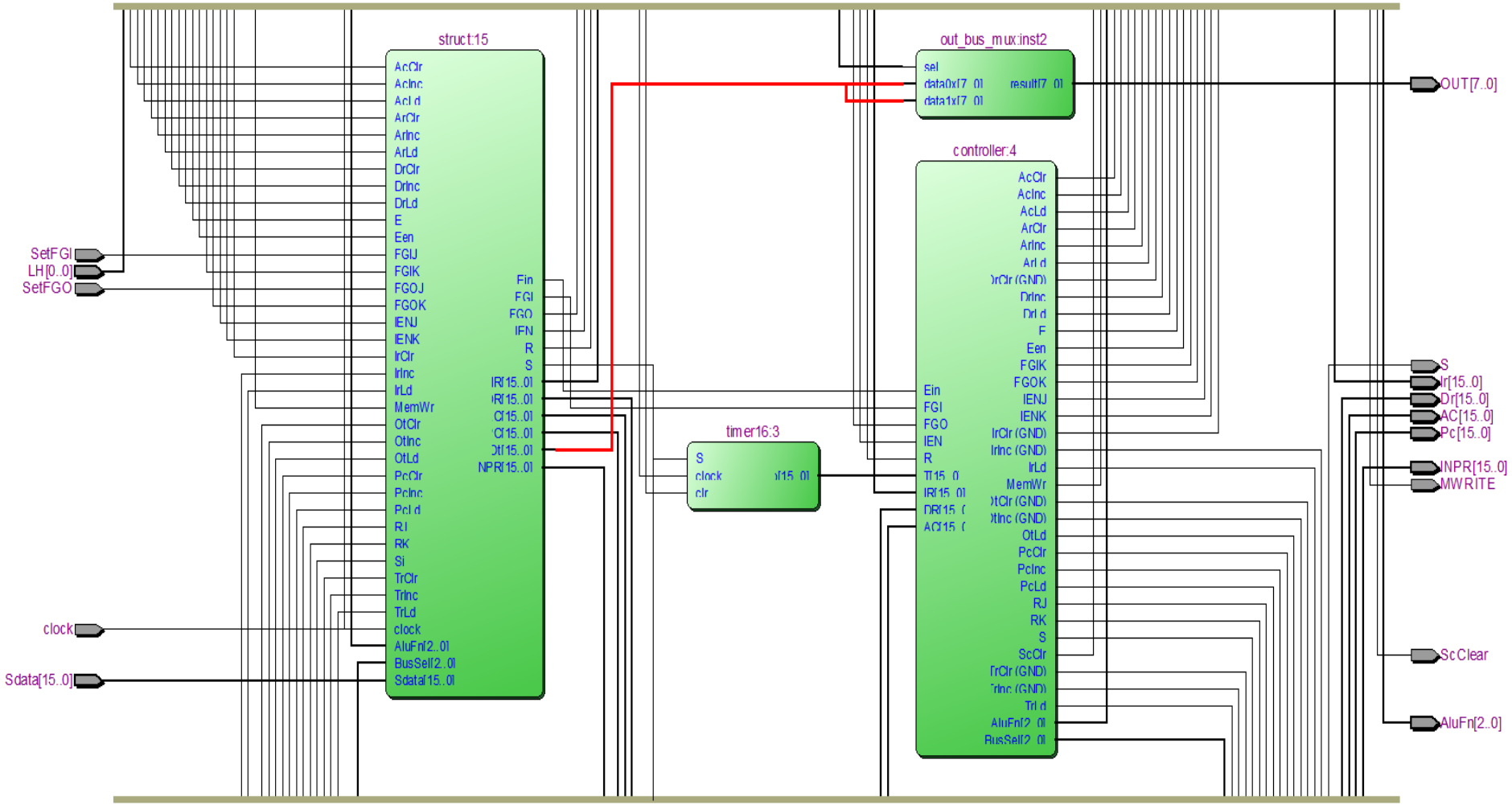
3.1.8.12. Tasarım Birimlerini Birleştirme

Mikroişlemcinin tüm birimleri “devices.vhd” (Bkz. Ek 7. Devices Yapısının VHDL Kodları) dosyasında tanımlanmış ve birleştirilmiştir. Birimde yapılacak her değişiklikte sistemin güncellenmesi ve derlenmesi gerekir.

3.1.8.13. 16-Bitlik Mikroişlemcinin Tamamlanması

Tüm birimler ve tanımlamalar tanımlandıktan sonra mikroişlemci hazır hale gelmiştir. Bundan sonra VHDL kodu yerine grafik arayüz kullanılmıştır. Grafiksel arayüz, I/O pinlerini yerleştirmemize ve ayrıca mikroişlemciyi simüle etmemize izin vermektedir. Burada amaç kontrol çıkış hatlarını ilgili yapı giriş hatlarına ve yapı çıkış hatlarını kontrol ünitesi biriminin ilgili giriş hatlarına bağlamaktır. Zamanlayıcı da kontrol ünitesine bağlıdır. 16 Bitlik mikroişlemci çevre birimleri ile birlikte bağlantıları

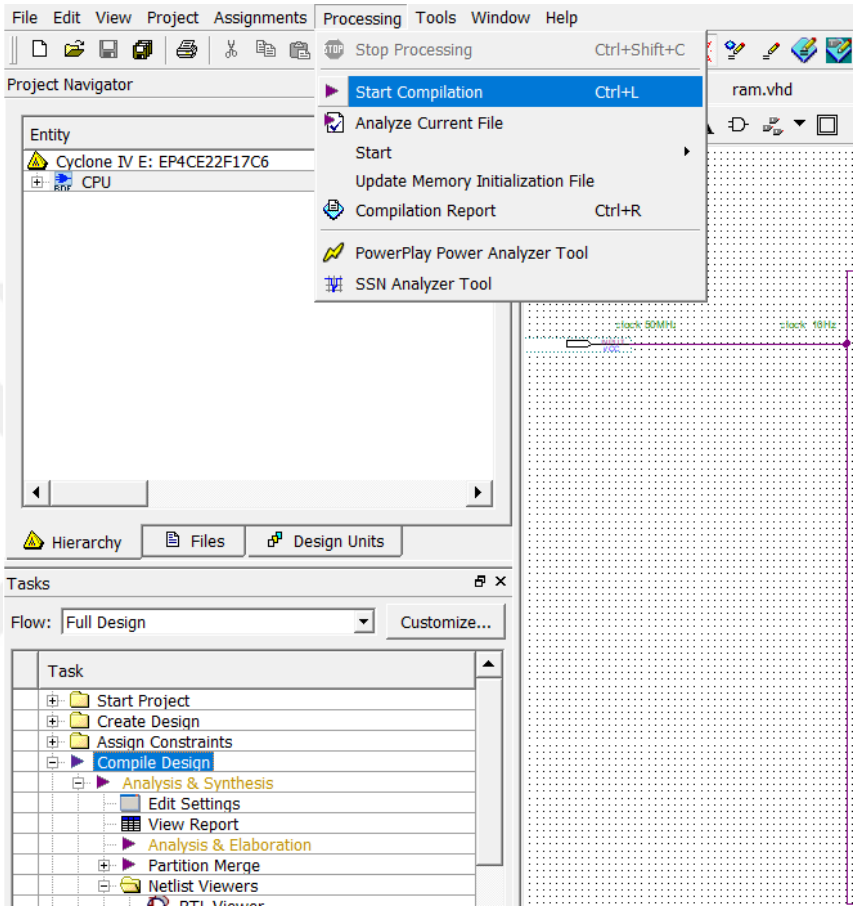
yapılmış, çalıştırılabilir hale gelmiştir. Bütün iç ve dış çevre birimleri ile birleştirildiğinde temel bir mikrobilgisayarı oluşturmaktadır. Şekil 3.22.'de mikroişlemci ve çevre birimler ile bağlantı yapısı gösterilmiştir.



Şekil 3.22. 16-Bitlik mikrobilgisayar yapısı

3.2. Tasarımın Analiz ve Sentezi

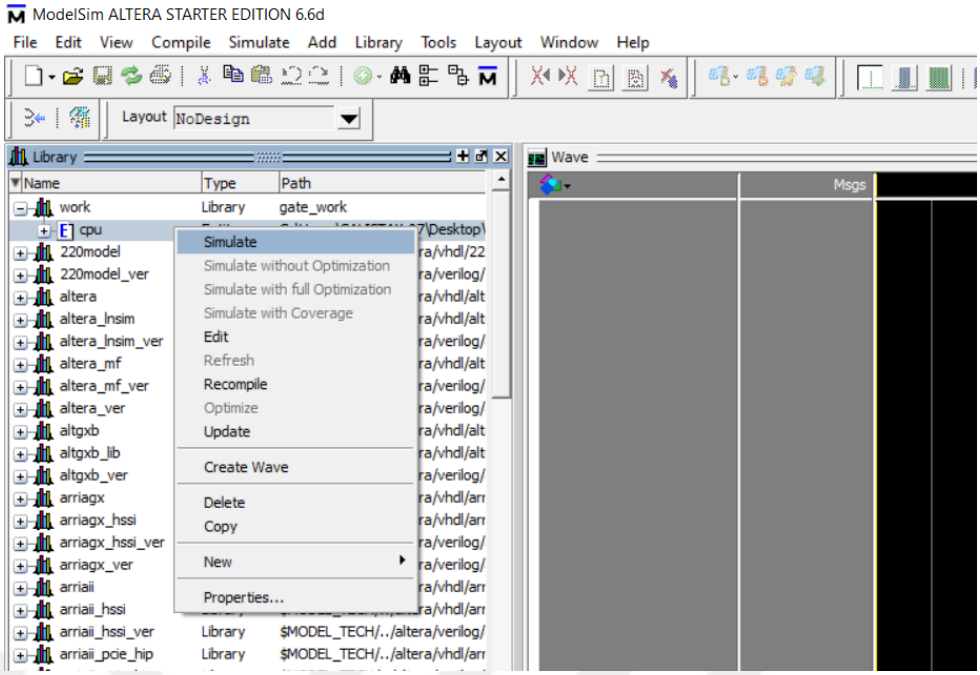
Tasarlanan mikroişlemcinin derlenmesi için Quartus programı menülerinden Şekil 3.23'de gösterildiği gibi derleyici çalıştırılır. Hata ayıklama ve düzeltmeler yapılır. Sorunlar düzeltildikten sonra derleme işlemi başarılı bir şekilde gerçekleştirilmiştir.



Şekil 3.23. Tasarımın derlenmesi

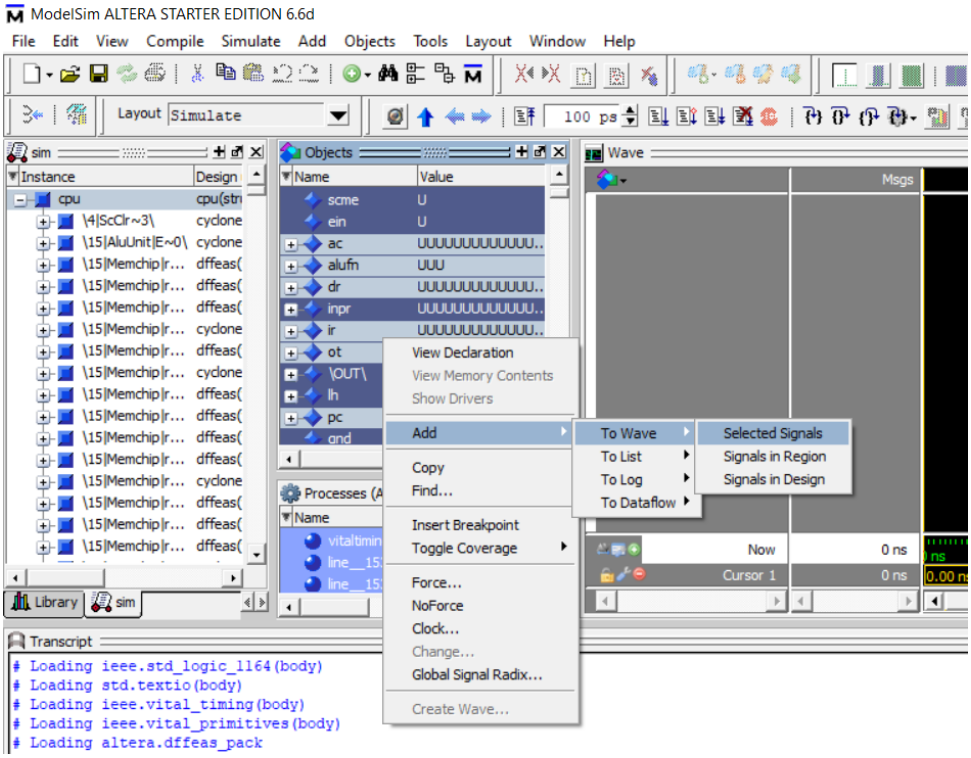
3.3. Tasarımın Simülasyon Adımları

Tasarımların doğruluğunu test etmek için Modelsim-Altera uygulaması kullanılmıştır. Uygulama başarılı bir şekilde derlendikten sonra Modelsim-Altera açılacaktır. Şekil 3.24.'de ki gibi ekrana geldikten sonra simule edip sinyallerin ilk olarak eklenmesi sağlanmıştır.



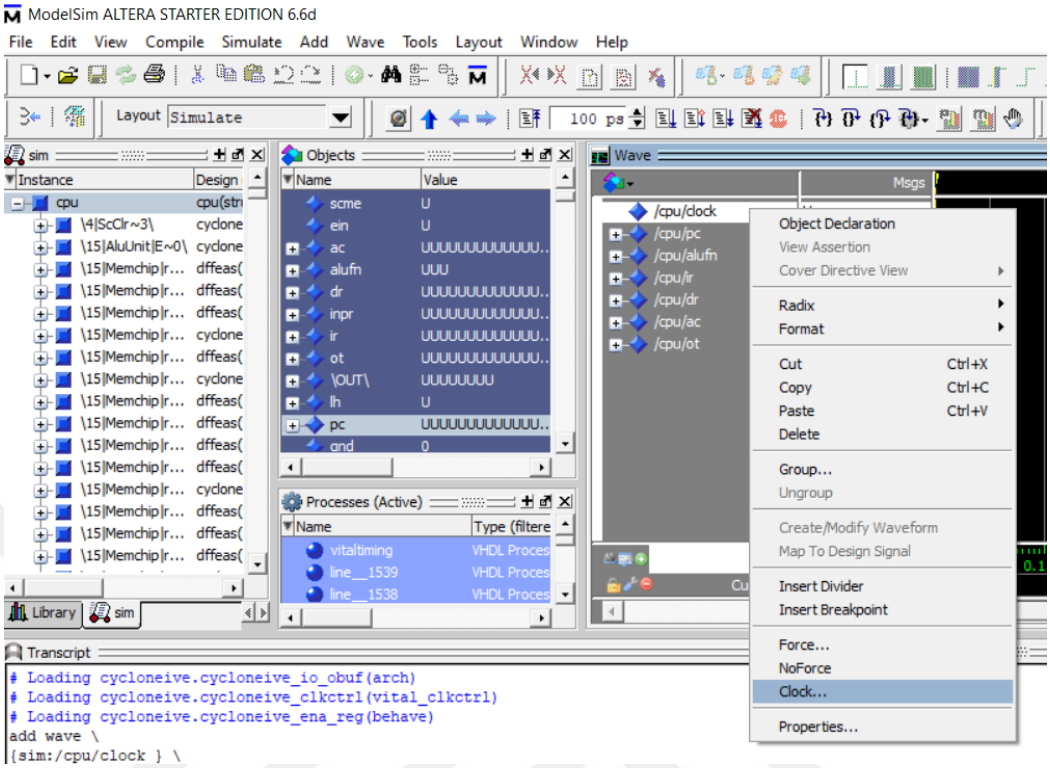
Şekil 3.24. Modelsim simülasyon uygulaması

Ardından eklenen sinyalleri gözlemleyebilmek için wave ekranına Şekil 3.25.'de görüldüğü gibi eklenmiştir.



