T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

SAHADA PROGRAMLANABİLİR KAPI DİZİLERİ (FPGA) TABANLI ROBOT KONTROLÜ

Şirin AKKAYA

YÜKSEK LİSANS TEZİ
KONTROL VE OTOMASYON MÜHENDİSLİĞİ ANABİLİM DALI
KONTROL VE OTOMASYON PROGRAMI

DANIŞMAN
DOC. DR. HALUK GÖRGÜN

İSTANBUL, 2013

T.C.

YILDIZ TEKNİK ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

SAHADA PROGRAMLANABİLİR KAPI DİZİLERİ (FPGA) TABANLI ROBOT KONTROLÜ

Şirin AKKAYA tarafından hazırlanan tez çalışması 04.12.2013 tarihinde aşağıdaki jüri tarafından Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Kontrol ve Otomasyon Mühendisliği Anabilim Dalı'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Tez Danışmanı

Doc. Dr. Haluk Görgün

Yıldız Teknik Üniversitesi

Jüri Üyeleri

Doc. Dr. Haluk Görgün

Yıldız Teknik Üniversitesi

Yrd. Doç. Dr. Türker TÜRKER

Yıldız Teknik Üniversitesi

Yrd. Doç Dr. Celal Sami Tüfekçi

Yıldız Teknik Üniversitesi

Bu çalışma, Yıldız Teknik Üniversitesi Bilimsel Araştırma Projeleri Koordinatörlüğü' nün 2012-04-04-YL02 numaralı projesi ile desteklenmiştir.

ÖNSÖZ

Bu tezin hazırlanması sırasında bana her konuda yardımcı olan, yardım ve katkıları ile beni yönlendiren, değerli hocam ve danışmanım Doç. Dr. Haluk GÖRGÜN'e ve çalışmanın uygulama aşaması gerekli malzemelerin temini için projemizi desekleyen Yıldız Teknik Üniversitesi Bilimsel Araştırmalar Koordinatörlüğü'ne teşekkürlerimi sunarım.

Ayrıca tasarım ve uygulama konularında karşılaştığım her problemde bana yardımcı olan ve destekleyen sevgili arkadaşım Arş. Gör. Onur AKBATI'ya ve beni bu günlere getiren aileme ve özellikle beni her konuda destekleyen sevgili anneme sonsuz sevgi ve saygılarımı sunarım.

Aralık, 2013

Şirin AKKAYA

İÇİNDEKİLER

Sayfa

SİMGE LİSTESİ

${}^{i-1}_{i}T$	Dönüşüm matrisi
x_i	i. x ekseni
y_i	i. y ekseni
z_i	i. z ekseni
a_{i-1}	İki eksen arasındaki bağ uzunluğu
α_{i-1}	(i-1) ile i eksenleri arasındaki bağ açısı
d_i	Üst üste çakışan bağlar arasındaki eklem kaçıklığı
θ_i	İki bağ arasında oluşan eklem açısı
$\mathrm{v_b}(.)$	Zıt elektromotor kuvvet
$\mathrm{K_b}$	Zıt elektromotor katsayısı
$i_a(.)$	Armatür akımı
$e_a(.)$	Armatür gerilimi
R_a	Armatür direnci
L_a	Armatür endüktansı
$\tau_{\mathrm{m}}(.)$	Motor torku
K_t	Tork sabiti
$\mathrm{J_m}$	Armatürün ve yükün eşdeğer ataleti
$\mathrm{D_m}$	Armatürün ve yükün eşdeğer viskoz sürtünme değeri
T_i	Giriş ve çıkışın periodu
f_i	Giriş ve çıkışın frekansı
w_i	Açısal frekans
ϕ_i	Faz farkı
E_i	Giriş sinyali genliği
θ_i	Çıkış sinyali genliği
w_{cutoff}	Kesim frekansı
$\mathrm{K_P}$	Oransal katsayı
$\mathrm{K_I}$	İntegral katsayısı

$\mathrm{K_D}$	Türev katsayısı
$u(.)$	Kontrolcü çıkışı
$e(.)$	Hata değeri
$r(.)$	Referans değer
$y(.)$	Sistem çıkışı
$\mathrm{u}[.]$	Ayrık kontrolcü çıkışı
$e[.]$	Ayrık hata değeri
$\mathrm{G_c}(.)$	Kontrolcü transfer fonksiyonu
$\mathrm{G}_p(.)$	Sistem transfer fonksiyonu
$\mathrm{G}(.)$	İleri yön transfer fonksiyonu
$\mathrm{T}(.)$	Kapalı çevrim transfer fonksiyonu
s_i	Sistemin laplace uzayında kökleri
z_i	Sistemin ayrık uzayda kökleri
T_{on}	PWM daldasının yüksekte kalma süresi
T_{off}	PWM dalgasının düşükte kalma süresi
T	PWM periodu
D	Sinyal Oranı(kullanım oranı, duty cycle)
V_{ss}	En yüksek gerilim değeri
V_{cc}	En düşük gerilim değeri
V	Çıkışta elde edilen gerilim değeri
$f(.)$	PWM frekansı

KISALTMA LİSTESİ

SoC	System on-Chip
UART	Universal Asynchronous Receiver Transmitter
GPRS	General Packet Radio Service
A/D	Analog to Digital
LUT	Look up Table
ASSP	Application Specific Standart Product
DSP	Digital Signal Processing
ASIC	Application Specific Integration Circuit
FFT	Fast Fourier Transform
FHT	Fast Hartley Transform
HMM	Hidden Markov Model
FFD	Free Form Deformation
DPA	Differential Power Analysis
VHDL	Very High Speed Integrated Developent Circuit
HD	High Definition
LCD	Liquid Crystal Display
AES	Advanced Encryption Standard
YSA	Yapay Sinir Ağları
DA	Doğru Akım
PWM	Pulse Width Modulation
PID	Proportional-Integral-Derivative
HDL	Hardware Description Language
PLD	Programable Logic Device
SPLD	Simple Programable Logic Device
CPLD	Complex Programable Logic Device
PROM	Programmable Read Only Memory
PLA	Programmable Logic Array
PAL	Programmable Array Logic
GAL	Generic Array Logic
EECMOS	Elektriksel Silinebilir CMOS
OLMC	Output Logic Macrocell
PLL	Phase Locked Loop
IP	Intellectuel Property

RTL	Register Transfer Level
GPIO	General Purpose Input Output
SDRAM	Synchronous Dynamic Random Access Memory
USB	Universal Serial Bus
JTAG	Joint Test Action Group
TDI	Test Data Input
TDO	Test Data Output
TMS	Test Mode Select
TCK	Test Clock
TRST	Test Reset
I2C	Inter Integrated Circuit

ŞEKİL LİSTESİ

Sayfa

ÇİZELGE LİSTESİ

Sayfa

ÖZET

SAHADA PROGRAMLANABİLİR KAPI DİZİLERİ (FPGA) TABANLI ROBOT KONTROLÜ

Şirin Akkaya

Kontrol ve Ototmasyon Mühendisliği Anabilim Dalı

Yüksek Lisans Tezi

Tez Danışmanı: Doç. Dr. Haluk Görgün

Günümüzde elektronik sistemlerin tasarımında system-on-chip (SoC) teknolojisi yaygın olarak kullanılmaya başlanmıştır. Sahada Programlanabilir Kapı Dizileri (FPGA), SoC teknolojisinin en gelişmiş üyelerinden biridir. FPGA, programlanabilir mantık blokları ve bu bloklar arasındaki ara bağlantılardan oluşan, geniş uygulama alanlarına sahip sayısal tümleşik devredir.

Tez kapsamında FPGA ile ilgili yapılan başlıca çalışmalara yer verildikten sonra sıra ile FPGA ve FPGA'in gelişimini ve VHDL hakkında detaylı bilgiye yer verilmiştir. Sonrasında uygulama geliştirilecek olan deney düzeneği elemanları tanıtılmış ve FPGA ile beş eksenli bir robot kolunun pozisyon kontrolü gerçekleştirilmiştir. Robot eklemi kapalı çevrim mekanizma olarak düşünülmüş ve kontrolcü olarak FPGA tabanlı dijital PID kontrolcü tasarlanmıştır. Ayrıca FPGA ile kontrol sistemi tasarımında en çok kullanılan algoritmalar hakkında detaylı bilgiler ve akış şemaları verilmiştir. Sonuç olarak FPGA birde fazla kapalı sistemin eş zamanlı kontrolü başarılı bir şekilde gerçeklenmiştir.

Anahtar Kelimeler: FPGA, Dijital PID, PWM, Robot, VHDL

ABSTRACT

FIELD PROGRAMMABLE GATE ARRAY (FPGA) BASED ROBOT CONTROL

Şirin AKKAYA

Department of Control and Automation Engineering

MSc. Thesis

Advisor: Assoc. Prof. Dr. Haluk GÖRGÜN

Recently, system-on-chip (SoC) technology has been widely used on electronic circuit design. SoC technology, the units in the system often are carried out with the help of a hardware programming language. FPGA is one of the most devoloped members of this technology. FPGA, programmable logic blocks and interfaces between these blocks with links, has wide applications for digital integrated circuits.

In this thesis, detailed information on FPGA, the history of FPGA and VHDL, one of the hardware programming languages that needed to be known to program FPGA is presented, after demonstrating the major studies about FPGA. The following, five-axis robot arm position control is performed by the FPGA. Considered as a closed-loop mechanism for each robot joint and as a controller FPGA-based digital PID controller is designed. In addition, detailed information about the algorithms and flow charts most commonly used FPGA-based control system design is presented. As a result, more then one closed system control with FPGA simultaneously successfully implemented.

Keywords: FPGA, Digital PID, PWM, Robot, VHDL

BÖLÜM 1

GİRİŞ

1.1 Literatür Özeti

FPGA'in, düşük maliyetli olması ve tasarım sırasında kullanıcıya esneklik sağlaması sebebiyle birçok alanda kullanımı hızla artmaktadır. Kullanım alanları genel olarak, sayısal işaret işleme (digital signal processing, DSP), video işleme, biyoenformatik, bilgisayarlı görü (computer vision), ses tanıma ve işleme, tıbbi görüntüleme, ASIC prototiplendirme, yeniden yapılanabilir hesaplama (reconfigurable computing), havacılık ve savuma sistemleri, endüstriyel otomasyon, kontrol sistemleri, otomotiv ve şifreleme sistemleri olarak sınıflandırılabilir. FPGA kontrol sistemlerinde kullanılmaya başlandıktan sonra, mekanik sistemlerin kontrollünde ve özellikle robot denetim uygulamalarında da kullanılmış ve verimli sonuçlar elde edilmiştir.

Dijital işaret işleme alanında C. Dick ve F. Harris sigma-delta modülasyon tekniklerinin FPGA teknolojisi ile belirli sinyal işleme problemleri üzerinde nasıl gerçekleştirileceğine değinmişlerdir. Sigma-delta modulasyonu, sürekli zamandaki sinyali ayrık zamana çevirmek için kullanılan bir yöntemdir ve günümüzde birçok analog-dijital dönüştürücü bu yöntemi kullanır [1]. G. R. Goslin çalışmasında FPGA'in bir DSP yardımcı işlemcisi olarak kullanılmasının yanında tek başına DSP cihazı olarak kullanılmasının faydalarını incelemiştir. FPGA'in sistem performansını nasıl artırdığı ve DSP uygulamasındaki

bileşen sayısını nasıl azalttığını görtermek için bir Viterbi[1] kod çözücü ve 16 sekmeli FIR filtre uygulaması geliştirmiştir [2]. R. J. Petersen ve B. L. Hutchings çalışmalarında nicelik olarak FPGA performansını CAD araçları, cihazlar ile gerçek bir uygulama kullanarak DSP işlemciler ve ASIC'lerle karşılaştırmışlardır. Performans ölçütü FPGA, DSP işlemci ve ASIC'ler ile gerçek bir çarpan (multiplexer) performansına dayandırılmıştır. Çalışma sonucunda FPGA'in DSP işlemcilerden çok daha iyi br performans gösterdiği ve ACIS performansına yaklaştığı ve bazı durumlarda geçtiği görülmüştür [3].

I.S. Uzun, A. Amira, A. Bouridane çalışmalarında sinyal ve görüntü işlemede kullanılan FFT uygulamalarını FPGA üzerinde gerçekleştirmişler ve bu uygulamalar için ihtiyaç duyulan olan yüksek performansın ve geliştirme kolaylığının FPGA tarafından sağlandığı gözlemlenmiştir. Çalışmada radix-2, radix-4, split-radix ve fast Hartley transform(FHT) dâhil olmak üzere FFT algoritmaları üzerinde çalışılmış ve geliştirilen paralel 2-D FFT algoritması frekans uzayında görüntü filtreleme uygulamalarında çözüm olarak sunulmuştur [4]. J. Batle ve diğerleri çalışmalarında video tabanlı bilgisayarlı görü uygulamaları için yeni bir yenien yapılandırılabilir paralel mimari üzerinde çalışmışlar ve bu mimariyi FPGA/DSP tabanlı işlemcilerin iki boyutlu dizileri ile yapılandırmışlardır [5]. J.T. Jontson, K. T, Gribbon ve D.G. Bailey, üç çeşit görüntü işleme işlemi için zamanlama kısıtlaması, band genişliği kısıtlaması ve kaynak kısıtlaması gibi çeşitli kısıtlamalar ve etkin dönüşümlerle ilgili bir takım genel teknikleri FPGA üzerinde gerçeklemişlerdir [6].

FPGA tabanlı ses tanıma ve işleme alanında yapılan bazı önemli çalışmalardan S. J. Melnikoff, S. F. Quigley ve M. J. Russel, FPGA Xilinx Vixtex XCV 1000 üzerinde sürekli Hidden Markov Modeli (HMM) tabanlı ses tanımlama sistemi geliştirmiştir. Sonuç olarak modelin gerçek zamandan 75 kat daha hızlı ses tanıyabildiği görülmüştür [7]. Aynı ekip bir başka çalışmalarında ise ayrık ve sürekli Hidden Markov Model kullanarak FPGA tabanlı ses tanımlama sistemleri geliştirmişlerdir [8]. F. L. Vargas, R. Fagundes ve D. B. Junior ses tanımlama sistemleri için FPGA tabanlı Virtebi algoritması üzerine

[1] Viterbi algoritması 1967 yılında Andrew Viterbi tarafından önerilen ve günümüzde en çok kullanılan kod çözme algoritmalarından biridir. Bu algoritma en yüksek olasılık yaklaşımını kullanarak veriyi başarılı bir şekilde çözer.

çalışmışlardır. Ayrıca genellikle ses tanıma sistemlerinde kullanılan ve olasılıksal bir model olan Hidden Markov Model(HMM) için yeni bir FPGA tabanlı yaklaşım geliştirmişlerdir [9].

FPGA'in ayrıca tıbbi görüntüleme alanında da kullanımı giderek yaygınlaşmaktadır. J. Jiang, W. Luk ve D. Rueckert medikal görüntü işlemede kullanılan serbest biçimli deformasyonları (FFD) destekleyen FPGA tabanlı sistemler üzerine çalışmışlardır. Serbest biçimli deformasyonlar üç boyutlu deforme olabilir objeleri modellemek için kullanılan B-spline[1] tabanlı güçlü bir araç kutusudur [10]. T. Yokota ve diğerleri medikal görüntü işleme için FPGA tabanlı yoğunlaştırılmış index filtreleme (concentration index filtering) yapan özel bir hesaplama makinesi üzerine çalışmışlardır [11]. M. Leeser, S. Coric, E. Miller ve H. Yu bilgisayarlı tomografide kullanılan paralel ışın geri yansıtma algoritmasını (paralel-beam back projection algorithm) FPGA üzerinde gerçeklemişlerdir [12].

R. Andraka CORDIC[2] algoritması ile çözülen belirli fonksiyonları FPGA kullanarak nasıl gerçeklendiği ve ilgili algoritmaların nasıl çalıştığı üzerine bir çalışma yayınlamıştır [13].
K. Tiri ve I. Verbauwhede, güvenli bir DPA dirençli şifreli yazı (kripto) işlemci gerçekleştirmek için yeni bir tasarım tekniği oluşturmuşlardır. FPGA ve ASIC'lerin standart hücreleri bu yöntemi gerçekleştirmek için uygun görülmüştür. Bu yöntemin en önemli avantajı güvenli bir şifreleme biriminin nasıl oluşturulacağın detaylarının tasarımcı tarafından saklanabilmesidir [14].

S. Brown ve J. Rosa ticari olarak temin edilebilen yüksek kapasiteli alan programlanabilir cihazları araştırmışlardır. Bu cihazları basit ve karmaşık programlanabilir lojik devreler ve alan programlanabilir kapı dizileri (FPGA) olarak üç başlık altında sınıflandırılmış ve bu cihazların en çok kullanılanların mimari yapıları incelenmiş ve her tip cihaz için uygulamalı örneklere yer verilmiştir [15].

M. Gokhale, J. Stone, J. Arnold çalışmalarında akış odaklı hesaplamayı (stream-oriented computation) destekleyen FPGA tabanlı paralel bilgisayarlar geliştirmek için kullanılan

[1] B Spline, bilgisayar çizimleride eğrileri tanımlayabilmek için kullanılan matematiksel bir fonksiyondur.

[2] CORDIC algoritması hiperbolik ve trigonometrik fonksiyonların çözülmesinde kullanılan bir algoritmadır.

Stream-C sisteminin dil yapılarını, derleyici özelliklerini ve yazılım donanım kütüphanelerini incelemişlerdir. Çalışmada Stream-C ve VHDL kodları ile gerçekleştirilen uygulamalar karşılaştırılmıştır. Sonuç olarak Stream-C ile oluşturulan devrelerin 3 kat daha fazla yer kapladığı ve saat sinyalinin yarısı hızında çalıştığı ve zaman açısından karşılaştırıldığında Stream-C ile uygulamaların çok daha hızlı geliştirildiği için on kat daha verimli olduğu görülmüştür [16].

FPGA ile ülkemizde yapılan bazı çalışmalara yer verirsek, E. Cesur, gerçek zamanlı bir hücresel sinir ağı yapısının tasarımı ve bu yapıyla yüksek çözünürlüklü, basit taramalı video işaretini işleyebilen bir ayrık zamanlı Gabor benzeri HSA filtresinin FPGA ile tasarımı ve gerçeklemesi üzerine bir çalışma yapmıştır [17]. N. Yıldız, bir hücresel sinir ağı emilatörünün tasarlanması ve FPGA üzerinde gerçeklenmesi üzerine bir çalışma yapmıştır. Tez çalımasında full HD 1080p@60 video görüntlerini işleyebilen gelişmiş bir gerçek zamanlı sayısal CNN mimarisi önerilmiş, VHDL dilinde kodlanmış ve iki ayrı FPGA üzerinde gerçeklenmiştir [18]. K. Kayaer hücresel sinir ağları yapısının FPGA platformu ile dijital emüsyonu ile kullanılabilecek yeni bir emülatör yapısı tasarlamış ve bir kenar belirleme algoritması ile tasarımını test etmiştir. Sonuç olarak oluşturulan yapı basit bir taramalı video işaretlerini gerçek zamanlı olarak işleyebilecek kadar hızlı olduğu görülmüştür [24]. Z. Peker FPGA ile veri gizleme algoritmaları üzerine bir çalışma yapmış ve Spartan 3E FPGA geliştirme kartı üzerinde belirlenen bir metin gizleme algoriması ile steganografi uygulamaları geliştirilerek uygulama sonuçları LCD ekran üzerinde gösterilmiştir [19]. U. Yazkurt ileri şifreleme standart algoritmasını (AES) içeren bir STM-1 verici-alıcı sistemi tasarlamaış ve Xilinx XCV1000E FPGA üzerinde tasarımını gerçeklemiştir. Uygulamada 128 bit anahtar uzunluklu AES blok şifreleme algoritması, 128 bit şifre geribesleme (CFB-128) çalışma modu kullanılarak paralel bir mimari gerçekleştirilmiştir [25]. T. Çayır çalışmasında medyan filtre uygulaması üzerinde durarak, filtrenin yapısı, uygulanış şekli, görüntü üzerindeki etkileri, avantaj ve dezavantajları gibi birçok noktaya değinilmiş, yapının FPGA üzerinde temel bir donanım algoritması kullanılarak yüksek çözünürlüklü gerçek zamanlı görüntüler kullanarak uygulaması gerçekleştirerek sonuçları monitör üzerinden gözlemlemiştir [26]. N. Isakova, gerçek zamanlı stereo görme sisteminin FPGA üzerinde tasarımı ve gerçeklemesi üzerine bir çalışma yapmıştır. Stereo Görme, 3 boyutlu nesne algılama,

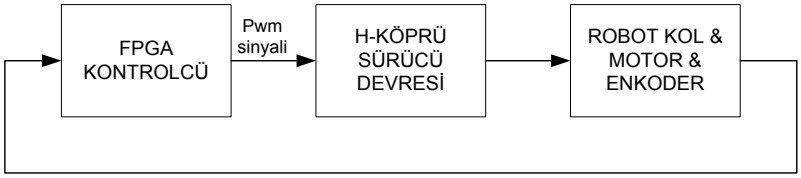
robotik, navigasyon, yol planlama, haritalama ve lokalizasyon, gibi birçok gerçek zamanlı uygulamalarda kullanılması nedeniyle bilgisayar görü alanında en önemli konulardan biridir. Stereo kayıklık paralel işlem gerektiren bir uygulama olduğu için gerçek zamanlı ve donanımsal olarak FPGA'de uygulanmıştır [21]. M. F. Özçelik görüntü işleme algoritmalarının FPGA üzerinde gerçeklenmesi ile ilgli bir çalışma yapmıştır. FPGA kartı üzerinde gerçek zamanlı olarak görüntüde renk değiştirme, morfolojik açma ve kapama, kırmızı renkli nesne takibi, ten rengi tanıma ve sobel filtresinin resim üzerine uygulanması gibi pek çok tasarım gerçekleştirmiştir [22]. T. Taş, ayrık sistemlerin tanımlanmasında kullanılan etkin bir matematiksel ve grafiksel modelleme yöntemi olan bulanık petri ağları ile modellenmiş demiryou trafik sisteminin sayısal devresinin tasarlanması ve uygulamasının FPGA üzerinde gerçeklenmesi üzerinde bir çalışma yapmıştır [23]. N. Yılmaz, yonga üzerinde eğitilebilir bir YSA yapısını, Altera FPGA devreleri ile gerçekleştirmiştir. Çalışmada XOR problemi ve bir sensör doğrusallaştırma problemi ile çalışılmış ve sabit noktalı sayı sistemi tabanlı ve hatanın geri yayılımı algoritması ile eğitilen bir YSA yapısı kullanılmıştır. Öğrenme kuralı olarak delta bar delta[1] kuralı seçilerek ve bu uygulamalar Altera QUARTUS II FPGA tasarım programı ve MATLAB ile tasarlanmış ve simüle edilmiştir. Bunlara ek olarak, basitleştirilmiş YSA yapısı ile XOR problemi Altera Cyclone EP1C6Q240C8 FPGA tabanlı UP3 geliştirme kartı ile gerçekleştirilmiştir [27]. E. Alaer 16-bit mikroişlemcinin FPGA mimarileri kullanılarak tasarımı üzerine bir çalışma yapmıştır. Tasarlanan mikroişlemci 16-bit adres yolu ve 16-bit veri yolu, 16 tane genel amaçlı saklayıcı, program sayıcı ve 3-bit durum saklayıcı içermektedir. Her bir komut 16-bit kelime uzunluğuna sahiptir ve mikroişlemci toplama, çıkarma, çarpma, bölme gibi aritmetik komutları; AND, OR, NOT, XOR gibi lojik komutları; yükleme ve saklama gibi yer değiştirme komutlarını yürütebilmektedir [28]. E. Öztürk de benzer bir çalışma yaparak FPGA platformu ile 16-bit mikroişlemci tasarımı gerçekleştirmiştir. Bu çalışmada ise 16 bit veri yolu, 1 adet akümülatör, 8 adet genel amaçlı yazmaç, 3 adet indeks yazmacı, 2 adet çarpım birimi yazmacı, MAR, MBR, ve IR olmak üzere toplam 17 adet yazmaca, 8 Kilobyte kapasiteli

[1] Delta bar delta, yapay sinir ağlarında, çok katmanlı algılıyıcılarda bağlantı ağırlıklarının yakınsama hızını artırmak için kullanılan sezgisel bir öğrenme algoritmasıdır.

bir bellek birimine, 33 adet komuttan oluşan bir komut setine ve 65.95 MHz maksimum çalışma frekansına sahip bir mikroişlemci (HSEO16) tasarlanmıştır [29].

FPGA ile endüstiriyel otomasyon, robotik ve kontrol sistemleri alanlarında yapılan çalışmaları incelersek, E. Monmasson FPGA tasarım tekniklerinin endüstriyel kontrol sistemleri uygulamalarına nasıl uyarlanabileceği konusunda genel bir çalışma yayınlamıştır. Çalışmanın ilk bölümünde FPGA'in genel yapısı ve araç kutuları incelenmiş sonra FPGA tabanlı kontrolcü tasarım kurallarına değinilmiştir. Bu kurallar algoritma geliştirme (algorittm refinement), modülerlik ve A3 metodu başlıklarında incelenmiş ve son olarak da önerilen tasarım yöntemleri kullanılarak FPGA üzerinde uygulamalar geliştirilmiştir [30]. K. A. Toker çalışmasında donanımsal ve yazılımsal olarak rüzgâr türbinleri için değişik kontrol sistemleri incelenmiştir. En uygun rüzgâr türbin kontrol algoritması kullanılarak bir FPGA/GPRS tabanlı bir kontrol uygulaması gerçekleştirilmiş ve test edilmiştir. Çalışmada algılayıcılardan gelen veriler kablosuz birimler ile FPGA ve GPRS birimlerine iletilmiştir. FPGA platformuna gömülen ve VHDL programlama dilinde yazılan bir rüzgâr türbini kontrol algoritması ile de türbin motorları kontrol edilmiştir [20]. U. Serdar çalışmasında fırçasız dogru akım motoru için pozisyon ve akım kontrol sistemleri tasarlanmış ve tasarımlarını FPGA üzerinde gerçeklemiştir. Deneysel sonuçlar güdümlü bir füzenin kanat tahrik sistemindeki fırçasız doğru akım motorları kontrol edilerek elde edilmiştir. Motor torkunu kontrol etmek için akım kontrolcü tasarlanmış ve FPGA üzerinde gerçelenmiş, sonra pozisyon komutlarını yerine getirmek için pozisyon kontrolcü tasarlanmıştır. Akım kontrolcü, pozisyon algılayıcı ara yüz ve haberleşme birimlerini (UART, Spacewire) birbirine bağlamak ve birimleri konfigüre etmek için FPGA'de yazılımsal işlemci kullanılmıştır. Buna ek olarak pozisyon kontrolcü yazılımsal işlemci üzerinde çalıştırılmıştır. Kontrol algoritmalarını, güç çevirici devresini çalıştırmak ve kullanıcı bilgisayarı ile haberleştirmek için FPGA tabanlı bir elektronik kart tasarlanmış ve üretilmiştir [32]. A. Mamur çalışmasında kararlı ve yük değişimlerinden etkilenmeyen bir çıkış gerilimi üreten DA-DA dönüştürücü devresi tasarlanmış, denetleyici ve PWM sinyal üreteci olarak Xilinx XC3S500E-4FG320 FPGA kullanmıştır. Bu FPGA'la 24.4 kHz frekansa sahip ve değişken görev periyotlu PWM sinyali üretilmiş ve IGBT anahtarın sürücü devresine uygulanmıştır [33]. F. Piltan ve diğerleri çalışmalarında robot manipulatorleri için FPGA

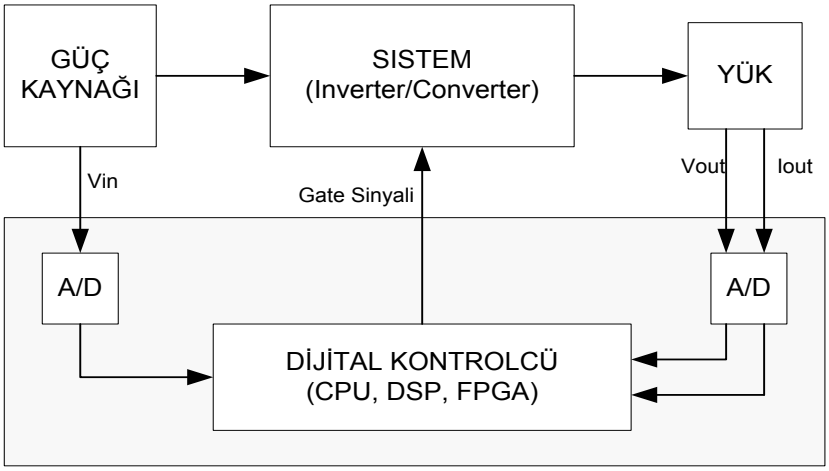
tabanlı kayan kipli kontrolcü tasarlamışlardır. Çalışmada önce uygulamada kullanılacak olan PUMA-560 robot manipulatörü denklemlerine yer verilmiş sonra PD Matlab tabanlı kayan kipli kontrolcü, PID Matlab tabanlı kayan kipli kontrolcü ve FPGA tabanlı kayan kipli kontrolcü test edilmiştir. Sonuç olarak FPGA tabanlı kayan kipli kontrolcü ile kabul edilebilir bir performansa karşın küçük çip boyutuna ve daha yüksek uygulama hızına ulaşılmıştır. Çatırdama problemi ve hata varlığında kayda değer ölçüde bu olguları azalttığı gözlemlenmiştir. Kontrolcünün çok çeşitli robot manipülatörlerini yüksek örnekleme oranları ile kontrol edebileceği sonucuna varılmıştır [31]. Meshram, U Devendra, and R. Harkare eğitici amaçlı hazırladıkları yayında beş eksenli robot kolun FPGA tabanlı kontrolü üzerine çalışmışlardır. Temel bir robot kontrol problemi için sistemi FPGA tabanlı yazılımsal kontrolcü, H-köprü sürücü devresi, robot kol modeli ve döner enkoder bulunduran DC motorlar olarak parçalamışlar ve her parçayı incelemişlerdir. Bir robot kontrol sisteminin en basit şekliyle blok diyagramını aşağıdaki gibi ifade etmişlerdir [34].



Şekil 1.1 Robot Kol Modeli

A.Z. Mansoor, M. R. Khalil ve O. A. Jasim robot kolu kontrolü için FPGA tabanlı soft-core işlemci kullanarak doğru akım servo motorları kontrolü üzerine çalışmışlardır. Bu işlemci motor hareketi ve yönü için gerekli olan PWM sinyalini oluşturulmuş ve servo motorların pozisyon ve hızı kontrol edilmitir. Uygulamada Xilinx Spartan-3E FPGA kullanılmış ve PicoBlaze soft-core işlemci VHDL dili kullanarak programlanmıştır. FPGA'in paralel çalışması motor kontrolü yaparken aynı zamanda başka işlemler yapmasına da izin verdiği için, ayrıca motor sayısı daha fazla olan sistemlerde FPGA içine başka bir PicoBlaze soft-core işlemci eklenmiştir. Uygulama sonucunda PicoBlaze soft-core işlemci kullanılarak yapılan kontrol çalışmasında robot eklemlerindeki servo motorların doğru bir şekilde kontrolü gerçekleşmiştir [35]. M. Xu, W. Zhu, Y. Zeu dinamik yeniden yapılandırılabilir FPGA ve yeniden yapılandırılabilir akış konusunda çalışma yapmış ve hazırlanan kontrolcü sistemi modüler robot üzerinde

gerçeklenmiştir. Kontrol birimi, iletişim, kontrol, algılama, hareket ve iletim görevlerini yerine getirmektedir [36]. K. Sugahara, S. Oida, T. Yokoyama güç elektroniği uygulamalarını dijital kontrolü için yüksek performanlı FPGA kontrolcü tasarımı üzerine çalışmışlardır. Uygulamada Altera Cyclone III FPGA, 1 Mhz A/D dönüştürücü, 12x4 I/O (3.3V, 5.0V), 40 dijital I/O (3.3V), RS 232C I/F, SD kart yuvası ve ethernet I/F kullanılmıştır. FPGA, tetikleme sinyaleri yanında, A/D dönüştürücü saat sinyali, kanal seçimi ve gecikme sürelerini gibi işlemleri de gerçekleştirmiştir. Sistemin bu esneklik özelliği tasarımın birçok güç elektroniği uygulamasında kullanılabileceğini gösterir. Aşağıda çalışmada kullanılan güç elektroniği sistemleri dijital kontrolü için blok diyagramına yer verilmiştir [37].