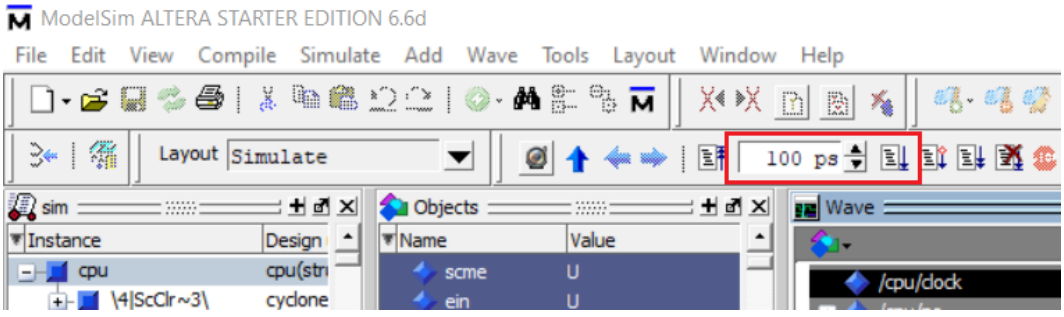
Şekil 3.25. Modelsim sinyallerin wave ekranına eklenmesi

Saat (clock) gibi gerekli olan sinyal değer atamaları Şekil 3.26.'daki gibi yapılmıştır.



Şekil 3.26. Modelsim sinyal değer atama

Daha sonra, simülasyonu başlatmak için Şekil 3.27.'deki gibi kırmızı çerçeve içerisi zaman değeri ayarlandı ve sağ tarafındaki run simgesine basarak çalıştırıldı.



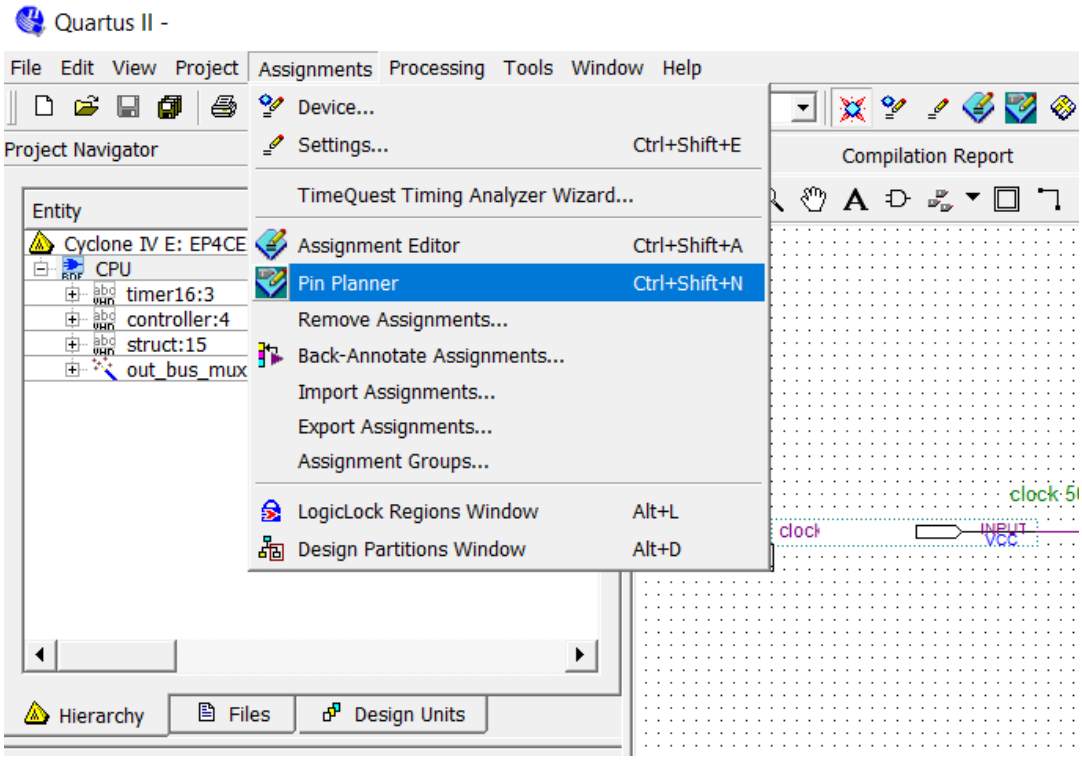
Şekil 3.27. Modelsim simülasyon zaman ayarı ve çalıştırma

3.4. Tasarımın FPGA Kartına Gömülmesi

Tasarlanan 16 bitlik mikroişlemcinin çıkış portu 16 bitliktir. Fakat çalışmada kullanılan FPGA kartında çıkış olarak 8 adet ledi vardır. Bu yüzden tasarımda çıkış portunu 15-8 bitleri, 7-0 bitleri gösterecek şekilde out mux tasarlanmıştır. Tasarım kodlarına, Bkz. EK 6. OUT BUS MUX Yapısının VHDL Kodları yer almaktadır. İlk olarak alçak seviye bitler (7-0) gösterilir, Yüksek seviye bitleri (15-8) göstermek için ise FPGA kartının giriş butonlarından biri tanımlanmıştır. Her iki seviye çıkışlar izlenebilecek şekilde ayarlanmıştır.

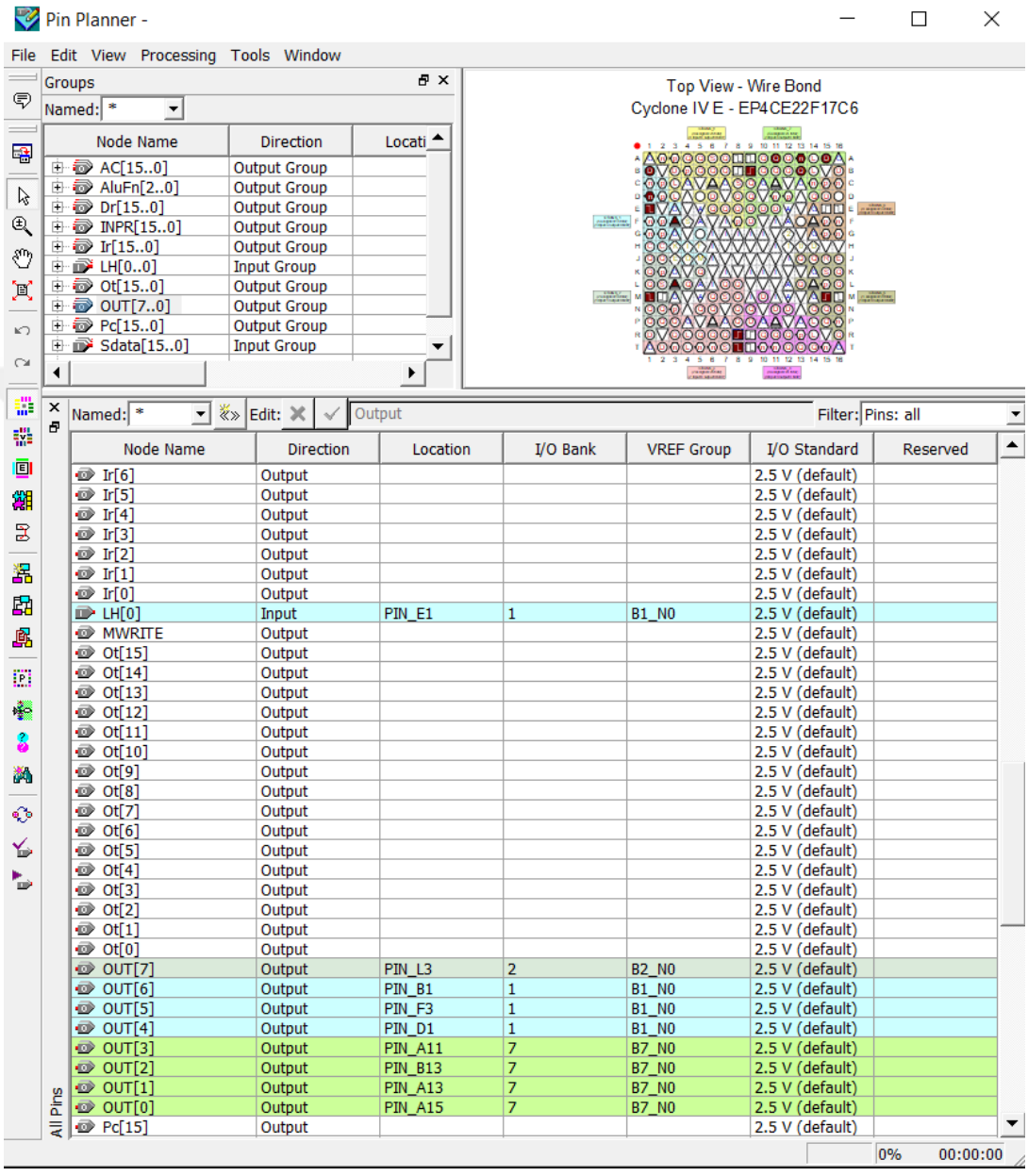
3.4.1. Pin Atamaları

Tasarlanan mikroişlemcinin I/O portlarını, FPGA kartının I/O bağlantı noktalarına Quartus pin planner kullanılarak eşleştirmeleri yapılmıştır. Pin planlayıcıya Şekil 3.28.'de gösterildiği gibi ilgili menüden girilmiştir.



Şekil 3.28. I/O Portlarının pin planlayıcı menüsü

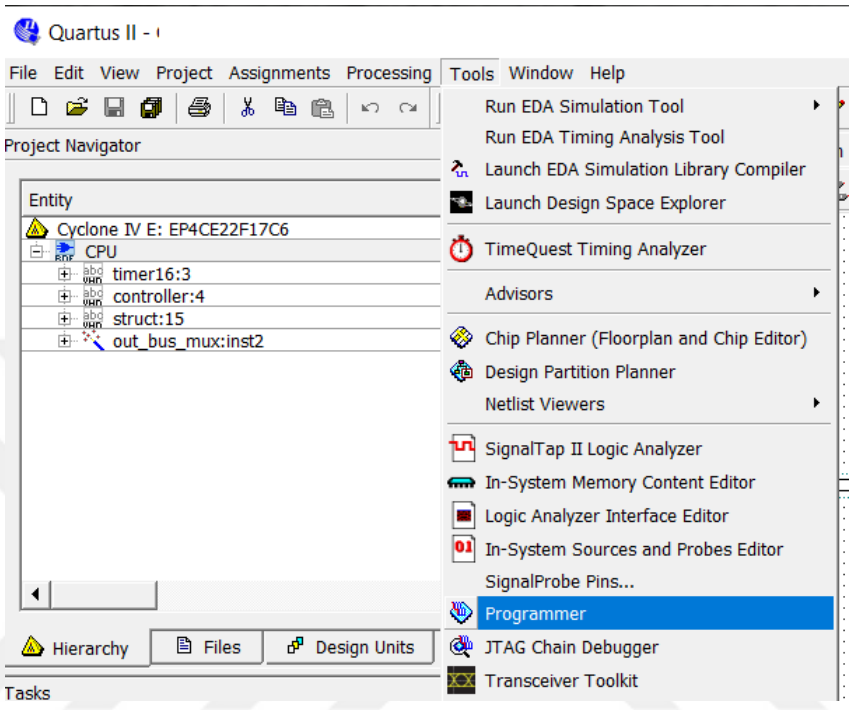
Pin planlayıcı ekranı üzerinden tasarlanan mikroişlemcinin I/O portları ile FPGA kartının I/O pin bağlantılarının eşleştirmesi Şekil 3.29.'da görüldüğü gibi yapılmıştır. FPGA kartı I/O pin şemasına üretici kılavuzunda yer almaktadır [39].



Şekil 3.29. Pin planlayıcı

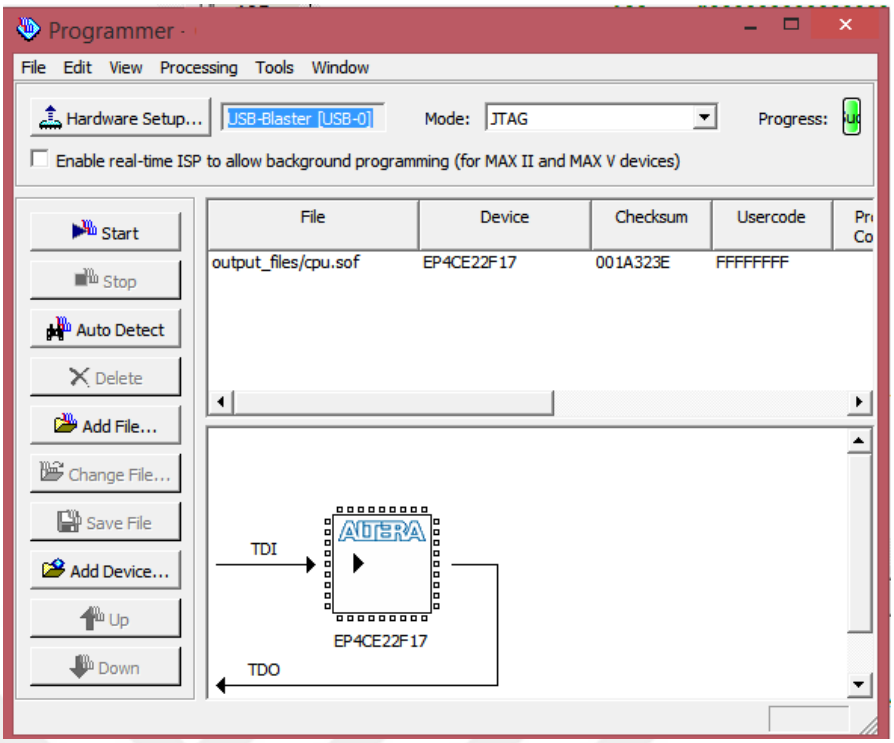
3.4.2. Tasarımın FPGA Karta Yüklenmesi

Tasarlanan mikroişlemcinin FPGA kartına yüklenmesi için Şekil 3.30.'da gösterildiği gibi ilgili menüden yapılır.



Şekil 3.30. Programlayıcı menüsü

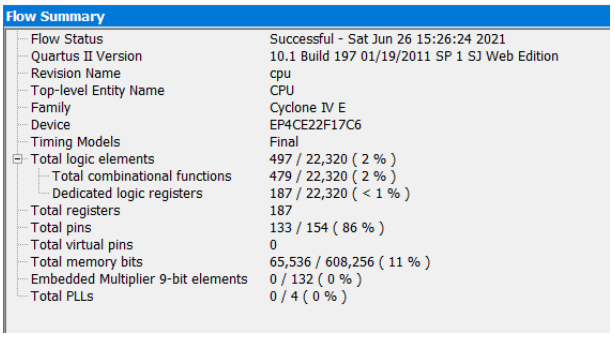
Şekil 3.31.'de görüldüğü gibi programlayıcı ekranına geçiş yapılmıştır. Bu ekrandan derlenen proje ilgili bağlantı yoluyla "Başlat" butonuna basılarak FPGA kartına gömülmektedir.



Şekil 3.31. Programlayıcı ekranı

3.5. Tasarımın FPGA Kartında Kullanmış Olduğu Kaynaklar

Tasarlanan 16 bitlik mikroişlemci Intel (Altera) Cyclone IV FPGA kartı üzerinde, toplam mantık devrelerinden 22320 elementten sadece %2 civarında yani 497 adeti kullanılmıştır. 187 adet register bloğu kullanılmıştır. 154 pin sayısından 133 adet pin kullanılmıştır. FPGA üzerinde bulunan 608256 bitlik ayrılmış RAM bloğundan 65536 (4096x16bit) kısmı kullanılmıştır. Şekil 3.32'de kullanılan kaynakların raporu yer almaktadır.



Şekil 3.32. Mikroişlemcinin FPGA kartı üzerinde kullandığı kaynaklar

4. BULGULAR

Tasarlanan 16 bitlik mikroişlemci için 4096 x 16-bitlik RAM tanımlıdır. Bu RAM içerisinde hem veri hem de program kodları bulunmaktadır. Modelsim-Altera Starter Edition (ücretsiz başlangıç sürümü) simülasyon programı maksimum 8x16-bit bellek desteğini çalıştırdığı için, bilgisayar ortamında simülasyon için RAM tasarımı 8x16-bit olacak şekilde geçici düzeltilmektedir. Program kısıtı sebebiyle program satırı sayısı maksimum 8 adettir. Bilgisayar ortamında bu şekilde geçici düzenleme ile mikroişlemcinin kodları test edilip simülasyonu yapılmıştır. Fakat FPGA üzerinde çalıştırılacağı zaman her hangi bir donanımsal ya da yazılımsal kısıt yoktur. Bu yüzden FPGA üzerinde çalışılacağı zaman 4096x16-bit olarak ayarlanmaktadır.

Bu bölümde tasarlanan 16 bitlik mikroişlemcinin kodları örnek programlar yazılarak doğrulaması test edilmiştir. Programlar RAM'e aşağıdaki formatta yazılmaktadır. Bellek adresi 0-4096 satır içerir. Her satır 16 bitlik binary kodu içerir. Binary kod içerisinde komut ve veri duruma göre işlenir. Program satırının karşısına "--" işaretinden sonra komut sembolik kodu ve HEX karşılığı açıklama olarak yazılmıştır. Programa her hangi bir etkisi bulunmamaktadır.

Bellek Adresi => "Binary Kodu,

Örnek: 0 => "0010000000000101",
1 => "0000000000000100",
.
.

Simülasyon ekranında mikroişlemcinin sinyalleri eklenmiş ve sonuçları gözlenmiştir. Çizelge 4.1.'de kullanılan sinyaller ve açıklamaları yer almaktadır.

Çizelge 4.1. Sinyaller ve açıklamaları

Sinyal	Bit	Açıklama
s	1	Start / Stop sinyalidir. 1 olduğu zaman mikroişlemci durur. 1 değerinin aldığı zaman mikroişlemcinin çalışma süresi hesaplamış olur.
sdata	16	Dışarıdan 16 bitlik veri giriş sinyalidir. Input komutuyla değer okunur. Çalışmada FPGA kartı üzerindeki switch ile sadece 3-0. Bitleri tanımlıdır.
clock	1	Clock girişi sinyalidir.
Alufn	3	ALU işlem seçim durumunu gösterir.
pc	16	Program Counter, işlem adımı
ir	16	Instruction register, işlem opcode gösterir.
dr	16	Data register, veri kaydedicidir.
ac	12	Accumulator
ot	16	Çıkış portu
mwrite	1	Bellek yazma okuma sinyali

4.1. Program 1 –AND İşlemi

4.1.1. Programın Tanımı

A ve B bellek adresinde tanımlı iki veriyi AND işlemine tabi tutup sonucu çıkış portuna gönder.

4.1.2. Örnek Program Kodları

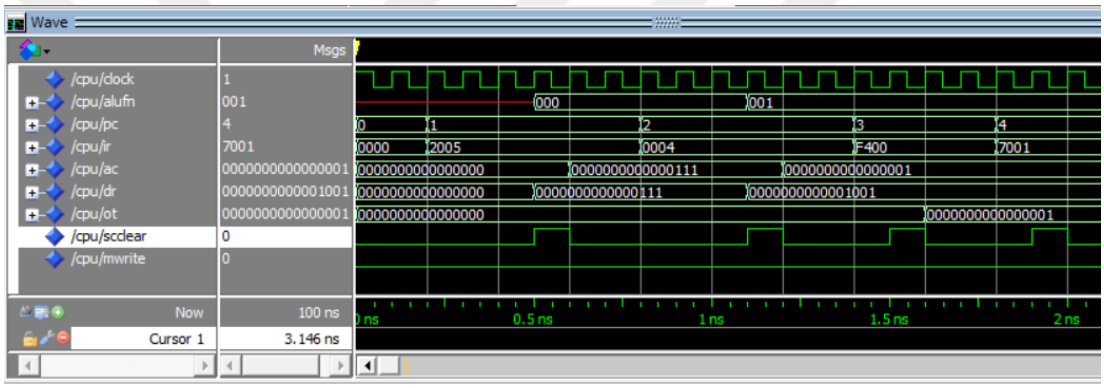
```
0 => "0010000000000101",-- LDA  A  (2005)
1 => "0000000000000100",-- AND  B  (0004)
2 => "1111010000000000",-- OUT   (F400)
3 => "0111000000000001",-- HLT    (7001)
```

4 => "0000000000001001", -- A: (DEC 9)

5 => "0000000000000111", -- B: (DEC 7)

4.1.3. Simülasyon Sonucu

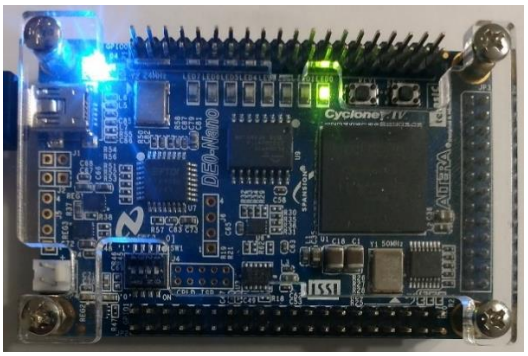
Programın Modelsim-Altera simülasyon sonuçları Şekil 4.1.'de görülmektedir. Burada bellekte yazılan program kod satırları ir (instruction register) tarafından adım adım işletildiği, dr (data register) ve ac (accumulator) ve sonucun ot (output register) çıkışına doğru bir şekilde aktarıdığı anlaşılmaktadır. Sonuç olarak bellekte yer alan A ve B sayılarının AND işlemi sonucu 1 dir.



Şekil 4.1. Program 1 simülasyon sonucu

4.1.4. Gerçekleme Sonucu

Şekil 4.2.'de de görüldüğü gibi simülasyonu yapılan program 1'in FPGA üzerinde 0.ledi aktif sonucu doğru olarak hesaplamıştır.



Şekil 4.2. Program 1 gerçekleme sonucu

4.2. Program 2 – ADD İşlemi

4.2.1. Problemin Tanımı

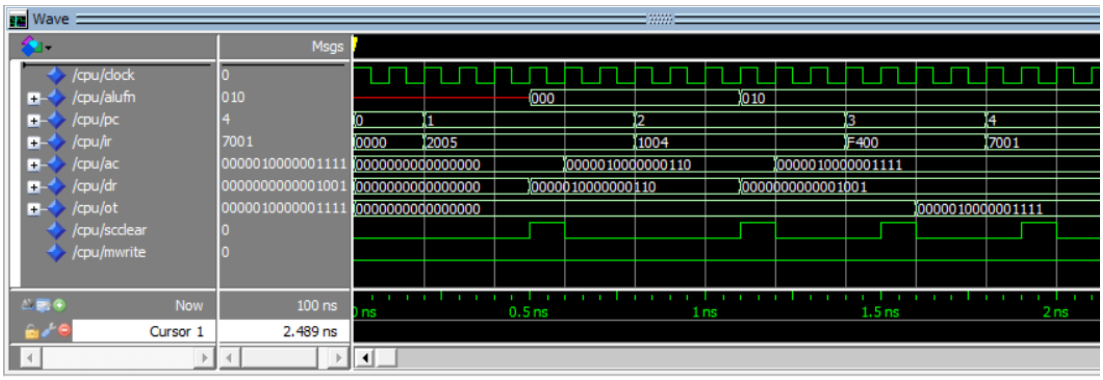
A ve B bellek adreslerindeki iki veriyi topla ve sonucu çıkış portuna gönder.

4.2.2. Örnek Program Kodları

```
0 => "0010000000000101",-- LDA (2005)
1 => "0001000000000100",-- ADD (1004)
2 => "1111010000000000",-- OUT (F400)
3 => "0111000000000001",-- HLT (7001)
4 => "0000000000001001", --  A
5 => "0000010000000110", --  B
```

4.2.3. Simülasyon Sonucu

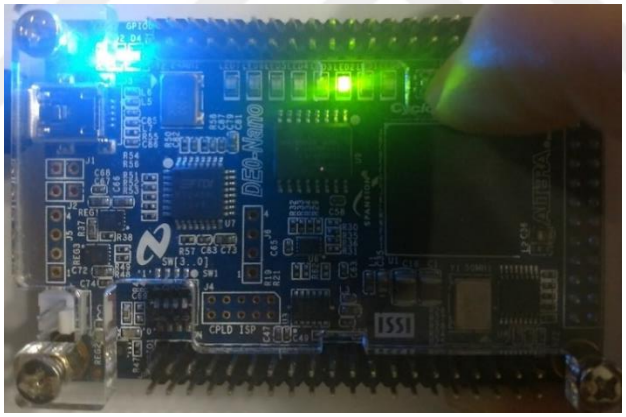
A ve B bellek adreslerindeki iki sayının toplamı doğru hesaplanmıştır. Şekil 4.3.'de görülmektedir.



Şekil 4.3. Program 2 simülasyon sonucu

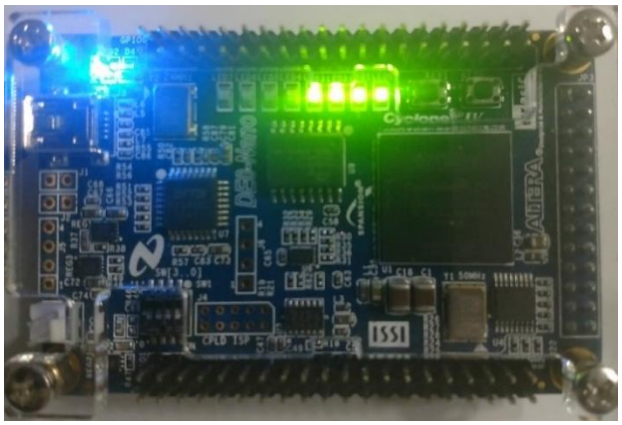
4.2.4. Gerçekleme Sonucu

Bu programın sonucuna göre FPGA kartı üzerinde 15-8 bitlerin çıkışını görmek için butona basılmıştır. Şekil 4.4.'de gösterilmiştir.



Şekil 4.4. Program 2 gerçekleme sonucu 15-8.bitler

Ardından buton bırakılmış ve 7-0 bitler de görüntülenmiştir. Şekil 4.5.'de gösterilmiştir.



Şekil 4.5. Program 2 gerçekleme sonucu 7-0.bitler

4.3. Program 3 – Negatif Sayı ile ADD İşlemi

4.3.1. Programın Tanımı

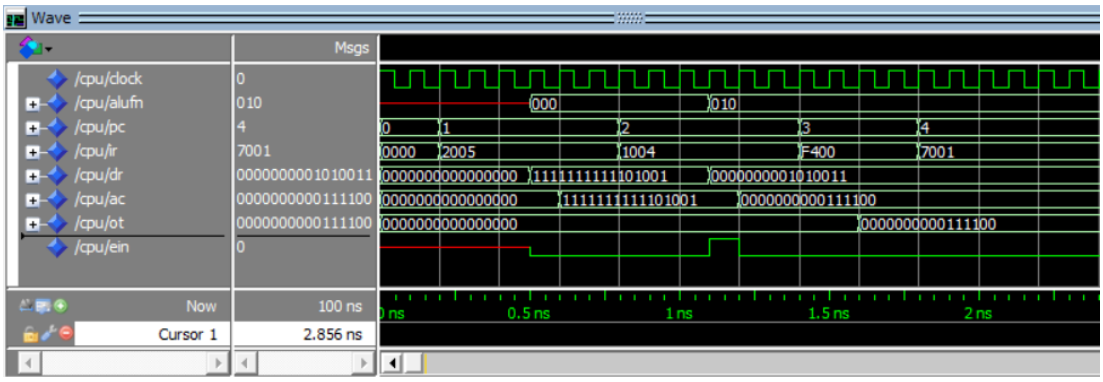
POZ ve NEG bellek adreslerinde bulunan iki veriyi topla ve sonucu yazan program. NEG sayısı işaretli negatif bir sayıdır.

4.3.2. Programın Kodları

```
0 => "0010000000000101",-- LDA NEG     (2005)
1 => "0001000000000100",-- ADD POZ     (1004)
2 => "1111010000000000",-- OUT   (F400)
3 => "0111000000000001",-- HLT   (7001)
4 => "0000000001010011", -- POZ (DEC 83)
5 => "1111111111101001", -- NEG (DEC -23)
```

4.3.3. Simülasyon Sonucu

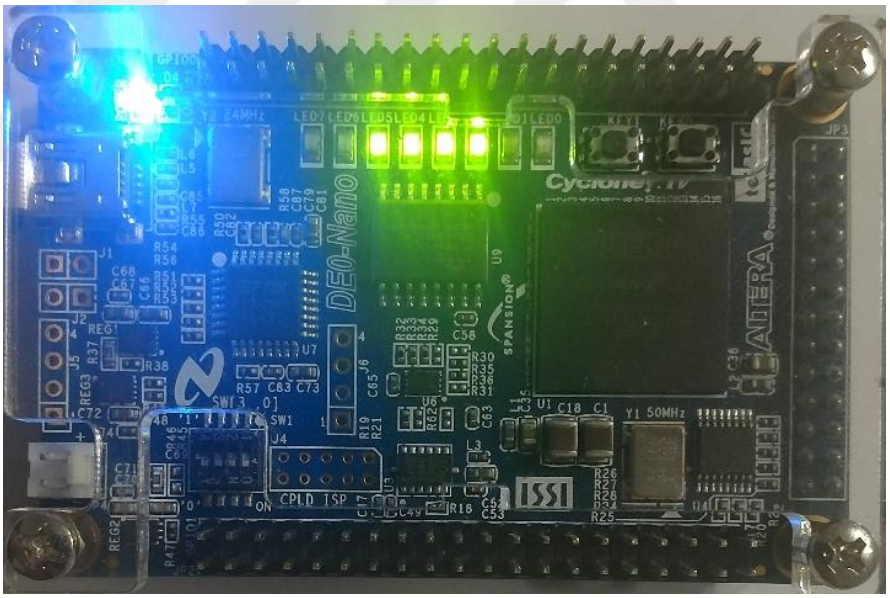
Toplam decimal olarak 60 bulunmuş ve Şekil 4.6.'da görüldüğü gibi doğru sonuç elde edilmiştir.



Şekil 4.6. Program 3 simülasyon sonucu

4.3.4. Gerçekleme Sonucu

Şekil 4.7.'de de görüldüğü gibi hesaplama başarılı bir şekilde gerçekleştirilmiştir.



Şekil 4.7. Program 3 gerçekleme sonucu

4.4. Program 4 – Dolaylı Adresleme ile ADD İşlemi

4.4.1. Programın Tanımı

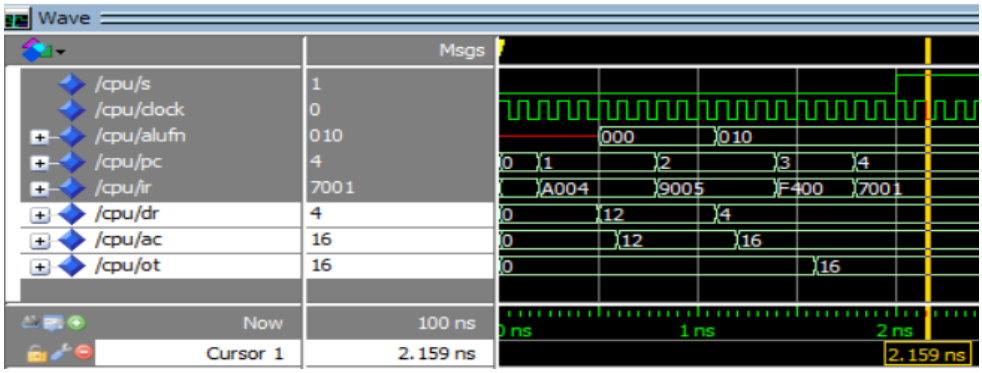
X ve Y bellek adreslerinde tanımlı dolaylı adresleme içeren iki sayıyı toplayan ve sonucu gösteren program.