Şekil 1.2 FPGA tabanlı güç elektroniği sistemi

U.Meshram ve diğerleri FPGA kullanarak robot kol kontrolü üzerine çalışma yapmışlardır. Robot kolun eklemlerinde yer alan altı adet adım motorunun FPGA ile hareket kontrolü yapılmıştır. Konrolcü sistemi Spartan II FPGA'in VHDL dili ile programlanması ile oluşturulmuştur. Kontrolcü çıkışı motorları sürmek için yeterli gerilimi sağlayamadığı için motor sürücü devreleri kullanılmıştır. Sonuç olarak FPGA'in işlem süresini artırmadan aynı anda altı adım motorunu sürebildiği görülmüştür [38]. C. Tsai, F. Tai ve S. Hsies gerçekleştirdikleri farklı bir FPGA tabanlı kontrol uygulamasında, iki tekerlekli non-holonomik gezici robotun düşük hızlarda anlık takip ve düzeltme işlemleri için FPGA SoPC tabanlı kinematik kontrolcü tasarlamışlardır. Kinematik kontrolcü Lyapunov kararlılık kriterleri dikkate alınarak tasarlanmış ve önerilen hareket

kontrol yöntemi Altera FPGA üzerinde gerçeklenmiştir. Önerilen kontrol yöntemi, mobile robotun belirlenen yörüngelerde etkili ve kullanışlı olduğunu göstermiştir [39].

Literatür araştırmasını biraz daha özelleştirip FPGA ile gerçeklenen PID algoritmalarını incelersek, M. Sultan ve diğerleri çalışmalarında FPGA tabanlı PID algoritması gerçekleştirmiştir. Oluşturulan algoritmada dağıtık aritmetik (Distributed Aritmatic) kullanılmış ve çalışmada DA tabanlı PID olarak işlenmiştir. Dağıtık aritmetik, LUT (Look-up Table) tabanlı şemalar oluşturmak için kullanılan bit düzeyinde dizi hesaplama algoritmasıdır. Hesaplamalar sonucunda PID'nin üç terimi içinde üç ayrı LUT tablosu oluşturulmuş ve elde edilen tasarım sıcaklık kontrol sistemine uygulanmıştır. Bu tasarım ayrıca FPGA içinde hafıza birimleri oluşturulabileceği ve esnekliği sayesinde antiwindup koruma ve uyarlamalı şema algoritmaları ile de birleştirilebileceğini göstermiştir [40]. B. R Mutlu ve diğerleri doğru akım motorunun kayan kipli konum kontrolünün FPGA ile gerçeklenmesi üzerine yaptıkları çalışmada kayan kipli kontrolcüye ek olarak bir PID kontrolcü ve bir komut ileri beslemeli kontrolcü tasarlamışlardır. Kontrolcülerin başarı kriterini, kullandıkları kaynak miktarları ve asgari döngü zamanları ile karşılaştırmışlardır. Tasarımı gerçekleştirmek için Xilinx Virtex5 FPGA yongası barındıran ML 505 gelirtirme kartı kullanılmıştır. Sonuç olarak kayan kipli denetleyici, PID denetleyiciye oranla %50 oranla daha fazla kaynak harcayarak FPGA yongası üzerinde gerçekleştirilebildiği görülmüştür [41]. M. Abdelati çalışmasında FPGA üzerinde PID kontrolcü yapılandırmak için gerekli olan birimleri ana hatları ile incelemiştir. Bu birimleri test etmek ve gerçekleştirmek için deneysel bir platform üzerinde hız ve pozisyon olmak üzere iki ayrı pid kontrolcü tasarlamıştır. Çalışmada PID kontrolcünün gerçeklenmesi için analog giriş birimi olarak 8 bit seri analog-dijital dönüştürücü, analog çıkış birimi olarak 8 bit dijital analog dönüştürücü, PWM üretici, optik encoder, kullanıcı arayüzü, dijital filtreler birimleri kullanılmıştır. Deneysel çalışma olarak da hazırlanan test düzeneğinde dc motorun PID kontrolcü ile hız ve pozisyon kontrolü bu birimler kullanılarak gerçeklenmiştir [42].

1.2 Tezin Amacı

Robot manüpülatörlerinin eksen sayısının artmasıyla birlikte, robotun hareket yeteneği de artmiş, fakat bununla beraber robot manüpülatörünün denetimi de zorlaşmış ve

denetim algoritmaları daha karışık hale gelmiştir. Bu nedenle daha hızlı işlem yapabilen sistemlere olan ihityaç artmış ve FPGA gibi paralel işem yapabilen sistemler avantajlı hale gelmiştir. Bu projede 5 eksenli bir robot kolunun bilinen mikrokontrolcülerden farklı olarak FPGA kullanılarak kontrol edilmiştir.

FPGA (Alan Programlanabilir Kapı Dizileri) istenen fonksiyona göre iç yapısı kullanıcı tarafından değiştirilebilen donanım programlanabilir entegre şeklinde tanımlanabilir. Tasarım sırasında büyük esneklik sağlaması ve paralel işlem yapabilme kabiliyeti sebebiyle FPGA'ler günümüzde oldukça yaygınlaşmıştır. Ülkemizde Aselsan A.Ş. başta olmak üzere FPGA kullanımı kurumlarda giderek yaygınlaşmaktadır. Buna rağmen FPGA programlama hakkında büyük bir kaynak sıkıntısı yaşanmaktadır. Bu çalışma ile bir yandan da konu ile ilgili ayrıntılı bilgi vererek bir miktar bu eksikliği kapatmayı amaçlanmıştır.

FPGA kullanmanın diğer nedenlerinden biri de yazılımsal ve donanımsal olarak kolay programlanabilir olması, kapasitesi, hızı ve birçok fonksiyona izin vermesidir. Kompakt ve güvenilirdir, maliyeti düşüktür, kolay ve hızlı bir şekilde yeniden programlanabilmektedir. Bu çalışmada FPGA programlamak için HDL (Donanım Programlama Dili) dileri ailesinden IEEE standartı olan VHDL dili şeçilmiştir.

Sayısal tabanlı bir yonga olduğu için, FPGA içinde gerçeklenmek istenen algoritmalar ayrık zaman uzayında incelenmelidir. Bu çalışma içersinde kontrol sistemlerin de kullanılan en temel algoritmarın FPGA ile nasıl gerçekleneceğine ve ilgili algoritmaların akış diaygramlarına yer verilmiştir.

Bu tezin uygulama bölümünde beş eksenli bir robot manipülatörünün ayrık PID kontrolcü ile kontrolünün gerçekleştirilmiştir. PID kontrolcü günümüzde endüstriyel sistemlerde en çok kullanılan kontrolcüdür. Proses kontrol, robotik, otomasyon, havacılık ve uzay, üretim ve iletim sistemleri başta olmak üzere birçok uygulamada PID kontrolcü kullanılır [40]. İncelenen sistemde robot kolun kontrolü her bir kolun birbirinden bağımsız kapalı çevrim sistemler olduğu kabulüne dayanmaktadır. Sistemde robot kolunu hedeflenen yörüngeye ya da bir noktadan diğer noktaya gitmesi eyleminde hareket etmesi gereken manüpülatör eksenine ait motoru sürmek için PWM (Darbe Genişlik Modülasyonu) sinyali FPGA kontrolcü tarafınfan üretilir. Bu sinyaller

robotun eklemlerinde yer alan motorların kontrol uçlarına iletilir, böylece robotun istenilen koordinatlarda konumlanması sağlanır. Çalışmada her bir kolun birbirinden bağımsız kapalı çevrim sistemler olduğu kabulünde robot eklemlerinin model tabanlı doğrulama yöntemi ile ayrı ayrı yaklaşık modelleri elde edilmiş ve bu modeller üzerinde kontrol algoritması gerçeklenmiştir.

Tez çalışmasını devamında Bölüm 2'de FPGA'in gelişimi hakında bilgi verilmiş, Bölüm 3'de FPGA programlamak için kullanılan VHDL dilinin yazpısı ve özellikleri incelenmiş, Bölüm 4'te deney düzeneği tanıtılmış ve Bölüm 5'te kontrolcü tasarımı anlatılarak Bölüm 6'da sonuçlara yer verilmiştir.

1.3 Hipotez

Teknolojinin hıza gelişmesiyle daha hızlı çalışan ve aynı anda birden fazla sistemleri kontrol edebilen birimlere olan ihtiyaç da artmaktadır. Buna paralel olarak yarı iletken teknojosiyle üretilen ürünler bu ihtiyacı karşılamak için gelişme göstermektedir. FPGA bu gelişim basamağını en son ve en gelişmiş elemanlarından biridir. Diğer programlanabilir lojik sistemlere karşılaştırıldığında daha hızlı çalışması ve paralel işlem yapabilme gibi özellikleri FPGA'in ön plana çıkmasını sağlar.

FPGA tabanlı sistemlerin kullanımı ülkemizde giderek atmaktadır. Buna rağmen FPGA tasarımcılarının sayısı çok azdır ve konu ile ilgili kaynak sıkıntısı bulunmaktadır. Bu çalışma gerek FPGA'in gelişimi gerekse FPGA programlamak için kullanılan VHDL dili hakkında temel seviyede bilgi vermektedir. Aynı zamanda FPGA tabanlı kontrol sistemleri tasarımı ile ilgili bilinmesi gereken en temel algoritmaların nasıl oluşturulacağı hakkında geniş bilgi vermektedir. Çalışmanın konu ile ilgilenenler için uygulamalı bir kaynak oluşturması amaçlanmıştır.

BÖLÜM 2

FPGA

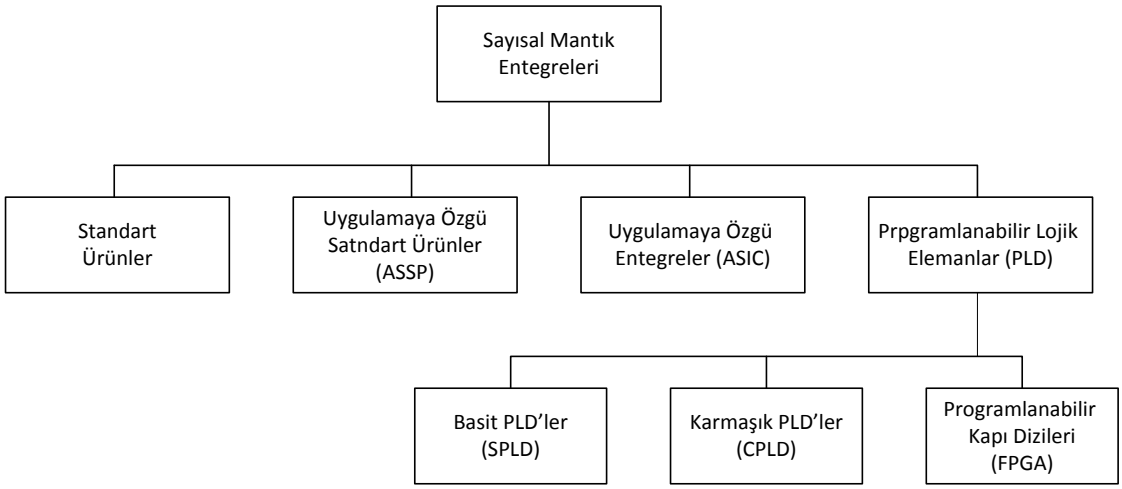
FPGA, Türkçe “Sahada Programlanabilir Kapı Dizileri” anlamına gelen “Field Programmable Gate Array” ifadesinin kısaltmasıdır [43]. FPGA, programlanabilir mantık blokları ve bloklar arasındaki ara bağlantılardan oluşan, geniş uygulama alanlarına sahip bir çeşit sayısal tümleşik devredir. Başka bir tanım da, üretimden sonra istenilen fonksiyona göre donanım yapısı kullanıcı tarafından değiştirilebilen bütünleşik devre şeklinde yapılabilir. FPGA 1984 yılında Xininx şirketi kurucularından Roos Freeman ve ekibi tarafından icat edilmiştir. FPGA, dijital tasarımlarda kullanılan farklı lojik kapıların milyonlarcasını tek bir entegre üzerinde barındırır ve bu lojik kapılar istenildiği gibi programlanarak, istenilen herhangi bir dijital tasarım bu elemanlar üzerinde gerçeklenebilir.

Bu bölümün devamında FPGA’in gelişimi hakkında, ilk programlanabilir lojik elemanlardan başlayarak, basit ve karmaşık PLD’ler anlatılmıştır. FPGA’in iç yapısı yapısı, çalışma şekli ve genel özellikleri anlatıldıktan sonra genel kullanım alanları sıralanmıştır.

2.1. FPGA ‘in Gelişimi

1970’li yılların başlarında üretilmeye başlanan programlanabilir mantık devreleri önce, belirli bir fonksiyonu gerçekleştirmek üzere özel olarak üretilen tek kullanımlık bütünleşik devreler şeklindeydi. Bu bütünleşik devrelerin en küçüğü birkaç mantık kapısından meydana gelmekte olup, çok sayıda entegreyi bir arada kullanmak hem performans kaybına, hem yüksek maliyete, hem de karmaşaya neden olmaktaydı. Bu

problemlere çözüm üretmek için tasarımcılar işlenecek fonksiyonu, Karnough haritaları ve Quine-McCluskey sadeleşme yöntemlerini kullanarak “toplamların çarpımı” şekline getirerek ve çarpım ifadelerini VE kapıları, toplam ifadelerini de VEYA kapıları kullanarak oluşturan yeni lojik devreler geliştirmeye başlamışlardır. Bu gelişimle beraber sayısal mantık devrelerinin genel sınıflandırması aşağıdaki gibi yapılabilir.



Şekil 2.1 Programlanabilir lojik elemanlar [43]

FPGA, Sayısal Mantık Entegreleri sınıflandırmasının “Programlanabilir Lojik Elemanlar” bölümünün alt ve en gelişmiş bileşenlerinden biridir.

Programlanabilir lojik elemanlar olarak isimlendirilen elemanların ‘programlanabilir’ olarak nitelendirilmesinin nedeni; entegre olarak üretimden sonra, içyapısının yapılmak istenen işleme göre şekillendirilebilmesidir. Programlama işleminin amacı ise; lojik eşitlikleri daha az bileşen ile oluşturmak ve kablolar veya baskı devre ile elemanları birbirine bağlamak şeklindeki fiziki bağlantılara ihtiyaç kalmadan lojik devreleri gerçekleştirebilmektir [44].

Programlanabilir lojik elemanlar, gelişim sıralarına göre Şekil 2.1’ de gösterildiği gibi ‘Basit PLD’ler’, ‘Karmaşık PLD’ler’ ve ‘Programlanabilir Kapı Dizileri’ şeklinde sınıflandırılabileceği gibi aynı zamanda iç yapılarına göre de VE/VEYA kapılarının oluşturduğu devreler olan ‘programlanabilir bileşik devreler’ ve flip-flop ve kapı devreleri ile birlikte flip-flop’ların programlandığı ‘programlanabilir ardışık devreler’ şeklinde de sınıflandırılabilir. FPGA programlanabilir ardışık devreler grubunda yer alır.

Programlanabilir lojik elemanların gelişimi aşağıda özetlenmiştir;

2.1.1. Basit PLD'ler (SPDL)

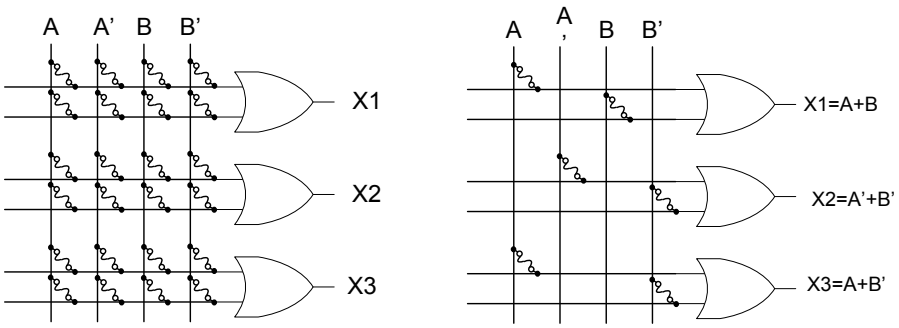
İlk basit programlanabilir mantık devreleri 1970'lerin başlarında üretilmeye başlanmıştır ve temelde VE/VEYA kapılarından oluşmaktadırlar. Farklı kaynaklarda farklı isimlerle sınıflandırılan bu devreler temel olarak aşağıdaki gibi listelenebilir;

- PROM: Programlanabilir Yalnızca Okunabilir Belekler (Programmable ROM)
- PLA: Programlanabilir Lojik Dizileri (Programmable Logic Array)
- PAL: Programlanabilir Dizi Lojik (Programmable Array Logic)
- GAL: Genel Kapsamlı Dizi Lojik (Generic Array Logic)

Programlanabilir lojik elemanların içyapılarında 'programlanabilir diziler' denilen yapılar bulunur. Bu yapılar her bir kesişim noktasında sigortalı bağlantı bulunan, satırlar ve sütunlar şeklinde düzenlenen iletkenler kombinasyonundan meydana gelirler. Programlama işlemi sütun ve satırların kesişme noktalarında bulunan sigortaların istenilen lojik işlemi gerçekleştirmek için olduğu gibi bırakılarak iletimde kalması ya da attırılarak yalıtıma geçmesi şeklinde olur. Bir sonraki aşamada tüm lojik fonksiyonlar belirli sadeleştirme işlemlerinden geçtikten sonra çarpımların toplamı formundaki eşitlikler (minterimlerin toplamı) şeklinde ifade edilebileceği kuramı kullanılarak istenilen lojik fonksiyondaki çarpımları ve toplamları ifade etmek için bağlantılar VE/VEYA kapılarından geçirilir ve programlama işlemi tamamlanmış olur.

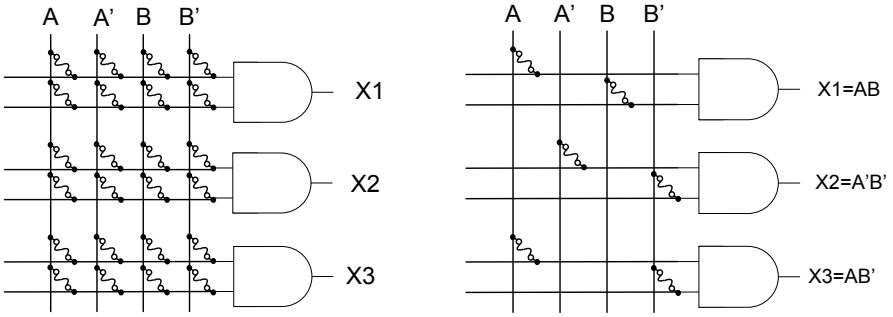
Programlanabilir elemanlarda yaygın olarak kullanılan diziler 'VEYA Dizisi' ve 'VE Dizisi' dir. Bu diziler aşağıda kısaca açıklanmıştır.

VEYA Dizisi (OR Array): Giriş değişkenlerinin çıkışa aktarılması için sütun ve satırlar arasında sigortaların şekillendirildiği ve çıkış fonksiyonu oluşturulmasında 'VEYA' kapılarının kullanıldığı programlanabilir lojik elemanlardır. Satır ve sütunlardan oluşan matris şeklindeki düzeneklerde giriş değişkenleri sütunlara uygulanır. Çıkışta oluşturulacak fonksiyona göre, sütunlar ve satırlar arasındaki bilgi geçişini sağlayacak sigortalar attırılır veya olduğu gibi bırakılır. Attırılan sigortaların tekrar eski haline dönmesi mümkün değildir. Şekil 2.2'de programlanmamış ve programlanmış VEYA dizileri gösterilmiştir.



Şekil 2.2 a) Programlanmamış VEYA Dizisi - b) Programlanmış VEYA Dizisi

VE Dizisi (AND Array) : Giriş değişkenlerinin çıkışa aktarılması için sütun ve satırlar arasında sigortaların şekillendirildiği ve çıkış fonksiyonu oluşturulmasında 'VE' kapılarının kullanıldığı programlanabilir lojik elemanlardır. Çıkışta oluşması istenmeyen değişkenleri temsil eden sigortalar attırılır ve yalnızca çıkış fonksiyonunda bulunan değişkenlere ait sigortalar sağlam bırakılır. 'VE' dizisinde bulunan sigortalar da yalnızca bir kere programlanabilir yapıdadırlar. Şekil 2.3'de programlanmamış ve programlanmış VE dizileri gösterilmiştir.



Şekil 2.3 a) Programlanmamış VE Dizisi - b) Programlanmış VE Dizisi

2.1.1.1 PROM: Programlanabilir Yalnızca Okunabilir Belekler

Genel kullanım alanı bulan ilk programlanabilir lojik devre PROM (Programmable Read Only Memory) devreleridir. PROM devreleri bir defa programlanabilirken EPROM ve EEPROM gibi PROM'lar silinip tekrar programlanabilir özelliğe sahiptirler.

PROM üzerine kaydedilen bilgiler enerji kesildiğinde silinmez. Üzerine program kodlarını veya verileri yazmak için PROM programlayıcı cihazlara ihtiyaç vardır. Bu bellek elemanının yapısında küçük sigorta telleri bulunur. Hepsi sağlam durumda bulunan sigortalar '1' i temsil eder. Yazılacak olan bilginin bit düzeninde '0' lara karşılık

gelen hücredeki sigortar küçük bir elektrik akımı ile attırılır ve böylece PROM programlanmış olur.

PROM'ların içyapısı incelendiğinde, sabit 'VE' dizisi ve programlanabilen 'VEYA' dizisinden oluştuğu görülür. PROM'lar sabit 'VE' kapılarının neden olduğu kısıtlamalar nedeni ile programlanabilir lojik elemanlardan daha çok adreslenebilir bellek olarak kullanılırlar. Bir PROM içersinde, bir mikroişlemci programı, basit bir algoritma veya durum makinesi kodu yüklenebilir. PROM'lar sınırlı sayıda giriş/çıkışı bulunan herhangi bir kombinasyonal devrenin gerekleştirilmesi için uygun cihazlardır [44].

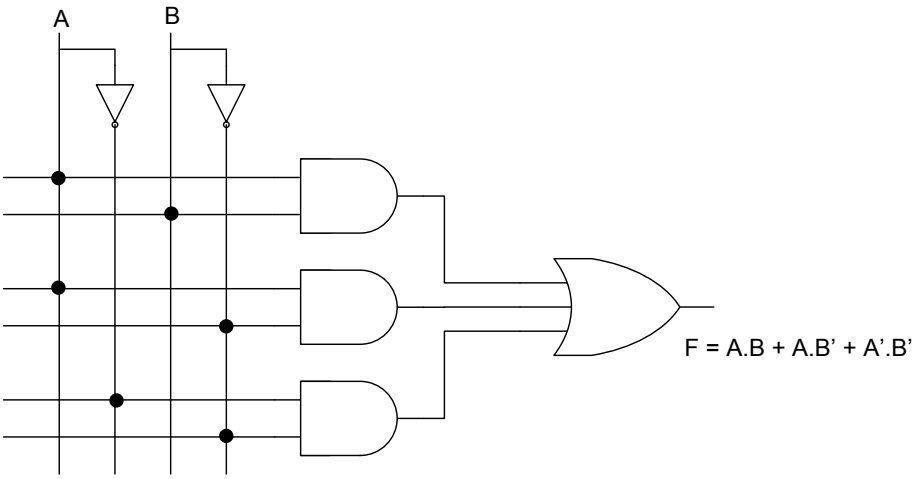
EPROM ve EEPROM devreleri ise kullanıcı tarafından programlanabilen devrelerdir. EPROM'lar bellek hücrelerine elektrik sinyali uygulanarak programlanır ve kaydedilen bilgiler enerji kesildiğinde silinmez. EPROM içersindeki programın silinmemesi için entegrenin üzerindeki cam pencereli kısım ışık geçirmeyen bantla kapatılmalıdır.

EPROM belleğe yeniden yazma işlemi yapabilmek için ise, cam bölmenin üzerindeki bant kaldırılarak entegreyi ultraviloye altında belirli bir süre tutmak gerekir. Bu şekilde içindeki bilgiler silinebilir. Böylece tekrar programlanabilir hale gelen ürün tekrar tekrar farklı programların denenmesi ve cihazın çalıştırılması için kullanılabilir. Silme işlemi esnasında belirli şartlara dikkat edilmemesi halinde silinebilme ömrü kısalan entegreler bir süre sonra kullanılamaz hale gelebilir.

EEPROM devreleri ise elektriksel olarak yazılabilen ve silinebilen bellek elemanlarıdır. EEPROM'u besleyen enerji kesildiğinde üzerindeki bilgiler kaybolmaz. Bilgilerin silinmesi ve yazılması için özel silme ve yazma cihazlarına gerek yoktur. Programlayıcılar üzerinden gönderilen elektriksel sinyallerle programlanır.

2.1.1.2 PAL: Programlanabilir Dizi Lojik (Programmable Array Logic)

PAL yapıları, girişinde programlanabilir VE dizisi ve çıkışında sabit VEYA dizileri içerir. PAL devreleri yaygın olarak kullanılan ve bir kez programlanabilen elemanlardır. PAL elemanları belirli sayıda giriş değişkenlerine sahip lojik fonksiyonları çarpımların toplamı şeklinde ifade etmek için uygun yapılardır. Şekil 2.4'de çarpımların toplamı işleminin PAL ile nasıl oluşturulduğu gösterilmiştir.

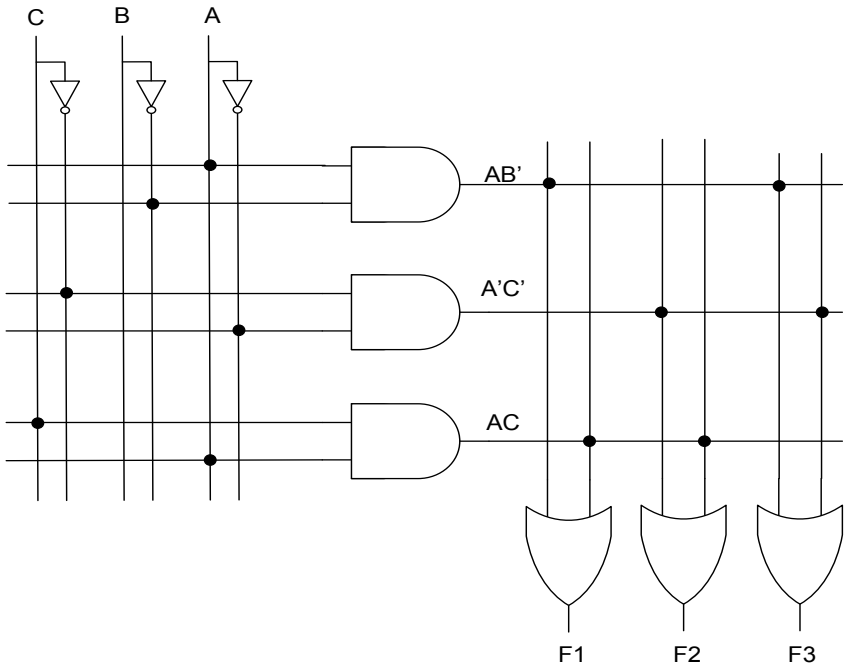


Şekil 2.4 Programlanabilir Dizi Lojik

PAL devrelerinde girişler istenilen şekilde programlandıktan sonra VE dizisine uygulanır. VE dizisinden elde edilen çıkışlar ise VEYA dizilerinin girişi kabul edilir. VEYA dizisinin çıkışları da PAL çıkışı olara kullanılan lojik devrelere uygulanır. Programlama ile satır ve sütunlar arasında bulunan sigortaların sağlam bırakılması veya attırılması sağlanır. Giriş değişkenlerinin veya terslerinin bir VE kapısına giriş olarak uygulanması, istenilen çarpma işleminin oluşmasını sağlar. Çarpma işlemini gerçekleştiren ve kapılarının çıkışları VEYA kapılarına uygulanır ve çarpımların toplamı işlemi gerçekleştirilmiş olur.

2.1.1.3 PLA: Programlanabilir Lojik Diziler (Programmable Logic Array)

PROM'lardaki hız ve sınırlı sayıda giriş/çıkış sorununa çözüm olarak PLA'lar geliştirilmiştir. Genel olarak bakıldığında bu cihazlar çok sayıda girişi destekleyebilirler ve daha hızlı çalışırlar. PLA'da girişler VE kapılarından oluşan programlanabilir bir yapıya bağlıdır. Bu kısımda gerçekleştirilecek tasarıma ait VE işlemleri yapılır. VE işlemlerinin çıkışları VEYA kapılarından oluşan başka bir programlanabilir yapıya bağlıdır. Burada tasarıma ait gerekli VEYA işlemleri yapılır ve çıkışlar elde edilir. PLA'lerin iki adet programlanabilir yapıya sahip olmaları, devrenin karmaşıklığını artırdığı gibi fazladan kullanılan her bir sigorta bağlantısı daha büyük gecikmelere neden olur.



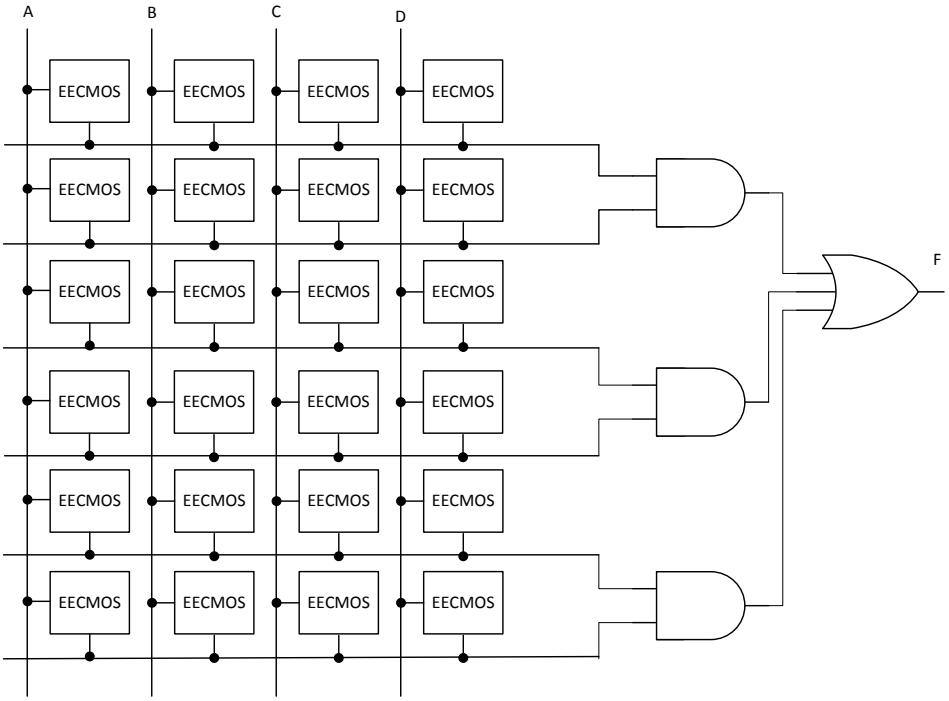
Şekil 2.5 Programlanabilir Lojik Diziler

2.1.1.4 GAL: Genel Kapsamlı Dizi Lojik (Generic Array Lojik)

Son geliştirilen PLD elemanlarından olan genel kapsamlı dizi lojik (GAL), PAL elemanları gibi programlanabilir VE dizisi ve sabit VEYA dizisi ile programlanabilir çıkış devresi içerir. GAL ve PAL arasındaki iki temel fark, GAL'in yeniden programlanabilir olması ve programlanabilir çıkış devresine sahip olmasıdır.