4.4.2. Programın Kodları

```
0 => "1010000000000100",-- LDA X (A004) indirect adres
1 => "1001000000000101",-- ADD Y (9005) indirect adres
2 => "1111010000000000",-- OUT (F400)
3 => "0111000000000001",-- HLT (7001)
4 => "0000000000000110",--  X    $6
5 => "0000000000000111",--  Y    $7
6 => "0000000000001100",--  (DEC 12)
7 => "0000000000000100",--  (DEC 4)
```

4.4.3. Simülasyon Sonucu

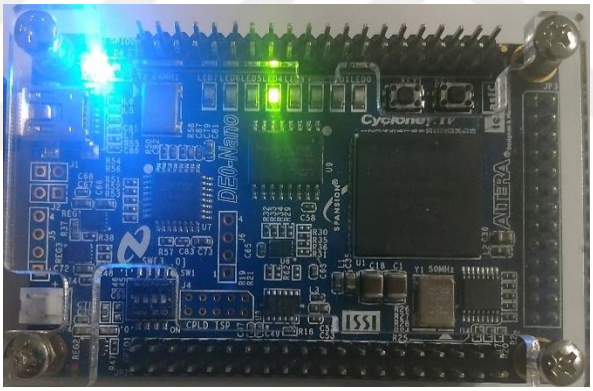
LDA'nın dolaylı adres komut kodu ile X ile tanımlı bellek adresindeki veri dolaylı adresi olmuş ve akümülatöre decimal olarak 12 sayısı yüklenmiştir. Aynı şekilde ADD'in dolaylı adres komut kodu ile Y ile tanımlı bellek adresindeki 4 verisi dolaylı adresi olmuş ve akümülatördeki 12 sayısıyla toplanmıştır. Şekil 4.8.'de toplam sonuç 16 olarak başarılı bir şekilde doğrulanmıştır. Programın çalışma süresi "s (start/stop)" sinyalinin durumuna bakılarak anlaşılabilir. Başlama 0, bitiş 1 olarak tanımlıdır. Bu programın ortalama çalışma süresi 2 ns dir.



Şekil 4.8. Program 4 simulasyon sonucu

4.4.4. Gerçekleme Sonucu

Şekil 4.9.'da da hesaplama başarılı bir şekilde gerçeklenmiştir.



Şekil 4.9. Program 4 gerçekleme sonucu

4.5. Program 5 – Kaydırma (CIR, CIL) ve Elde Silme (CLE) İşlemi

4.5.1. Programın Tanımı

6. bellek adresindeki veriyi ilk önce sağa kaydıran sonra eldeyi temizleyen ardından sola kaydıran program.

4.5.2. Programın Kodları

0 => "0010000000000110",-- LDA $6 (2006)

1 => "0111000010000000",-- CIR (7080)

2 => "0111010000000000",-- CLE (7400)

3 => "0111000001000000",-- CIL (7040)

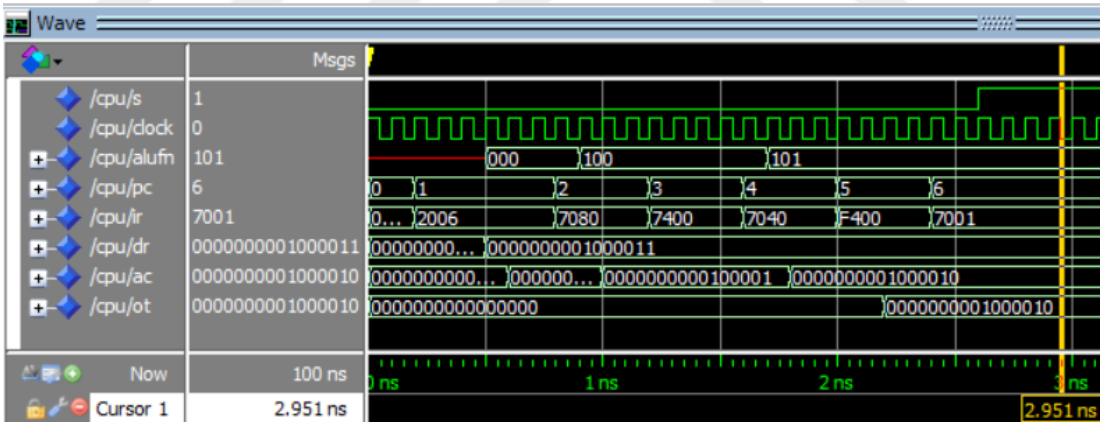
4 => "1111010000000000",-- OUT (F400)

5 => "0111000000000001",-- HLT (7001)

6 => "0000000001000011",

4.5.3. Simülasyon Sonucu

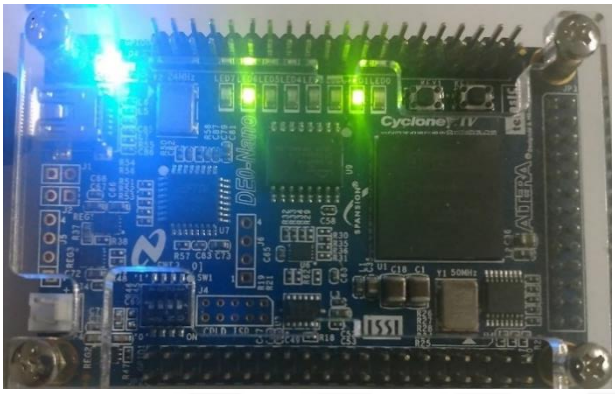
Şekil 4.10.'da görüldüğü gibi sağa kaydırma, elde silme ve sola kaydırma işlemi sonucu başarılı bir şekilde doğrulanmıştır. Bu programın ortalama çalışma süresi 2,6 ns dir.



Şekil 4.10. Program 5 simülasyon sonucu

4.5.4. Gerçekleme Sonucu

Programın gerçekleme sonucu Şekil 4.11.'de gösterilmiştir.



Şekil 4.11. Program 5 gerçekleme sonucu

4.6. Program 6 – Sayının Tersini (CMA) Alma ve INC İşlemi

4.6.1. Programın Tanımı

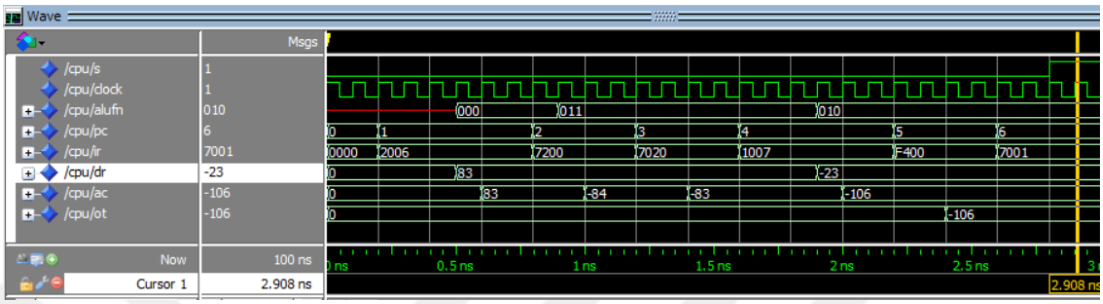
6. bellek adresindeki verinin negatifini bulan ve 7. bellek adresindeki veri ile toplayan program.

4.6.2. Programın Kodları

```
0 => "0010000000000110",-- LDA A  (2006)
1 => "0111001000000000",-- CMA     (7200)
2 => "0111000000100000",-- INC      (7020)
3 => "0001000000000111",-- ADD B  (1007)
4 => "1111010000000000",-- OUT      (F400)
5 => "0111000000000001",-- HLT      (7001)
6 => "0000000001010011",-- A  83
7 => "1111111111101001",-- B  -23
```

4.6.3. Simülasyon Sonucu

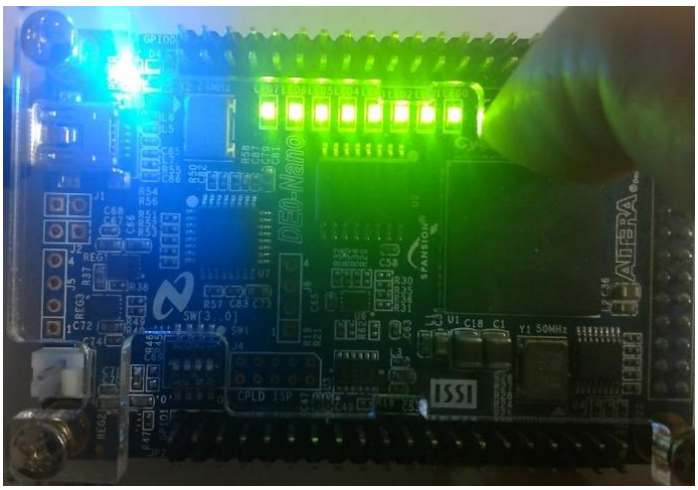
6. bellek adresindeki decimal 83 verisinin tersi alınmış ve değeri bir artırılarak negatif değeri elde edilmiştir. Her iki negatif sayının toplamı da Şekil 4.12'de görüldüğü gibi başarılı bir şekilde doğrulanmıştır. Çalışma süresi 2,8 ns dir.



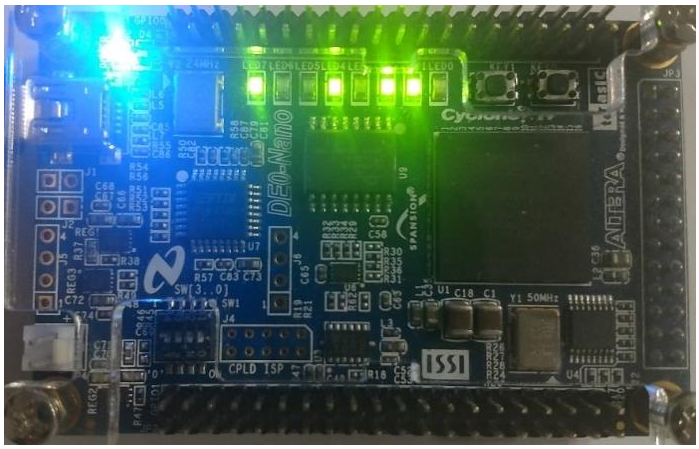
Şekil 4.12. Program 6 simülasyon sonucu

4.6.4. Gerçekleme Sonucu

Elde edilen decimal -106 değerinin binary karşılığı "1111111110010110" dır. Şekil 4.13'de 15-8. bitleri, 7-0. bitleri ise Şekil 4.14'de gösterilmiştir.



Şekil 4.13. Program 6 gerçekleme sonucu 15-8. bitler



Şekil 4.14. Program 6 gerçekleme sonucu 7-0. Bitler

4.7. Program 7 – Döngü Oluşturma (SZI) ve Dallanma (BUN) İşlemi

4.7.1. Programın Tanımı

Bellek organizasyonunda 100. Bellek adresinden başlamak üzere, 16 ayrı bellek adresinde yer alan sayıların toplamını bulan program.

4.7.2. Programın Kodları

```
100 => "0010000001110001",-- LDA DATA
101 => "0011000001110010",-- STA PTR
102 => "0010000001110011",-- LDA NUM_S
103 => "0011000001110100",-- STA COUNT
104 => "0111100000000000",-- CLA
105 => "1001000001110010",-- ADD PTR I / LOOP
106 => "0110000001110010",-- ISZ PTR
107 => "0110000001110100",-- ISZ COUNT
108 => "0100000001101001",-- BUN LOOP
109 => "0011000010000101",-- STA SUM
110 => "0010000010000101",-- LDA SUM
111 => "1111010000000000",-- OUT (F400)
```

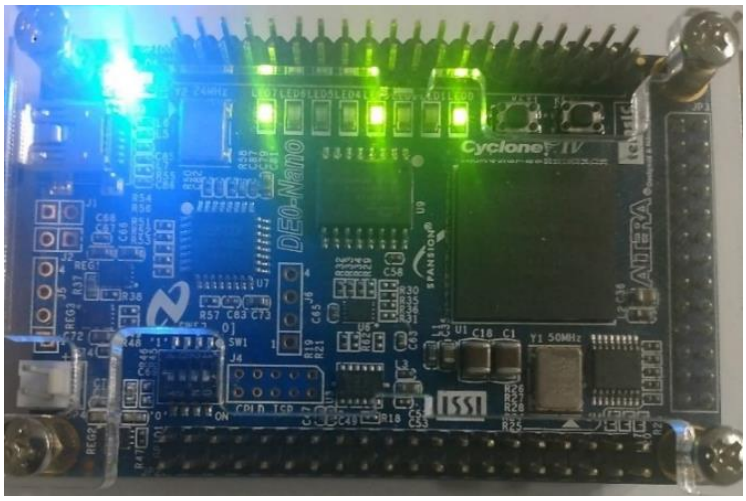
```
112 => "0111000000000001",-- HLT
113 => "0000000001110101",-- DATA
114 => "0000000000000000",-- PTR
115 => "1111111111110000",-- NUM_S
116 => "0000000000000000",-- COUNT
117 => "0000000000000001",-- 1 FIRST
118 => "0000000000000010",-- 2
119 => "0000000000000011",-- 3
120 => "0000000000000100",-- 4
121 => "0000000000000101",-- 5
122 => "0000000000000110",-- 6
123 => "0000000000000111",-- 7
124 => "0000000000001000",-- 8
125 => "0000000000001001",-- 9
126 => "0000000000001010",-- 10
127 => "0000000000001011",-- 11
128 => "0000000000001100",-- 12
129 => "0000000000001101",-- 13
130 => "0000000000001110",-- 14
131 => "0000000000001111",-- 15
132 => "0000000000010000",-- 16
133 => "0000000000000000",-- SUM
```

4.7.3. Simülasyon Sonucu

Modelsim-Altera Starter Edition bellek organizyonu kısıtı sebebiyle simülasyonu gösterilememiştir.

4.7.4. Gerçekleme Sonucu

FPGA üzerinde mikroişlemcinin program yanıtına göre işlemlerin başarılı bir şekilde gerçekleştiği Şekil 4.15.'de görülmektedir.



Şekil 4.15. Program 7 gerçekleme sonucu

4.8. Program 8 – Dallanma (SPA, SNA, SZA) İşlemi

4.8.1. Programın Tanımı

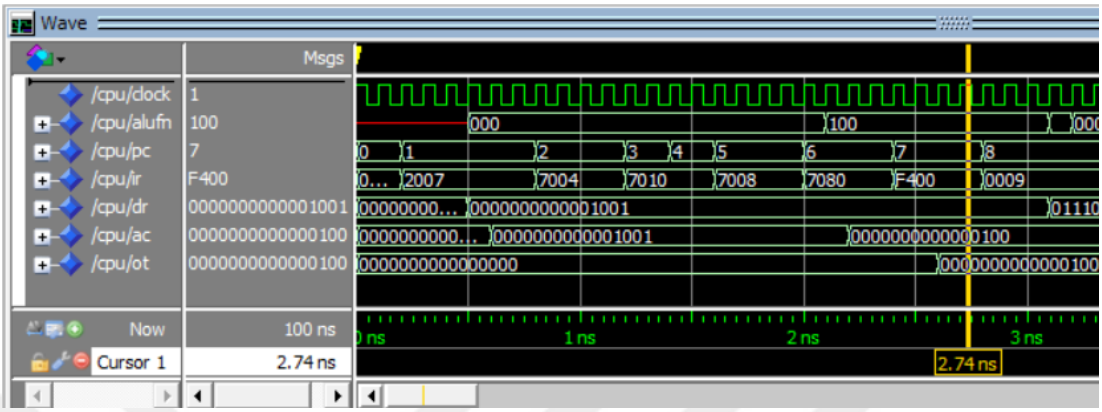
7. bellek adresindeki verinin negatif, pozitif veya sıfır durumuna göre işlem yapan program.

4.8.2. Programın Kodları

0 => "0010000000000111",-- LDA (2007)
1 => "0111000000000100",-- SZA (7004) Sifir ise PC+1
2 => "0111000000010000",-- SPA (7010) Pozitif ise PC+1
3 => "0111000000100000",-- INC (7020)
4 => "0111000000001000",-- SNA (7008) negatif ise PC+1
5 => "0111000010000000",-- CIR (7080)
6 => "1111010000000000",-- OUT (F400)
7 => "0000000000001001", --

4.8.3. Simülasyon Sonucu

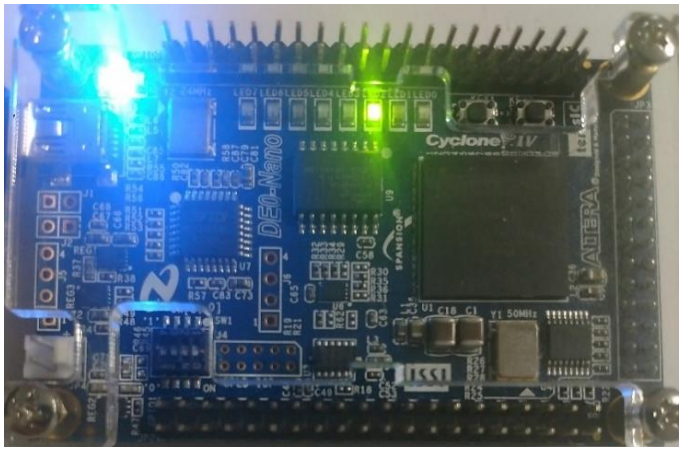
Şekil 4.16.'da programın simülasyonu başarılı bir şekilde doğrulanmıştır.



Şekil 4.16. Program 8 simülasyon sonucu

4.8.4. Gerçekleme Sonucu

Programın gerçeklenmesi Şekil 4.17.'de görülmektedir.



Şekil 4.17. Program 8 gerçekleme sonucu

4.9. Program 9 – Veri Girişi (INP) ve Kesme Kontrolü (SKI, SKO) İşlemi

4.9.1. Programın Tanımı

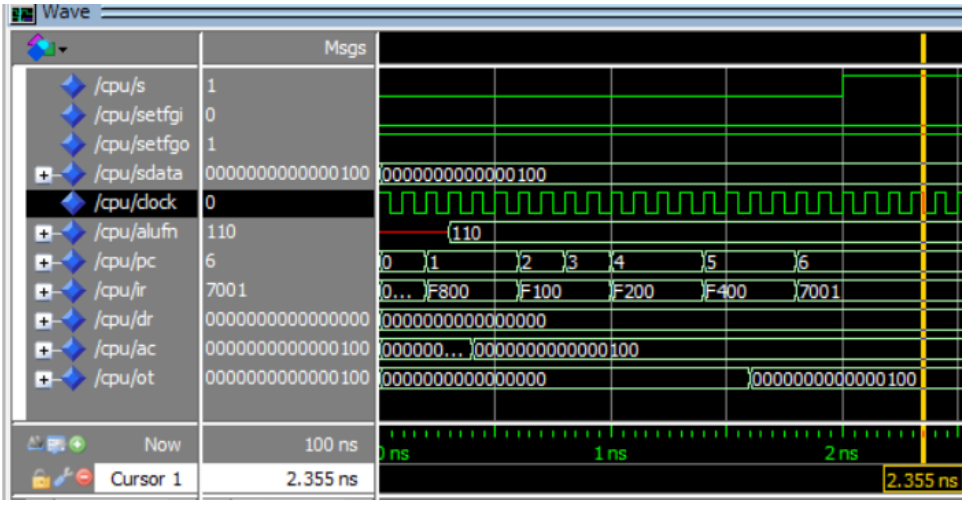
Mikroişlemcinin sdata veri girişine decimal 4 değeri, setFGI girişine 0 ve setFGO girişine 1 uygulanarak, giriş kesme kontrollü olarak sdata değerini çıkışa aktaran program.

4.9.2. Programın Kodları

```
0 => "1111100000000000",-- INP (F800)
1 => "1111000100000000",-- SK0 (F100) FG0=1? PC=PC+1
2 => "0111000000000001",-- HLT (7001)
3 => "1111000100000000",-- SKI (F200) FGI=1? PC=PC+1
4 => "1111010000000000",-- OUT (F400)
5 => "0111000000000001",-- HLT (7001)
```

4.9.3. Simülasyon Sonucu

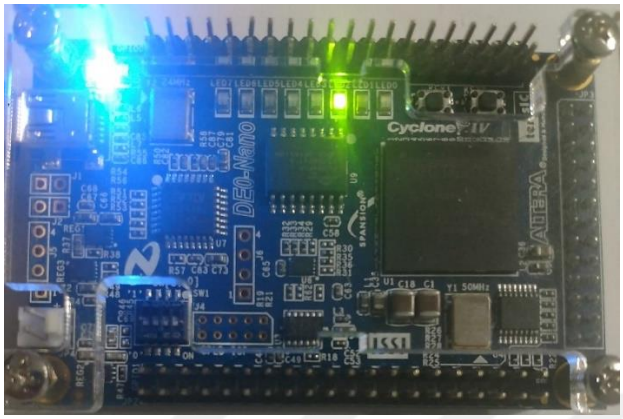
Giriş değerlerine göre kesme sinyali durumlarına göre programın simülasyon sonucu Şekil 4.18.'de doğrulanmıştır. Çalışma süresi 2 ns dir.



Şekil 4.18. Program 9 simülasyon sonucu

4.9.4. Gerçekleme Sonucu

Şekil 4.19'da gerçekleme sonucu başarılı bir şekilde gösterilmiştir.



Şekil 4.19. Program 9 gerçekleme sonucu

4.10. Program 10 – Eldenin Durumuna Göre Dallanma (SZE) İşlemi

4.10.1. Programın Tanımı

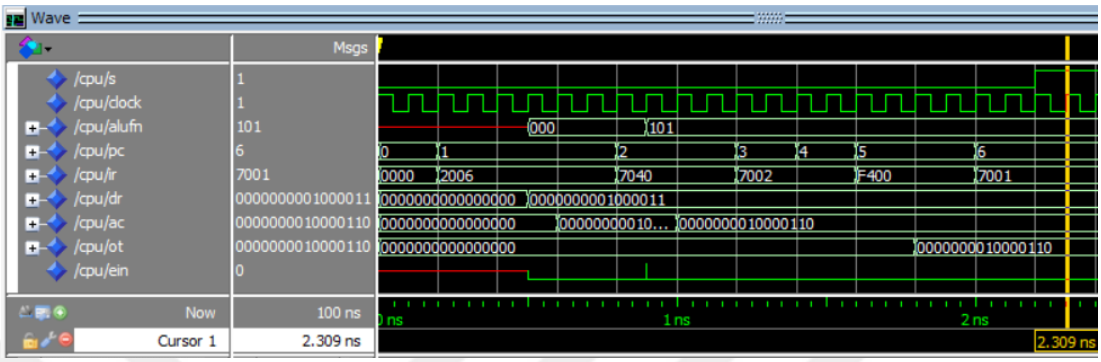
6. bellek adresinde yer alan verinin sola kaydırma sonucunda oluşan elde bitinin durumuna göre işlem yaparak çıkış portuna değeri gönderen program.

4.10.2. Programın Kodları

```
0 => "0010000000000110",-- LDA (2006)
1 => "0111000001000000",-- CIL (7040)
2 => "0111000000000010",-- SZE (7002) -- e=0? PC+1
3 => "0111000010000000",-- CIR (7080)
4 => "1111010000000000",-- OUT (F400)
5 => "0111000000000001",-- HLT
6 => "0000000001000011",
```

4.10.3. Simülasyon Sonucu

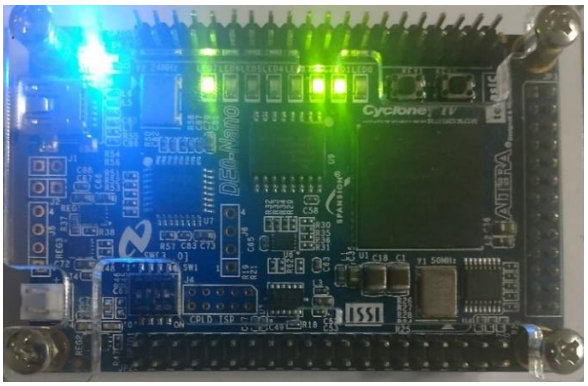
Programın simülasyonu Şekil 4.20.'de görüldüğü gibi başarılı bir şekilde doğrulanmıştır. Çalışma süresi 2,2 ns dir.



Şekil 4.20. Program 10 simülasyon sonucu

4.10.4. Gerçekleme Sonucu

Şekil 4.21.'de FPGA üzerinde mikroişlemcinin program yanıtı başarılı olarak gerçeklenmiştir.



Şekil 4.21. Program 10 gerçekleme sonucu

4.11. Program 11- Çarpma Operatörü Kullanmadan Çarpma İşlemi

4.11.1. Programın Tanımı

13. bellek adresinde yer alan veri ile 14. Bellek adresinde yer alan veriyi çarpma operatörü kulanmadan toplama işlemi yaparak sonucu çıkış portuna gönderen program.

4.11.2. Programın Kodları

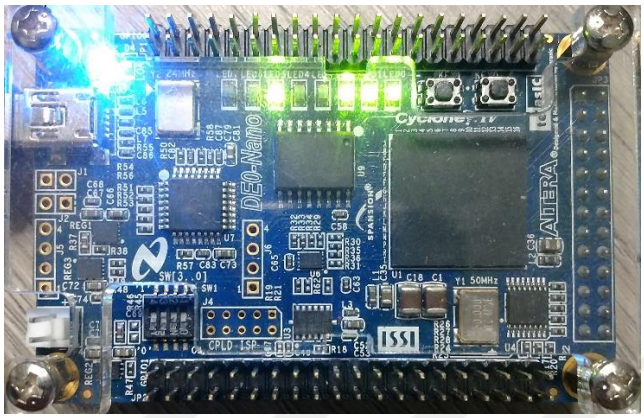
```
0 => "0010000000001101",-- LDA A        (200D)
1 => "0111000000000100",-- SZA          (7004) ac=0? pc+1
2 => "0100000000000100",-- BUN NZR      (4004)
3 => "0100000000001011",-- BUN SON      (400B)
4 => "0111001000000000",-- CMA          (7200) -- NZR
5 => "0111000000100000",-- INC          (7020)
6 => "0011000000001111",-- STA CTR      (300F)
7 => "0111100000000000",-- CLA          (7800)
8 => "0001000000001110",-- ADD B        (100E) -- LOP
9=>  "0110000000001111",-- ISZ CTR      (600F)
10 =>"0100000000001000",-- BUN LOP      (4008)
11 =>"1111010000000000",-- OUT          (F400) -- SON
12 =>"0111000000000001",-- HLT          (7001)
13 =>"0000000000000011",-- A: 3
14 =>"0000000000001101",-- B: 13
15 =>"0000000000000000",-- CTR
```

4.11.3. Simulasyon Sonucu

Modelsim-Altera Starter Edition bellek organizyonu kısıtı sebebiyle simülasyonu gösterilememiştir.

4.11.4. Gerçekleme Sonucu

Şekil 4.22.'de FPGA üzerinde mikroişlemcinin program yanıtı A (3) x B (13) = 39 şeklinde beklenildiği gibi başarılı olarak gerçeklenmiştir.



Şekil 4.22. Program 11 gerçekleme sonucu

5. TARTIŞMA

Morris Mano'nun işlemci modelini temel alan "Mano's Micro" [6] çalışmasındaki komut setinde sadece AND, ADD, LDA, STA, BUN, BSA ve ISZ komutlarının yer aldığı, Mano'nun tarif ettiği mimariye göre eksik kalan komut ve yapıların olduğu görülmüştür. Buradan hareketle Morris Mano'nun temel işlemcisinde yer alan eksik komutları tamamlama, hatta üzerinde değişiklikler yaparak işlemcinin daha da geliştirilmesi fikri doğmuştur.

Eksik komutların yapısı ve çalışmasına göre çeşitli sinyaller eklenmiş ve başarılı bir şekilde test edilmiştir. Bazı komutların gerçeklenmesi için de farklı modüllerin gerektiği anlaşılmıştır. Örneğin, eldeli işlemler için geçici hafıza birimi olarak elde Flip-Flop'u geliştirilmiştir.

Bu tez çalışmasında elde edilen bulgular ve kazanımlarla eğitim ortamında basit bir mikroişlemcinin yazılım ile tasarlanabilirliği ve gerçeklenebilirliği gösterilmiştir. Tasarlanan mikroişlemcinin simülasyonunu ve FPGA kartı üzerinde çalışması başarılı bir şekilde gerçekleştirilmiştir.

6. SONUÇ ve ÖNERİLER

Bu tez çalışmasının amacı, Morris Mano'nun 16 bitlik mikroişlemci modelini, eksiksiz bir şekilde FPGA üzerinde VHDL yazılım dili kullanarak tasarlamak ve gerçeklemektir.

Referans alınan "Mano's Micro" çalışması ile bu tez çalışmasının farklılıkları Çizelge 6.1.'de görülmektedir.

Çizelge 6.1. Tez çalışmasının farklılıkları

	Mano's Micro isimli çalışma	Bu Tez Çalışması
Komut Seti	AND, ADD	Tersleme için COM/CMA, sağa kaydırma için CIR, sola kaydırma için CIL, veri girişi için INP, eldenin sıfırlanması için CLE, eldenin tersini almak için CME, dallanma komutları için SPA SNA, SZA, SZE, mikroişlemcide işlemleri durdurmak için HLT, kesme komutları için ise SKI SKO, ION, IOF eklenmiş ve komut seti tamamlanmıştır.
Elde Flip-Flop	YOK	VAR
CPU Başlatma / Durdurma Flip-Flop	YOK	VAR
INPUT ve OUTPUT Kaydedicisi	YOK	VAR
Bellek Mimarisi	Harvard	Von-Neumann

Çalışmanın amacına uygun olarak Morris Mano'nun anlattığı 16 bitlik mikroişlemcisi, eksiksiz bir şekilde VHDL yazılım dilini kullanarak tasarlanmıştır. Tasarımı tamamlanan 16 bitlik temel mikroişlemci, RAM ve I/O çevre birimleriyle birlikte mikrobilgisayar haline getirilmiştir. Simülasyon çalışmaları ile doğruluğu test edilmiş ve FPGA üzerinde de başarılı bir şekilde çalıştırılmıştır.

VHDL dili kullanarak fonksiyonel hale getirilen bu mikroişlemci, adım adım anlatılmış ve geliştirmeye açık hale getirilmiştir. Böylelikle eğitim ortamında teoride kalan mikroişlemciler hakkındaki edinimler somutlaştırılmış ve uygulanabilirliği sağlanmıştır.

Çalışma sırasında birkaç kısıt ile karşılaşılmıştır. Bunlardan biri; Modelsim-Altera Starter Edition Ver. 6.6 uygulamasında tasarımı yapılan 4096x16 bitlik RAM modülünün simule edilememesidir. Bunun yerine RAM kapasitesi 8x16 bitlik modüle çevrilip test edilmiş ve başarılı bir şekilde sonuçlandırılmıştır. Mikroişlemci FPGA karta gömüleceği zaman RAM yeniden 4096x16 bit modüle çevrilmiştir. Bu kısıt FPGA kartı üzerinde yaşanmamıştır. Çünkü FPGA kartı üzerinde 608256 bitlik RAM bloğu desteği vardır.

Diğer bir kısıt; çalışmada kullanılan DE0 Nano FPGA kartının 8 adet led çıkış portunun olmasıdır. Çünkü tasarlanan mikroişlemcinin 16 bitlik çıkışı vardır. Bu duruma çözüm olarak, tasarıma Out Bus Mux modülü eklenmiş ve 15-8.bitler ile 7-0.bitler ayrı ayrı seçilebilir duruma getirilmiştir.

Bir başka kısıt ise; FPGA saat frekans üretecinin maksimum 50 Mhz olmasıdır.
Çalışma eğitim ortamında rehber niteliği taşıması için detaylı olarak anlatılmıştır. Çalışma geliştirilmeye açık olup, kullanılan mikroişlemcinin komut kümesi genişletilebilir, bit sayısı artırılabilir ve aynı mikroişlemciden birden fazla kullanılarak çoklu çekirdek yapısı elde edilebilir. Bellek yapısı daha da genişletilebilir. RAM için assembly dilinde program yazma editörü ayrıca geliştirilebilir.

KAYNAKLAR

[1] E. A. Bezerra ve M. P. Gough, "A guide to migrating from microprocessor to FPGA coping with the support tool limitations", *Microprocess. Microsyst.*, c. 23, sy 10, ss. 561-572, Mar. 2000, doi: 10.1016/S0141-9331(99)00063-0.

[2] N. Sağlam ve F. Kaçar, "Design and Simulation of 64 Bit FPGA Based Arithmetic Logic Unit", *Electrica*, c. 19, sy 2, ss. 158-165, Tem. 2019.

[3] E. Sarıtaş ve S. Karataş, *Her yönüyle FPGA ve VHDL*. sedat karatas, 2013.