GAL elemanı yapısında bipolar transistor ve sigortalı hatlar yerine EECMOS (Elektriksel Silinebilir CMOS) teknolojisi kullanılması nedeni ile birçok kez programlanabilme özelliğine sahiptir. Programlanabilir VE, sabit VEYA ve programlanabilir çıkış devresinden oluşan GAL yapılarında VE yapısının programlanabilir olması çarpımların toplamı şeklindeki eşitliklerin GAL kullanılarak gerçekleştirilmesine olanak verir. GAL'ın temel yapısı aşağıda gösterilmiştir.

GAL elemanlarında bulunan VEYA kapısı ve bileşik lojik çıkış devreleri birlikte çıkış lojik makro hücreleri (output logic macrocell - OLMC) olarak adlandırılır. OLMC kısmında bulunana bileşik lojik devre programlanabilir özelliktedir. OLMC devresi birleşik lojik devresi ya da kaydedici olarak programlanabilir.



Şekil 2.6 Genel Kapsamlı Dizi Lojik

Diğer programlanabilir lojik elemanlardan satır ve sütunların kesişiminde bulunan sigortalardan farklı olarak GAL yapılarında EECMOS (elektriksel silinebilir CMOS) hücreler bulunur. Satır ve sütunların arasında bulunan EECMOS hücreleri programlayarak iletim 'ON' ve yalıtım 'OFF' durumunda olması belirlenir. Hücrelerin programlanması ile uygun değişkenlerin çarpma işlemlerini gerçekleştirmek için VEYA kapısına uygulanması sağlanır. Hücrenin iletim durumunda olması, sütundaki bilgiyi satıra aktarırken, yalıtımda olması iletimi engeller. Programlanabilen hücrelerin silinerek tekrar programlanması mümkündür.

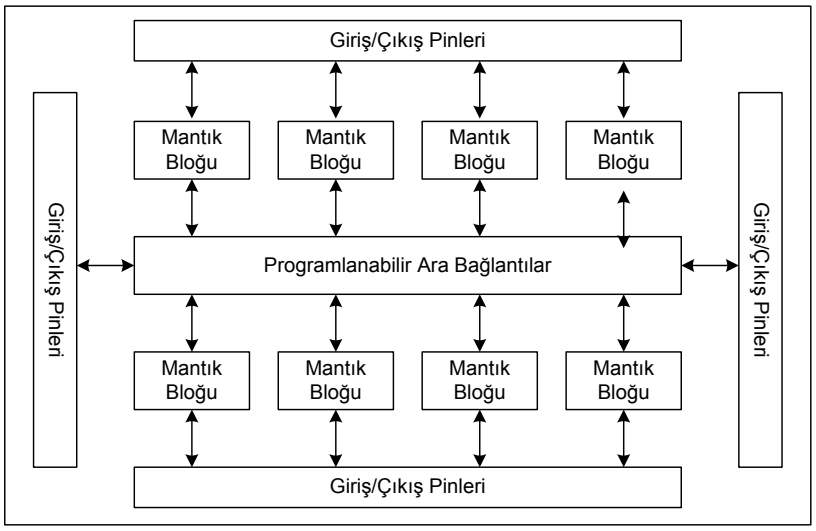
2.1.2. Karmaşık PLD'ler (CPLD'ler)

PAL ve PLD'ler küçük devlelerin gerçeklenmesi için uygun özeliklere sahip olmasına karşın çok fazla giriş/çıkış gerektiren daha büyük ve karmaşık devrelerin tasarımları için kullanışlı değillerdir. Karmaşık devrelerde kullanılan kapı sayısının artması kullanılan entegre sayısını artırır. Kullanılan entegre sayısının artması devrenin boyutunu artırır, bununla beraber farklı çalışma frekansları ve farklı besleme girişleri gibi durumlar meydana getirebilir. Bunlara ek olarak büyüyen devre tasarımlarında, büyüklükle orantılı olarak parazitler ve besleme dengesizliği gibi sorunlar meydana gelebilir. Bu

nedenle daha büyük devrelerin tasarımları için tek bir bütünleşmiş devre içerisinde birbiri ile etkileşimli birden fazla PAL'lerin işlevini yerine getirebilen sistemlere ihtiyaç duyulmuş ve sonuç olarak da CPLD'ler üretilmeye başlanmıştır. CPLD'ler Verilog veya VHDL dilleri ile programlanabilir. PAL ve PLA'larla sadece birkaç yüz mantık kapısı eşdeğerinde olan küçük devreler gerçekleştirilebilirken, CPLD'lerde binler hatta yüz binlerce mantık kapısı eşdeğerindeki devreleri gerçekleştirmek mümkün olmaktadır.

CPLD'lerin genel yapısına bakıldığı zaman tek bir ara bağlantının etrafında sıralanmış mantık hücreleri bulunduğu görülür. Gerçekleştirilmesi istenilen fonksiyona göre mantık hücrelerinin sayısı arttıkça hücreler arasındaki bağlantı sayısı da katlanarak artar. Bu ise çok büyük tasarımlarda bağlantı sorunlarını beraberinde getirdiğinden CPLD'leri kullanışsız hale getirmektedir.

Basit bir CPLD tümleşik devresinin içyapısı aşağıda gösterilmiştir;



Şekil 2.7 CPLD'nin içyapısı

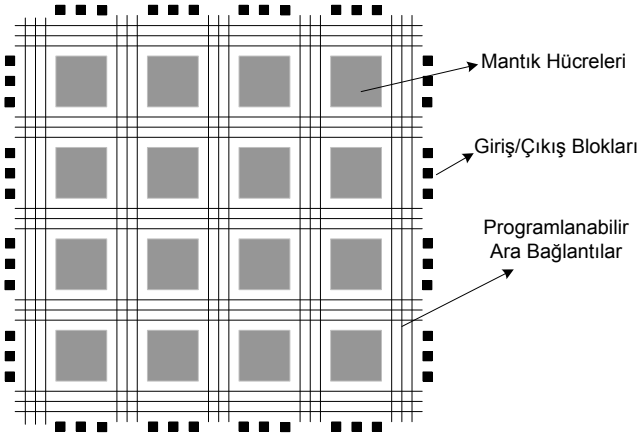
Buna paralel olarak FPGA'da mantık hücreleri CPLD'lerden farklı olarak dizi şeklinde yerleştirilmiştir. Mantık hücrelerinin aralarındaki bağlar yatay ve dikey programlanabilir ara bağlantılar sayesinde sağlanır ve mantık hücreleri dizi şeklinde sıralanmıştır. Bu yapı sayesinde hücreler arasında ihtiyaç duyulan bağlantı sayısı, mantık hücresi sayısındaki artış ile doğru orantılı olarak artar. Bu özelliği ile FPGA, daha karmaşık ve büyük tasarımlarda bağlantı sorunları en aza indirgenmiş olur.

2.1.3. FPGA'ler

FPGA'lar yapısal olarak SPLD ve CPLD'lerden daha farklılardır. Daha yüksek lojik kapasiteye sahiptirler. Basit bir FPGA dizi şeklinde sıralanmış programlanabilir ara bağlantılar ile birbirine bağlanabilen programlanabilir mantık blokları ve bunların etrafını saran programlanabilir giriş/çıkış bloklarından oluşmaktadır. Mikroişlemciler gibi adım adım işlem yapabilme özelliği yanında paralel işlem yapabilme özelliği ve tekrar tekrar programlanabilir olması, FPGA'in çok geniş bir alanlarda kullanılmasına olanak vermektedir. Gömülü işlemciler, sayısal işaret işleme (DSP) blokları, PLL'ler, yüksek hızlı paralel, seri giriş-çıkış desteği, harici bellek ara yüzleri, PCI Express ve Gigabit Ethernet gibi alıcı/verici ara yüzleri, gelişmiş saat ve güç dağıtım yönetimi gibi özellikleri FPGA'lerin tercih edilmesini sağlamaktadır. FPGA'lerin tercih edilmesinin bir başka sebebi de üretici firmaların sağladığı, PCB tasarım, modelleme ve benzetim yazılımları ve hazır program blokları (IP) gibi güçlü yazılım desteğidir.

2.2. FPGA'in Yapısı ve Özellikleri

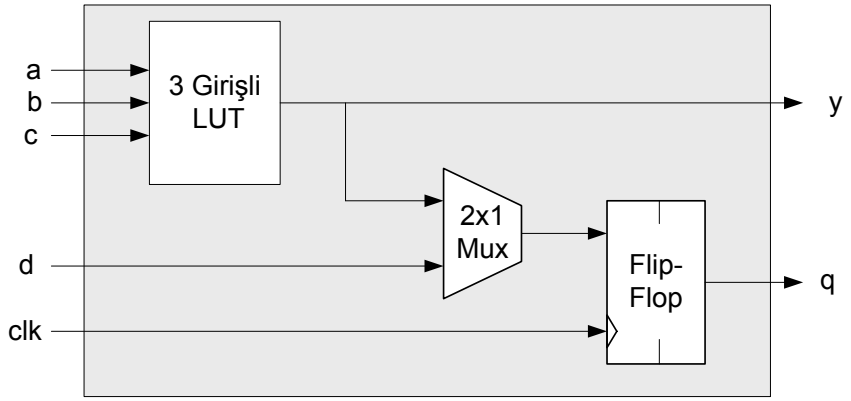
FPGA, yarı iletken teknolojisi ile üretilir ve en temel şekliyle, mantık mücreleri (logic cell), giriş/çıkış blokları (IO Block) ve programlanabilir ara bağlantılar'dan oluşur. Mantık hücreleri FPGA içersine matris şeklinde dizilmiştir ve programlanabilir ara bağlantılar da bu mantık hücrelerini birbirine bağlamıştır.



Şekil 2.8 FPGA'in içyapısı

FPGA'in temel yapısı SRAM mantık hücrelerinden oluşturur. Mantık hücrelerinin yapıları aynı üreticinin farklı FPGA modellerinde bile farklılık gösterebilir. İlk tasarlanan

programlanabilir FPGA'ler genellikle üç girişli lookup table(LUT), kaydedici birim (flip flop) ve çoklayıcı birimden(multiplexer) oluşmaktadırlar. Şekil 2.9'da basit bir programlanabilir mantık hücresinin iç yapısını gösterilmiştir.



Şekil 2.9 Sadeleştirilmiş bir mantık Hücresinin içyapısı

Her FPGA bu programlanabilir mantık hücrelerinden çok sayıda içerirler. Uygun SRAM programlama hücreleri vasıtasıyla cihaz içindeki her lojik blok farklı bir fonksiyonu gerçekleştirmek için yapılandırılabilir. Kaydedici olarak işlev gören flip-flop saat sinyalinin pozitif ya da negatif kenarında tetiklenecek şekilde yapılandırılır ve multiplexer uygun LUT çıkışı ile flip flopu besler.

Programlanabilir mantık hücreleri ve FPGA pinleri arasındaki bağlantılar programlanabilir giriş/çıkış blokları ile sağlanır. Bu bloklar FPGA'in dış dünya ile iletişimini sağlar. Bu yapılar uygulanan tasarıma göre yalnızca giriş, yalnızca çıkış ya da hem giriş hem çıkış şeklinde yapılandırılabilir. Mantık hücrelerinin ara bağlantıları matris şeklindeki veri yolları ve programlanabilir anahtarlarla sağlanır. Bu anahtarlama işlemi FPGA içersine yüklenen program ile yapılır. Programlanabilir anahtarlar sayesinde mantık hücrelerinin birbirleri ve giriş/çıkışlar arasındaki sinyal iletimi sağlanmış olur. Bir tasarımda kullanılan FPGA'in diğer çevre elemanları ile bağlantılarını sağlanması üzerindeki pinler aracılığı ile olur. Bu pinler genel olarak "ayrılmış pinler" ve "kullanıcı pinleri" olarak iki sınıfa ayrılabilir. Tasarım sırasında yapılan pin atama işlemi ile tasarımda oluşturulan portlar FPGA pinlerine bağlanır.

Ayrılmış pinler güç pinleri, yapılandırma (konfigürasyon) pinleri ve saat pinleri olarak sınıflandırılabilir. Güç pinleri FPGA için gerekli olan güç ve toprak bağlantısını sağlayan noktalardır. Konfigürasyon pinleri, oluşturulan programın FPGA'a yüklenmesi için

kullanılan bağlantı noktalarıdır. Saat pinleri, saat sinyallerinin bağlanabilmesi için ayrılmış özel bağlantı noktalarıdır. Kullanıcı pinleri ise kullanıcı tarafından yapılandırılabilen standart giriş/çıkış pinleridir.

Senkron tasarımlarda FPGA içindeki flip flopların tek bir saat sinyali ile aynı anda tetiklenmesi gerekir. Aksi takdirde tasarımda zamansal problemler meydana gelebilir. FPGA'lerde global hat olarak adlandırılan özel iç bağlantılar bulunur. Bu bağlantılar sayesinde saat sinyalinin FPGA içersindeki bütün flip floplara aynı anda iletilmesi sağlanmış olur. Doğru bir senkron tasarım için saat beslemelerinin FPGA'in saat için ayrılmış pinlerinden verilmesi gerekir.

FPGA'in en önemli özelliklerinden biri **paralel işlem yapabilme** yeteneğidir. Aynı anda birden fazla işlemi yapabilme anlamına gelen bu özellik, FPGA'in diğer programlanabilir lojik cihazlardan göre daha hızlı çalışmasını sağlar. Sıradan entegreler ya hiç paralel işlem yapamazlar ya da çok sınırlı yapabilirler, fakat FPGA'ler uygulamaya ve kapasiteye bağlı olarak, birbirinden bağımsız yüzlerce işlemi aynı anda yapmak mümkündür.

FPGA'in diğer bir önemli özelliği de **sahada programlanabilir** olmasıdır. Yani, çalıştığı ortamda programlanabilmesidir. FPGA'in son ürün haline getirildikten sonra defalarca programlanabilir olma özelliği zaman ve maliyet açısından tasarımcılara oldukça büyük kolaylıklar sağlar.

Paralel işlem yapabilme kabiliyeti başta olmak üzere, sahada programlanabilme, tasarımın test edilip doğrulanabilmesi ve istenirse içine işletim sistemi gömülebilmesi gibi birçok özelliği FPGA'in tercih edilme sebepleri olarak sıralanabilir. Tasarlanan programın FPGA'e yüklenmeden önce bilgisayar ortamında simülasyonu yapılabilmektedir. Bu işlem tasarımcının programı FPGA'e yüklemeden önce test edip doğrulayabilmesine olanak sağlamaktadır. Bunun yanında FPGA üretici firmalarının geliştirdiği özel araçlar ile program FPGA'e yüklendikten sonra ürün üzerinde de simülasyon ve hata giderme işlemleri yapılabilmektedir. İşlemci gerektiren tasarımlarda FPGA içersine soft işlemci gömülebilmekte ve C/C++ gibi yazılım dilleri ile bu işlemciler programlanabilmektedir. Kullanılan FPGA türüne ve ihtiyaca göre, birden fazla işlemci dahi kullanılabilmekte ve FPGA'in artan bölümleri de paralel işlemler için

kullanılabilmektedir. Bununla beraber içersinde fiziksel olarak gömülü işlemciler bulunduran FPGA'ler de üretilmektedir.

2.3. FPGA'in Kullanım Alanları

FPGA'in kullanım alanları aşağıdaki gibi sıralanabilir;

- **Uzay ve Savunma Sanayi;** güvenli (kriptolu) iletişim, radarvesonar cihazları, elektronik harp, aviyonik sistemler ve silah sistemlerinde.

- **Endüstriyel Otomasyon ve Kontrol Sistemleri;** motor kontrol, endüstriyel görüntüleme, endüstriyel ağlar

- **Otomotiv;** görüntü işleme, araç kontrol sistemleri, araç bilgi ve eğlence sistemleri.

- **Bilgisayar/Depolama;** sunucular, disk sürücüler, yazıcı/fotokopi.

- **Kablolu/Kablosuz İletişim;** 3G teklolojisi, GSM, ADSL, VSDL, radyo dalgaları, optik ağlar, bilgisayar ağları.

- **Tıbbi Elektronik;** yansılanım (ultrason) görüntüleme, bilgisayarlı tomogrofi (CT), MRI görüntüleme, PET görüntüleme, yaşam destek üniteleri, hastane ve laboratuvar elektroniği.

- **Tv/ Radyo Yayıncılığı;** gerçek zamanlı ve yüksek çözünürlüklü grafik işlemcileri, video dağıtım cihazları, video anahtarlama ve yönlendirme cihazları, profosyonel kameralar, profosyonel görüntü sistemeleri.

- **Test/Ölçüm alanında;** ağ/protokol analizörleri, spektrum analizörleri, bit-hata test cihazları, yarı iletken tabanlı test cihazları, osiloskoplar, lojik analizörleri, sinyal üreteçleri.

- **Tüketici Elektroniği;** akıllı telefonlar, LED, LCD, plazma TVler, uydu alıcıları, projektörler, multimedya cihazları, digital kameralar, navigasyon ve GPS.

- **Güvenlik /Şifreleme;** kripto cihazları, askeri iletişim cihazları, gömülü şifreleme, ağ şifrelemesi, datalink uygulamaları, tektik telsizler, uydu iletişimi, finans ve bankacılık, güvenli veri depolama.

BÖLÜM 3

VHDL

Dijital sistemlerin hızlı bir şekilde gelişmesi birçok bileşenin tek bir chip üzerinde çalışmasına izin veren tümleşik devre (Integrated Circuit) teknolojisini ortaya çıkarmıştır. Tümleşik devreler uygulamalarda birçok kolaylık sağlar, fakat bu yapıların kapı ve flip-flop seviyesinde tasarımı çok detaylı, karmaşık ve zaman alıcıdır. Bu nedenle dijital sistem tasarımında donanım tanımlama dilleri geliştirilmiştir. Donanım tanımlama dilleri, dijital sistemin kapı ve flip-flop seviyesine dönüştürülmeden önce, yüksek seviyede tasarlanmasını ve hata ayıklama işlemlerini yapılmasına olanak tanır. [43]

FPGA programlamak için en çok kullanılan iki temel donanım programlama dili VHDL ve Verilog dilleridir. Tez kapsamında yapılan çalışmada VHDL dili kullanılmıştır. VHDL, çok yüksek hızlı bütünleşik devre donanımı tanımlama dili anlamına gelen "**V**ery High-Speed Integrarated Circuit **H**ardware **D**escription **L**anguage" ifadesinin kısaltmasıdır. VHDL, birkaç kapıdan, çok karmaşık tümleşik devrelerin birbirine bağlanmasına kadar çok çeşitli dijital sistemlerin tasarımında ve simülasyonunda kullanılan genel amaçlı bir donanım tanımlama dilidir. 1970 yılların başlarından itibaren geliştirilmeye başlanan dil, ilk olarak Amerikan Savunma Bakanlığı tararından askeri projelerde kullanılmak üzere geliştirilmiştir. Daha sonra 1987 yılında resmi olarak IEEE standardı olarak kabul edilmiş ve endüstride kullanılmaya başlanmıştır. IEEE tarafından 1076 ve 1164 standartları ile standartlaştırılmış ilk donanım programlama dilidir. Daha sonra VHDL 93, VHDL 2002 ve son olarak VHDL 2008 versiyonları geliştirilmiştir [45]. VHDL dilinin, Programlanabilir Lojik Devrelerin tasarımı (CPLD ve FPGA) ve ASIC sistemlerin tasarımı olmak üzere iki temel uygulama alanı vardır.

Bu bölümün devamında VHDL dili tasarım yapabilmek için bilinmesi gereken temel bilgilere yer verilmiştir. Önce hazırlanılan programın cihaz üzerine yüklenenince kadar geçtiği aşamaları belirten tasarım akışı anlatıldıktan sonra sıra ile VHDL dilinin temelini oluşturan kod yapısı, veri nesneleri, veri tipleri, operatörler ve nitelikler anlatılmış ve son olarak da VHDL'de aş zamanlı ve sıralı kod yazabilmek için bilinmesi gereken yapılar hakkında bilgi verilmiştir.

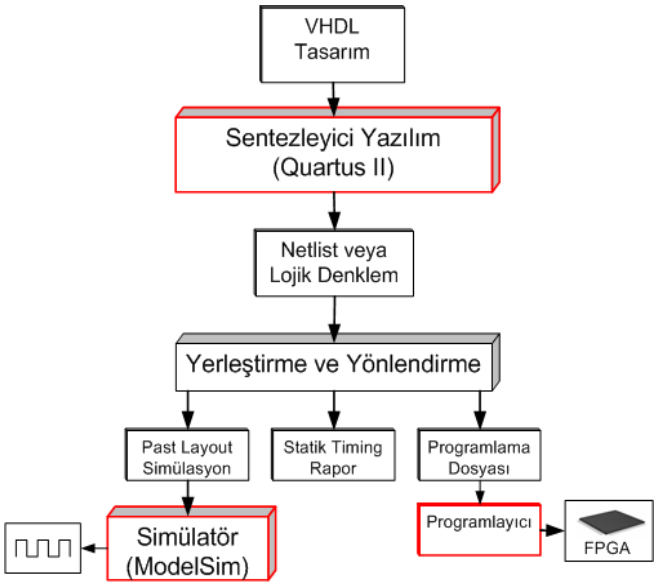
Bu çalışmada, Altera firmasının Quartus II programlama platformu ile VHDL dilinde tasarımlar oluşturulmuş ve Mentor Graphics firmasının ModelSim programı kullanılarak ilgli simülasyonlar hazırlnamıştır.

3.1. VHDL Tasarım Akışı

VHDL tasarımı kodlama, simülasyon ve sentezleme olmak üzere üç temel kısımdan oluşur. **Kodlama,** VHDL kodunun yazıldığı kısımdır. Tasarımcı kodun hepsini kendi yazabileceği gibi, program sihirbazı kullanarak da bir kısmını derleyiciden alabilir. **Simülasyon,** VHDL kodunun simülasyonunun gerçekleştirilip, programın doğru çalışıp çalışmadığının kontrol edildiği kısımdır. Böylece test aşamasında çıkan hatalar, program FPGA'e yüklenmeden düzeltilebilir. **Sentezleme** bölümünde ise yazılım dili olan VHDL programı, donanım diline çevrilir. Yani RTL şeması çıkarılır. Sentezleme sonucunda yazılan kod compiler tarafından FPGA'e yüklenecek olan konfigürasyon dosyasına çevrilir [43].

VHDL ile bir devrenin tasarımı ilk olarak VHDL kodunun yazılması ile başlar ve bu kod .vhd uzantısı ile kaydedilir. Kaydedilen dosya ismi ve tasarımın *entity*[1] ismi aynı olmalıdır. Sonra tasarımın derleme (compilation) işlemi gerçekleştirilir. Derleme işleminde RTL seviyesinde yazılan VHDL kodu Netlist seviyesine dönüştürülür. Bu seviye "lojik seviye" ya da "gate level" olarak da isimlendirilir. Sistemin simülasyonu gerçekleştirilir ve programlama dosyası bir programlayıcı vasıtasıyla PLD/FPGA chipine yüklenir. Böylece tasarlanan devre fiziksel bir sisteme dönüştürülmüş olur. Şekil 3.1'de VHDL dili ile hazırlanmış bir programın akış şeması verilmiştir.

[1] Tez içinde, okuma kolaylığı sağlaması için VHDL diline özgü kelimeler italik olarak yazılmıştır.



Şekil 3.1 VHDL akış şeması

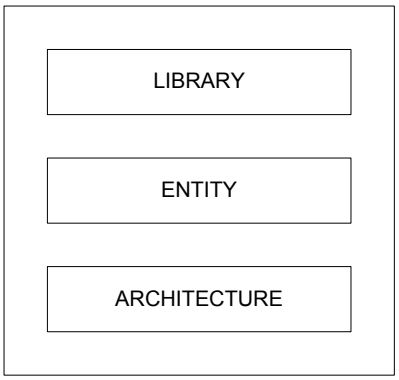
VHDL dilinde kodlar sıralı ve eşzamanlı olarak yazılabilir. Eş zamanlı ifadeler uygun uyarıcı tarafından uyarıldığı ya da saat sinyali geldiği zaman aynı anda çalışacakları için yazım sırası önemli değildir. Sıralı kodların ise diğer programlama dillerinde olduğu gibi çalışma sırasına göre yazılması gerekir. VHDL kodunda satırlar noktalı virgül (;) ile sonlandırılır. Kod yazımında büyük küçük harf duyarlılığı yoktur. Boşluklar ve satırların bir anlamı yoktur, kaynak kodu etkilemez, fakat boşlukların düzenli kullanılması programın okunurluğunu artırır. Açıklamalar yan yana iki çizgi "--" ile başlar ve derlenmezler. VHDL dilinde *entity, process, function* ve *procedur* gibi VHDL yapılarına verilen isimlere tanımlayıcı denir. VHDL dilindeki ilk tanımlayıcılar VHDL'87 standartları ile belirlenmiştir. Her zaman harf ile başlamak şartı ile harflerden, sayılardan ve altçizgilerden oluşabilirler. Latin alfabesinde bulunan karakteler kullanılır. Tanımlayıcılar boşluk içermez ve VHDL diline özel sözcükler kullanılmaz. Daha sonra VHDL'93 standardı ile tanımlayıcılar geliştirilmiş ve çoğu kısıtlama kaldırılmıştır [63].

3.2. VHDL Kod Yapısı

VHDL programını oluşturan üç temel yapı vardır; ***Library***, ***Entity*** ve ***Architecture***

Bu yapılara ek olarak ayrıca tasarımlarda kullanım kolaylığı sağlayan, fakat kullanım zorunluluğu olmayan ***package, component, function*** ve ***precedure*** yapılarıda mevcuttur.

Library bölümünde (library decleration) tasarımda kullanılacak bütün kütüphanelerin tanımlanması gerekir. *Entity*, tasarlanacak devrenin giriş ve çıkış pinlerinin tanımlandığı bölümdür. *Arcitecture* kısmında ise tasarlanacak devrenin davranış şekli yani nasıl çalışacağı ile ilgili kodlar yer alır. Bütün VHDL programlarında bu üç temel yapı bulunmak zorundadır. Basit bir VHDL programını Şekil 3.2'deki gibi gösterilebilir.



Şekil 3.2 Basit bir VHDL kodunun yapısı [45]

3.2.1 Library (Kütüphane)

Kütüphane çok sık kullanılan kod parçalarının bir araya getirilip derlenmesi ile oluşturulan yapılardır. Kodların başka tasarımlarda yeniden kullanılabilir ve paylaşılabilir hale gelmesini sağlar. Genel olarak kütüphaneyi oluşturan kodlar bir *paket* içersinde *fonksiyon, prosedur* ya da *component* olarak tasarlanır.

VHDL ile tasarlanan her projenin kaynak kodlarının derlenebilmesi için kaydolduğu bir kütüphane vardır. Bunun içinde kullanıcı genelde tasarımları için kendine ait bir kütüphane oluşturur, fakat belirtilmemişse derleyici programlar varsayılan (default) olarak "*work*" kütüphanesini kullanılır. Kullanıcı tasarımlarında kendi oluşturduğu kütüphaneler dışında derlenmiş kütüphaneleri de kullanabilir. Bunun için öncelikle bu kütüphaneyi tasarımına tanıtması gerekir. Bir kütüphanenin tasarım içinde tanımlanması ve kullanıma açılması aşağıdaki gibi yapılır;

```
library kütüphane_ismi;
use kütüphane_ismi.paket_ismi.paket_bölümü
```

Temel bir VHDL tasarımında üç farklı kütüphanede bulunan üç tane paketin programa tanıtılması gerekir. Bunlar *ieee.std_logic_1164 (*ieee kütüphanesinden*), standard* (std kütüphanesinden), *work* (work kütüphanesinden) kütüphaneleridir.[5] *"std"* ve *"work*"

kütüphaneleri VHDL tasarımlarında varsayılan olarak tanımlandığı için yeniden tanımlanmasına gerek yoktur. Fakat *ieee.std_logic_1164* paketi her programda aşağıdaki gibi tanıtılmalıdır.

```
library ieee;
use ieee.std_logic_1164.all;
```

Aşağıda bu kütüphaneler ve içerdikleri paketler hakkında yeterli bilgi verilmiştir.

3.2.1.1 IEEE kütüphanesi

Çizelge 3.1'de IEEE kütüphanesinin en çok kullanılan paketleri ve içerikleri sıralanmıştır.

Çizelge 3.1 IEEE kütüphanesinde bulunan paketler ve açıklamaları

Paket adı	Açıklama
std_logic_1164	std_logic, std_ulogic, std_logic_vector ve std_ulogic_vector türleri ile ilgili fonksiyonlar
std_logic_arith	signed, unsigned, integer, std_ulogic, std_logic ve std_logic_vector türleri için aritmetik, çevirme ve karşılaştırma fonksiyonları
std_logic_unsigned	işaretsiz aritmetik fonksiyonlar
std_logic_signed	işaretli aritmetik fonksiyonlar
numeric_std	std_logic türünde verilen aritmetik işlem fonksiyonlar

3.2.1.2 STD kütüphanesi

Standart (std) kütüphanesi ön tanımlı VHDL bileşenlerinin bulunduğu kaynak kütüphanedir. Bu nedenle program başında *library* komutu ile tanıtılmasına gerek yoktur. Bu kütüphane standart ve textio adında iki paket içerir.

Standart paketinin içinde, bit, boolean, integer, real, time basit veri türleri ve bu türler için uygulanan fonksiyonlar bulunur. Bu paketin programın başında tanıtılmasına gerek yoktur. **Textio paketinin içinde** ise, metin dosyası işlemleri ile ilgili prosedür, fonksiyon ve türleri içerir. Bu paket kullanılırken program başında tanıtılması gerekir [43].

3.2.1.3 Work kütüphanesi

Varsayılan olarak tanımlı olduğu için yeniden tanımlanmasına gerek yoktur. Tasarımın kaydedildiği kütüphanedir. Bu kütüphanenin içinde ayrıca compiler ve simülator için oluşturulan dosyalar da kaydedilir.

3.2.2 Entity

Entity yapısı tasarımın dış dünya ile iletişimini tanımlamak için kullanılır. Tasarlanan elektronik devrenin giriş ve çıkışlarını oluşturan portlar bu bölümde tanımlanır. *Entity* ifadesi *library* tanımlamasından sonra yazılır ve bu sayede paket içinde tanımlanmış bütün ifadeler burada kullanılabilir hale gelir. *Entity* ismi aynı zamanda tasarımın ismini ifade ettiği için tasarım bu isimle kaydedilmelidir.

Her VHDL tasarımında sadece bir adet entity bulunur ve *paket* bildiriminden sonra yapılır. Fakat bir *entity* için birden fazla *architecture* ve *konfigurasyon* kullanılabilir. Genellikle bir sistem için tek *architecture* ve tek konfigurationdan oluşan kodlar tercih edilir [43].

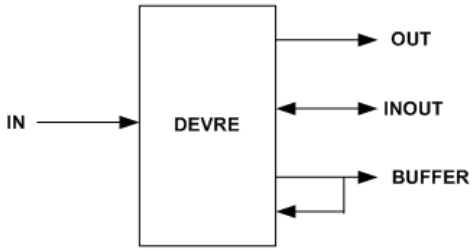
Entity yapısının tanımlaması aşağıdaki gibi yapılır.

```
entity entity_adi is port (
   port_adi: port_modu port_türü;
   diger potlar…);
end entity_adi;
```

VHDL dilinde ***port*** devrenin dış dünya ile iletişimini sağlayan kanallara verilen addır. Bir tasarımda kullanılacak portların sayısı, modu ve türü entity içinde tanımlanır. VHDL içersinde ***in, out, inout*** ve ***buffer*** olmak üzere dört adet port bulunmaktadır. Bu portların özellikleri Çizelge 3.2'de açıklanmış ve Şekil 3.3'te şematik olarak gösterilmiştir.