[4] E. Ayeh, K. Agbedanu, Y. Morita, O. Adamo, ve P. Guturu, "FPGA Implementation of an 8-bit Simple Processor", içinde *2008 IEEE Region 5 Conference*, Kansas City, MO, USA, Nis. 2008, ss. 1-5. doi: 10.1109/TPSD.2008.4562743.

[5] Sumedh. S. Jadhav ve C. N. Bhoyar, "FPGA Based Embedded Multiprocessor Architecture", *Int. J. Electron. Electical Eng.*, ss. 217-223, Oca. 2013, doi: 10.47893/IJEEE.2013.1042.

[6] N. Narasimhamurthi, "Mano's Micro Tecnical Note".

[7] C. Thimmannagari, *CPU Design: Answers to Frequently Asked Questions*, 1. bs. Springer, 2005.

[8] K. Arnold, *Embedded Controller Hardware Design*. 2000.

[9] R. Cable, *Microprocessor design*. Delhi: Global Media, 2009.

[10] J. Floyd, *Digital_Fundamentals__8th_Edition_Floyd.pdf*. 2009.

[11] R. C. Cofer ve B. F. Harding, *Rapid System Prototyping with FPGAs: Accelerating the Design Process*. Elsevier, 2011.

[12] S. D. Brown, R. J. Francis, J. Rose, ve Z. G. Vranesic, *Field-Programmable Gate Arrays*. Springer Science & Business Media, 1992.

[13] I. Kuon, R. Tessier, ve J. Rose, *FPGA Architecture: Survey and Challenges*. Now Publishers Inc, 2008.

[14] P. Wilson, *Design Recipes for FPGAs: Using Verilog and VHDL*. Newnes, 2015.

[15] E. Ölmez, "FPGA Tabanlı Mikrobilgisayar Mimarisi Kullanılarak DC Motor Sürücü Tasarımı ve Uygulaması", Yüksek Lisans Tezi, Bozok Üniversitesi Fen Bilimleri Enstitüsü, Yozgat, 2012.

[16] “Cyclone® IV FPGA Devices - Intel® FPGA”, *Intel*. https://www.intel.com/content/www/us/en/products/details/fpga/cyclone/iv.html (erişim Haz. 13, 2021).

[17] “Cyclone® IV EP4CGX150 FPGA - Product Specifications”, *Intel*. https://www.intel.com/content/www/us/en/products/sku/210476/cyclone-iv-ep4cgx150-fpga/specifications.html (erişim Haz. 15, 2021).

[18] C. Maxfield, *FPGAs: World Class Designs*. Newnes, 2009.

[19] “Spartan-6 FPGA Embedded Kit”, *Xilinx*, Haz. 15, 2021. https://www.xilinx.com/products/boards-and-kits/dk-s6-embd-g.html (erişim Haz. 15, 2021).

[20] “ISE Design Suite”, *Xilinx*, Haz. 15, 2021. https://www.xilinx.com/products/design-tools/ise-design-suite.html (erişim Haz. 15, 2021).

[21] “Intel Cyclone 10 GX FPGA Development Kit User Guide”. Intel Co., 2018.

[22] “Intel Cyclone 10 GX FPGA Development Kit Web Page”, Haz. 15, 2021. https://www.intel.com/content/www/us/en/programmable/documentation/hvu1509010715799.html (erişim Haz. 15, 2021).

[23] “Download Center for FPGAs”, Haz. 15, 2021. https://fpgasoftware.intel.com/ (erişim Haz. 15, 2021).

[24] E. O. Hwang, *Digital Logic and Microprocessor Design with VHDL*. 2005.

[25] “IEEE Standard for VHDL Language Reference Manual”, IEEE, 2019. doi: 10.1109/IEEESTD.2019.8938196.

[26] B. Cohen, *VHDL - Coding Styles and Methodologies.pdf*. 2002.

[27] H. R. Charles, *Digital Systems Design Using VHDL*. 1998.

[28] M. Bryan ve T. Fabrizio, *Free Range VHDL*. 2012.

[29] D. K. M. Al-Aubidy, R. F. Al-Bader, ve A. A. Smadi, “SIMULATION AND FPGA IMPLEMENTATION OF A SIMPLE COMPUTER”, s. 8, 2005.

[30] E. Alaer, A. Tangel, ve M. Yakut, “‘MIB-16’ FPGA Based Design and Implementation of a 16-Bit Microprocessor for Educational Use”, c. 5, sy 5, s. 5, 2008.

[31] E. Öztürk ve H. Sedef, “FPGA Kullanarak 16 Bitlik Mikroişlemci Tasarımı Designing of a 16 – bit Microprocessor by Using FPGA”, s. 6, 2010.

[32] S. Görgünoğlu, M. Teke, ve M. Peker, “Hardware Design and Simulation of a Microprocessor”, 2010.

[33] D. Taşkın, K. Baysal, ve N. Topçubaşı, "VHDL ile Mikroişlemci Tasarımı ve Eğitimde Uygulanabilirliği", s. 6, 2011.

[34] N. V. Nandanwar ve P. R. Thorat, "8 Bit Instructional Processor using FPGA", c. 4, sy 11, s. 4, 2013.

[35] M. Kumar, S. K. Jha, ve R. Sharma, "A Case Study: Design of 16 bit Arithmetic and Logical unit using Xillinx 14.7 and Implementation on FPGA Board", 2017.

[36] N. Kostadinov ve N. Bencheva, "An Approach for Teaching Processor Design", içinde *2018 28th EAEEIE Annual Conference (EAEEIE)*, Eyl. 2018, ss. 1-9. doi: 10.1109/EAEEIE.2018.8534262.

[37] A. Wanjari, N. Bisen, M. Chaudhari, S. Rajak, ve S. P. Washimkar, "To Design 16 bit Synchronous Microprocessor using VHDL on FPGA", c. 05, sy 03, s. 3, 2018.

[38] M. M. Mano, *Computer system architecture*, 3. ed., International ed. Upper Saddle River, NJ: Pearson Education Prentice-Hall International, 1993.

[39] "De0 Nano User Manuel". terasic.com.tw, 2012.

EKLER

EK 1. ALU Yapısının VHDL Kodları

```
LIBRARY ieee; USE ieee.std_logic_1164.all;
library work; USE work.datatypes.all, work.defines.all, work.devices.all;
-- alu.vhd
-- Developer: BAKACAK, M.
-- Computer Engineer
-- MB_MMC16 Ver. 2
entity alu is
port(
  Ein: in std_logic;
  D0, D1: in Tword;
  INPR: in Tword;
  fn : in aluinst;
  E: out std_logic;
  q: out Tword
);
end alu;

architecture a of alu is
  signal  Cout1,Cout2,Cout3: std_logic;
  signal notD1, and2, add2, shr, shl, notINPR: Tword;
begin
  chip1: andnbit generic map(n=>16) port map(D0, D1, and2);
  chip2: fan     generic map(n=>16) port map(D0, D1, add2,Cout1);  --e2
  chip3: shift   generic map(n=>16) port map(D1,Ein,'0',shr,Cout2);--e1
  chip4: shift   generic map(n=>16) port map(D1,Ein,'1',shl,Cout3);--e0
  chip5: com     generic map(n=>16) port map(D1, notD1);

        q <=    D0       when fn=aluPass
        else INPR        when fn=aluINPR
        else and2        when fn=aluAND
        else add2        when fn=aluADD
        else notD1       when fn=aluCOM
```

```
        else shr        when fn=aluSHR
        else shl        when fn=aluSHL
        else zeroword;

  E <=   Cout1 when fn=aluADD
        else COut2 when fn=aluSHR
        else COut3 when fn=aluSHL
        else Ein;

end a;
```

EK 2. Structure Yapısının VHDL Kodları

```
LIBRARY ieee; USE ieee.std_logic_1164.all;
library work; USE work.datatypes.all, work.defines.all, work.devices.all;
-- struct.vhd
-- Developer: BAKACAK, M.
-- Computer Engineer
-- MB_MMC16 Ver. 2
entity struct is
port (
  MemWr: in std_logic; -- memory write varsa degeri 1
  AluFn: in std_logic_vector(2 downto 0); -- ALU Fonksiyon Belirleyici
  BusSel: in std_logic_vector(2 downto 0); -- Bus yolu belirleyici

  ArLd, ArInc, ArClr,                    -- AR icin sinyaller
  PcLd, PcInc, PcClr,                    -- PC icin sinyaller
  DrLd, DrInc, DrClr,                    -- DR icin sinyaller
  AcLd, AcInc, AcClr,                    -- AC icin sinyaller
  IrLd, IrInc, IrClr,                    -- IR icin sinyaller
  TrLd, TrInc, TrClr,                    -- TR icin sinyaller
  OtLd, OtInc, OtClr,                    -- Output Register sinyalleri

  RJ, RK,                                -- R Flipflop Sinyalleri
  IENJ, IENK,                            -- IEN FF
  FGIJ, FGIK,                            -- FGI FF
  FGOJ, FGOK,                            -- FGO FF
  clock                                  -- Clock
  : in std_logic;

  Sdata: in Tword;--std_logic_vector(2 downto 0);
  E,E1: in std_logic;                      -- E Flipflop
  Si: in std_logic;                      -- S Flipflop
  sCLE: in std_logic:='0';
  sCME : in std_logic:='0';
```

```
  -- Control Sinyalleri
  IR, DR, AC: out TWord;
  R, IEN, FGI, FGO : out std_logic;

  -- Diger Cikis Sinyalleri
  PC, Ot: out TWord;
  INPR : out Tword;
  Ein,S: out std_logic
  );
end struct;

architecture a of struct is
  signal busdata, aluout, Memout,
                ArS, PcS, DrS, AcS, IrS, TrS, INPRS: Tword;

  signal Ein1,Ein2,Ein3,soCLE, soCME,tempE,tempE1,tempE2: std_logic;
begin
-- Cikislarin Dahili Sinyallere Baglanmasi
  IR <= IrS;    DR <= DrS; AC <= AcS; PC <= PcS; INPR <= INPRS;

-- Flipflops
  FFE  : JKFFLOP port map(J=>E,K=>E1,clock=>clock,Q=>Ein1);
  FFS  : JKFFLOP port map(J=>Si,K=>Si,clock=>clock,Q=>S);
  FFR  : JKFFLOP port map(J=>RJ, K=>RK, clock=>clock, Q=>R);
  FFIEN: JKFFLOP port map(J=>IENJ, K=>IENK, clock=>clock, Q=>IEN);
  FFFGI: JKFFLOP port map(J=>FGIJ, K=>FGIK, clock=>clock, Q=>FGI);
  FFFGO: JKFFLOP port map(J=>FGOJ, K=>FGOK, clock=>clock, Q=>FGO);

-- Registers
  RegIr: reg16 port map(IrInc, busdata, IrClr, IrLd, clock, IrS);
  RegDr: reg16 port map(DrInc, busdata, DrClr, DrLd, clock, DrS);
  RegPc: reg12 port map(PcInc, busdata, PcClr, PcLd, clock, PcS);
  RegTr: reg16 port map(TrInc, busdata, TrClr, TrLd, clock, TrS);
  RegOt: reg16 port map(OtInc, busdata, OtClr, OtLd, clock, Ot);
```

```
  RegAr: reg12 port map(ArInc, busdata, ArClr, ArLd, clock, ArS);
  RegINPR: reg16 port map('0', Sdata, '0', '1', clock, INPRS);
  RegAc: reg16 port map(AcInc, aluout , AcClr, AcLd, clock, AcS);

-- Memory
  Memchip: mem port map(busdata, Ars(11 downto 0), MemWr, clock, Memout);

-- Bus
  BusSwitch: buslines port map(ArS, PcS, DrS, AcS, IrS, TrS, Memout, Bussel,
busdata);

-- Alu
  AluUnit: Alu port map(Ein1, DrS, AcS, INPRS, AluFn,Ein, aluout);

end a;
```