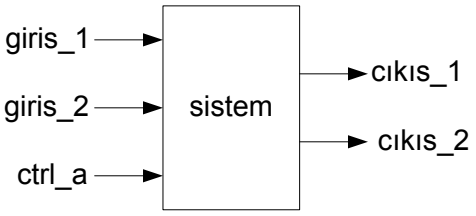
Çizelge 3.2 VHDL dilindeki port isimlri ve açıklamaları [43]

Port adi	Açıklama
in	Giriş portudur, modele dışarıdan gelen verileri tanımlar ve bu veriler sadece okunurlar, üzerlerinde hiçbir değişiklik yapılmaz.
out	Çıkış portudur ve modelden dışarıya çıkan sinyalleri tanımlar.
inout	Hem giriş hem de çıkış özelliğine sahip olan porttur. Çift yönlü olarak kullanıldığı için genellikle bellek uygulamalarında tercih edilir.
buffer	Out portruna benzemekte fakat farklı olarak değer okunabilmektedir.



Şekil 3.3 Sinyal Modları[46]

Aşağıdaki şematik olarak tanımlanan devrenin giriş ve çıkış portlarını *entity* içinde nasıl yazılacağını gösterilmiştir. Bu devrenin üç tane girişi ve iki tane çıkışı vardır.




```
entity system is port(
   giris_1: in std_logic;
   giris_2: in std_logic;
   ctrl_a: in std_logic;
   cikis_1: out bit;
   cikis_2: out std_logic);
end system;
```

Şekil 3.4 Bir sistemin giriş çıkış portları ve *entity* yapısı

Devrenin girişleri *std_logic* olarak tanımlanmış ve çıkışlarınında biri *bit* diğeri *std_logic* olarak tanımlanmıştır. *Entity* içinde giriş ve çıkış portları yukarıdaki gibi yazılır. Burada önemli olan nokta son tanım satırında noktalı virgül işareti parantez işaretinden sonra yazılmasıdır.

Entity kullanımında karşılaşılacak bir diğer yapı da ***"generic"*** yapısıdır. *Generic*, entity yapısına ait bazı parametrelerin bileşenlere aktarılması için kullanılan bir VHDL yapısıdır. *Generik* yapısı, *sabit* yapısı gibi tasarıma sabit bilgi girişinde kullanılır ve architecture içinde de işlenebilirler. *Generik* yapısının *sabit* yapısından farklı dışarıdan beslenebilme özelliğidir. *Generic* yapısı *entity* içinde *port* tanımlamasından önce tanımlanır. *Generic* yapıları genelikle port boyutlarını, alt bileşen sayısını, zamanlama özelliklerini, döngü sayısını, tasarımın fiziksel özelliklerini ve *architecture* içindeki vektörlerin uzunluğunu belirtmek için kullanılır.

Aşağıdaki örnekte adres boyutu *generic* yapısı ile 16 bit olarak bildirilen bir ram yapısı için *entity* tasarımı verilmiştir [46].

```
entity ram is
   generic (adres_boyutu := 16);
   port ( adres: std_logic_vector(adres_boyutu-1 downto 0));
end ram;
```

Aşağıdaki örnekte *generic* yapısının *architecture* içersinde nasıl kullanılacağı gösterilmiştir.

```
entity and_gate is
   generic(temp: time);
   port (a,b: in bit; y: out bit);
end and_gate;
architecture data of and_gate is
begin
   y <= a and b after temp;
and data;
```

3.2.3 Architecture

Bu yapı bir VHDL tasarımının davranışını yani içyapısını tanımlayan bölümdür. *Entity* içersinde giriş/çıkış portları tamamlandıktan sonra çıkışların girişin hangi fonksiyonu ile tanımlanacağı *architecture* bölümünde yazılır. Başka bir deyişle *arctitecture* bölümünde *entity* içersinde tanımlanmış olan portlar arasındaki ilişkiler gösterilir.

Tasarım içersinde *architecture* tanımlaması aşağıdaki gibi yapılır;

```
architecture architecture_adi of entity_adi is
```

```
   [bildirimler(sinyal, component tanımlamaları)];
begin
   architecture_gövdesi;
end arctitecture_adi;
```

İlk satırda *architecture* ismi ve *architecture* yapısının bağlı olduğu *entity* ismi belirtildikten sonra *begin* ifadesine kadar tasarım içersinde kullanılacak sabitler, sinyaller, alt programlar, bileşenler ve veri türleri yazılır. *Architecture* gövdesi bölümünde eş zamanlı olarak çalışacak olan kodlar yazılır ve yazım sırası önemli değildir. Her *architecture* yapısının bir *entity'ye* bağlanması gerekir. Bir *entity'ye* birden fazla *architecture* bağlanabilir, fakat tersi olmaz.

3.2.4 Package (Paket)

İki ya da daha fazla birim tarafından ortak olarak kullanılan veri türleri, alt programlar, sabitler, sinyaller ve bileşenler gibi VHDL bileşenlerini içeren yapılara *paket (package)* adı verilir. *Paket* tasarımı, *paket* bildirimi ve paket gövdesi olarak iki bölümden oluşur. Aşağıda tasarım içinde *paket* bildiriminin ve paket gövdesinin nasıl yapılacağı gösterilmiştir [43].

```
-- paket bildirimi
package paket_adi is
   [paket_bildirimleri];
end package paket_adi;
```

```
-- paket gövdesi
package body paket_adi is
   [alt_program_tanımlamaları];
end package body paket_adi;
```

Aşağıda program içersinde paket kullanımının nasıl gerçekleştirileceğine ilişkin örnek verilmiştir.

```
-- paket bildirimi örneği
package ornek is
   constant gecikme: integer
   type renk is (sari, yesil);
   function sayici (k: integer) return integer;
end package ornek;
--paket gövdesi örneği
package body ornek is
   constant gecikme: integer := 20;
```

```
   function sayici (k:integer)return integer is
begin
   ...
end;
end ornek;
```

Paket tanımlama işlemi tamamlandıktan sonra, *paketin* tasarımda kullanıma açılması için kütüphane tanımlama bölümüne aşağıdaki ifade yazılır.

```
use kutuphane_adi.paket_adi.all
```

Paket oluşturma işleminin mevcut tasarımın VHDL kaynak kodu içersinde yapılması kullanım amacına uygun değildir. Bu yüzden *paketin work* kütüphanesin de ya da özel bir kütüphane de tanımlanması daha uygundur. *Work* kütüphanesi varsayılan olarak tanımlı olduğu için bu kütüphane içinde tanımlanan paket kullanılırken sadece paketin kullanıma açılması yeterlidir.

3.2.5 Component (Bileşen)

VHDL dili kullanılarak hazırlanan bir tasarım bir bütün olarak tanımlanabileceği gibi, her biri özel bir işlevi yerine getiren küçük alt birimlerden de oluşabilir. Bu alt birimlere tasarımın bileşenleri denir. Bu bileşenler hiyerarşik düzende kendi içlerinde bir devre özelliği taşırlar ve her birinin kendisine ait *library, entity* ve *architecture* yapısal birimleri bulunur. *Component* tanımlama işlemi daha önceden hazırlanan bir tasarımın mevcut tasarıma ekleme işlemi olarak da düşünülebilir. Bileşen kullanımı çok karmaşık programların daha kolay anlaşılmasını sağlar ve programın okunurluğunu artırır.

Component yapısının kullanılması için önce *component* bildirimi yapılır ve sonra *component* yapısı program içersine çağırılır. *Component* bildirimi *paket*, *entity*, *architecture* ya da *blok* içersinde yapılabilir. *Paket* içersinde yapılan *component* bildirimi, *paketin* bildirim kısmında yapılır. *Architecture* içinde yapılan *component* bildirimi ise *architecture* ve *begin* ifadeleri arasında yapılır ve sadece o *architecture* içersinde kullanılır.

Aşağıda VHDL programı ile *component* bildiriminin nasıl yapılacağı ve bileşenin program içersine nasıl çağrılacağını gösteren kod parçalarına yer verilmiştir.

```
--bileşen bildirimi
component component_adi
   generic(generic_adi tür := deger);
   port(port_adi mod :=tür);
end component;
--bileşenin program icersine cagrilmasi
etiket: component bileşen_adi
  generic map (generic_adi => deger, diger generic …)
  port map (port_adi => sinyal_adi, diger portlar…);
```

Generic map ifadesi ile component içersinde kullanılan generic yapısı mevcut tasarıma tanıtılır ve *port map* ifadesi ile de port eşleştirilmesi yapılır.

3.2.6 Function (Fonksiyon)

Sıralı olarak çalışan *fonksiyon* yapısının amacı tip dönüşümleri, lojik işlemler, aritmetik hesaplamalar ve yeni operatör ve nitelikler gibi tasarım içersinde sıklıkla kullanılan ifadeleri kullanarak bunları işleyen ve bir sonuç üreten komut grubu oluşturmaktır. *Fonksiyon* yapısının içerisinde *process* yapısında olduğu gibi *if, wait, case* ve *loop* yapıları kullanılır, fakat *signal* ve *component* bildirimi yapılamaz.

Tasarım içersinde *fonksiyon* kullanabilmek için önce *fonksiyonun* yazılması ve daha sonra bu fonksiyonun program içinde çağrılması gerekir. *Fonksiyonlar* program içersinde genellikle *paket* içerisinde tanımlanır ve *paketler* de ilgili kütüphaneye gömülür. Tasarımda *paket* içerisinde bir *fonksiyon* kulanılacaksa, ilgili *paketin* tasarıma tanıtılması gerekir. *Fonsiyonlar* ayrıca *arctitecture* ya da *entity* içinde de tanımlanabilir. Fonksiyon gövdesi ve fonksiyonu tasarım içinde çağırma işlemi aşağıdaki gibi yapılır.

```
--fonsiyon gövdesi
function fonksiyon_adi[giris_parametreleri] return data_tipi is
    [tanımlamalar];
begin
    [sıralı çalısan kodlar];
end fonksiyon_adi;
--fonksiyon çağırma işlemi
geri_donus_degeri <= fonksiyon_adi (giris_parametreleri);
```

3.2.7 Precedure (Prosedür)

Prcedure, yapı ve kullanış olarak *fonksiyona* çok benzer ve *fonksiyon* yapısından tek farkı birden fazla geri dönüş değeri gönderebilmesidir. *Procedure* yapısını oluşturmak için de önce *procedure* gövdesi oluşturulur, sonra da tasarım içinde *procedure* yapısı çağrılır. Procedure gövdesi aşağıdaki gibi yazılır;

```
-- procedure gövdesi
procedure procedure_adi[parametre_listesi] is
      [tanımlamalar];
begin
     [sıralı çalışan kodlar];
end procedure_adi;
```

Procedure ifadesi, *paket* yapısının içerisinde ya da *enttiy* ya da *architeture* yapısının icersinde tanımlanabilir. *Paket* içerisinde tanımlanan *procedure* kullanılmadan önce ilgili paketin tasarıma tanıtılması gerekir.

Procedure ifadesinin parametre listesinde *in, out* ve *inout* modlarında *signal, variables* ya da *constant* veriler bulunur. Giriş parametreleri için varsayılan veri tipi *constant* iken, çıkış parametreleri için varsayılan değer *variable* veri tipindedir.

Tasarım içerisinde procedure çağırma işlemi aşağıdaki gibi yapılır;

```
-- prosedür çağırma işlemi
procedure_adi(input_1, input_2, input_3, output_1, output_2);
   ifadeler;
```

3.3 VHDL Veri Nesneleri, Veri Tipleri, Operatörler ve Nitelikler

3.3.1 Veri Nesneleri

Program içersinde belirtilen veri türündeki değerleri tutan yapılara veri nesneleri denir. Bu veri nesnelerinin programa dâhil edilebilmesi için program içersinde bildirimini yapılarak tür, isim ve sınıfının belirtilmesi gerekir.

VHDL dilinde ***sabit, sinyal, değişken*** ve ***dosya*** olmak üzere dört adet veri nesnesi bulunur. Bu veri nesneleri hakkında yeterli bilgi aşağıda verilmiştir;

3.3.1.1 Sinyal (Signal)

Sinyal, sistemin giriş ve çıkışları ile iç hatları arasındaki ilişkiyi tanımlayan veri nesnesidir. Başka bir deyişle, *architecture* içerisinde eş zamanlı yapılar arasındaki fiziksel iletişimi sağlarlar. *Entity, package, architecture* ya da *blok* içerisinde tanımlanabilmelerine rağmen, *process, procedure* ve *fonksiyon* içinde sinyal tanımlaması yapılmaz. Tasarım içerisinde *sinyal* tanımlamasının aşağıdaki gibi yapılır;

```
signal sinyal_adi: sinyal_tipi [aralık] [:= başlangıç değeri];
signal saniye: integer range 0 to 59;
signal enable: bit;
signal my_data: std_logic_vector(1 to 8):= ”00001111”;
```

3.3.1.2 Sabit (Constant)

Bir kez belirtildikten sonra değeri değişmeyen sabit veri nesnesidir. Tasarımın okunabilirliğini artırır. *Entity, architecture, package, component, block, configuration* ve *altprogram* yapıları içerisinde tanımlanabilir. *Dosya* veya *Access* türünde tanımlanmaz. Tanımlaması aşağıdaki gibi yapılır;

```
constant sabit_adi: sabit_tipi := sabit_değeri
constant number_of_bytes: integer :=4;
constant e: real:= 2.718281828;
constant mask: bit_vector(31 downto 0) := (others => ‘1’);
```

Not: Sinyal atamalarında “<=” sembolü kullanılır. “:=” ifadesi ile ilk değer ataması yapılır. İlk değer ataması yapılmamış sinyallerin varsayılan değeri o türün eleman kümesinin en başındaki değeridir. “*others*” ifadesi ile vektörün tüm değerlerine aynı anda atama yapılabilir.

3.3.1.3 Değişken (Variable)

Değişken yapısı sadece altprogram (*process, function, procedure*) yapıları içerisinde tanımlanan ve *yerel* (local) bilgi taşıyan veri nesnesidir. Değeri hızlı bir şekilde değiştirilebilir ve çoklu sinyal atamalarına izin verir. Sadece tanımlandığı altprogram içerisinde kullanılır. Program içersindeki atama işlemleri ile değişkenlerin değerleri değiştirilebilir. Tanımlaması aşağıdaki gibi yapılır;

```
variable deg_adi: deg_tipi [aralik] [:=baslangic degeri];
variable saniye: integer 0 to 59;
variable count: integer := 0;
```

Değişkenlere değer atama işlemleri sinyallerden farklı olarak ":=" sembolü ile yapılır. Bir değişkene bir değer atandığı zaman işlem o anda gerçekleşir ve değişken bu değeri başka bir atama oluncaya kadar tutar.

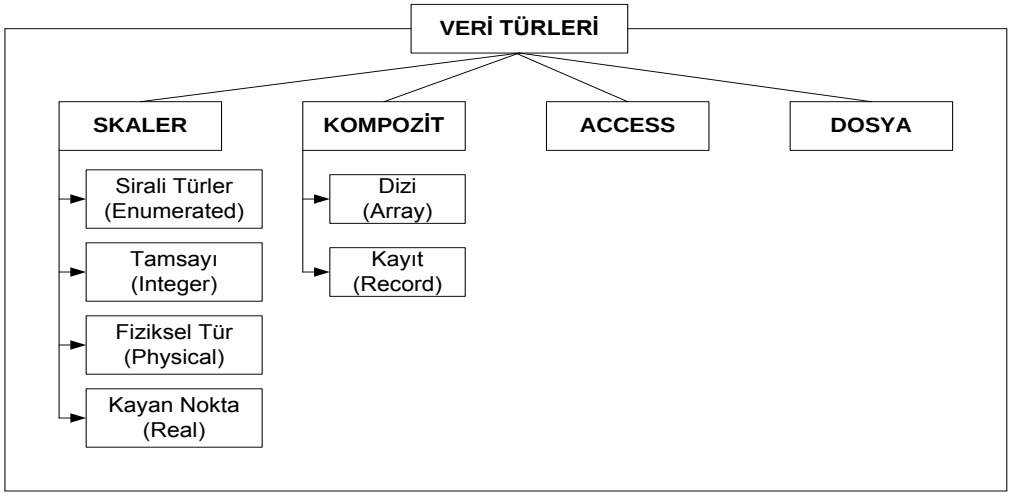
3.3.1.4 Dosya (File)

Dosya veri nesnesi VHDL tasarımı ile bilgisayarda fiziksel olarak kayıtlı olan dosya arasındaki iletişim için bir yol oluşturur. Dosya açma, dosya kapama, dosyadan okuma, dosyaya yazma ve dosya sorunu bulma gibi bazı işlemleri vardır.

3.3.2 Veri türleri

Sinyallerin, değişkenlerin ve sabitlerin program içersinde alabilecekleri değerleri belirlemek için veri türleri kullanılır. Veri türleri program içersindeki bildirimler esnasında belirlenir ve program çalışırken hiçbir şekilde değiştirilemezler. Veri türü bildirimleri *type* ve *subtype* komutları ile yapılır. *Type* komutu ile yeni bir veri türü tanımlanırken, *subtype* komutu ile var olan bir türün alt türü tanımlanır.

VHDL'de dört ana veri türü bulunur, bunlar *Scaler, Kompozit, Access* ve *Dosya* türleridir. Şekil 3.5'de VHDL dilinde bulunan bütün veri türleri gösterilmiştir.



Şekil 3.5 VHDL veri türleri

Bu veri türlerinden skaler ve dizi türlerinin altındaki bazı türler ön tanımlı veri tiplerindendir. Bu türlerin program içersinde tekrar tanımlanmalarına gerek yoktur. Ön tanımlı veri tipleri *standard (std)* kütüphanesi içinde yer almaktadır. Ayrıca IEEE kütüphanesinin altında bulunan bazı paketlerdeki veri türleri de VHDL tasarımlarında çok sık kullanılmaktadır. Bunun yanında VHDL dili bu kullanıcı tanımlı yeni veri tipi tanımaya da olanak sağlamaktadır.

Aşağıdaki tabloda VHDL tasarımlarında sıkça kullanılan veri türleri ilgili kütüphane ve paket ve ilgili paket içindeki veri türleri listelenmiştir. Listede "*synopsys veri türü*" veri türü olarak belirtilen türler Synopsys firması tarafından geliştirilmiş ve çok yaygın olarak kullanılan veri türleridir.

Çizelge 3.3 Std ve ieee kütüphanesinde bulunan veri tipleri

Kütüphane	**Paket**	**Veri Tipi**
STD	standard	*bit, bit_vector, boolean*, integer, natural, positive, *character*, string, real, time
IEEE	std_logic_1164	std_(u)logic, std_(u)logic_vector (synopsys veri türü)
IEEE	std_logic_arith numeric_std	signed, unsigned (synopsys veri türü)

3.3.2.1 Skaler Türler;

Herhangi bir anda sadece tek bir değer tutabilen veri türleridir. Tür olarak birçok değer içermesine rağmen, skaler olarak tanımlanan bir obje zamanın herhangi bir anında bu skaler değerlerden sadece bir tanesini tutabilmektedir.

Sıralı Türler (Enumerated) : Elemanlarını sıralı listeleme yöntemi ile tanımlayan veri tipidir. Ön tanımlı veri tiplerinden standart kütüphanede bulunan *boolean, bit (bit_vector), character* ve IEEE kütüphanesinde bulunan *std_logic, std_ulogic* veri tipleri bu gruba dâhil edilebildiği gibi ayrıca kullanıcı kendisine özel veri tipi de tanımlayabilir. Kullanıcı sıralı bir türü aşağıdaki gibi tanımlayabilir;

```
type tür_ismi is (eleman_1, eleman_2, …);
```

```
type color is (red, yellow, blue, gren, orange);
type my_logic is ('0', '1', 'Z');
```

Standart kütüphane içerİsinde ön tanımlı sıralı türler aşağıda incelenmiştir;

- **bit**: '0' ve '1' olmak üzere iki seviye lojik bilgi içerir. Değer ataması aşağıdaki gibi yapılır;

```
signal x: bit := '1';
signal y: bit := '0';
```

- **boolean**: True ya da False değerlerinden birini alır. Koşullu ifadelerde kullanılır. Varsayılan değeri False'dir.
- **character**: ISO 8859-1 karakter seti içersinde bulunan 256 karakteri içerir.

IEEE *kütüphanesi* içinde bulunan sıralı veri türleri de aşağıda incelenmiştir;

- **std_logic ve std_ulogic:** IEEE 1164 standartında tanımlanmış lojik sistemlerdir. Bu veri türlerinde değer ataması aşağıdaki gibi yapılır;

```
signal x: std_logic :='1';
signal y: std_ulogic := '0'
```

std_logic veri türü sekiz değerli lojik bilgi içerir, *std_ulogic* ise bu sekiz değerli lojik sisteme ek olarak 'U' lojik değerini de içerir. Aşağıdaki tabloda bu lojik sistemin elemanları verilmiştir;

Çizelge 3.4 Std_logic - std_ulogic lojik elemanları

Değer	Açıklama
U	Tanımlanmamış
X	Bilinmeyen
0	Mantıksal 0
1	Mantıksal 1
Z	Yüksek empedans
W	Zayıf bilinmeyen
H	Zayıf 1
L	Zayıf 0
-	Önemsiz (don't care)

Tamsayı (integer): *Standart paketinde* ön tanımlı olarak tanımlanmıştır. Değer aralığı 2,147,483,647 den +2,147,483,647'e kadardır. Öntanımlı olduğu için tasarım sırasında tekrar tanımlanmasına gerek yoktur. Fakat kullanıcıya kendi tamsayı türünü tanımlama fırsatı verir ve bu işlem aşağıdaki gibi yapılır;

```
type tür_adi is range alt_sınır to üst_sınır;
type tür_adi is range üst_sınır downto alt_sınır;
type tamsayi is range -3276 to 2000;
type deger is range 1000 downto 0;
```

Ayrıca *integer* veri tipinin *natural* ve *positive* olmak üzere iki adet alt türü bulunmaktadır. *Natural* türü sıfıra eşit ve sıfırdan büyük *integer* değerlerini ifade eder. *Positive* türü ise sıfırdan büyük olan *integer* değerlerini ifade eder.

Fiziksel Tür (Physical): Fiziksel tür, uzaklık, akım, zaman gibi fiziksel büyüklükleri nicelik belirten değerleri ifade etmek için kullanılır. Fiziksel tip kullanılarak 'temel birim' oluşturulur ve sonra bu temel birime bağlı alt ve üst birimler tanımlanır.

Fiziksel tür kullanılarak temel birim ve katları aşağıdaki gibi tanımlanır;

```
type tür_adi is range alt_sinir to ust_sinir
units
    temel_birim_adi;
    ikincil_birim_adi = deger Temel_birim_adi;
   ...
end units tür_adi
```

Ayrıca standart kütüphanede ön tanımlı *time* fiziksel türü bulunur.

- **time**: Time standart paketinde yer alan ön tanımlı bir fiziksel türdür. Zaman değerini gösterir ve değer aralığı -2147483647 ile +2147483647 arasındadır. Aşağıda time türünün kullanımı ile ilgili örnek verilmiştir;

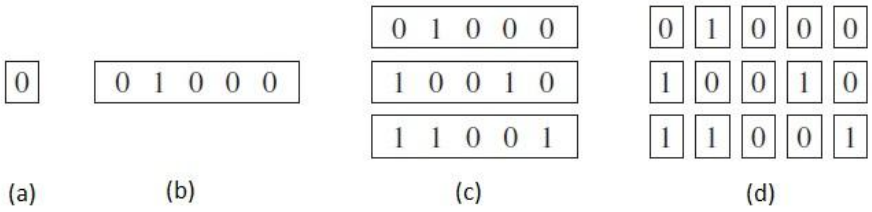
```
constant saat_periodu: time := 2 us;
```

Kayan Nokta (Reel): Standart paketinde ön tanımlı olarak tanımlanmıştır ve reel sayıları ifade eder. Değer aralığı -1.7014111e+308'den 1.7014111e+308'e kadardır. Ayrıca kullanıcı tamsayı türünde olduğu gibi kayan nokta türünde de kendi veri türünü tanımlayabilir.

3.3.2.2 Kompozit Türler

Kompozit türünün *array* ve *record* olmak üzere iki alt türü vardır.

Dizi (Array): Aynı türdeki bir ya da daha fazla elemanı gruplayarak tek bir nesne olarak tanımlayan veri türüdür. Dizi içindeki her elemanın belli bir aralık içinde index numarası vardır. Dizi elemanlarının her birine bağımsız olarak değer ataması yapılabilir.[1]



Şekil 3.6 VHDL dilinde veri dizileri (a) Skaler, (b) 1D, (c) 1Dx1D, (d) 2D

Şekil 3.6 VHDL dilinde yer alan tüm veri dizilerini göstermektedir [45]. Bu veri dizileri tek bir değer (skaler), vektör (1D), vektör dizileri (1Dx1D) ve skaler değer dizilerinden (2D) oluşur. Bu dizi çeşitlerinden tek bitlik scaler dizileri ve bir boyutlu vektör dizileri öntanımlı VHDL veri tiplerinden bazılarını içerir. *Bit, std_logic, std_ulogic*, ve *boolean* scaler grubunda, *bit_vector, std_logic_vector, std_ulogic_vector, integer, signed, unsigned* tek boyutlu vektör grubunda yer alır.

1Dx1D ve 2D diziler öntanımlı olmadığı için tasarımda ihtiyaç duyulduğu zaman kullanıcı tarafından tanımlanması gerekir. Aşağıda 1D, 1Dx1D ve 2D dizi tanımlama örnekleri verilmiştir;

```
-- 1D dizi tanımı
type vector is array(0 to 3) of std_logic;
signal x: vector;
-- 1Dx1D dizi tanımı
type mtx is array(0 to 3) of std_logic_vector(7 downto 0);
signal x: mtx ;
-- 2D dizi tanımı
type 2D_dizi is array (0 to 3, 7 downto 0) of std_logic;
```

Standart kütüphanede bulunan ön tanımlı dizi türleri aşağıda incelenmiştir;

- **bit_vector:** Elemanlarının her biri ‘0’ ya da ‘1’lerden oluşan dizidir. Tasarım içersinde aşağıdaki gibi kullanılır;

```
signal data: bit_vector (7 downto 0)  ;
data(3) <= '0';  -- dizinin üçüncü elemanına değer atama
data(7 downto 4) <= "0000";
```

- **string**: Karakterlerden oluşan tek boyutlu bir dizi türüdür. Tasarım içersinde aşağıdaki gibi kullanılır;

```
constant isim: string := "FPGA";
```

IEEE kütüphanesinde bulunan dizi türleri aşağıda incelenmiştir;

- **std_logic_vector, std_ulogic_vector:** IEEE kütüphanesi std_logic_1164 paketinin içinde tanımlanmışlardır. Elemanları Çizelge 3.4’de bulunan lojik sistemden oluşan dizilerdir. Tasarımda içersinde aşağıdaki gibi kullanılablir.

```
signal a: std_logic_vector(7 downto 0):= "0011ZZZZ";
signal b: std_ulogic_vector (5 downto 0):= "010011";
```

- **signed, unsigned:** IEEE kütüphanesi içindeki std_logic_arith paketinin içinde tanımlanmışlardır. *Unsigned* veri tipi işaretsiz sayıları temsil eder. En küçük değeri sıfırdır. N bitlik işaretsiz bir sinyal ya da değişkenin alabileceği değer aralığı 0 ile $2^N - 1$ ‘dir. *Signed* veri tipi işaretli sayıları temsil eder ve en soldaki bit işaret bitidir. Bu bit değeri ‘1’ ise sayı negatif, ‘0’ ise pozitifdir. Negatif sayılar ikinin tümleyeni şeklinde gösterilir. N bitlik işaretli bir sinyal ya da değişken -2^{N-1} ile $2^{N-1} - 1$ arasında sayısal değer alır. Bu veri türlerinde atama yapılırken değer aralığı belirtilmelidir. Aşağıda bu türlerin tasarım içersinde nasıl kullanılacağı gösterilmiştir;

```
variable temp: unsigned (1 to 8);
temp := "11110000";
temp(5):= '0 ;
signal sinyal: signed (5 downto 0);
sinyal <="011010";
```

Kayıt (Record) : Yapı olarak dizilere benzer fakat dizilerden farkı ise farklı türdeki veri tiplerini gruplayarak tek bir nesne olarak tanımlayan veri türüdür. VHDL tasarımı içinde *record* kullanımı aşağıdaki gibi olur;

```
type kayit_adi is record
     eleman_adi: eleman_türü;
     eleman_adi: eleman_türü;
      ...
end record;
```

```
-- record örneği
type birthday is record
     day: integer range 1 to 31;
     month: month_name;
end record;
```

3.3.2.3 Access Türü

Access türü diğer yazılım dillerindeki 'pointer'lara benzerler. Değeri derleme sırasında ortaya çıkabilecek olan nesneler için kullanılan veri türüdür. Access türü ile nesnelere dinamik olarak depolama alanı tahsis edilebilir veya soyut veri yapıları ile ilgili uygulamalar için kullanılır.

3.3.2.4 Dosya (File) Türü;

Dosya içine veri yazma ya da dosyadan veri okuma işlemleri için kullanılan veri türüdür. Bu veri türü genellikle gerekli verileri kayıtlı dosyadan almak ya da simülasyon sonucu oluşan verileri bir başka dosyaya kaydetmek için kullanılır. VHDL dilinde standart kütüphanedeki textilo paketinde ve IEEE kütüphanesinde std_logic_textilo paketinde dosya işlemleri için bir takım tanımlı özel prosedür ve türler bulunur.

3.3.3 Operatörler

VHDL dilinde *standart* kütüphanede bulunan ön tanımlı operatörler ve IEEE kütüphanesinde bazı veri türlerine özgü operatörler bulunur. Ayrıca kullanıcı tarafından da yeni bir operatör tanımlanabilir. Atama, lojik, aritmetik, karşılaştırma, dönüşüm ve birleştirme operatörleri VHDL dilindeki ön tanımlı operatörlerdir.

3.3.3.1 Atama Operatörleri

Atama operatörleri: " <= ", " := ", " => "

" <= ": Sinyale değer atamak için kullanılır.

" := ": Değişkene, sabite veya generic'e değer atamak için ya da başlangış değeri atamak için kullanılır.

" => " : Tek bir vektöre veya "others" ifadesine değer atamak için kullanılır [47].

```
signal x: std_logic_vector(0 to 7);
variable y: std_logic_vector(3 downto 0);
x <= "10000000";
y := "0101";
x <= (0 => '1', others => '0');
```

3.3.3.2 Birleştirme Operatörleri

Birleştirme operatörleri ; " & ", " , "

Bit, std_logic ve std_ulogic (bit_vector, std_logic_vector, std_ulogic_vector) türündeki verileri birleştirmek ya da bir dizinin sağına ve soluna eleman eklemek için kullanılır [47].

```
k: constant bit_vector(1 to 4 ) :="1100";
x <= ('Z', k(2 to 3),"1111");     --sonuc: x <= "Z101111"
y <= ('Z' & k(2 to 3) & "1111");  --sonuc: y <= "Z101111"
```

3.3.3.3 Mantıksal (Lojik) Operatörler:

Lojik operatörler; NOT, AND, OR, NAND, NOR, XOR, XNOR

Bu operatörler bit(bit_vector), std_(u)logic(std_(u)logic_vector) ve boolean tipi için tanımlanmıştır. Boolean türü için '0'ın karşılığı "false", '1'in karşılığı "true" olarak tanımlanmıştır. Mantıksal işleme girecek olan verilerin aynı tür ve boyutta olması gerekir. Not operatörü diğer mantıksal operatörlerden önceliklidir [47].

```
y <= not a and b        --(a'.b')
y <= not (a nand b)     --(a.b)'
y <= a nand b;          --(a.b)'
```

3.3.3.4 Aritmetik Operatörler

Aritmetik operatörler; +, -, *, / , **, mod, rem, abs

Aritmetik operatörler, tamsayı(*integer*), reel sayı(*real*), *signed*, *unsigned* türleri için tanımlıdırlar, fakat *std_logic_vector* türü için sadece toplama ve cıkarma operatörleri tanımlıdır. Matematikte geçerli olan işlem sıraları ve operatör anlamları VHDL dili için de geçerlidir[47].

```
x <= (a+b)**N
y <= ABS(a) + ABS(b);
z <= a/(a + b);
```

3.3.3.5 Karşılaştırma Operatörleri

Karşılaştırma operatörleri ; =, */=, <, <=, >, >=*

VHDL verileri arasında karşılaştırma işlemleri yapmak için kulanılırlar. İşlem sonucunda ortaya çıkan değer boolean türünde olur yani doğru ya da yanlıştır. Karşılaştırma işlemine girecek dizilerin türlerinin ve boyutlarının aynı olması gerekmektedir.

```
if a >= b then x <= '1';
```

3.3.3.6 Kaydırma/Döndürme Operatörleri

Bit ve boolean türündeki tek boyutlu dizilerde kaydırma ve döndürme işlemleri gerçekleştirmek için kullanılırlar. VHDL dilinde kullanılan kaydırma ve döndürme operatörleri Çizelge 3.5'te listelenmiştir. Lojik ve aritmetik kaydırma operatörlerinde dizi elemanını operatörün sağındaki değer kadar sola ya da sağa kaydırır. Fakat lojik kaydırma işleminde giden elemanın yerine '0' gelirken aritmetik kaydırma işleminde, SLA operatörü için giden elemanın yerine dizinin en son elemanın değeri gelirken SRA operatörü için giden elemanın yerine dizinin en başındaki değer gelir.

Çizelge 3.5 VHDL dilindeki kaydırma ve döndürme operatörleri

Op.	Görev	Örnek	Sonuç (x)
SLL	Lojik Sola kaydırma	x <= "10010101" sll 2	"01010100"
SRL	Lojik Sağa kaydırma	x <= "10010101" srl 3	"00010010"
SLA	Arimetik sola kaydırma	x <= "10010101" sla 3	"10101111"
SRA	Arimetik sağa kaydırma	x <= "10010101" sra 2	"11100101"
ROL	Sola döndürme	x <= "101000" rol 3	"100010"
ROR	Sağa döndürme	x <= "101000" ror 2	"011010"

3.3.4 Nitelikler (Attributes)

Nitelikler, veri tipleri, diziler ve sinyaller hakkında daha fazla bilgi edinmek için kullanılan VHDL araçlarıdır. VHDL dilinde ön tanımlı nitelikler olduğu gibi kullanıcı kendi tasarımına uygun olarak da nitelik tanımlayabilir.

Çizelge 3.6'da, bit_vector, std_(u)logic_vector, integer, natural, positive (un)signed veri türleri kullanılan nitelikler ve özellikleri listelenmiştir. Bu nitelikler, veri türlerinin tanımlanma aralığı hakkında kullanıcıya değer geri döner.

Çizelge 3.6 VHDL dilinde diziler içinen kullanılan nitelikler

Nitelik Adı	Özellik
x'low	x verisinin en küçük index numarasını verir.
x'high	x verisinin en büyük index numarasını verir.
x'right	x verisinin en sağdaki index numarasını verir.
x'left	x verisinin en soldaki index numarasını verir.
x'lenght	x verisinin boyutunu verir.
x'range	x verisinin tanımlama aralığını verir.
x'reverse_range	x verisinin ters tanımlama aralığını verir.

```
signal x: std_logic_vector(7 donto 0);
x'low = 0, x'high = 7, x'right = 0, x'left = 7, x'length = 8
x'range = (7 downto 0), x'reverse_range = (0 to 7)
```

Çizelge 3.7'de sıralı veri türleri (enumerated) için kullanıcıya veri türü elemanları pozisyonları ve değerleri hakkında değer geri döndüren nitelikler listelenmiştir;

Çizelge 3.7 VHDL dilinde sıralı türler için kullanılan nitelikler

Nitelik Adı	Özellik
x'val (position)	x verisinin belirtilen konumdaki değerini verir
x'pos (value)	x dizisinin belirtilen değerinin konumunu verir,
x'leftof (value)	x dizisinin belirtilen değerinin solundaki bulunan veriyi verir
x'low (row, position)	x dizisinin belirlenen konumdaki değerini verir.

VHDL dilinde yukarıda ifade edilen niteliklere ek olarak tasarımlarda sıkça kullanılan olay ilişkili niteliklerde bulunmaktadır.

Çizelge 3.8 VHDL dilinde olay ilişkili nitelikler

Nitelik Adı	Özellik
x'event	x değerinde değişim olduğunda *true* değeri döner.
x'stable	x değerinde değişim olmadığı zaman *true* değeri döner.
x'active	x = 1 ise *true* değeri döner.
x'quiet (zaman)	Belirleren zamanda x değeri değişmezse true değeri döner.

Olay ilişkili nitelikler VHDL tasarımlarında genellikle saat sinyallerinin değişimleri, yükselen ve düşen kenar tespitleri için kullanılır. Aşağıda verilen dört atama işlemi de eşdeğer ve sentezlenebilirdir.

```
if (clock'event and clock = 1) …
if (not clock'stable and clock = 1) …
wait until (clock'event and clock = 1) …
if (rising_edge(clock)) …
```

3.4 Sıralı İfadeler

Sıralı ifadeler bilinen diğer programlama dillerinde olduğu gibi yazıldığı sıraya göre işleyen ifadelerdir. Bu ifadeler VHDL dilinde yalnızca *process* ve alt programlar

(*fonksiyon ve procedure*) içinde kullanılırlar. VHDL dilinde kullanılan sıralı ifadeler, *if* yapıları, *case* yapısı, *wait* yapısı ve *döngü* yapılarıdır [43][46].

3.4.1 If/then Yapısı

Bir ya da birden çok koşula bağlı işlemleri kontrol etmek için kullanılan yapılardır. Koşullu ifadenin sonucu 'doğru' veya 'yanlış' mantığına göre belirlenir. Yani koşul sağlanıyorsa lojik 1 değeri geri döner ve sıralı ifadeler işleme girer, koşul sağlanmıyora lojik 0 değeri geri döner ve sıralı ifadeler işlenmeden atlanır. Kullanımı aşağıda gösterilmiştir.[43]

```
if (koşul) then
  --sıralı ifadeler;
end if;
```

```
if (koşul) then
  --sıralı ifadeler;
else
  --sıralı ifadeler;
end if;
```

```
if (koşul) then
  --sıralı ifadeler;
elsif (koşul) then
  --sıralı ifadeler;
else
  --sıralı ifadeler;
end if;
```

3.4.2 Case Yapısı

Bir değişken veya sinyalin alabileceği değerlere göre yapılacak işlemlerin seçiminde kullanılan yapıdır. *Case* yapısı da diğer sıralı ifadele gibi sadece *process* ve alt programlar içersinde kullanılırlar [43][47].

```
case anahtar_kelime is
   when değer_1 => atamalar;
   when değer_2 => atamalar;
   . . .
   when others;
end case;
```

```
case secim is
   when "00" => sonuc <= a;
   when "01" => sonuc <= b;
   when "10" => sonuc <= c;
   when others => sonuc <= d;
end case;
```

3.4.3 Wait Yapısı

Bu yapı herhangi bir işlemi bekletmek için kullanılır ve *process* içersinde bu ifade kullanıldığı zaman duyarlılık listesi (*sensitivity list*) kullanılmaz. *Wait* ifadesi tasarım içersinde aşağıdaki şekillerde kullanılabilir;

Ayrıca process içersinde wait ifadesinin yanında herhangi bir koşul belirtilmemişse *process* burada sonsuza kadar durdurulur[43][47].

```
wait until sinyal_şartı;
wait on sinyal_listesi
wait for zaman;
```

```
wait until clk'event and clk = '1'
wait on clock, reset;
wait for 5 ns;
```

3.4.4 Döngü Yapıları

VHDL dilinde diğer programlama dillerinde olduğu gibi belirli işlemlerin tekrarlı bir şekilde gerçekleştirilmesi için döngüler kullanılır. Sonsuz döngü, for döngüsü ve while döngüsü en çok kullanılan döngü yapılarıdır.

```
--for döngüsü
[etiket:]for sayac in ara loop
    [sıralı ifadeler];
end loop [etiket];
```

```
U1: for i in 0 to 10 loop
    i <= i + 1;
end loop U1;
```

```
--while döngüsü
[etiket:] while sart loop
    [sıralı ifadeler];
end loop [etiket];
```

```
döngü: while (i < 10) loop
    wait for 5 ns;
    i <= i + 1;
end loop döngü;
```

```
--sonsuz döngü
[etiket:] loop
    [sıralı ifadeler];
end loop [etiket];
```

```
U1: loop
    sayici <= sayici + 1;
end loop U1;
```

VHDL tasarımlarında döngü içinde kullanılan bazı yardımcı yapılar bulunmakadır; *Next* Yapısı ve *Exit* Yapısı.

Next Yapısı: Bu yapı döngü içinden çıkmadan döngüyü yeniden başlatmak için kullanılır. Tasarımımın yapısına göre tek başına kullanılablieceği gibi bir şarta bağlı olarak da kullanılablir. Kullanımı aşağıdaki gibidir;

```
-- next yapısı
[etiket1:] ... loop
   [etiket:] next [etiket1]
   [when kosul];
    . . .

end loop etiket1;
```

```
temp := 0;

U1: for i N-1 downto 0 loop
  next when x(i) = '1';
  temp := temp +1;

end loop U1;
```

Exit Yapısı: Exit yapısı ile döngü tamamlanmadan döngüden çıkma işlemi gerçekleştirilir. Exit yapısının önünde bir koşul varsa döngüden çıkmak bu koşula bağlı olur. Kullanım şekli aşağıdaki gibidir;

```
-- exit yapısı
[etiket1:] ... loop
   [etiket:] exit [etiket1]
   [when kosul];
    ...

end loop etiket1;
```

```
temp := 0;

U1: for i N-1 downto 0 loop
  exit when x(i) = '1';
  temp := temp +1;

end loop U1;
```

3.5 Eş zamanlı İfadeler

Sıralı işlem yapan diğer bilgisayar programların aksine VHDL dili yapısal olarak eşzamanlı (paralel) çalışır. Yani bütün ifadelerin öncelik sırası aynıdır. Eş zamanlı ifadeler program içersinde yazılış sırasına bakmaksızın aynı anda işleme alınan ifadelerdir.

VHDL dili yapısı gereği eş zamanlı çalışan tasarımlar için çok kullanışlıdır, fakat *process, fonksiyon* ve *prosedür* yapıları sayesinde sıralı işlem gerektiren sistemlerde de kullanılır. Fakat tasarıma genel olarak bakıldığı zaman bu yapılar eş zamanlı olarak çalışır. VHDL dilinde *process* ve alt programlar (*fonksion, prosedür*) ifadeleri dışında eş zamanlı olarak çalışan ifadeler, architecture içersinde operatörler kullanılarak yapılan atamalar, when yapısı, generate yapısı ve blok yapısıdır.

3.5.1 Process

Sıralı işlem gerektiren yerlerde kullanılan ve içerisindeki komutların sıralı bir şekilde çalışan VHDL yapısıdır. Bu yapının eş zamanlı ifadeler bölümünde incelenmesinin nedeni ise, içersindeki kodlar sıralı olarak çalışmasına rağmen *architecture* içersinde process'ler eş zamanlı olarak çalışmaktadır. Prosess yapılarını tetiklemek için kontrol listesi *(sensitivity list)* ya da *wait on* ifadesi kullanılır. Kontrol listesinin içine ya da *wait on* komutundan sonra yazılan tetikleyici ifadelerden herhangi biri değiştiği zaman process içindeki komutlar çalışmaya başlar.

Kullanım şekli aşağıdaki gibidir;

```
-- process yapısı
[etiket:] process(kontrol_list)
   [nesne_bildirimleri];
begin
    [sıralı_ifadeler];
end loop etiket;
```

```
-- process yapısı
[etiket:] process
   [nesne_bildirimleri];
begin
   [sıralı_ifadeler];
   wait on kontrol_listesi;
end loop etiket;
```

Ardışıl devre tasarımlarında genellikle iki process yapısı bulunur. Bunlardan biri sistemin çalışma şeklinin belirlendiği kombinasyonel devre yapısını oluşturur. Diğeri ise saat sinyalini oluşturan yapıdır ve sistemin depolama elemanını ifade eder.

3.5.2 Architectre İçinde Operatör Kullanılarak Yapılan Atamalar

VHDL dilinde paralel kod yazmanın en kolay yolu architecture içine yazılan ifadelerdir. Operatörler kullanılarak kodlar architecture içine yazılan ifadeler eş zamanlı olarak çalışırlar.

3.5.3 When Yapısı

When yapısı ile sinyallere eşzamanlı olarak atama yapmak için kullanılır. Bu yapı iki türlü kullanımı mevcuttur. İlk olarak *when/else* yapısı ile koşullu sinyal ataması, ikinci olarak da *with/select/when* yapısı ile de seçimli sinyal ataması yapılır.

Kullanım şekilleri aşağıdaki gibidir;

```
sinyal<=
    atama_1 when koşul_1 else
    atama_1 when koşul_1 else
    . . .
    atana_n;
```

```
with secim select
sinyal <=
     atama_1 when secenek_1
     atama_1 when secenek_1
     . . .
     atama_n when others;
```

3.5.4 Gererate Yapısı

Generate ifadesi if ve for yapılarının eş zamanlı olarak kullanılmasına olanak sağlar. For/generate yapısı tekrarlı ifadelerde ve if/generate yapısıda koşullu ifadelerde kullanılır. Karmaşık programlarda for/generate ve if/generate yapılarında da etiket kullanılabilir.

Kullanım şekilleri aşağıdaki gibidir.

```
for sayac in aralık generate
    [bildirimler];
begin
    [eş zamanlı ifadeler];
end generate;
```

```
If koşul generate
    [bildirimler];
begin
    [eş zamanlı ifadeler];
end generate;
```

3.5.5 Blok Yapısı

Blok yapısı, büyük ve karmaşık programlarda architecture içindeki eşzamanlı ifadeleri gruplamak için kullanılan bir VHDL yapısıdır. Basit blok ve koşullu (guarded) blok olmak üzere iki türü vardır. Architecture içersinde blok yapıları eş zamanlı olarak işlenirler. Hiçbir bloğun diğerine göre önceliği yoktur.

Kullanım şekilleri aşağıdaki gibidir.

```
-- basit blok
[etiket:] block
    [nesne bildirimleri];
begin
    [eş zamanlı ifadeler];
end block [etiket];
```

```
-- koşullu blok
[etiket:] block (koşul)
    [nesne bildirimleri];
begin
    [eş zamanlı ifadeler];
end block [etiket];
```

BÖLÜM 4

BEŞ EKLEMLİ ROBOT KOL

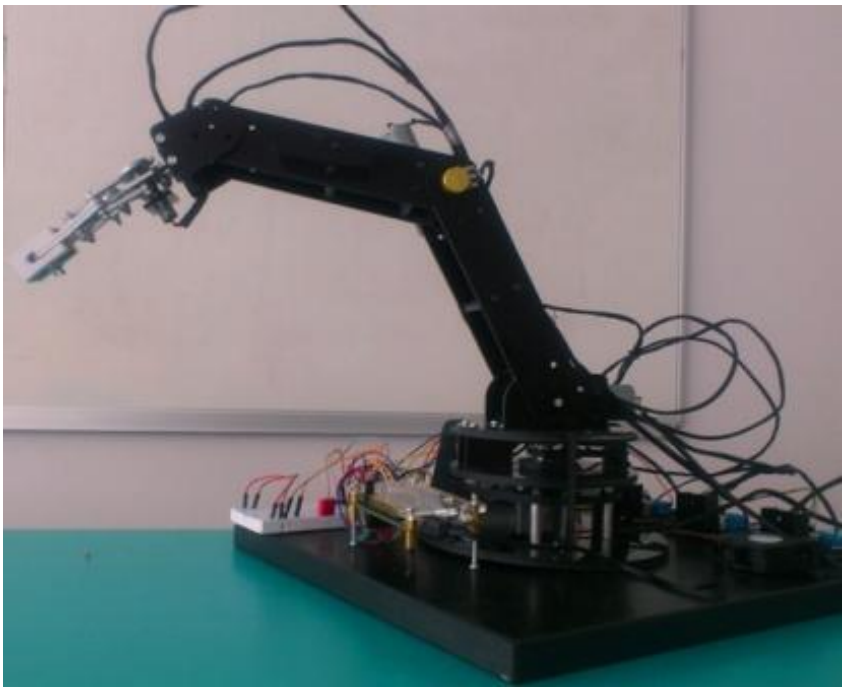
Tezin uygulama bölümünde FPGA kullanılarak beş eksenli bir robot kolunun pozisyon kontrolü üzerinde çalışılmıştır. Bu bölümde deney düzeneğinde kullanılan elemanlar hakkında bilgiler verilmiştir. Bu elemanlar aşağıda listelenmiştir;

- Beş eksenli robot kolu
- FPGA kontrol kartı
- Motor sürücü devreleri
- Quanser Q2 veri toplama modülü

4.1. Robot Modelinin Elde Edilmesi

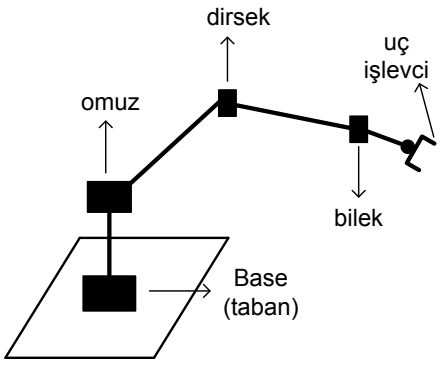
Çalışma kapsamında kullanılan robot kol hobi ve akademik çalışmalarda kullanılmak üzere tasarlanmıştır. İşlevi mümkün olduğu kadar endüstriyel uygulamalardaki robot kollarına benzetilmeye çalışılmıştır. Robot kolu beş ekleme sahiptir. Bu eklemlerden üç tanesi menteşe hareketi, bir eklem ekseni etrafında dönme hareketi yapmakta ve son eklem de ise tutaç (gripper) bulunmaktadır. İlk dört ekleminde dc motor ve bu dc motorların dönme hareketlerini ölçmek için motor şaftlarına bağlı potansiyometreler bununmaktadır. Uç işlevcide ise sadece açma ve kapama hareketini gerçekleştiren dc motor bulunmaktadır. Bu bölümde robot kolunun her bir eklemi ayrı birer sistem olarak ele alınmış ve ayrı ayrı doğrusal modelleri elde edilmiştir. Modelleme işlemi model tabanlı dogrulama (model based verification) yöntemi ile gerçekleştirilmiştir.

Şekil 4.1'de robot kolunun resmi görülmektedir;



Şekil 4.1 Beş eksenli robot kol

Şekil 4.2'de robot kolunun taban, omuz, dirsek bilek ve uç işlevci olarak bölümleri gösterilmiştir.

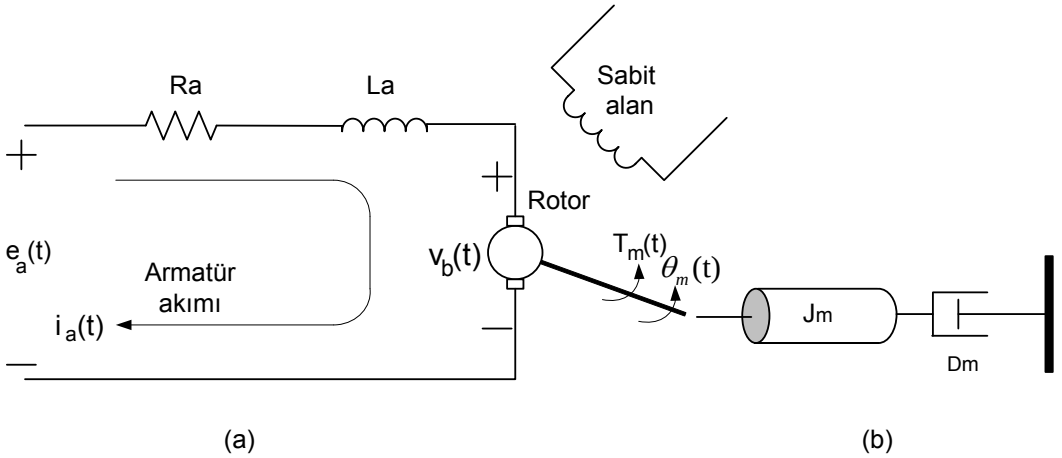


Şekil 4.2 Robot kol bölümleri

Robot sisteminin modelleme işlemi, her robot ekleminin matematiksel olarak ifade eden transfer fonksiyonlarının elde edilmesi işlemidir. Robot kolu lineer olmayan bir sistem olmasına rağmen bu uygulama için doğrusal model tasarlanmıştır. Modelleme işlemi sisteme gerçek zamanlı giriş işaretleri uygulayarak elde edilen çıkış işaretlerinin yorumlanması ile gerçekleştirilmiştir ve sonuç olarak robot kolunun yaklaşık modeli elde edilmiştir. Her robot ekleminde bir adet redüktörlü dc motor ve sisteme geri besleme sağlayan ölçme cihazı olarak potansiyometre bulunmaktadır. Her eklem kendi ağırlığı ile birlikte üst eklemlerinde ağırlığını taşımaktadır. Elde edilen model robotun

yüksüz durumu için elde edilmiştir ve robotun taşıyacağı yük ağırlığının küçük olduğu düşünülerek bu model uygulama için yeterli görülmüş ve başarılı sonuçlar elde edilmiştir.

Modelleme işlemine başlamadan önce, dc motor ve redüktör sisteminin transfer fonksiyonunun nasıl elde edildiği ve sonucunda da her bir eklem sisteminin nasıl ikinci dereceden bir sistem olduğu gösterilmiştir. Şekil 4.3'de redüktörlü bir DC motorun elektriksel devre şeması ve eşdeğer mekanik yükleme şemasına yer verilmiştir.



Şekil 4.3 a) Dc motor eşdeğer devresi b) Dc motor eşdeğer mekanik şeması

Dc motor, elektronik ve mekanik kısımdan oluşan ve giriş olarak uygulanan gerilim sonucunda çıkışında yer değişimi oluşturan elektromekanik komponenttir. Dc motor stator, armatür ve rotor bölümlerinden oluşur. Stator, üzerindeki mıknatıslar sayesinde sabit manyetik alan oluşturur ve armatür akımı bu manyetik alandan dik açılı bir şekilde geçtiği zaman üzerine bir manyetik kuvvet etti eder. Bu manyetik kuvvetin oluşturduğu tork etkisi de motorun hareketli kısmı olan rotoru döndürür. Akım taşıyan rotor manyetik alanda döndüğü için rotor gelirimi hızı ile orantılıdır ve bu ifade aşağıdaki matematiksel denklem ile ifade edilir;

$$v_b(\mathrm{t}) = \mathrm{K}_b \frac{d\theta_m(\mathrm{t})}{dt} \qquad (4.1)$$

Burada $\mathrm{v_b(t)}$ zıt elektromanyetik kuvvet (back emf), $\mathrm{K_b}$ zıt elektromotor katsayısı ve $\mathrm{d\theta_m(t)/dt}$ ise motor açısal hızını ifade eder.

Denklem 4.1'in Laplace dönüşümü alınırsa Denklem 4.2 elde edilir;

$$V_b(\mathrm{s}) = \mathrm{K}_b\, s\theta_m(\mathrm{s}) \tag{4.2}$$

Ayrıca armatür akımı ($i_a(\mathrm{t})$), armatür gerilimi ($e_a(\mathrm{t})$) ve zıt elektronanyetik kuvvet ($\mathrm{v_b}(\mathrm{t})$), arasındaki ilişki aşağıdaki gibi ifade edilir;

$$R_a i_a(\mathrm{t}) + \mathrm{L}_a \int_0^t i_a(\mathrm{t})dt + v_b(\mathrm{t}) = \mathrm{e}_a(\mathrm{t}) \tag{4.3}$$

Burada R_a armatür direnci ve L_a armatür endüktans değerini ifade eder.

Denklem 4.3'in Laplace dönüşümü alınırsa Denklem 4.4 elde edilir;

$$R_a I_a(\mathrm{s}) + \mathrm{L}_a\, sI_a(\mathrm{s}) + V_b(\mathrm{s}) = E_a(\mathrm{s}) \tag{4.4}$$

Son olarak da motorun ürettiği tork ($\tau_{\mathrm{m}}(\mathrm{t})$) ve armatür akımı ile arasındaki ilişki Denklem 4.5 ile verilir. Burada K_t tork sabitidir.

$$\tau_m(\mathrm{t}) = K_t\, \mathrm{i}_a(\mathrm{t}) \tag{4.5}$$

Denklem 4.5'in Laplace dönüşümü alınırsa Denklem 4.6 elde edilir;

$$T_m(\mathrm{s}) = K_t I_a(\mathrm{s}) \tag{4.6}$$

Denklem 4.6 yenidem düzenlenirse;

$$I_a(\mathrm{s}) = \frac{1}{K_t} T_m(\mathrm{s}) \tag{4.7}$$

Denklem 4.2 ve 4.7, Denklem 4.4'de yerine yazılırsa Denklem 4.38 elde edilir;

$$\frac{(R_a + \mathrm{L}_a\, s)T_m(\mathrm{s})}{K_t} + \mathrm{K}_b\, s\theta_m(\mathrm{s}) = E_a(\mathrm{s}) \tag{4.8}$$

Sistemin transfer fonksiyonunu ($\theta_m(\mathrm{s}) / E_a(\mathrm{s})$) elde etmek için $T_m(\mathrm{s})$'nin $\theta_m(\mathrm{s})$ cinsinden yazmak gerekir. Bu ifade Şekil 4.3 b'de gösterilen bir motorun mekanik yüklenmesi eşdeğer şeması yardımıyla elde edilir.

$$\tau(t) = \mathrm{J}_m \frac{d^2\theta_m(t)}{dt^2} + D_m \frac{d\theta_m(t)}{dt} \tag{4.9}$$

Burada $\mathrm{J_m}$ armatürün ve yükün eşdeğer ataleti ve $\mathrm{D_m}$ de armatürün ve yükün eşdeğer viskos sürtünme katsayısıdır.

Denklem 4.9'in Laplace dönüşümü alınırsa Denklem 4.10 elde edilir;

$$T_m(\mathrm{s}) = (\mathrm{J}_m s^2 + D_m s)\theta_m(\mathrm{s}) \tag{4.10}$$

Denklem (4.10), denklem(4.8)'de yerine yazılırsa;

$$\frac{(R_a + \mathrm{L}_a s)(\mathrm{J}_m s^2 + D_m s)\theta_m(\mathrm{s})}{K_t} + \mathrm{K}_b s\theta_m(\mathrm{s}) = E_a(\mathrm{s}) \tag{4.11}$$

Armatür endüktans değeri (L_a), armatür direnç (R_a) değerinin yanında çok küçük kaldığı için ihmal edilebilir.

$$\left[\frac{R_a}{K_t}(\mathrm{J}_m s + D_m) + \mathrm{K}_b\right] s\theta_m(\mathrm{s}) = E_a(\mathrm{s}) \tag{4.12}$$

Denklem (4.13)'den bir dc motor redüktör sisteminin transfer fonksiyonu elde edilir [51].

$$\frac{\theta_m(\mathrm{s})}{E_a(\mathrm{s})} = \frac{1/(R_a \mathrm{J}_m)}{s\left[\mathrm{s} + \frac{1}{J_m}(D_m + \frac{\mathrm{K}_t \mathrm{K}_b}{R_a})\right]} \tag{4.13}$$

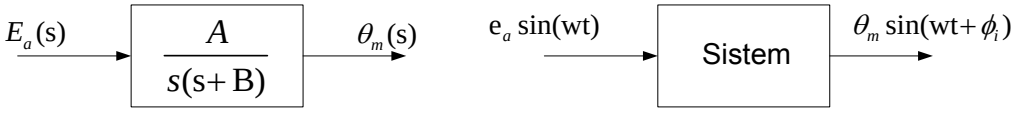
Bu transfer fonksiyonunun genelleştirilmiş şekli Denklem 4.14'te gösterilmiştir;

$$\frac{\theta_m(\mathrm{s})}{E_a(\mathrm{s})} = \frac{A}{s(\mathrm{s} + \mathrm{B})} \tag{4.14}$$

Sonuç olarak dc motor-redüktör sistemi ikinci dereceden bir transfer fonsiyonu ile ifade edilir. Uygulamada robot kolunun dört ekleminde dc motor ve enkoder bulunmaktatır ve sistemi kontrol etmek için bu eklemlerin matematiksel modellerinin bilinmesine ihtiyaç duyulur. Bu bölümün devamında her bir eklem sistemi için denklem 4.14'te ifade edildiği gibi dört adet transfer fonksiyonu elde edilmiştir. Bu modeller sistemin gerçek zamanlı analizleri sonucunda elde edilen grafikler yardımı ile hesaplanmıştır. Osiloskop yardımı ile fiziksel sistemden frekans cevabı bilgileri

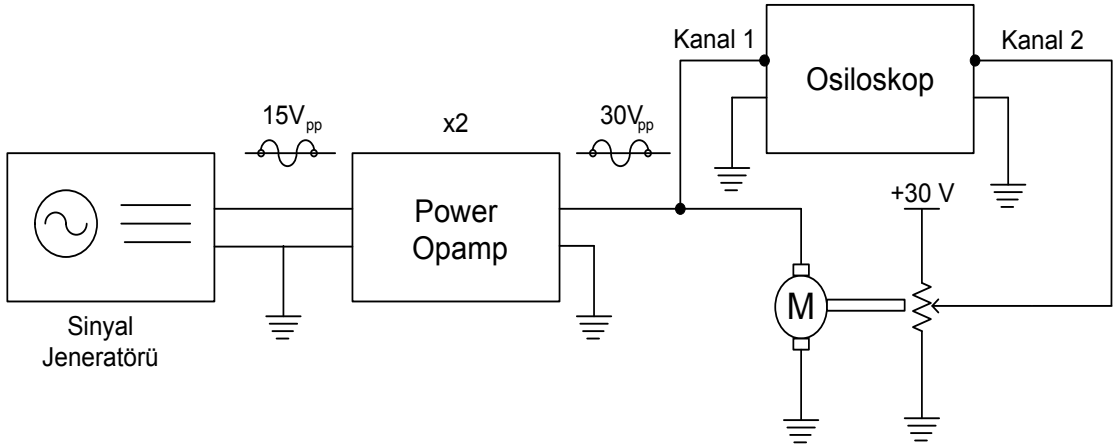
toplanmış ve bu bilgiler sistemin matematiksel modelinin elde edilmesinde kullanılmıştır.

Kararlı halde, lineer bir sisteme sinüzoidal bir giriş uygulandığında sistem aynı frekansta bir sinüzoidal çıkış üretir. Bu çıkış cevabı giriş ile aynı frekansta olmasıyla beraber genlik ve faz açısı girişten farklı olur. Bu farklılılar girişin bir fonksiyonudur. Bu yaklaşım Şekil 4.4 ile ifade edilebilir [52].



Şekil 4.4 Sinüzoidal giriş uygulanan dc motor sistemi blok diyagramı

Bu bilgiler ışığında frekans cevabı kullanılarak robot eklemlerinin modellenmesi için Şekil 4.5'de verilen aşağıdaki test düzeneği hazırlanmıştır.