EK 3. Kontrol Ünitesinin VHDL Kodları

```
LIBRARY ieee; USE ieee.std_logic_1164.all;
library work; USE work.datatypes.all, work.devices.all, work.defines.all;
-- controller.vhd
-- Developer: BAKACAK, M.
-- Computer Engineer
-- MB_MMC16 Ver. 2
entity controller is
  port(
        T: in std_logic_vector(15 downto 0);
        IR: in Tword;
        DR: in Tword;
        R,IEN, FGI, FGO, Ein: in std_logic;
        AC: in Tword;

        MemWr: out std_logic;
        AluFn: out aluinst;
        BusSel : out busctrl;

        ArLd, ArInc, ArClr,
        PcLd, PcInc, PcClr,
        DrLd, DrInc, DrClr,
        AcLd, AcInc, AcClr,
        IrLd, IrInc, IrClr,
        TrLd, TrInc, TrClr,
        OtLd, OtInc, OtClr,

        RJ, RK,
        IENJ, IENK,             -- IEN FF
        FGIK,                   -- FGI FF (Set by i/o device)
        FGOK,                   -- FGO FF (Set by i/o device);
        ScClr,                  -- Clear for sequencer (end of current instruction)
```

```
        E,E1,S,sCLE,sCME                    -- E Flipflop
  : out std_logic

  );
  end controller;

architecture a of controller is
  signal I, LCr, DRZflag, P, tempE,E2, tempE1,ACZflag,AC15flag : std_logic;
  signal SelPC, SelMem, SelTr, SelAc, SelIr, SelAr:std_logic;
  signal D: std_logic_vector(7 downto 0);
  signal      cADD,cCIR,cCIL,cCMA,cAND,cPASS,cINPR,cCME,cCLE,cHLT      :
std_logic;
  signal say: integer :=0;
begin
  --
  I <=    IR(15);
  P <=    D(7) and I and T(3);
  LCr <=  D(7) and (Not I) and T(3);--r
  DRZflag  <= '1' when DR = "0000000000000000" else '0'; --ISZ komutu sarti
  ACZflag  <= '1' when AC = "0000000000000000" else '0'; --SZA komutu sarti
  AC15flag <= '1' when AC(15)='1' else '0';             --SPA ve SNA icin komut sarti
  insdecode: dec3x8 port map(IR(14 downto 12), D);

  cINPR<= (P    and IR(11));
  cPASS<= (D(2) and T(5));
  cAND <= (D(0) and T(5));
  cADD <= (D(1) and T(5));
  cCIR <= (LCr  and Ir(7));
  cCIL <= (LCr  and Ir(6));
  cCMA <= (LCr  and Ir(9));
  cCME <= (LCr  and Ir(8));
  cCLE <= (LCr  and Ir(10));
  cHLT <= (LCr  and Ir(0));  --HLT: S <- 0
```

```
--         S flipflop+
  S<= cHLT;-- Start/Stop CPU

-- E flipflop
  tempE <= (cCIR) or (cCIL) or (Ein and cADD) or cCME or cCLE ;--EFF Enable

--elde islemleri
  E <=     '0' when tempE= cCLE
  else not Ein when tempE= cCME
  else Ein;
  E1<=tempE;

  sCLE<=cCLE;
  sCME<=cCME;

-- R flipflop +
  RJ <= not T(0) and not T(1) and not T(2) and IEN and (FGI or FGO); -- R<-1
  RK <= R and T(2); -- R<-0

-- IEN flipflop
  IENJ <=  P and Ir(7);
  IENK <= (P and Ir(6)) or
                    (R and T(2));

-- FGI flipflop +
  FGIK <= not (P and Ir(11));

-- FGO flipflop +
  FGOK <= not (P and Ir(10));

-- Sequence Counter (SC) +
  ScClr <= (R  and T(2)) or
           (D(0) and T(5)) or --AND: SC<-0
                    (D(1) and T(5)) or --ADD: SC<-0
```

```
                (D(2) and T(5)) or --LDA: SC<-0
                (D(3) and T(4)) or --STA: SC<-0
                (D(4) and T(4)) or --BUN: SC<-0
                (D(5) and T(5)) or --BSA: SC<-0
                (D(6) and T(6)) or --ISZ: SC<-0
                 LCr  or P              ;   --SC <- 0


-- Alu KOMUTLAR
  AluFn <= aluINPR when cINPR ='1' else
                        aluPass when cPASS ='1' else

                        aluAND  when cAND  ='1' else
                        aluADD  when cADD  ='1' else
                        aluCOM  when cCMA  ='1' else --  CMA AC<=AC'
                        aluSHR  when cCIR  ='1' else
                        aluSHL  when cCIL  ='1' ;

-- Memory write line
  MemWr <=  (D(3) and T(4)) or            --STA: M[AR] <- AC
                        (D(5) and T(4)) or    --BSA: M[AR] <- PC
                        (D(6) and T(6)) or    --ISZ: M[AR] <- DR
                        (R and T(1));         --Interrupt: M[AR] <- TR


-- Address Register
  ArLd <=       ((not R) and T(0)) or               --Fetch: AR <- PC
                        ((not R) and T(2)) or       --Decode: AR <- IR(0-11)
                        ((not D(7)) and I and T(3)); --Indirect: AR <- M[AR]
  ArClr <= R and T(0);                              --Interrupt: AR <- 0
  ArInc <= D(5) and T(4);                           --BSA: AR <- AR+1

-- Program Counter  -
  PcLd <=  (D(4) and T(4)) or                       --BUN: PC <- AR
```

```
                (D(5) and T(5));                --BSA: PC <- AR

  PcClr <= (R and T(1));                        --Interrupt: PC <- 0

  PcInc <= ((Not R ) and T(1)) or               --Fetch: PC <- PC+1
           ((R) and T(2)) or                    --interrupt: PC <- PC+1
   (D(6) and T(6)  and DRZflag) or              --ISZ  DR=0: PC <- PC+1
  (LCr  and IR(4) and not AC15flag) or          --SPA  AC(15)=0 : PC <- PC+1
  (LCr  and IR(3) and AC15flag) or              --SNA: AC(15)=1 : PC <- PC+1
  (LCr  and IR(2) and ACZflag)  or              --SZA: AC=0  : PC <- PC+1
  (LCr  and IR(1) and not Ein)  or              --SZE: E=0     : PC <- PC+1
  (P  and IR(9) and FGI)      or                --SKI: FGI=1   : PC <- PC+1
  (P  and IR(8) and FGO)       ;                --SKO: FGO=1   : PC <- PC+1

-- Data register DR  +
  DrLd <= (D(0) and T(4))     or                --AND: DR <- M[AR]
          (D(1) and T(4))     or                --ADD: DR <- M[AR]
          (D(2) and T(4))     or                --LDA: DR <- M[AR]
              (D(6) and T(4));                  --ISZ: DR <- M[AR]
  DrInc <= D(6) and T(5);                       --ISZ: DR <- DR + 1
  DrClr <= '0';

-- Temporary Register TR   +
  TrLd <= R and T(0);                   --Interrupt: TR <- PC
  TrInc <= '0';
  TrClr <= '0';

-- Instruction Register IR; +
  IrLd <= (Not R) and T(1);  -- Fetch    : IR <- M[AR]
  IrInc <= '0';
  IrClr <= '0';

-- Accumulator Ac; +
  AcLd <= (D(0)  and T(5)) or         --AND          : AC <- AC ^ DR
```

```
                (D(1)  and T(5)) or   --ADD          : AC <- AC + DR
                (D(2)  and T(5)) or   --LDA(DR)  : AC <- DR
                (LCr   and IR(7))or   --CIR/SHR     : AC <- shr AC , AC(15) <- E
                (LCr   and IR(6))or --CIL/SHL       : AC <- shl AC , AC(0)  <- E
                (LCr   and IR(9))or --CMA/COM   : AC <- com AC
                (P     and IR(11));    --INPR          : AC <- INPR
  AcInc <= LCr   and IR(5);           --INC           : AC <- AC + 1
  AcClr <= LCr   and IR(11);          --CLA           : AC <- 0

-- Output register OT
  OtLd <= P and IR(10);                -- OT <- AC
  OtInc <= '0';
  OtClr <= '0';
  ---------------------------------
  SelPc <= (T(0) and '1') or    --Fetch(R'T(0)): AR <- PC , Interrupt(RT(0)): TR <- PC
         (D(5) and T(4)) ;              --BSA           : M[AR] <- PC

  SelMem <=((not R) and T(1))or --Fetch     : IR <- M[AR]
                        (D(0) and T(4)) or   --AND   : DR <- M[AR]
                        (D(1) and T(4)) or   --ADD   : DR <- M[AR]
                        (D(2) and T(4)) or   --LDA   : DR <- M[AR]
                        (D(6) and T(4)) or   --ISZ      : DR <- M[AR]
         ((not D(7)) and I and T(3)); --Indirect        : AR <- M[AR]

  SelTr <=  R and T(1);         --Interrupt       : M[AR] <- TR

  SelAc <=       (P and IR(10)) or       --OUT: OT <- AC
                 (D(0) and T(5)) or   --AND: AC <- AC ^ DR
                 (LCr and IR(5)) or   --INC: AC <- AC + 1
                 (D(3) and T(4)) ;       --STA: M[AR] <- AC




  SelIr <= ((not R) and T(2));          --Decode: AR <- IR(0-11)
```

```
SelAr <=  (D(4) and T(4)) or          --BUN:        PC <- AR
          (D(5) and T(5)) ;           --BSA:        PC <- AR

-- Bus selection
BusSel <=    BusPc  when SelPc ='1'
                 else BusMem when SelMem='1'
                 else BusTr  when SelTr ='1'
                 else BusAc  when SelAc ='1'
                 else BusIr  when SelIr ='1'
                 else BusAr  when SelAr ='1'
                 else BusDr;

end a;
```

EK 4. Timer16 Yapısının VHDL Kodları

```
LIBRARY ieee; USE ieee.std_logic_1164.all;
library work; use work.devices.all;
--timer16.vhd
-- Developer: BAKACAK, M.
-- Computer Engineer
-- MB_MMC16 Ver. 2
entity timer16 is
  port(
        clock: in std_logic;
        clr: in std_logic;
        S: in std_logic;
        o: out std_logic_vector(15 downto 0)
  );
end timer16;

architecture a of timer16 is
  signal zero, coutS: std_logic;
  signal dataS,qs, qbars: std_logic_vector(3 downto 0);
begin
  zero <= '0';
  dataS <="0000";
  counter: regn
  generic map(n=>4)
  port map(
        inc =>not S, data=>dataS, clr=>clr, load=>zero, clock=>clock,
        q=>qs, qbar=>qbars, cout=>couts);

  dec: dec4x16 port map(sel=>qs, o=>o);

end a;
```

EK 5. Defines Yapısının VHDL Kodları

```
library ieee; USE ieee.std_logic_1164.all;
--defines.vhd
-- Developer: BAKACAK, M.
-- Computer Engineer
-- MB_MMC16 Ver. 2

PACKAGE defines IS
  -- Subtypes
        subtype aluinst is std_logic_vector(2 downto 0);
        subtype busctrl is std_logic_vector(2 downto 0);

  --  Constants
        -- ALU instructions
        constant aluPass: aluinst := "000";
        constant aluAND:     aluinst := "001";
        constant aluADD:     aluinst := "010";
        constant aluCOM:     aluinst := "011";
        constant aluSHR:     aluinst := "100";
        constant aluSHL:     aluinst := "101";
        constant aluINPR:    aluinst := "110";

  -- Bus select codes
        constant BusAR:      busctrl := "001";
        constant BusPC:      busctrl := "010";
        constant BusDR:      busctrl := "011";
        constant BusAC:      busctrl := "100";
        constant BusIR:      busctrl := "101";
        constant BusTR:      busctrl := "110";
        constant BusMem:     busctrl := "111";

end defines;
```

EK 6. OUT BUS MUX Yapısının VHDL Kodları

```
LIBRARY ieee; USE ieee.std_logic_1164.all;
LIBRARY lpm; USE lpm.lpm_components.all;
--out_bus_mux.vhd
-- Developer: BAKACAK, M.
-- Computer Engineer
-- MB_MMC16 Ver. 2

ENTITY out_bus_mux IS
  PORT
  (
        data0x          : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        data1x          : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
        sel             : IN STD_LOGIC ;
        result          : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
  );
END out_bus_mux;

ARCHITECTURE SYN OF out_bus_mux IS

--type STD_LOGIC_2D is array (NATURAL RANGE <>, NATURAL RANGE <>)
of STD_LOGIC;

  SIGNAL sub_wire0 : STD_LOGIC_VECTOR (7 DOWNTO 0);
  SIGNAL sub_wire1 : STD_LOGIC_VECTOR (7 DOWNTO 0);
  SIGNAL sub_wire2 : STD_LOGIC_2D (1 DOWNTO 0, 7 DOWNTO 0);
  SIGNAL sub_wire3 : STD_LOGIC_VECTOR (7 DOWNTO 0);
  SIGNAL sub_wire4 : STD_LOGIC ;
  SIGNAL sub_wire5 : STD_LOGIC_VECTOR (0 DOWNTO 0);

BEGIN
  sub_wire3    <= data0x(7 DOWNTO 0);
  result    <= sub_wire0(7 DOWNTO 0);
```

```
  sub_wire1    <= data1x(7 DOWNTO 0);
  sub_wire2(1, 0)    <= sub_wire1(0);
  sub_wire2(1, 1)    <= sub_wire1(1);
  sub_wire2(1, 2)    <= sub_wire1(2);
  sub_wire2(1, 3)    <= sub_wire1(3);
  sub_wire2(1, 4)    <= sub_wire1(4);
  sub_wire2(1, 5)    <= sub_wire1(5);
  sub_wire2(1, 6)    <= sub_wire1(6);
  sub_wire2(1, 7)    <= sub_wire1(7);
  sub_wire2(0, 0)    <= sub_wire3(0);
  sub_wire2(0, 1)    <= sub_wire3(1);
  sub_wire2(0, 2)    <= sub_wire3(2);
  sub_wire2(0, 3)    <= sub_wire3(3);
  sub_wire2(0, 4)    <= sub_wire3(4);
  sub_wire2(0, 5)    <= sub_wire3(5);
  sub_wire2(0, 6)    <= sub_wire3(6);
  sub_wire2(0, 7)    <= sub_wire3(7);
  sub_wire4    <= sel;
  sub_wire5(0)    <= sub_wire4;

  LPM_MUX_component : LPM_MUX
  GENERIC MAP (
        lpm_size => 2,
        lpm_type => "LPM_MUX",
        lpm_width => 8,
        lpm_widths => 1
  )
  PORT MAP (
        data => sub_wire2,
        sel => sub_wire5,
        result => sub_wire0
  );

END SYN;
```

EK 7. Devices Yapısının VHDL Kodları

```
LIBRARY ieee; USE ieee.std_logic_1164.all;
library work; USE work.datatypes.all, work.defines.all;
--devices.vhd

PACKAGE devices IS

  component fa1
        port(
              x,y,z: in std_logic;
              s,c: out std_logic);
  end component;

  component hadder
        port(
              x,y: in std_logic;
              s,c: out std_logic);
  end component;

  component fan
        generic (n: integer);
        port(
              x,y: in std_logic_vector(n-1 downto 0);
              s: out std_logic_vector(n-1 downto 0);
              Cout: out std_logic
        );
  end component ;


  component and1bit
        port (
              x, y: in std_logic;
              z: out std_logic
```

```
        );
end component;

component andnbit
        generic (n: integer);
        port(
                x,y: in std_logic_vector(n-1 downto 0);
                s: out std_logic_vector(n-1 downto 0)
        );
end component ;

component dec1x2e
        port(
                sel: in std_logic;
                en: in std_logic;
                o: out std_logic_vector(1 downto 0)
        );
end component ;

component dec1x2
        port(
                sel: in std_logic;
                o: out std_logic_vector(1 downto 0)
        );
end component ;

component dec2x4e
        port(
                sel: in std_logic_vector(1 downto 0);
                en: in std_logic;
                o: out std_logic_vector(3 downto 0)
        );
end component;
```

```
component dec2x4
	port(
		sel: in std_logic_vector(1 downto 0);
		o: out std_logic_vector(3 downto 0)
	);
end component ;

component dec3x8
	port(
		sel: in std_logic_vector(2 downto 0);
		o: out std_logic_vector(7 downto 0)
	);
end component ;

component dec4x16
	port(
		sel: in std_logic_vector(3 downto 0);
		o: out std_logic_vector(15 downto 0)
	);
end component ;

component jkfflop
port(
	J,K		: IN	STD_LOGIC;
	clock		: IN	STD_LOGIC;
	Q,Qbar			: OUT	STD_LOGIC);
end component;

component dfflop
	PORT
	(
		D		: IN	STD_LOGIC;
		clock		: IN	STD_LOGIC;
		Q		: OUT	STD_LOGIC
```

```
        );
end component;

component reg1
        port
        (
                cin: in std_logic;
                data: in std_logic;

                clr: in std_logic;
                ld: in std_logic;

                clock: in std_logic;

                q, qbar: out std_logic;
                cout: out std_logic);
end component;

component regn
        generic (n: integer);
        port (
        inc: in std_logic;
        data: in std_logic_vector(n-1 downto 0);

        clr: in std_logic;
        load: in std_logic;

        clock: in std_logic;

        q, qbar: out std_logic_vector(n-1 downto 0);
        cout: out std_logic);

end component;
```

```
component reg16
	port
	(
		inc: in std_logic;
		data: in std_logic_vector(15 downto 0);
		clr: in std_logic;
		load: in std_logic;

		clock: in std_logic;

		q: out std_logic_vector(15 downto 0)
		);
end component;

component reg12
	port
	(
		inc: in std_logic;
		data: in std_logic_vector(15 downto 0);

		clr: in std_logic;
		load: in std_logic;

		clock: in std_logic;

		q: out std_logic_vector(15 downto 0)
		);
end component;

component timer16
	port(
		clock: in std_logic;
		clr:in std_logic;
		S:in std_logic;
```

```
            o: out std_logic_vector(15 downto 0)
        );
end component;
component ram
PORT(
        address :       IN TAddress;
        clock   :       IN STD_LOGIC;
        data    :       IN Tword;
        we              :       IN STD_LOGIC;
        q               :       OUT Tword
);
END component ;

component mem
PORT(
        data    :       IN Tword;
        address :       IN TAddress;
        we              :       IN STD_LOGIC;
        clock:  IN STD_LOGIC;
        q               :       OUT Tword
);
END component;

component alu
port(
        Ein: in std_logic;
        D0, D1: in Tword;
        INPR: in Tword;
        fn : in aluinst;
        E: out std_logic;
        q: out Tword
);
end component;
```

```
component shift
        generic (n: integer);
        port(
                x: in std_logic_vector(n-1 downto 0);
                Ein: in std_logic;
                mode: in std_logic;
                y: out std_logic_vector(n-1 downto 0);
                E: out std_logic
        );
end component ;

component com
        generic (n: integer);
        port(
                x: in std_logic_vector(n-1 downto 0);
                y: out std_logic_vector(n-1 downto 0)
        );
end component ;

component buslines
        port(
                Ar, PC, DR, AC, IR, TR, Memory: in Tword;
                sel: in busctrl ;
                q: out Tword
        );
end component;

component controller
        port(
                T: in std_logic_vector(15 downto 0);
                IR: in Tword;
                DR: in Tword;
                R, IEN, FGI, FGO: in std_logic;
```

```
                    Ein: in std_logic;
            MemWr: out std_logic;
            AluFn: out aluinst;
            BusSel : out busctrl;


            ArLd, ArInc, ArClr,
            PcLd, PcInc, PcClr,
            DrLd, DrInc, DrClr,
            AcLd, AcInc, AcClr,
            IrLd, IrInc, IrClr,
            TrLd, TrInc, TrClr,
            OtLd, OtInc, OtClr,

            RJ, RK,
            IENJ, IENK,
            FGIK,
            FGOK,

            ScClr,

            E,E1,S : out std_logic

    );
  end component;

END devices;
```

EK 8. and1bit Biriminin VHDL Kodları

```
LIBRARY ieee; USE ieee.std_logic_1164.all;
library work; use work.devices.all;
--and1bit.vhd

entity and1bit is
	port(
		x,y: in std_logic;
		z: out std_logic
	);
end and1bit;

architecture a of and1bit is
begin
  z <= x and y;
end a;
```

EK 9. Andnbit Biriminin VHDL Kodları

```
LIBRARY ieee; USE ieee.std_logic_1164.all;
library work; use work.devices.all;
-- andnbit.vhd

entity andnbit is
  generic (n: integer := 16);
  port(
        x,y: in std_logic_vector(n-1 downto 0);
        s: out std_logic_vector(n-1 downto 0)
  );
end andnbit;

architecture a of andnbit is
begin

        gen_loop: for i in 0 to n-1 generate
                chip: and1bit port map (x=>x(i), y=>y(i), z=>s(i));
        end generate;
end a;
```