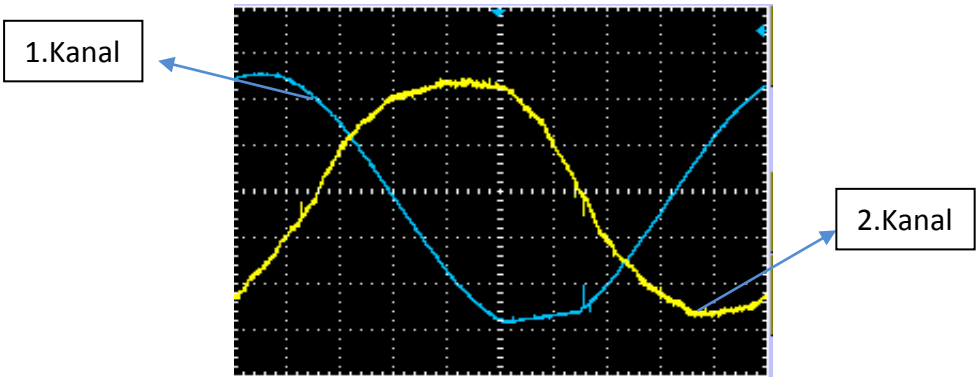


Şekil 4.5 Test düzeneği

Bu test düzeneği yardımıyla ayrı ayrı her eklemdeki motorun girişine uygulanan sinüzoidal işaret ve bu işaret sonucunda çıkışta oluşan genliği ve fazı farklı sinüzoidal işaret osiloskop yardımı ile görüntülenmiş ve sonuçlar ışığında robot eklemlerinin yaklaşık modelleri elde edilmiştir. Modeller oluşturulurken sisteme etkiyen bozucu etkiler ihmal edilmiştir ve sonuçlar bu sistem için yeterli görülmüştür. Oslioskop ile alınan grafiklerin daha iyi yorumlanabilmesi için sinyal jeneratöründen alınan tepeden tepeye 15 Volt sinüzoidal gerilim güç opampı ile ikiye katlanarak tepeden tepeye 30 Voltluk sinüzoidal gerilime dönüştürülmüştür. Motorun kesim frekansına yakın seçilen bu değer motor girişlerine uygulanmış ve osiliskobun birinci kanalına verilmiştir. Buna

karşılık olarak da çıkış olarak, potansiyometrelerin üzerine uygulanan 30 V dc gerilimin ac dalgalanmaları osiloskobun ikinci kanalında gözlemlenmiştir. Potansiyometre üzerine düşen gerilim daha sonra 3.3 V için normalleştirilmiştir. Elde edilen giriş ve çıkış grafiklerinin genlik değerleri ve faz kaymalarına bakarak eksenlerin matematiksel modeli elde edilmiştir. Dc motor sisteminin (ikinci dereceden sistemin) kesim frekansına yakın frekanslarda davranışı bilindiği için bu çalışma civarı seçilmiştir. Robotun dört eklemine, Şekil 4.5'deki test düzeneği uygulanması sonucunda elde edilen grafiklerle oluşturulan modellemeler aşağıda anlatılmıştır.

Birinci eklemin modellenmesi: Şekil 4.5'deki deney düzeneği kuruldktan sonra birinci eklem için osiloskoptan[1] alınan giriş ve çıkış grafikleri Şekil 4.6'de verilmiştir.



Şekil 4.6 Birinci eklem için giriş ve çıkış grafikleri

Osiloskop ayarları: Kanal 1: 5.00 V /Div - Kanal 2: 100,0 mV/Div - Time/Div: 50.0 ms/Div

T_i : giriş ve çıkışın periodu f_i : giriş ve çıkışın frekansı

w_i : açısal frekans E_i : Giriş sinyali genliği

θ_i : Çıkış sinyali genliği i=1, 2, 3, 4

Hesaplamalar;

$$T_1 \simeq (5.1\times 2)\times 50\times 10^{-3} = 0.51s \quad (4.15)$$

$$f_1 = \frac{1}{T_1} = \frac{1}{0.51} = 1.96Hz \quad (4.16)$$

[1] Kullanılan osiliskop UNI-T UTD 2025C modelindedir ve elde edilen grafikleri osiloskoptan gerçek zamanlı olarak flash bellek ile alınmıştır.

$$w_1 = 2\pi f_1 = 2\pi \times 1.96 = 12.31 \quad (4.17)$$

$$\phi_1 = \frac{3.5}{10.2} \times 360 = 123.5^\circ \quad (4.18)$$

$$E_1 = \frac{5.4 \times 5}{2} = 13.5V \quad (4.19)$$

$$\theta_1 = \frac{5 \times 100 \times 3.33}{2 \times 30} = 27.75mV \quad (4.20)$$

$$13.5\angle 0 \longrightarrow \boxed{\frac{A_1}{s(s+B_1)}} \longrightarrow 27.75\times10^{-3}\angle -123.5^\circ$$

$$\frac{13.5\,A}{-151.54 + j12.31B} = -0.0153 - j0.0231$$

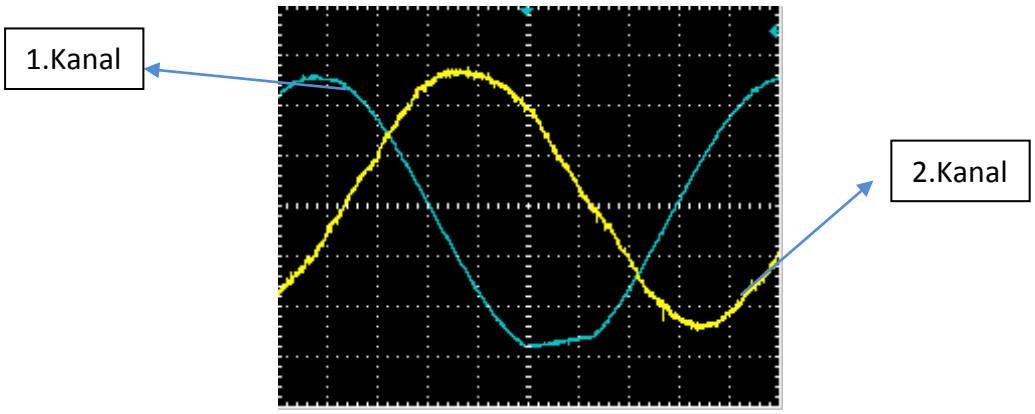
$$13.5\,A = 2.3186 + 0.2844B + j(3.5 - 0.18B)$$

$$B_1 = \frac{3.5}{0.18} = 19.44, \quad A_1 = 0.58 \quad (4.21)$$

Birinci eklemin modeli; $$\frac{\theta_1(s)}{E_1(s)} = \frac{0.58}{s(s+19.44)} \quad (4.22)$$

Kesim frekansı; $$w_{cutoff-1} = 3.09\,rad/sn$$

<u>İkinci eklemin modellenmesi</u>: İkinci eklem için osiloskoptan alınan giriş ve çıkış grafikleri Şekil 4.7'de verilmiştir.



Şekil 4.7 İkinci eklem için giriş ve çıkış grafikleri

Osiloskop ayarları: Kanal 1: 5.00 V /Div - Kanal 2: 100,0 mV/Div - Time/Div: 50.0 ms/Div

Hesaplamalar;

$$T_2 = (4.9 \times 2) \times 50 \times 10^{-3} = 0.49s \tag{4.23}$$

$$f_2 = \frac{1}{T_2} = \frac{1}{0.49} = 2.04\,Hz \tag{4.24}$$

$$w_2 = 2\pi f_2 = 2\pi \times 2.04 = 12.81\,rad/sn \tag{4.25}$$

$$\phi_2 = \frac{3.2}{9.8} \times 360 = 117.55^\circ \tag{4.26}$$

$$E_2 = \frac{5.4 \times 5}{2} = 13.5V \tag{4.27}$$

$$\theta_2 = \frac{5.1 \times 100 \times 3.3}{2 \times 30} = 28.05mV \tag{4.28}$$

$$13.5\angle 0 \longrightarrow \boxed{\frac{A_2}{s(s+B_2)}} \longrightarrow 28.05 \times 10^{-3} \angle -117.5^\circ$$

$$\frac{13.5\,\mathrm{A}}{-164.09 + j12.81B} = -0.0130 - j0.0249$$

$$13.5\,\mathrm{A} = 2.1232 + 0.319B + j(4.0875 - 0.1665\,\mathrm{B})$$

$$B_2 = \frac{4.0875}{0.1665} = 24.55, \quad \mathrm{A}_2 = 0.7374 \tag{4.29}$$

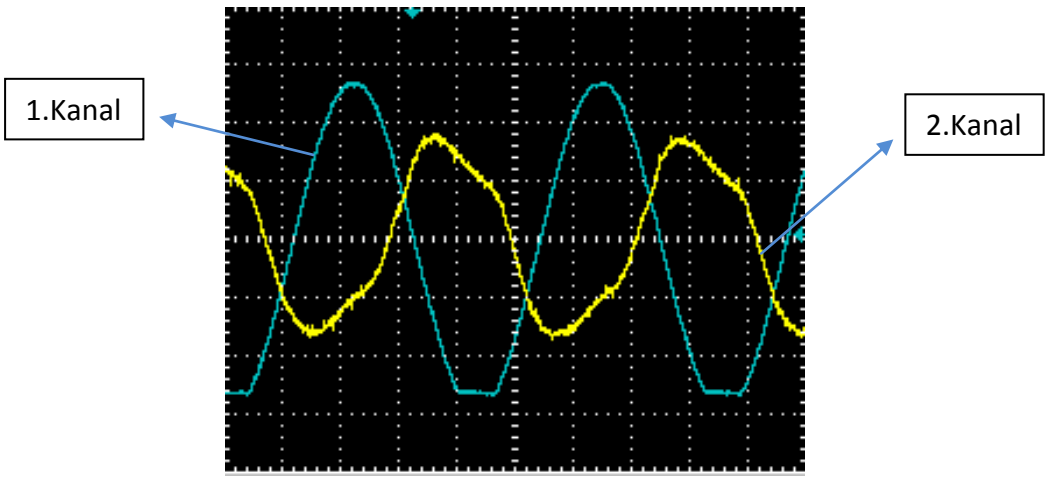
İkinci eklemin modeli; $$\frac{\theta_2(\mathrm{s})}{E_2(\mathrm{s})} = \frac{0.7374}{s(\mathrm{s}+24.55)} \tag{4.30}$$

Kesim frekansı; $$w_{ctoff-2} = 3.90\,rad/sn \tag{4.31}$$

<u>Üçüncü eklemin modellenmesi</u>: Üçüncü eklem için osiloskoptan alınan giriş ve çıkış grafikleri Şekil 4.8'de verilmiştir.

Osiloskop ayarları;

Kanal 1: 5.00 V /Div - Kanal 2: 50,0 mV/Div - Time/Div: 50.0 ms/Div



Şekil 4.8 Üçüncü eklem için giriş ve çıkış grafikleri

Hesaplamalar;

$$T_3 = 4.2 \times 50 \times 10^{-3} = 0.21s \tag{4.32}$$

$$f_3 = \frac{1}{T_3} = \frac{1}{0.21} = 4.76\,Hz \tag{4.33}$$

$$w_3 = 2\pi f_3 = 2\pi \times 4.76 = 29.92\,rad/sn \tag{4.34}$$

$$\phi_3 = \frac{1.6}{4.2} \times 360 = 137.1^\circ \tag{4.35}$$

$$E_3 = \frac{5.4 \times 5}{2} = 13.5V \tag{4.36}$$

$$\theta_3 = \frac{3.3 \times 50 \times 3.3}{2 \times 30} = 9.075mV \tag{4.37}$$

$$13.5\angle 0 \longrightarrow \boxed{\frac{A_3}{s(s+B_3)}} \longrightarrow 9.075 \times 10^{-3} \angle -137.1^\circ$$

$$\frac{13.5\,\mathrm{A}}{-895.2 + j29.92B} = -0.0066 - j0.0062$$

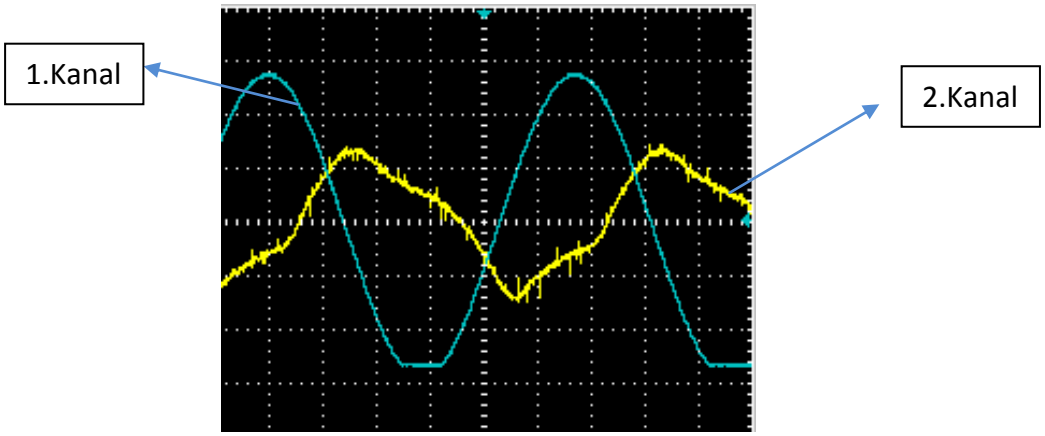
$$13.5\,\mathrm{A} = 5.9 + 0.1855B + j(5.55 - 0.1975B)$$

$$B_3 = \frac{5.55}{0.1975} = 28.10\,, \quad A_3 = 0.8232 \tag{4.38}$$

Üçüncü eklemin modeli; $$\frac{\theta_3(s)}{E_3(s)} = \frac{0.8232}{s(s+28.10)} \quad (4.39)$$

Kesim frekansı; $$w_{cutoff-3} = 4.47\, rad/sn \quad (4.40)$$

<u>Dördüncü eklemin modellenmesi</u>: Dördüncü eklem için osiloskoptan alınan giriş ve çıkış grafikleri Şekil 4.9'de verilmiştir..



Şekil 4.9 Dördüncü eklem için giriş ve çıkış grafikleri

Osiloskop ayarları; Kanal 1: 5.00 V /Div - Kanal 2: 50,0 mV/Div - Time/Div: 50.0 ms/Div

Hesaplamalar;

$$T_4 = 5.7 \times 50 \times 10^{-3} = 0.2850s \quad (4.41)$$

$$f_4 = \frac{1}{T_4} = \frac{1}{0.2850} = 3.51Hz \quad (4.42)$$

$$w_4 = 2\pi f_4 = 2\pi \times 3.51 = 22.04 \quad (4.43)$$

$$\phi_4 = \frac{2}{5.7} \times 360 = 126.4^\circ \quad (4.44)$$

$$E_4 = \frac{5.4 \times 5}{2} = 13.5V \quad (4.45)$$

$$\theta_4 = \frac{2.9 \times 50 \times 3.3}{2 \times 30} = 7.98V \quad (4.46)$$

$$13.5\angle 0 \longrightarrow \boxed{\frac{A_4}{s(s+B_4)}} \longrightarrow 7.98 \times 10^{-3} \angle -126.32^\circ$$

$$\frac{13.5\,\mathrm{A}}{-485.76+j22.04B} = -0.0047 - j0.0064$$

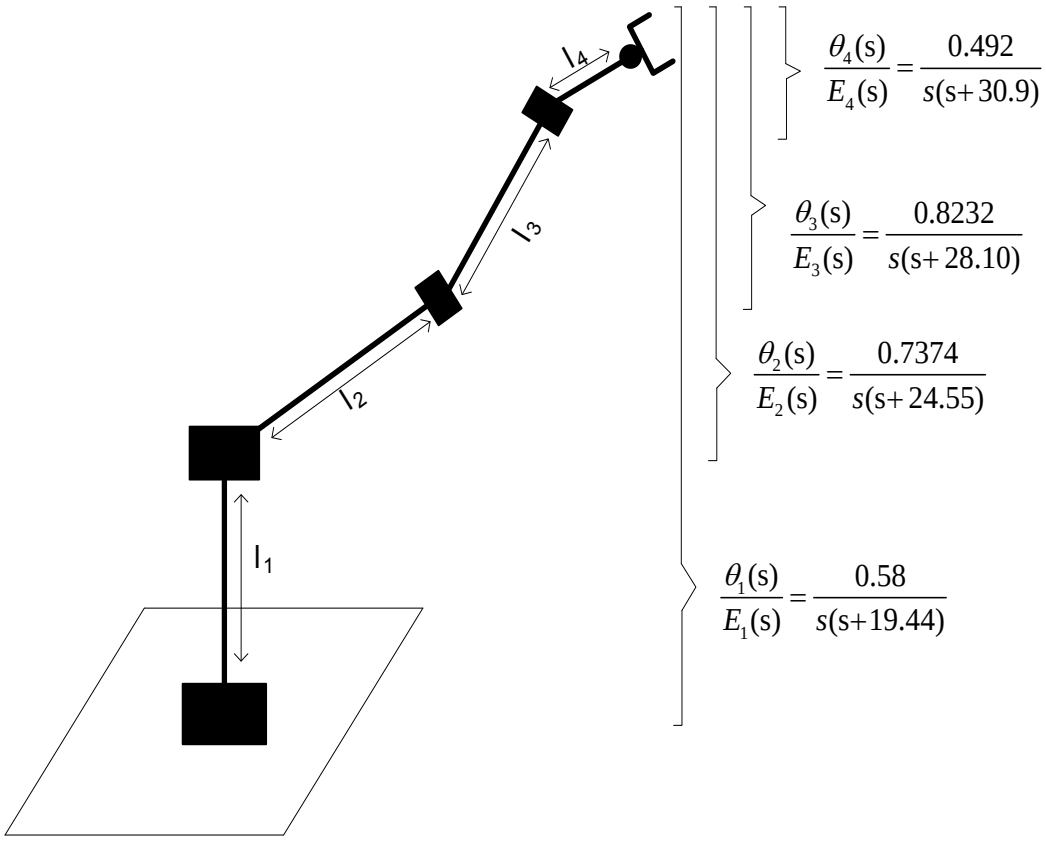
$$13.5\,\mathrm{A} = 2.28 + 0.141B + j(3.09 - 0.1B)$$

$$B_4 = \frac{0.1}{3.06} = 30.9\,, \quad \mathrm{A}_4 = 0.492 \tag{4.47}$$

Dördüncü eklemin modeli; $$\frac{\theta_4(\mathrm{s})}{E_4(\mathrm{s})} = \frac{0.492}{s(\mathrm{s}+30.9)} \tag{4.48}$$

Kesim frekansı; $$w_{ctoff-4} = 4.91 rad/sn \tag{4.49}$$

Sonuç olarak elde edilen doğrusal modeller robot kol üzerinde Şekil 4.10'daki gibi gösterilir.



Şekil 4.10 Robot eklemleri matematiksel modelleri

4.2 DE0 Nano Altera FPGA Kartı

Proje kapsamında uygulamaları gerçeklemek için Altera firmasını üretmiş olduğu DE0 Nano FPGA geliştirme ve eğitim kartı kullanılmıştır. Bu kart robot ve mobil projeler olmak üzere birçok projede kullanılabilecek özelliklere sahip sıkıştırılmış boyutta FPGA geliştirme platformudur.

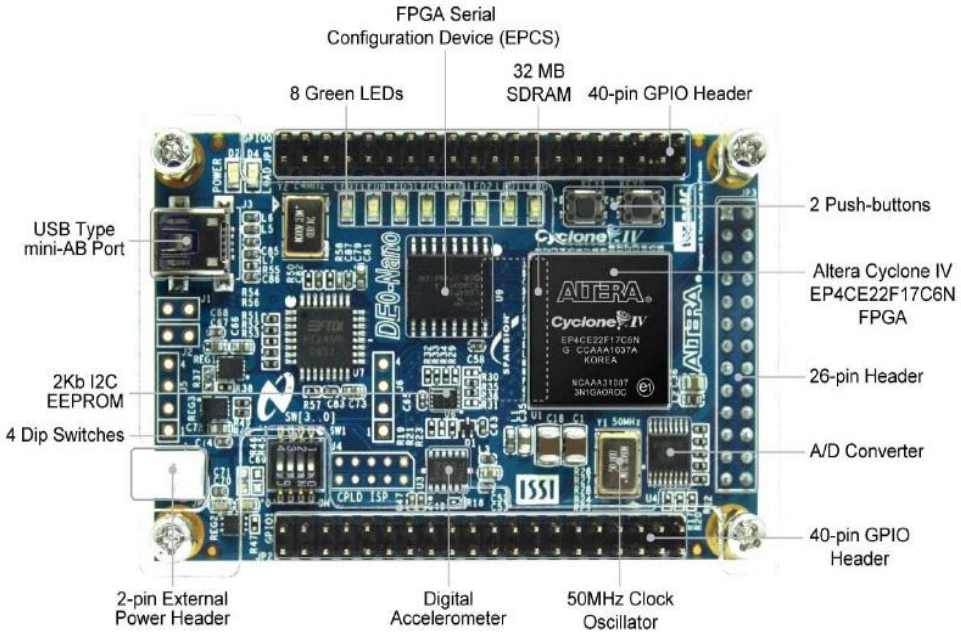
4.2.1 Genel Özellikleri

- FPGA cihazı
    - Altera Cyclone IV EP4CE22F17C6N FPGA
    - Maximum 153 giriş çıkış birimi (I/O pins)
- Yapılandırma birimi ve kurulum elemanları
    - Programlama için kart üzerinde USB-Blaster devresi
    - Spansion EPCS64
- Harici giriş/çıkış birimleri
    - İki adet 40-pin genel amaçlı bağlantı birimi,
    - 72 adet giriş çıkış pini
    - 2 adet 5V besleme pini
    - 2 adet 3.3 V besleme pini
    - 4 adet toprak pini
- Hafıza birimleri
    - 32 MB SDRAM
    - 12 KB I2C EEPROM
- Genel kullanıcı giriş çıkışları
    - 8 adet yeşil led
    - 2 adet buton
    - 4 adet DIPswitch
- İvme Ölçer (G Sensor) Kart üzerinde yüksek çözünürlüklü
    - ADI ADXL345, 3 eksen yüksek çözünürlüklü ivmeölçer (13 bit)

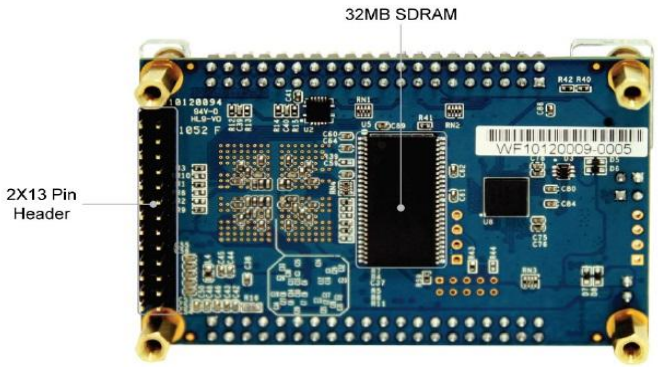
- Analog/Dijital Dönüştürücü
    - NS ADC128S022, 12 bit, 8 kanal analog-dijital dönüştürücü (50Ksps-200Kspa)
- Saat sistemi
    - Dâhili 50 MHz clock osilatör
- Güç Kaynağı
    - USB tipi mini AB portu (5 V)
    - İki pin harici besleme girişi (3.6V - 5.7 V)
    - Genel amaçlı I/O bağlantısı için DC 5 Volt pin

4.2.2 Komponentlerin Yerleşimi ve Blok Diyagramı

Aşağıdaki şekillerde DE0 Nano FPGA geliştirme kitinin üzerindeki bileşenlerin yerleşimi gösterilmiştir. Kart sıkıştırılmış boyutta olduğu için komponentler kartın üst ve alt bölümlerine dağıtılmıştır.

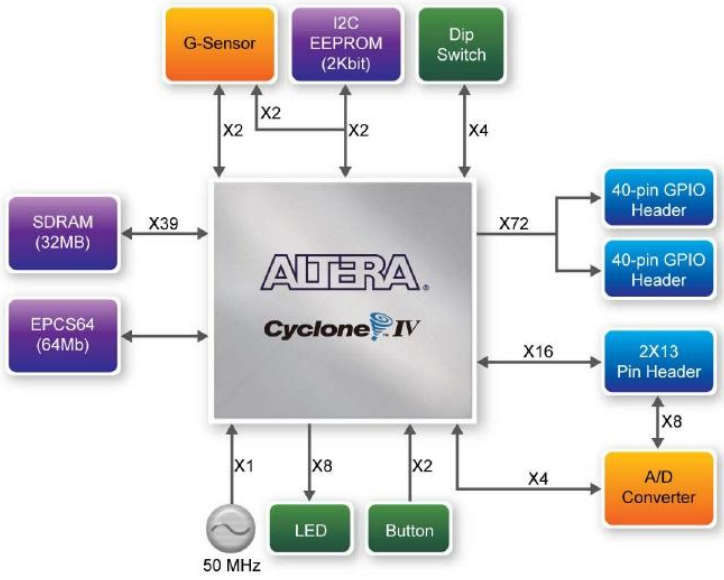


Şekil 4.11 DE0 Nano FPGA kitinin üsten görünümü



Şekil 4.12 DE0 Nano FPGA kitinin alttan görünümü

Aşağıda FPGA geliştirme kiti üzerinde bulunan tüm bileşenlerin FPGA cihazı ile bağlantısını gösteren genel blok diyagramı yer almaktadır. Şekilden anlaşılacağı gibi bileşenlerin FPGA'a bağlantısı gösterilirken kullanılan oklar veri akış yönlerini göstermektedir. Oklar üzerindeki yazan sayı da komponentin FPGA'e kaç pin ile bağlı olduğunu gösterir.



Şekil 4.13 FPGA DE0 Nano kart veri akış diyagramı

4.2.3 Cyclone IV FPGA'in yapılandırılması

Cyclone IV FPGA içeren DE0 Nano JTAG programlama kullanılarak programlanabilir. JTAG (Joint Test Action Group) ara yüzü 1980 yılında geliştirilmiş bir IEEE standardıdır. Günümüzde entegre devrelerde hata ayıklamak için kullanılan en yaygın yöntemdir. JTAG ile FPGA programlama kullanıcıya Quartus II programı ile yazdığı kendine özgü

programa göre FPGA'i konfigüre etme imkânı sunmaktadır. FPGA'in bütün giriş/çıkış pinleri JTAG ara yüzü kontrol edilebilmektedir. Böylelikle JTAG'in kendine özgü komutları kullanılarak FPGA programlanabilmektedir.

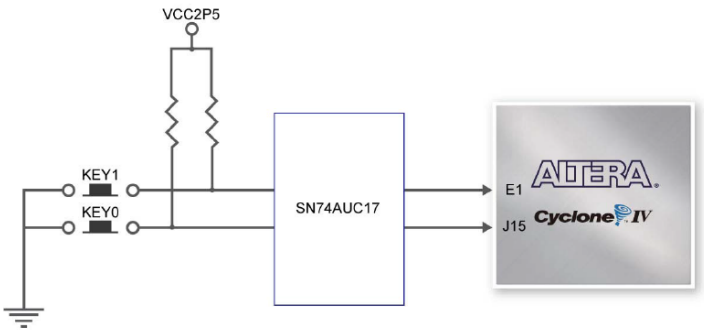
JTAG pinleri genelde ayrılmış özel pinlerdir ve yalnızca bu amaç için kullanılır. Test edilecek ya da program yüklenecek devreyi bilgisayara bağlamak için JTAG kablo kullanılır. Bu kablo ethernet, USB ya da paralel ara yüzlerden birine sahip olabilir.

FPGA'e yüklenilen yapılandırma verisi bellekte (SRAM) depolandığı için ve bellek uçucu özelliğe sahip olduğundan, FPGA enerji aldığı sürece, ya da yeni bir program yüklenilene kadar FPGA içinde kalır, fakat enerji kesildiğinde FPGA konfigürasyon bilgileri kaybolduğu için program silinir. Konfigürasyon dosyası Quartus II programı ile oluşturulmış <.sof> uzantılı bir dosyadır ve bu dosya Ouartus II Programlayıcı arayüzü kullnılarak JTAG programlama ile Cyclone 4 FPGA içine yüklenir.

FPGA'e yüklenilen yapılandırma verisinin kalıcı olması için kart üzerinde bulunan Spansion EPCS64 seri yapılandırma elemanı kullanılabilir. Bu yapılandırma elemanı üzerinde uçucu olmayan belleğe sahiptir. Konfigürasyon verisi bu belleğe kaydedilir ve FPGA'in gücü kesilse bile veri kaybolmaz. FPGA tekrar enerjilendirildiği zaman yapılandırma verisi otomatik olarak FPGA içine yüklenir.

4.2.4 Genel Kullnıcı Giriş Çıkış Pinleri

Butonlar: DE0-Nano FPGA geliştirme kartının üzerinde uygulama geliştirmek için iki adet buton bulunmaktadır. Bu butonların FPGA'a bağlantısı aşağıdaki şekilde gösterilmiştir;



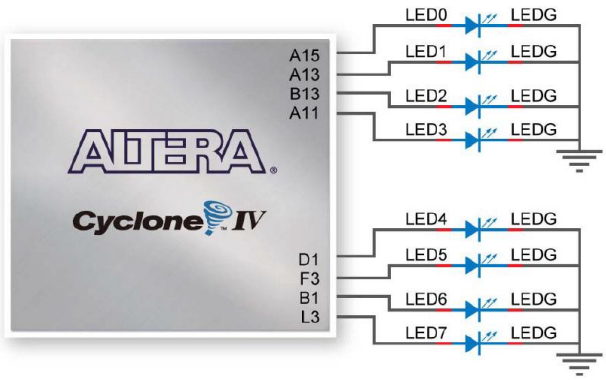
Şekil 4.14 FPGA ve butonlar arasındaki bağlantı şeması

Yukarıdaki şekildende anlaşılacağı gibi butonlar normalde açık pozisyondalardır ve butonlara basılmadığı durumda, her butonun FPGA girişi yüksek seviye lojik (lojik 1) belirtirken, butonlara basıldığı zaman düşük seviye lojik (lojik 0), belirtiler. Bu sayede bu butonlar clock ya da saat girişi için kullnılabilirler.

Butonlara basıldığı zaman meydana gelebilecek zıplama (debounce) problemlerini engellemek için butonlar ve FPGA cihazı arasında SN74AUC17 Schmitt-Trigger devresi kullanılmıştır ve devrenin çıkışları direk olarak FPGA pinlerine bağlanmıştır.

Dip Switch'ler: DE0 Nan0 kart üzerinde 4 adet dip switch yer amaktadır. Bu dipswitchler anahtag görevi üslenerek istenilen durumlarda sisteme yüksek seviye ya da düşük seviye lojik sağlamaktadır. Dip switch'lerin FPGA cihazına doğrudan bağlanmaktadır.

Ledler: DE0-Nano kart üzerinde uygulama geliştirmek için kullanıcı kontrollü 8 adet led bulunmaktadır. Her led direkt olarak Cyclone 4 FPGA'in bir pinine bağlıdır ve bu pinler FPGA tarafından ile sürülür. İlgili pin yüksek logic seviyesinde sürüldüğü zaman led yanar ve aynı şekilde düşük seviye lojik ile sürüldüğü zaman led söner. Aşağıdaki şekilde ledlerin FPGA cihazına bağlantı şekli gösterilmiştir.

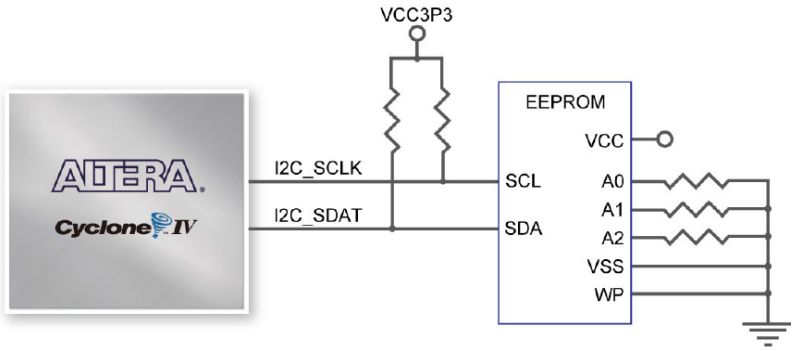


Şekil 4.15 FPGA ve ledler arasındaki bağlantı

Hafıza Birimi: DE0 nano kartın üzerinde FPGA'a direk olarak bağlı 16 bit veri hattı ve 32MB hafızaya sahip SDRAM (Synchronous Dynamic Random Access Memory) bulunmaktadır. Bu entegre 3.3V LVCMOS sinyal standartını kullanmaktadır. Kaydetme işlemi ana saat sinyalinin ve DRAM saat sinyalinin pozitif kenarında yapılır. SDRAM ve FPGA cihazın birbirine bağlantı şekli aşağıda gösterilmiştir.

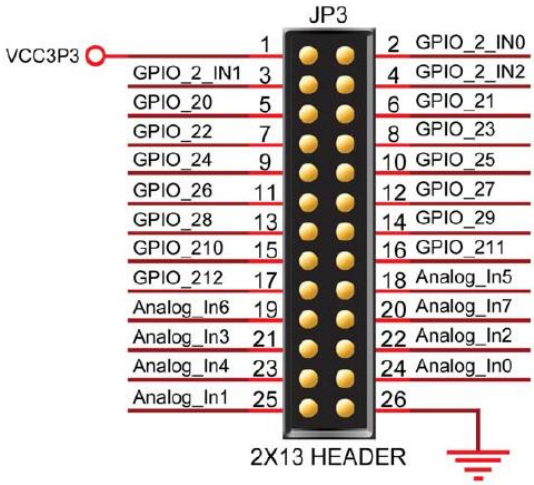
Şekil 4.16 FPGA ve SDRAM arasındaki bağlantı

I2C Serial EEPROM: DE2 Nano FPGA geliştirme kartı 2KB EEPROM'a (Electrically Erasable PROM) sahiptir. EEPROM küçük boyuttaki verileri kalıcı olarak saklamak için kullanılan bir yongadır. I2C ise EEPROM'a veri gönderek için kulanılan bir haberleşme protokolüdür. EEPROM ve FPGA arasında iletişimi sağlayan iki tane bağlantı bulunur. Bu bağlantılar saat ve veri bağlantılarıdır. Aşağıda EEPROM ve FPGA'in bağlantı şekli gösterilmiştir.



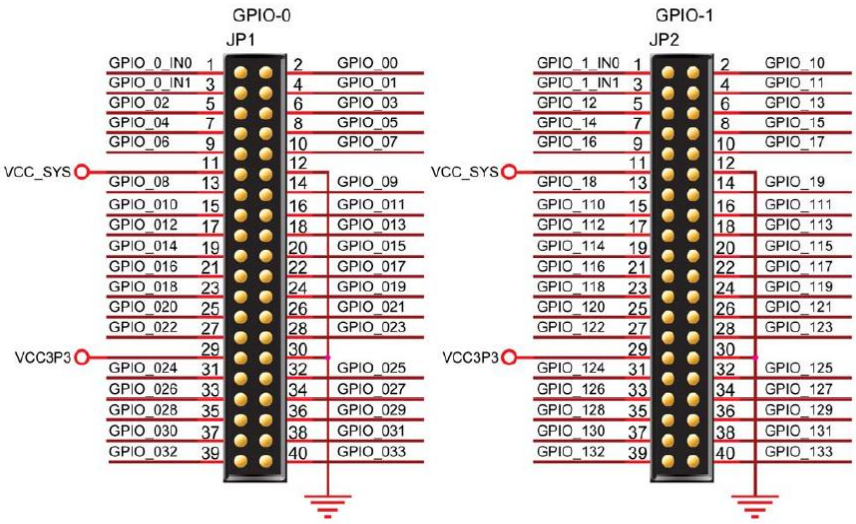
Şekil 4.17 FPGA ve EEPROM arasındaki bağlantı

Analog / Digital Dönüştürücü ve 2x13 Bağlantı: Analog-dijital dönüştürücüler, analog cihazları dijital sistemlere bağlamak için kullanılan ve sürekli değerli analog sinyalleri, ayrık değerli dijital sinyallere dönüştüren cihazlardır. DE0 Nano kart üzerinde ADC128S022 A/D dönüştürücü çipi bulunmaktadır. Bu chip 8 kanal analog girişi 12 bit dijital sinyale dönüştürür. ADC'nin sekiz giriş kanalı FPGA kartı üzerinde bulunan 3x13 harici bağlantı girişlerine aşağıdaki şekilde gösterildiği gibi IN0'dan IN7'ye kadar yapılandırılmıştır. Harici bağlantı diğer 16 pini doğrudan FPGA'a bağlanırken bir tane toprak ve bir tane de DC 3.3V (VCC33) pini bulunmaktadır.



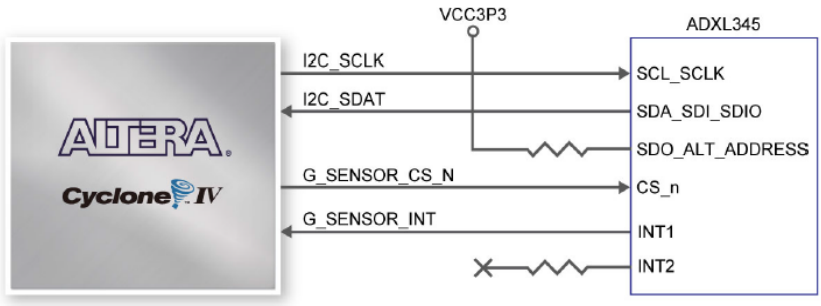
Şekil 4.18 A/D dönüştürücü giriş pinleri ve genel amaçlı giriş/çıkış pinleri

Harici Giriş/Çıkış Pinleri: DE0 Nano kart iki tane 40 pin harici giriş/çıkış pinleri içerir. Her iki 40 pinin de 36 pini direk olarak FPGA'e bağlıdır. Birer pinleri DC+5V (VCC5) ve DC+3.3V (VCC33) ve ikişer pinleri de toprak bağlantısıdır. Bu genel amaçlı giriş/çıkış pinlerinin bağlantı şemaları aşağıda gösterilmiştir.



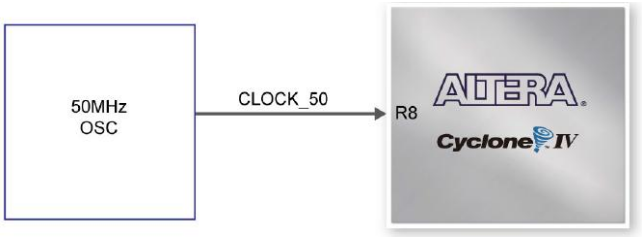
Şekil 4.19 Genel amaçlı giriş/çıkış pinleri

Dijital İvme Sensörü: Kart üzerinde üç eksende de çok yüksek çözünürlükte ölçüm yapabilen, küçük ince ve çok düşük güçle çalışabilen ADXL345 ivme sensörü yer almaktadır. Bu dijital sensöre SPI ya da I2C iletişim protokolleri kullanılarak ulaşılabilir. Aşağıdaki dijital ivme sensörününü FPGA'e bağlantısı gösterilmiştir.



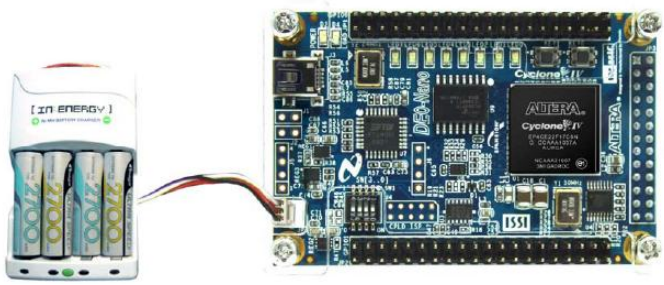
Şekil 4.20 FPGA ve dijital ivme sensörü arasındaki bağlantı şeması

Clock Devresi: DE0 Nano kart üzerinde ayrıca 50MHz sinyal üreten osilatör vardır. Bu osilatör FPGA'in clock için ayrılmış özel pinine bağlanmıştır. PLL devrelerinin sürülmesi için kaynak saat sinyali olarak kullnılabilir. FPGA'a bağlantı şekli aşağıda verilmiştir.



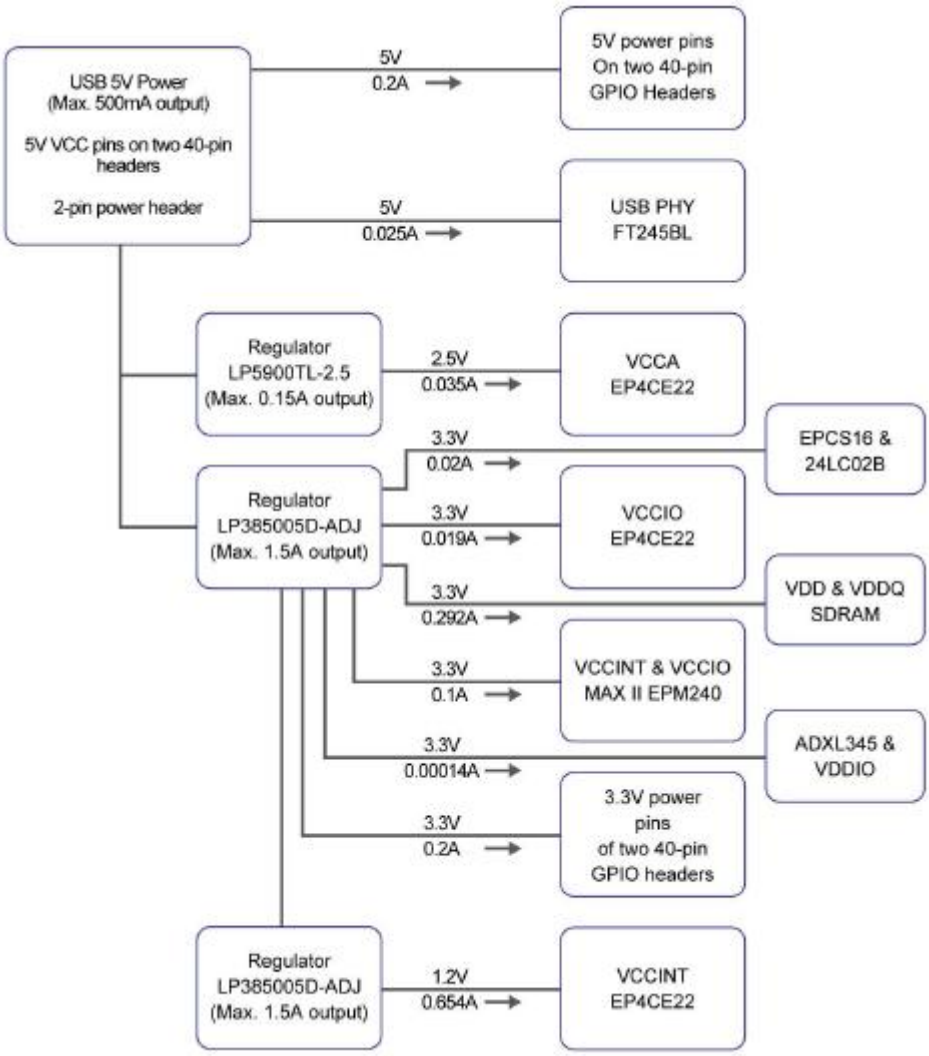
Şekil 4.21 FPGA ve saat üreteci arasındaki bağlantı şeması

Güç Kaynağı: DE0 Nano Kart USB üzerinden 5V ile beslenir. Ayrıca kart üzerindeki 2 adet 40 pin headerler üzerinde de 5V VCC ve ayrıca taşınabilir uygulamalar için harici 5V pinleri bulunmaktadır. Aşağıdaki şekilde taşınabilir uygulamalar için FPGA'e pil bağlantısı gösterilmiştir.



Şekil 4.22 FPGA ve harici güç bağlantısı

Ayrıca aşağıdaki şemada kart üzerindeki komponentlere güç dağılımının nasıl olduğu gösterilmiştir.



Şekil 4.23 DE0 Nano güç dağılım şeması

4.3 Motor Sürücü Devreleri

Robot eklemlerinde bulunan dc motorları sürmek için uygun sürücüler araştırılmış ve piyasadan, maximum 2A akım çekebilen ve üzerinde LN298 motor sürücü entegresi olan Şekil 4.24'deki motor sürücü devresi temin edilmiştir. Bu sürücü devresi aynı anda iki dc motoru sürebilecek özelliğe sahiptir ve robot kol üzerinde 5 adet motor bulunduğu için 3 adet sürücü devresi yeterlidir. Sürücü devre üzerinde harici 5V regülatör ve ledli besleme göstergesi bulunmaktadır. Kontrol işlemi sonucunda oluşan duty-cycle değeri FPGA'den motor sürücü devrelere gelir ve motorlar üzerinde verilecek gerilim değeri hesaplanarak gönderilir.

Şekil 4.24 Motor sürücü devresi

Çizelge 4.1‘ te motor sürücü devresinin özellikleri listelenmiştir;

Çizelge 4.1 Motor sürücü devresi özellikleri

Ağırlık:	35 gr
Boyut:	43 x 43 x 27
Güç:	25 W
Besleme:	5 – 35 V
Max. Akım:	2 A
Motor sayısı:	2

4.4 Quanser Q2 veri toplama modülü

Sistem üzerinden gerçek zamanlı veri toplamak için Quanser Q2 modülleri kullanılmışır. Bu modüler ile robot motorlarında bulunan potansiyometrelerder alınan konum bilgieri USB ile Matlab platformuna aktarılarak referans konum bilgisi ile karşılaştırılmıştır. Şekil 4.25’te uygulamada kullanılan Quanser Q2 veri toplama modülü gösterilmiştir.

Şekil 4.25 Quanser Q2 veri toplama modülü

BÖLÜM 5

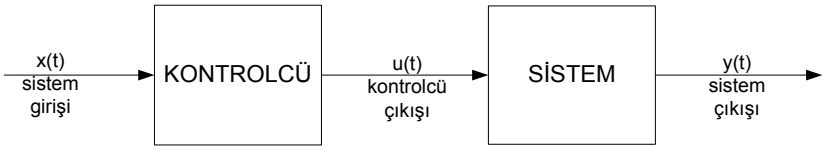
KONTROL YÖNTEMİ

Kontrol, “Sistemdeki değişkenlerin istenilen amaca uygun bir şekilde davranmasını sağlayabilmek ve sisteme uygun bir giriş uygulanarak arzu edilen çıkışı elde etmektir.” şeklinde tanımlanabilir. Robotik kontrol sistemlerinin amacı da robotun uç işlevcisinin verilen yörüngeye en uygun şekilde izlemesini veya istenilen noktaya gitmesini sağlamaktır. Bu işlem için öncelikle robotik sistemin ve uygun kontrolcünün matematiksel modellemesi yapılmalıdır. Bir sistemin matematiksel modelini çıkarmak için sistemin iyice tanınması gerekmektedir. Bölüm 4.1’de robot eklemlerinin belirli kısıtlamalar çerçevesinde modellemeleri yapılmıştır.

Robot kontrol sistemlerinin yapısı, yerine getirilmesi istenilen görevin karmaşıklığına göre değişir. Basit robotlarda, hareketi kontrol etmek için, pnomatik, mekanik veya basit elektriksel mantık kontrolcüleri kullanılırken, karmaşık endüstriyel robotlarda her eklemin konumu veya hareket ekseni kapalı çevrim servo sistemleri ile kontrol edilir. Bu kontrolcüler, eklem hareketlerini kontrol etmek için algılayıcılardan sürekli konum bilgisi alırlar. İncelenen sistemdeki robot kolunun kontrolü de bu mantıkla her bir kolun birbirinden bağımsız kapalı çevrim sistemler olduğu yaklaşımına dayanmaktadır. Sistemin kontrolü için, günümüzdeki endüstriyel sistemlerde en çok kullanılan PID kontrolcü kullanılmıştır. PID kontrolcü geri beslemeli yani kapalı çevrim bir kontrolcüdür. Az sayıda tasarım parametresi vardır ve bunlar performans kriterleri ile kolayca ilişkilendirilebilmektedir. [54][55]. FPGA dijital tabanlı çalışan bir bütünleşik devre olduğu için sistem kontrolü “ayrık PID” yaklaşımı ile gerçekleştirilmiştir.

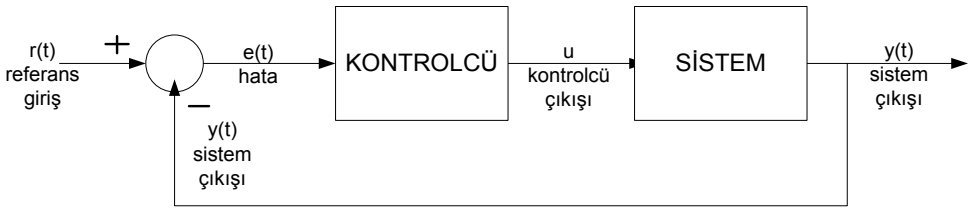
Bu bölümün devamında mühendislik kontrol sistemleri hakkında genel bir bilgi verildikten sonra PID kontrolcü ve ayrık PID kontrolcü yapısı anlatılmıştır. PID katsayılanısın hangi kriterlere göre elde edildiği belirtildikten sonra, VHDL dili ile analog veri okuma algoritması, PWM algoritması ve PID algoritması detaylı olarak anlatılmış ve sisteme uygulanan kontrolcü sonucunda elde edilen sonuçlara Bölüm 6'da yer verilmiştir. Bu çalışmada PID kontrolcü yapısının FPGA tabanlı gerçeklenmesinin amacı çoklu bağımsız (ya da uygulamaya göre bağımlı) kapalı çevrim mekanizmanın tek bir merkezden denetiminin sağlanabildiğini göstermektir. Çünkü bu tür sistemleri kontrol etmek için genellikle her eyleyiciyi kontrol eden çevresel kontrolcüler ve bunları kontrol eden merkezi kontrolcü kullanılır. FPGA kullanılarak sistem karmaşıklığını azalttığı gibi FPGA'in kendine özgü özelliklerinden de faydalanılmasına olanak sağlamaktadır.

Mühendislik kontrol sistemleri açık çevrim kontrol ve kapalı çevrim kontrol olmak üzere iki gruba ayrılır. Açık çevrim kontrol sistemlerinde kontrolcü çıkışı sistemin çıkışlarına bağlı olarak belirlenmez. Bu tür kontrol sistemleri genellikle sistemin yapısının ve sistem girişlerinin önceden çok iyi bilindiği uygulamalarda kullanılır. Açık çevrim kontrol yapısının blok diyagramı aşağıda gösterilmiştir.



Şekil 5.1 Açık çevrim kontrol sistemi blok şeması

Kapalı çevrim kontrol sisteminde ise, istenilen referans girişten sistem çıkışı çıkartılarak elde edilen hata sinyali kontrolcünün girişine uygulanır. Böylece kontrolcü çıkışı sistemin çıkışına bağlı olarak değiştirilerek çıkışın istenilen girişi takip etmesi sağlanır. Kapalı çevrim kontrol sisteminin blok diyagramı aşağıda gösterilmiştir.



Şekil 5.2 Kapalı çevrim kontrol sistemi blok şeması

5.1 PID Kontrol Yapısı

PID kontrolcü yapısı sürekli zamanlı sistemlerin kontrolünde çok geniş uygulamalarda kullanılan bir kapalı çevrim kontrol yapısıdır. PID kontrolcü lineer bir kontrolcüdür ve hata sinyali e(t)'yi kontrol sinyali u(t)'ye çevirir. Lineer bir kontrolcü olduğu için zaman ve frekans bölgelerinde ayrı ayrı incelenebilir. Denlem 5.1 ve 5.2'de PID kontrolcünün zaman bölgesinde kullanılan denklemleri verilmiştir.

$$u(\mathrm{t}) = K\left(e(\mathrm{t}) + \frac{1}{T_i}\int_0^t e(\mathrm{t})\,\mathrm{dt} + T_d\,\frac{d}{dt}e(\mathrm{t}) \right) \tag{5.1}$$

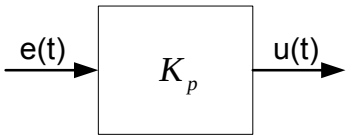
$$u(\mathrm{t}) = Ke(\mathrm{t}) + \frac{K}{T_i}\int_0^t e(\mathrm{t})\,\mathrm{dt} + KT_d\,\frac{d}{dt}e(\mathrm{t}) \tag{5.2}$$

$\mathrm{K_P} = \mathrm{K}\ (\text{oransal katsayı})$

$\mathrm{K_I} = \dfrac{\mathrm{K}}{\mathrm{T_i}}\ (\text{integral katsayısı})$

$\mathrm{K_D} = \mathrm{KT_d}\ (\text{türevsel katsayı})$

Denklemden de anlaşılacağı gibi PID kontrolcünün oransal, integral ve türevsel olmak üzere üç bileşeni bulunmaktadır. Oransal etki, kontrolcü çıkışına hatanın o andaki değerinin belli bir kazanç değeri ile çarpımı kadar etki gösterir. Bu kazanç değerine yani K_p'ye orantı kazancı ya da orantı sayısı denir. Kazanç büyüdükçe kontrolcü reaksiyonu artar. Oransal etki hatanın şu andaki değeri ile ilgilenir.



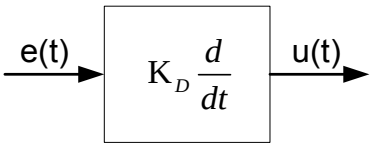
$$u(\mathrm{t}) = \mathrm{K}_p\, e(\mathrm{t})$$

Şekil 5.3 Oransal kontrolcü

Oransal kontrol hata azaldıkça daha düşük bir kontrol etkisi gösterir ve çıkışın referansa yumuşak bir şekilde yaklaşmasını sağlar. Böylelikle on/off kontrolde oluşan salınım etkisi azaltılmış olur. Fakat hata azaldıkça aynı zamanda oransal kontrolün kontrol etkisi çok azalır ve belirli bir değerin altında kontrol sinyalinin bağlandığı tahrik sisteminin sisteme etkisi çok azalır ve sistem çıkışı referansa asla tam olarak ulaşamaz. Bu nedenle kalıcı hal hatası oluşabilir. Bu hatanın oluşmasını engellemek için belirli giriş

değerlerine karşılık kontrolcü çıkışına sabit bir değer eklenebilir. Oransal etki sistemin yükselme zamanını azaltır ve aşımı artırır.

Türevsel kontrol etkisi sistemdeki hatanın değişimi ile orantılı olarak belirlenir. Türev işlevi sistem çıktısının hesaplandığı andan bir sonraki anda alacağı değere ilişkin bir veri üretir. Dolayısıyla türev etkisi sistemin gelecek hatası ile ilgili bir öngörü yapar denilebilir.

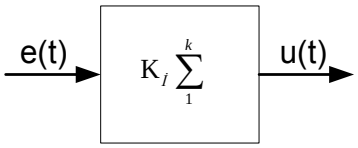


$$u(t) = K_D \frac{de(t)}{dt}$$

Şekil 5.4 Türevsel kontrolcü

Türev etkisi, kontrol sisteminin hızlı çalışmasını sağlar. Fakat hata sabit iken türevsel sisteme herhangi bir etki uygulamaz, bu nedenle türev sisteme tek başına uygulanmaz. Türevsel kontrol geçici durumda meydana gelen salınımları azaltır.

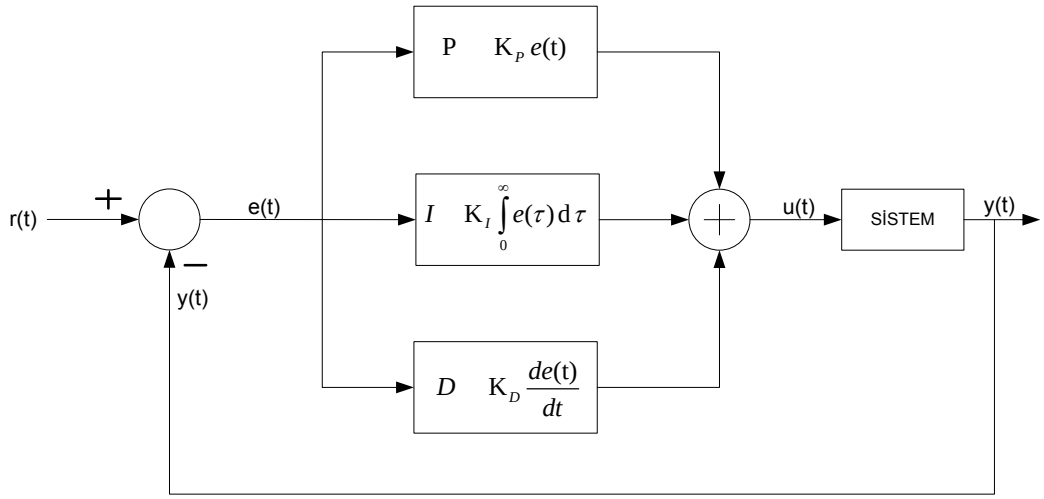
İntegral kontrol etkisi sistemin çalışmaya başladığı andan itibaren tüm anlardaki hatalar toplamının bir kazanç değeri ile çarpılması ile hesaplanır. Bu yaklaşımdan integral kontrolcünün sistemin geçmiş hataları ile ilgilendiği sonucuna varılabilir. Sistemin cevabının referans değere ulaması geciktikçe, integral kontrolcünün sistem üzerindeki etkisi de artar.



$$u(t) = K_I \sum_1^k e(t)$$

Şekil 5.5 İntegral kontrolcü

PID kontrolcü ile oransal, türevsel ve integral etkiler sisteme ayrı ayrı uygulanmış olur. Şekil 5.6'da paralel PID kontrolcünün blok diyagramı verilmiş ve Çizelge 5.1'de K_p, K_I ve K_D nin kapalı cevrim sistem yapısına etkileri listelenmiştir. [60].



Şekil 5.6 PID kontrolcü

Çizelge 5.1 K_p, K_I ve K_D'nin kapalı çevrim sisteme etkisi

Kapalı Çevrim Cevabı	**Yükselme Zamanı**	**Aşım**	**Yerleşme Zamanı**	**Kararlı Hal Hatası**
K_p	Azalır	Artar	Küçük oranda artar	Azalır
K_I	Küçün oranda azalır	Artar	Artar	Büyük oranda azalır
K_D	Küçük değişim	Azalır	Azalır	Küçük Değişim

5.2 Ayrık PID Yapısı

Denklem 5.1'de verilen analog PID kontrolcü denkleminin FPGA ile gerçeklenebilmesi için denklemin ayrık zaman uzayında yeniden düzenlenmesi gerekir. Bu düzenleme işleminde bir takım dönüşümler ve değişimler kullanılır. Şekil 5.6 'da verilen PID kontrolcü yapısı, paralel PID kontrolcüdür ve oransal, integral ve türevsel parçaların her biri ayrı ayrı ayrık zaman uzayına çevrilebilmektedir. Dönüşüm işlemi yapılırken z-dönüşümünün zamanda kayma (time shifting) özelliğinden faydalanılmıştır [56].

Z-dönüşümü zamanda kayma özelliği (time-shifting teorem);

$$\begin{aligned} Z\{x(\mathrm{k})\} &= X(z) \\ Z\{x(\mathrm{k}-n)\} &= z^{-n}X(z) \\ Z\{x(\mathrm{k}+n)\} &= z^{n}X(z) \end{aligned} \qquad (5.3)$$

Paralel PID yapısı aşağıdaki gibi üç terimin toplamı şeklinde gösterilebilir. Burada $\mathrm{u}_p(\mathrm{t})$ oransal kontrolden, $\mathrm{u}_d(\mathrm{t})$ türevsel kontrol ve $\mathrm{u}_i(\mathrm{t})$ integral kontrolde den gelir.

$$u(\mathrm{t}) = \mathrm{u}_p(\mathrm{t}) + \mathrm{u}_d(\mathrm{t}) + \mathrm{u}_i(\mathrm{t}) \tag{5.4}$$

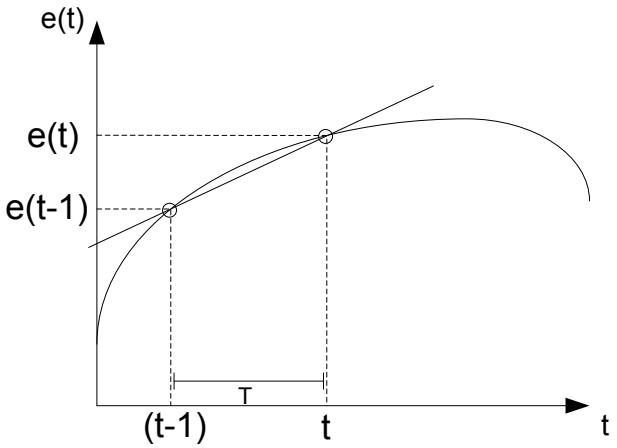
Oransal kontrolcü ayrık uzaya aşağıdaki gibi çevrilir.

$$\mathrm{u}_p(\mathrm{t}) = K_p e(\mathrm{t}),\ U_p(z) = K_p E(z),\ \mathrm{u}_p[\mathrm{k}] = K_p e[\mathrm{k}] \tag{5.5}$$

Oransal kontrolcünün ayrık uzaya dönüşümü yukarıda gösterildiği gibi oldukça basit iken aynı durum türevsel ve integral kontrolcü için geçerli değildir. Bu kontrolcülerin dönüşümleri için literatürde pek çok yaklaşım bulunmaktadır. Ayrık zamanlı türev almak için genelikle aşağıdaki yöntemler kullanılır;

- Forward difference approximation (İleri fark yaklaşımı)
- Backward difference approximation (Geri fark yaklaşımı)
- Central difference approximation (Merkez fark yaklaşımı)

Bu çalışmada geri fark yaklaşımı (bacward difference approximation) kullanarak türev alma işlemini aşağıdaki gibi gerçekleştirilmiştir.



Şekil 5.7 Geri fark yaklaşımı ile ayrık zamanlı türev hesabı

$$y(\mathrm{t}) = \frac{de(t)}{dt} = \frac{e(t) - e(\mathrm{t}-1)}{T},\ Y(\mathrm{z}) = \frac{E(\mathrm{z}) - \mathrm{z}^{-1}E(\mathrm{z})}{T} \tag{5.6}$$

$$\frac{Y(\mathrm{z})}{E(\mathrm{z})} = \frac{z-1}{zT} \tag{5.7}$$

Bu yaklaşımı kendi sistemimize uygulayarak türev hesabı yaparsak;

$$u_d = K_d \frac{d}{dt} e(\mathrm{t})$$

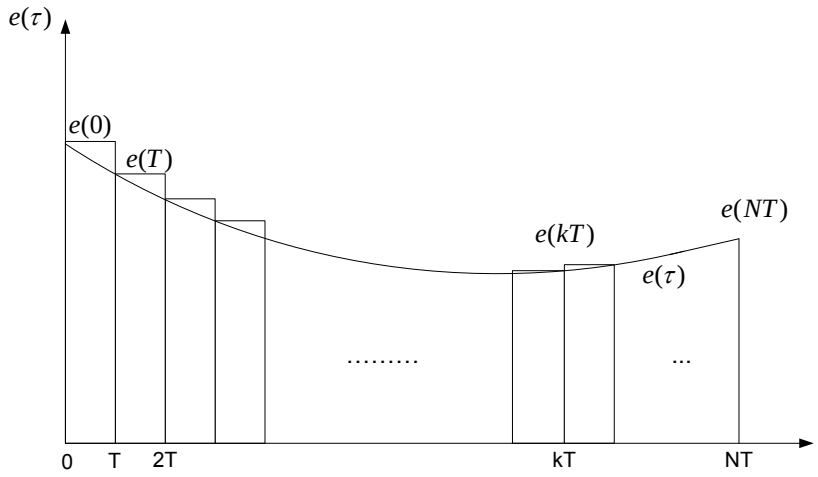
$$U_d(\mathrm{z}) = K_d \frac{E(\mathrm{z}) - \mathrm{z}^{-1} E(\mathrm{z})}{T_s} \tag{5.8}$$

$$u_d[\mathrm{k}] = \frac{\mathrm{K}_d}{T_s} (\mathrm{e}[\mathrm{k}] - \mathrm{e}[\mathrm{k}-1]) \tag{5.9}$$

Ayrık zamanlı integral almak için genellikle için aşağıda verilen üç dönüşümden biri kullanılmaktadır;

- Backward-Rectangular Integration (Geri Yol Entegrasyonu)
- Forward- Rectangular Integration (İleri Yol Entegrasyonu)
- Bilinear- Transformation Integration (Poligonal Entegrasyon)

Bu çalışma için bazı kabuller yapılarak geri yol entegrasyonu (backward rectangular approximation) kullanılmıştır.