EK 10. Reg1 Biriminin VHDL Kodları

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY work;
USE work.devices.all;
--reg1.vhd
entity reg1 is
  port
  (
        cin: in std_logic;
        data: in std_logic;

        clr: in std_logic;
        ld: in std_logic;

        clock: in std_logic;

        q, qbar: out std_logic;
        cout: out std_logic);
end reg1;

architecture a of reg1 is
  signal js, ks, qs, qbars: std_logic;
begin
        js <= cin or (data and ld);
        ks <= cin or (not data and ld) or clr;

        ff1: jkfflop port map (js, ks, clock, qs, qbars);
        q <= qs;
        qbar <= qbars;
        cout <=qs and cin;

end a;
```

EK 11. Reg12 Biriminin VHDL Kodları

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
--reg12.vhd

library work;
use work.devices.all;

entity reg12 is
  port
  (
        inc: in std_logic;
        data: in std_logic_vector(15 downto 0);

        clr: in std_logic;
        load: in std_logic;

        clock: in std_logic;

        q: out std_logic_vector(15 downto 0)
        );
end reg12;

architecture a of reg12 is

  signal qs: std_logic_vector(11 downto 0);
begin

  chip: regn generic map(n=>12)
        port  map (inc, data(11 downto 0), clr, load, clock, qs);
  q <= "0000" & qs;
end a;
```

EK 12. Reg16 Biriminin VHDL Kodları

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-- reg16.vhd

library work;
use work.devices.all;

entity reg16 is
  port
  (
        inc: in std_logic;
        data: in std_logic_vector(15 downto 0);

        clr: in std_logic;
        load: in std_logic;

        clock: in std_logic;

        q: out std_logic_vector(15 downto 0)
        );
end reg16;

architecture a of reg16 is
  signal qbar: std_logic_vector(15 downto 0);
  signal cout: std_logic;
begin
  chip: regn generic map(n=>16)
        port map(inc, data, clr, load, clock, q, qbar, cout);
end a;
```

EK 13. Regn Biriminin VHDL Kodları

```
--regn.vhd
LIBRARY ieee;
USE ieee.std_logic_1164.all;

library work; use work.devices.all;



entity regn is
  generic (n: integer := 16);

  port
  (
        inc: in std_logic;
        data: in std_logic_vector(n-1 downto 0);--:=(others => '1') dene verisini 0
lamak icin

        clr: in std_logic;
        load: in std_logic;

clock: in std_logic;

        q, qbar: out std_logic_vector(n-1 downto 0);
        cout: out std_logic);
end regn;

architecture a of regn is
  signal coS: std_logic_vector(n-2 downto 0);
  signal cin,  ld: std_logic;
begin
        cin <= inc and not load and not clr;
        ld <= load and not clr;
```

```
        r0: reg1 port map (cin, data(0), clr,ld, clock, q(0), qbar(0), coS(0));
        gen_loop: for i in 1 to n-2 generate
                ri: reg1 port map (coS(i-1), data(i), clr,ld, clock, q(i), qbar(i), coS(i));
        end generate;
        rlast: reg1 port map (coS(n-2), data(n-1), clr,ld, clock, q(n-1), qbar(n-1), cout);
end a;
```

EK 14. ClockOrganizer Biriminin VHDL Kodları

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--ClockOrganizer.vhd
-- Developer: BAKACAK, M.
-- Computer Engineer
-- MB_MMC16 Ver. 2

entity ClockOrganizer is
    port(
        clock   : in STD_LOGIC;  -- 50 Mhz
        cout    : out STD_LOGIC -- 100 Hz
    );
end ClockOrganizer;

architecture CO of ClockOrganizer is
    constant Hz:integer:=1;
    constant KHz:integer:=1000;
    constant MHz:integer:=1000000;
    constant clock_speed: integer:=50*MHz; --DE0-NANO Ossilator 50 MHz
    constant desired_clock_speed: integer:=100*Hz; --Clock Out
    -- referenceClock (clock_speed/desired_clock_speed)/2 (2 : clock rising_edge)
    signal referenceClock: integer := (clock_speed/desired_clock_speed)/2;
--referenceClock
    signal referenceClock_counter: integer :=0;
    signal newClock : std_logic := '0';
begin
    cout <= newClock;
    countClock: process(clock, newClock)
    begin
        if rising_edge(clock) then
            referenceClock_counter <= referenceClock_counter + 1;
```

```
            if(referenceClock_counter > referenceClock) then
                newClock <= not newClock;
                referenceClock_counter <= 0;
            end if;
        end if;
    end process;
end CO;
```

EK 15. Com Biriminin VHDL Kodları

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
--com.vhd

library work;
use work.devices.all, work.defines.all;

entity com is
  generic (n: integer := 16);
        port(
                x: in std_logic_vector(n-1 downto 0);
                y: out std_logic_vector(n-1 downto 0)
        );
end com;

architecture cm of com is
  begin
        process(x)
                begin
                for i in 0 to n-1 loop
                  y(i) <= not x(i);
                end loop;
        end process;
end cm;
```

EK 16. Buslines Biriminin VHDL Kodları

```
LIBRARY ieee; USE ieee.std_logic_1164.all;
library work; USE work.datatypes.all, work.defines.all;
--buslines.vhd

entity buslines is
  port(
        Ar, PC, DR, AC, IR, TR, Memory: in Tword;
        sel: in busctrl ;
        q: out Tword
  );
end buslines;

architecture a of buslines is
begin
 q <= Ar when sel = BusAR  else
        PC when sel = BusPC  else
        DR when sel = BusDR  else
        AC when sel = BusAC  else
        IR when sel = BusIR  else
        TR when sel = BusTR  else
  Memory when sel = BusMem else
        BusDefault;
end a;
```

EK 17. Dec1x2 Biriminin VHDL Kodları

```
LIBRARY ieee; USE ieee.std_logic_1164.all;
library work; use work.devices.all;
-- dec1x2.vhd

entity dec1x2 is
  port(
        sel: in std_logic;
        o: out std_logic_vector(1 downto 0)
  );
end dec1x2 ;

architecture a of dec1x2 is
begin
  o(0) <= (not sel);
  o(1) <= sel;
end a;
```

EK 18. Dec1x2e Biriminin VHDL Kodları

```
LIBRARY ieee; USE ieee.std_logic_1164.all;
library work; use work.devices.all;
-- dec1x2e.vhd

entity dec1x2e is
  port(
        sel: in std_logic;
        en: in std_logic;
        o: out std_logic_vector(1 downto 0)
  );
end dec1x2e ;

architecture a of dec1x2e is
begin
  o(0) <= (not sel) and en;
  o(1) <= sel and en;
end a;
```

EK 19. Dec2x4 Biriminin VHDL Kodları

```
LIBRARY ieee; USE ieee.std_logic_1164.all;
library work; use work.devices.all;
-- dec2x4.vhd

entity dec2x4 is
  port(
        sel: in std_logic_vector(1 downto 0);
        o: out std_logic_vector(3 downto 0)
  );
end dec2x4 ;

architecture a of dec2x4 is
  signal chipsel: std_logic_vector(1 downto 0);
  signal one: std_logic;

begin
  sel1: dec1x2 port map(sel(1), chipsel);
  chip0: dec1x2e port map(sel(0), chipsel(0), o(1 downto 0));
  chip1: dec1x2e port map(sel(0), chipsel(1), o(3 downto 2));
end a;
```

EK 20. Dec2x4e Biriminin VHDL Kodları

```
LIBRARY ieee; USE ieee.std_logic_1164.all;
library work; use work.devices.all;
-- dec2x4e.vhd

entity dec2x4e is
  port(
        sel: in std_logic_vector(1 downto 0);
        en: in std_logic;
        o: out std_logic_vector(3 downto 0)
  );
end dec2x4e ;

architecture a of dec2x4e is
  signal chipsel: std_logic_vector(1 downto 0);
begin
  sel1: dec1x2e port map(sel(1), en, chipsel);
  chip0: dec1x2e port map(sel(0), chipsel(0), o(1 downto 0));
  chip1: dec1x2e port map(sel(0), chipsel(1), o(3 downto 2));
end a;
```

EK 21. Dec3x8 Biriminin VHDL Kodları

```
LIBRARY ieee; USE ieee.std_logic_1164.all;
library work; use work.devices.all;
--dec3x8.vhd

entity dec3x8 is
  port(
        sel: in std_logic_vector(2 downto 0);
        o: out std_logic_vector(7 downto 0)
  );
end dec3x8;

architecture a of dec3x8 is
  signal chipsel: std_logic_vector(1 downto 0);
begin
  sel1: dec1x2 port map(sel(2), chipsel);
  chip0: dec2x4e port map(sel(1 downto 0), chipsel(0), o(3 downto 0));
  chip1: dec2x4e port map(sel(1 downto 0), chipsel(1), o(7 downto 4));
end a;
```

EK 22. Dec4x16 Biriminin VHDL Kodları

```
LIBRARY ieee; USE ieee.std_logic_1164.all;
library work; use work.devices.all;
--dec4x16.vhd

entity dec4x16 is
  port(
        sel: in std_logic_vector(3 downto 0);
        o: out std_logic_vector(15 downto 0)
  );
end dec4x16;

architecture a of dec4x16 is
  signal chipsel: std_logic_vector(3 downto 0);
begin
  sel1: dec2x4 port map(sel(3 downto 2), chipsel);
  chip0: dec2x4e port map(sel(1 downto 0), chipsel(0), o(3 downto 0));
  chip1: dec2x4e port map(sel(1 downto 0), chipsel(1), o(7 downto 4));
  chip2: dec2x4e port map(sel(1 downto 0), chipsel(2), o(11 downto 8));
  chip3: dec2x4e port map(sel(1 downto 0), chipsel(3), o(15 downto 12));

end a;
```

EK 23. Datatypes Biriminin VHDL Kodları

```
library ieee; USE ieee.std_logic_1164.all;
--datatypes.vhd

PACKAGE datatypes IS
  --  Constants
        constant wordsize : integer := 16;
        constant adrsize: integer := 12;
        --constant memUnitSize: integer := 12;
  --  SubType Declaration
        SUBTYPE Tword IS std_logic_vector(wordsize-1 downto 0);
        SUBTYPE Taddress IS std_logic_vector(adrsize-1 downto 0);
        --SUBTYPE TMemAddress IS std_logic_vector(memUnitSize-1 downto 0);
  -- Special constants
        constant zeroword : Tword := "0000000000000000";
        constant BusDefault: Tword := "1111111111111111";

end datatypes;
```

EK 24. DFFlop Biriminin VHDL Kodları

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-- dfflop.vhd

ENTITY dfflop IS

  PORT
  (
        D               : IN    STD_LOGIC;
        clock           : IN    STD_LOGIC;
        Q               : OUT   STD_LOGIC
  );

end dfflop;

architecture a OF dfflop IS

  SIGNAL      Q_signal        : STD_LOGIC;

begin

  process (clock)
  begin
        if (clock'event AND clock = '1') then
                        Q_signal <= D;
        end if;
  end process;
  Q <= Q_signal;
end a;
```

EK 25. Fa1 Biriminin VHDL Kodları

```
library ieee; use ieee.std_logic_1164.all;
--fa1.vhd

entity fa1 is
  port(
        x,y,z: in std_logic;
        s,c: out std_logic);
end fa1;

ARCHITECTURE a OF fa1 IS
  SIGNAL xeory: STD_LOGIC;
BEGIN
        xeory <= x xor y;
        s <= xeory xor z;
        c <= (x and y) or (xeory and z);
END a;
```

EK 26. Fan Biriminin VHDL Kodları

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
library work;
use work.devices.all;
--fan.vhd

entity fan is
  generic (n: integer := 16);

        port(
                x,y: in std_logic_vector(n-1 downto 0);
                s: out std_logic_vector(n-1 downto 0);
                Cout: out std_logic
        );
end fan;

architecture a of fan is
  signal coS: std_logic_vector(n-2 downto 0);
begin
        a0: hadder port map (x(0), y(0), s(0), coS(0));
        gen_loop: for i in 1 to n-2 generate
                ai: fa1 port map (x(i), y(i), coS(i-1),  s(i),  coS(i));
        end generate;
        alast: fa1 port map (x(n-1), y(n-1), coS(n-2), s(n-1), Cout);
end a;
```

EK 27. Hadder Biriminin VHDL Kodları

```
library ieee; use ieee.std_logic_1164.all;
--hadder.vhd

entity hadder is
  port(
        x,y: in std_logic;
        s,c: out std_logic);
end hadder;

ARCHITECTURE a OF hadder IS
BEGIN
        s <= x xor y;
        c <= x and y;
END a;
```

EK 28. JKFFlop Biriminin VHDL Kodları

```
LIBRARY ieee; USE ieee.std_logic_1164.all;
--jkfflop.vhd

 library work; use work.devices.all;

entity jkfflop is
  port(
        J,K             : IN     STD_LOGIC;
        clock: IN       STD_LOGIC;
        Q,Qbar                   : OUT  STD_LOGIC);
end jkfflop;

architecture a of jkfflop is

  signal qs, notqs, ds: std_logic;
begin
  notqs <= not qs;
  Q <= qs; Qbar <= notqs;
  ds <= (J and notqs) or (not K and qs);
  df: dfflop port map(D=>ds, clock=>clock, Q=>qs);
end a;
```

EK 29. Mem Biriminin VHDL Kodları

```
LIBRARY ieee; USE ieee.std_logic_1164.all;
LIBRARY work; USE work.datatypes.all; USE work.devices.all;
--mem.vhd

ENTITY mem IS
PORT(
  data   :      IN Tword;
  address :     IN TAddress;
  we            :       IN STD_LOGIC;
  clock :       IN STD_LOGIC;
  q             :       OUT Tword
);
END mem;

ARCHITECTURE example OF mem IS
BEGIN
  ramchip: ram port map(address, clock, data, we,  q);
END example;
```

EK 30. Ram Biriminin VHDL Kodları

```
library IEEE;use IEEE.STD_LOGIC_1164.ALL;
LIBRARY work;USE work.datatypes.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--ram.vhd
-- Developer: BAKACAK, M.
-- Computer Engineer
-- MB_MMC16 Ver. 2

entity ram is
port (
        address :in TAddress;
      clock : in std_logic;
      data : in Tword ;
          we : in std_logic;
      q : out Tword
   );
end ram;

architecture Behavioral of ram is
type ram_type is array (4096 downto 0) of std_logic_vector (15 downto 0);
signal RAM: ram_type:=
--

----+ ADD  2 adres verisini topla sonucu yaz program  + s: 11111
                    (
                    0 => "0010000000000101",-- LDA (2005)
                    1 => "0001000000000100",-- ADD (1004)
                    2 => "1111010000000000",-- OUT (F400)
                    3 => "0111000000000001",-- HLT (7001)
                    4 => "0000000000001001", --  A
                    5 => "0000010000000110", --  B
                    others => (others => '0'));
```

```
begin
PROCESS(clock)
BEGIN
   if (clock'event and clock='1') then
                  if(we='1') then
                                 RAM(conv_integer(address)) <= data;
                  end if;
   end if;
END PROCESS;
  q <= RAM(conv_integer(address));
end Behavioral;
```

EK 31. Shift Biriminin VHDL Kodları

```
LIBRARY ieee; USE ieee.std_logic_1164.all;
library work; use work.devices.all, work.defines.all;
--shift.vhd
-- Developer: BAKACAK, M.
-- Computer Engineer
-- MB_MMC16 Ver. 2

entity shift is
  generic (n: integer := 16);
        port(
                x: in std_logic_vector(n-1 downto 0);
                Ein: in std_logic;
                mode: in std_logic;
                y: out std_logic_vector(n-1 downto 0);
                E: out std_logic
        );
end shift;

architecture sh of shift is
  begin
  process(mode,Ein,x)
  begin
        IF mode='0' then--SHR
                E<=x(0);
                y<=Ein & x(15 downto 1);
        elsif mode='1' then--SHL
                E<=x(15);
                y<=x(14 downto 0) & Ein;
        end if;
  end process;
end sh;
```