Şekil 5.8 Geri dikdörtgen yaklaşımı ile ayrık zamanlı integral hesabı

Eğer $u(\mathrm{t}) = \int_0^t e(\tau) d\tau$ ise; $u(\mathrm{nT}) = \int_0^{\mathrm{nT}} e(\tau) d\tau \cong \sum_{k=0}^{n-1} e(\mathrm{kT}) \Delta t = \sum_{k=0}^{n-1} e(\mathrm{kT}) T$ (5.10)

$k = 0,1,2,...,n-1,\ \Delta t = T,\ n > 0$

Bu duruma benzer şekilde;

$$u((\mathrm{N}-1)\,\mathrm{T}) = \int_0^{(\mathrm{N}-1)\mathrm{T}} e(\tau)d\tau \cong \sum_{k=0}^{n-2} e(\mathrm{kT})T \tag{5.11}$$

Sonuç olarak fark denlemi aşağıdaki gibi olur;

$$\begin{aligned} u(NT) - u((N-1)T) &= Te((N-1)T) \\ u(NT) &= u((N-1)T) + Te((N-1)T) \end{aligned} \tag{5.12}$$

$$u[\mathrm{k}] = u[\mathrm{k}-1] + Te[\mathrm{k}-1] \tag{5.13}$$

Bu yaklaşımı kendi sistemimize uygulayarak integral hesabı yaparsak;

$$\mathrm{u}_i(\mathrm{t}) = K_I \int_0^t e(\mathrm{t})\,\mathrm{dt}$$

$$\mathrm{u}_i[\mathrm{k}] = \mathrm{u}_i[\mathrm{k}-1] + K_I T_s\, \mathrm{e}[\mathrm{k}-1] \tag{5.14}$$

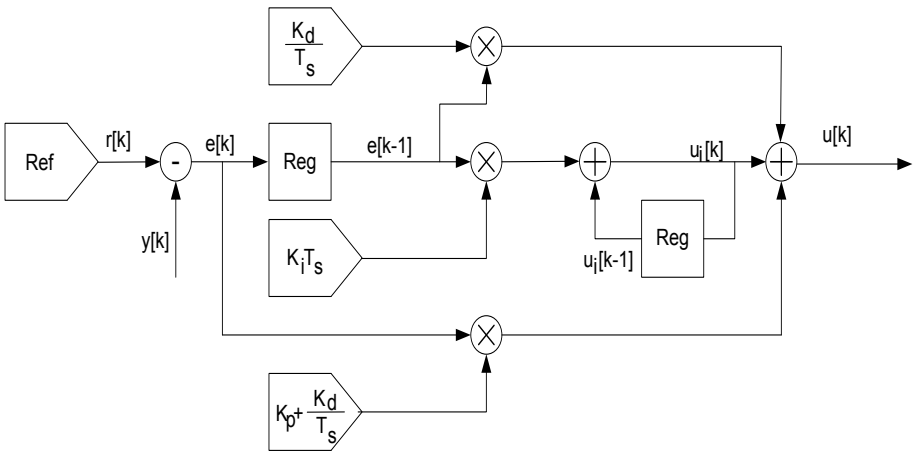
Sonuç olarak toplam ayrık PID denklemi aşağıdaki gibi olur;

$$u(\mathrm{t}) = \mathrm{u}_p(\mathrm{t}) + \mathrm{u}_i(\mathrm{t}) + \mathrm{u}_d(\mathrm{t})$$

$$u[\mathrm{k}] = K_p e[\mathrm{k}] + \mathrm{u}_i[\mathrm{k}-1] + K_I T_s\, \mathrm{e}[\mathrm{k}-1] + \frac{\mathrm{K}_d}{T_s}(\mathrm{e}[\mathrm{k}] - \mathrm{e}[\mathrm{k}-1]) \tag{5.15}$$

$$u[\mathrm{k}] = (K_p + \frac{\mathrm{K}_d}{T_s})\,\mathrm{e}[\mathrm{k}] + (-\frac{\mathrm{K}_d}{T_s})\,\mathrm{e}[\mathrm{k}-1] + (K_I T_s)\,\mathrm{e}[\mathrm{k}-1] + \mathrm{u}_i[\mathrm{k}-1] \tag{5.16}$$

Bu algoritma aynı zamanda kontrol çıkışının pozisyon algoritması olarak bilinir ve bu çalışma için yeterli görünmüştür[58]. Algortitmanın blok diyagramı Şekil 5.9'da verilmiştir;

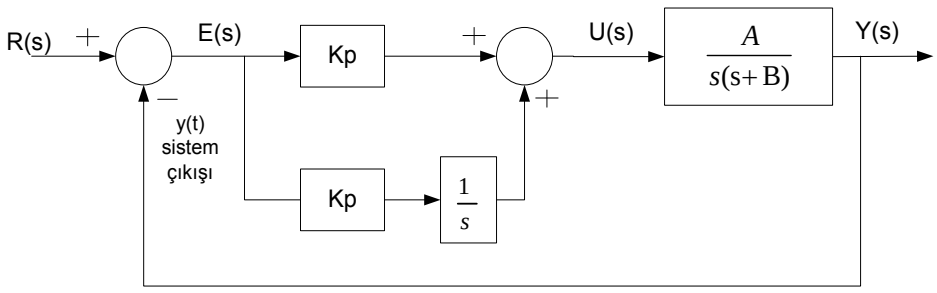


Şekil 5.9 Ayrık PID blok diyagramı

5.3 Kontrolcü Katsayılarının Bulunması

Beş eksenli robot kolunun ayrık PID kontrolcü ile kontrolü hedeflenmiş, fakat PI kontrolcünün yeterli olduğu görülmüştür. Teorik çalışmada oransal kontrolcü ile istenilen sonuç alınabiliyorken, integralin kullanılmasının sebebi, gerçek sistemdeki modellenemeyen bozucu etkilerden (sürtünme problemleri, boşluklar vs…) dolayı meydana gelen kararlı hal hatasını yok etmektir. Ayrıca oransal kontrolcü küçük PWM miktarlarında motorü sürmek için yetersiz kalmaktadır ve integral kontrolcü ile bu problem çözülmüştür. Bölüm 5.2'de oluşturulan paralel PID yapısı yazılımsal olarak da bu çalışma için oldukça uygundur. Algoritmada istenilen durumlara göre pid yapısının her üç terimi de aktif olarak kullanılmakla beraber, herhangi bir terimin etkisi de kolaylıkla ortadan kaldırılabilmektedir.

PI kontrolcü için genelleştirilmiş bir sistemin blok diyagramları Şekil 5.9 ve şekil 5.10'da gösterilmiştir.



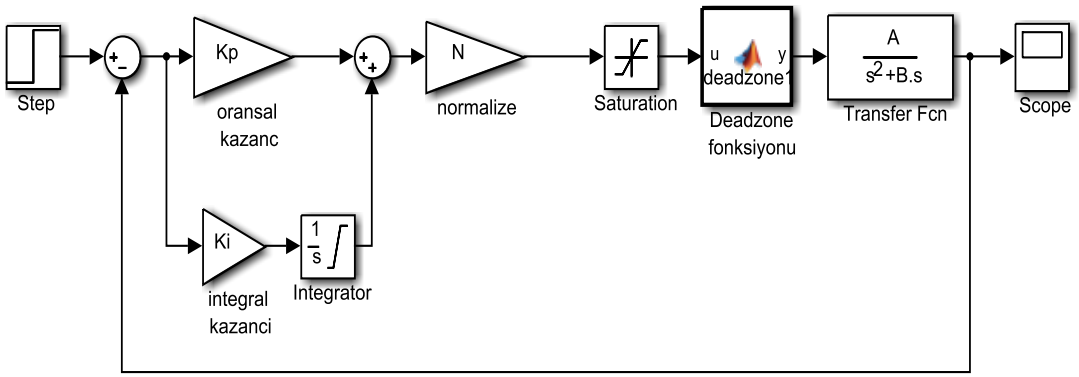
Şekil 5.10 PI kontrolcü blok diyagramı

$G_c(s)$: Kontrolcü transfer fonksiyonu
$G_p(s)$: Sistem transfer fonksiyonu
G(s): İleri yön transfer fonksiyonu
T(s): Kapalı çevrim transfer fonksiyonu

Bu bölümün devamında, robot eklemlerinin Bölüm 4.1'de elde edilen sistem modelleri için ayrı ayrı PI kontrolcü tasarımı için uygun K_p ve K_i katsayıları belirlenmiştir. Robotik kontrol sistemlerinin temel hedefi sistemi bir noktadan diğer noktaya istenilen sürede ve aşım olmadan hareket ettirebilmektir. Bu doğrultuda katsayılar hesaplanırken aşımın kabul edilebilir miktarda olması ve kararlı hal hatasının sıfır olması göz önüne alınmıştır. Tasarımda Matlab, Simulink ve Siso Design Tool

araçlarından faydalanılmıştır. Sisteme önce istenilen yükselme zamanını sağlayacak oransal deneteyici eklendikten sonra sistemin yapısını etkilemeyecek şekilde kararlı hal hatasını yok etmek için küçük bir integral denetleyici eklenmiştir. Elimizde robot eklemlerinin yaklaşık modelleri olduğu için kapalı çevrim köklerin reel eksen üzerinde olmasına dikkat edilmiştir.

Matlab Simulink elde edilen simülasyonlar Şekil 5.11'deki blok diyagramı ile gerçeklenmiştir;



Şekil 5.11 Sistem blok diyagramı (simulink)

Diyagram mümkün olduğu kadar sistemin gerçek modeline uygun hale getirilmeye çalışılmıştır. N değeri, sistemde geri besleme değerinin 0-3.3V aralığında alınırken motora verilen gerilim değerinin 0-12 V arasında değiştiği için eklenmiştir. Bu normalleştirme ifadesinin sistemin yapısına eklemek için her sistem transfer fonksiyonunua $N = 12/3.3 = 3.6364$ değeri eklenmiştir.

Burada dikkat edilmesi gereken nokta ise kontrolcü ayrık zamanlı olmasına karşılık PI parametreleri sürekli sistemde belirlenmiştir. Fakat bu parametreler ayrık zamanlı sistemde de kulanılabilmektedir. Sürekli sistemde bulunan PI parametreleri kapalı çevrim transfer fonksiyonunda yerine yazılıp, bu fonksiyon ayrık zaman uzayına dönüştürüldüğünde sistemin kökleri birim çemberin içinde kalmaktadır. Bu durum sistemin kararlı olduğunu göstermekte ve parametrelerin kullanılmasına olanak sağlamaktadır. Fakat burada öenmli olan diğer bir etkende sistemin örnekleme periyodudur. Örnekleme periyodunun farklı seçilmesi sistemin karakteristik yapısını bozabilir.

Birinci eklem için PI kontrolcü tasarımı:

Sistemin transfer fonksiyonu;

$$G_{p1}(\mathrm{s}) = \frac{\mathrm{N}^{*}0.58}{s(\mathrm{s}+19.44)} = \frac{2.1091}{s(\mathrm{s}+19.44)} \tag{5.17}$$

Sisteme sadece oransal denetleyici eklenip sistem performansı incelenirse;

İleri yön transfer fonksiyonu;

$$G_1(\mathrm{s}) = G_{c1}(\mathrm{s})^{*}\mathrm{G}_{p1}(\mathrm{s}) = \frac{2.1091K_{p1}}{s(\mathrm{s}+19.44)} \tag{5.18}$$

Kapalı çevrim transfer fonsiyonu.;

$$T_1(\mathrm{s}) = \frac{G_1(\mathrm{s})}{1+G_1(\mathrm{s})} = \frac{2.1091K_{p1}}{s^2+19.44s+2.1091K_{p1}} \tag{5.19}$$

Bu kriterler doğrultusunda Matlab Siso Design Tool kullanılarak sistemin kökleri ve K_{p1} değeri belirlenmiştir. $K_{p1}=40$ için istenilen kökler reel eksende ve istenilen sonuçlar yaklaşık olarak elde edilmiştir [59].

İkinci aşamada sisteme integral denetleyici eklenip sistem performansı incelenirse;

PI transfer fonksiyonu;

$$G_{c1}(\mathrm{s}) = \mathrm{K}_{p1} + \frac{K_{i1}}{s} = K_{p1}\left(\frac{s+K_{i1}/K_{p1}}{s}\right) \tag{5.20}$$

İleri yön transfer fonksiyonu;

$$G_1(\mathrm{s}) = \left(\frac{2.1091K_{p1}(s+K_{i1}/K_{p1})}{s^2(\mathrm{s}+19.44)}\right) \tag{5.21}$$

Kapalı çevrim transfer fonksiyonu;

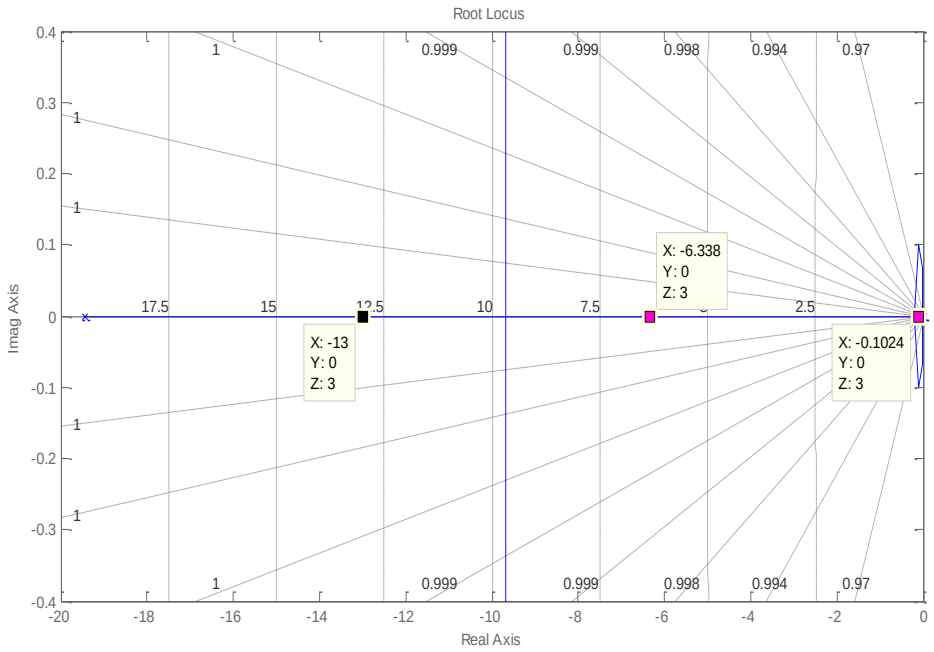
$$T_1(\mathrm{s}) = \frac{2.1091K_{p1}(s+K_{i1}/K_{p1})}{s^3+19.44s^2+2.1091K_{p1}s+2.1091K_{i1}} \tag{5.22}$$

Sistemin yapısını çok fazla etkilemeyecek şekilde $\frac{K_{i1}}{K_{p1}}=0.1$ seçilirse, istenilen tasarım kriterleri sağlanmış olur.

Seçilen kontrolcü;

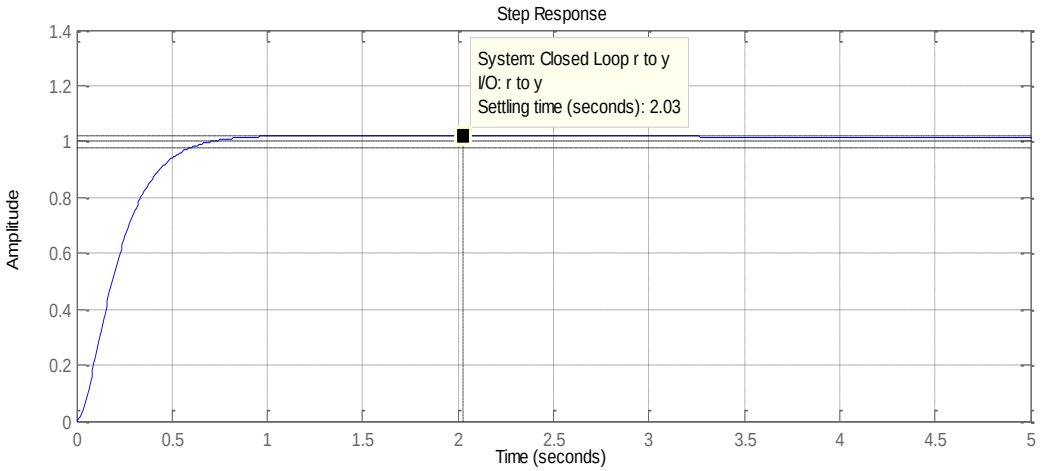
$$G_{c1}(s) = 40\left(\frac{s+0.1}{s}\right) \tag{5.23}$$

Matlab Siso Design Tool ile sistemin kökleri Şekil 5.12'deki gibi yerleştirilmiş ve basamak cevabı Şekil 5.12'deki gibi elde edilmiştir.



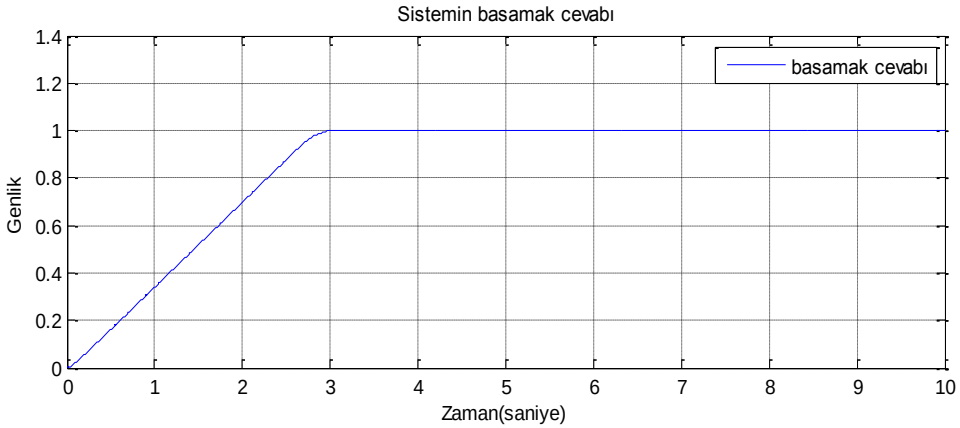
Şekil 5.12 Birinci eklem için Matlab/Sisotool ile köklerin yerleştirilmesi

$K_{p1} = 40$ ve $K_{i1} = 4$, Kökler; $s_1 = -13$, $s_2 = -6.34$, $s_3 = -0.1024$



Şekil 5.13 Sistemin basamak cevabı

Şekil 5.12'de verilen gerçek sistem modelinin basamak cevabı Şekil 5.14'de verilmiştir. Bu grafik Şekil 5.13'de Matlab Siso Tool ile elde edilen grafik ile karşılaştırıldığı zaman geçici reçici rejim cevaplarında farklılıklar görülmektedir. Bu farklılılar gerçek sistemde motorun maksimum hızı sınırlandırıldığı için oluşmaktadır.



Şekil 5.14 Gerçek sistem cevabı

Elde edilen kontrolcü, kapalı çevrim transfer fonksiyonunda yerine yazılırsa;

$$T_1(\mathrm{s}) = \frac{84.36s + 8.436}{s^3 + 19.44s^2 + 84.36s + 8.436} \tag{5.24}$$

Bölüm 5.6'da anlatılan PID algoritmasının periyodu 20 ms'dir ve bu değer kontrolcünün örnekleme periyodudur. Bu örnekleme periyodu kullanılarak Denklem 5.29 ayrık zaman uzayına çevirip kökleri incelenirse köklerinin birim çember içersinde kaldığı görülür.

$$T_1(\mathrm{z}) = \frac{0.01485z^2 - 0.001767z - 0.01303}{z^3 - 2.65z^2 + 2.328z - 0.6779} \tag{5.25}$$

Ayrık zamanlı transfer fonksiyonun kökleri;

$$z_1 = 0.99795, \quad z_2 = 0.88096, \quad z_3 = 0.77104$$

<u>İkinci eklem için PI kontrolcü tasarımı</u>

Sistemin transfer fonksiyonu;

$$G_{p2}(\mathrm{s}) = \frac{N * 0.7374}{s(\mathrm{s} + 24.55)} = \frac{2.6815}{s(\mathrm{s} + 24.55)} \tag{5.26}$$

Sisteme sadece oransal denetleyici eklenip sistem performansı incelenirse;

İleri yön transfer fonksiyonu;

$$G_2(\mathrm{s}) = G_{c2}(\mathrm{s}) * \mathrm{G}_{p2}(\mathrm{s}) = \frac{2.6815K_{p2}}{s(\mathrm{s}+24.55)} \tag{5.27}$$

Kapalı çevrim transfer fonksiyonu;

$$T_2(\mathrm{s}) = \frac{G_2(\mathrm{s})}{1+G_2(\mathrm{s})} = \frac{2.6815K_{p2}}{s^2+24.55s+2.6815K_{p2}} \tag{5.28}$$

Bu kriterler doğrultusunda Matlab Siso Design Tool kullanılarak sistemin kökleri ve K_{p2} değeri belirlenmiştir. $K_{p2}=50$ için kökler reel eksende ve istenilen sonuçlar yaklaşık olarak elde edilmiştir [59].

İkinci aşamada sisteme integral denetleyici eklenip sistem performansı incelenirse;

PI transfer fonksiyonu;

$$G_{c2}(\mathrm{s}) = \mathrm{K}_{p2} + \frac{K_{i2}}{s} = K_{p2}\left(\frac{s+K_{i2}/K_{p2}}{s}\right) \tag{5.29}$$

İleri yön transfer fonksiyonu;

$$G_2(\mathrm{s}) = \left(\frac{2.6815K_{p2}(s+K_{i2}/K_{p2})}{s^2(\mathrm{s}+24.55)}\right) \tag{5.30}$$

Kapalı çevrim transfer fonksiyonu;

$$T_2(\mathrm{s}) = \frac{2.6815K_{p2}(s+K_{i2}/K_{p2})}{s^3+24.55s^2+2.6815K_{p2}s+2.6815K_{i2}} \tag{5.31}$$

Matlab Siso Design Tool yardımıyla sistemin yapısını çok fazla etkilemeyecek şekilde $\frac{K_{i2}}{K_{p2}}=0.1$ seçilirse, istenilen tasarım kriterleri sağlanmış olur.

Seçilen kontrolcü;

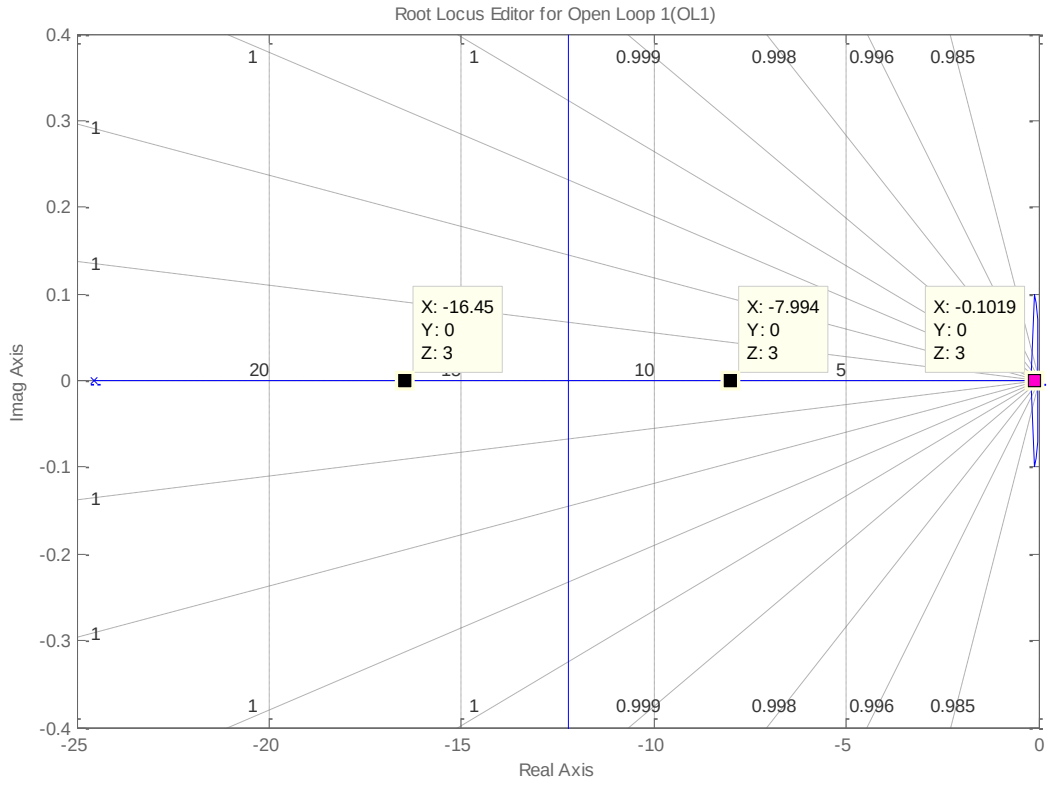
$$G_{c2}(\mathrm{s}) = 50\left(\frac{s+0.1}{s}\right) \tag{5.32}$$

Elde edilen kontrolcü, kapalı çevrim transfer fonksiyonunda yerine yazılırsa;

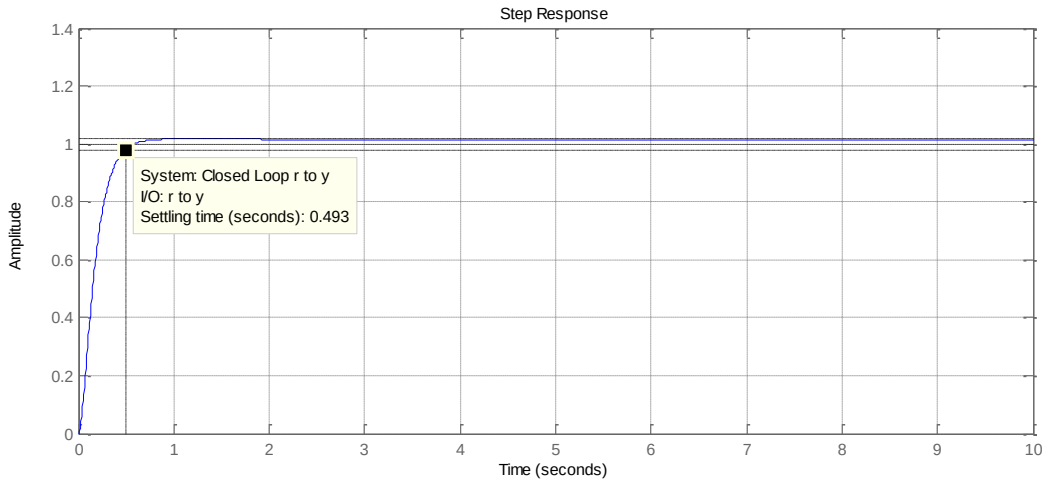
$$T_2(s) = \frac{134.1s + 13.41}{s^3 + 24.55s^2 + 134.1s + 13.41} \qquad (5.33)$$

$K_{p2} = 50$ ve $K_{i2} = 5$ Sistemin kökleri; $s_1 = -16.45$, $s_2 = -7.994$, $s_3 = -0.1019$

Matlab Siso Design Tool ile sistemin kökleri Şekil 5.15'deki gibi yerleştirilmiş ve basamak cevabı Şekil 5.15'deki gibi elde edilmiştir.

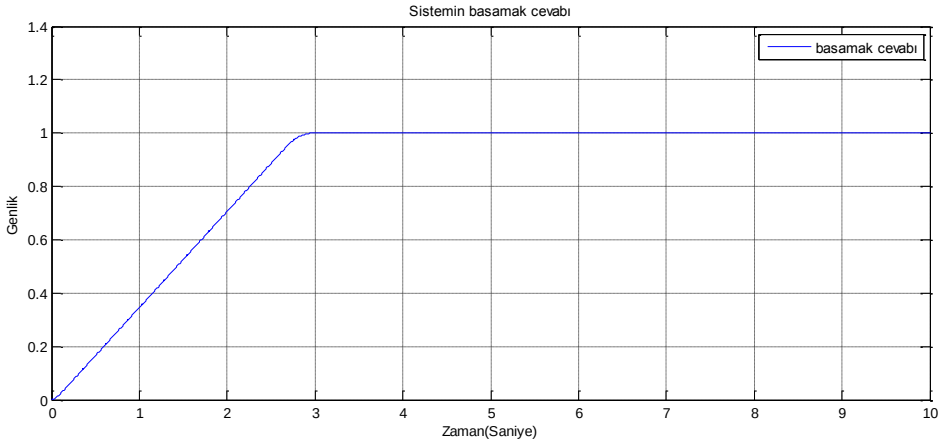


Şekil 5.15 İkinci eklem için Matlab/Sisotool ile köklerin yerleştirilmesi



Şekil 5.16 Sistemin basamak cevabı

Gerçek sistem modelinin basamak cevabı Şekil 5.17'de verilmiştir.



Şekil 5.17 Gerçek sistem cevabı

Denklem 5.38, $T = 0.02\,sn$ örnekleme periyodu ile ayrık zaman uzayına çevririp kökleri incelenirse köklerinin birim çember içersinde kaldığı görülür.

$$T_2(\mathrm{z}) = \frac{0.02283z^2 - 0.003389z - 0.01936}{z^3 - 2.57z^2 + 2.182z - 0.612} \tag{5.34}$$

Ayrık zamanlı transfer fonksiyonun kökleri;

$$z_1 = 0.99796, \quad z_2 = 0.85215, \quad z_3 = 0.7197$$

<u>Üçüncü eklem için PI kontrolcü tasarımı</u>

Sistemin transfer fonksiyonu;

$$G_{p3}(\mathrm{s}) = \frac{N * 0.8232}{s(\mathrm{s} + 28.10)} = \frac{2.9855}{s(\mathrm{s} + 28.10)} \tag{5.35}$$

Sisteme sadece oransal denetleyici eklenip sistem performansı incelenirse;

İleri yön transfer fonksiyonu;

$$G_3(\mathrm{s}) = G_{c3}(\mathrm{s}) * \mathrm{G}_{p3}(\mathrm{s}) = \frac{2.9855K_{p3}}{s(\mathrm{s} + 28.10)} \tag{5.36}$$

Kapalı çevrim transfer fonksiyonu;

$$T_3(\mathrm{s}) = \frac{G_3(\mathrm{s})}{1 + G_3(\mathrm{s})} = \frac{2.9855K_{p3}}{s^2 + 28.10s + 2.9855K_{p3}} \tag{5.37}$$

Bu kriterler doğrultusunda Matlab Siso Design Tool kullanılarak sistemin kökleri ve K_{p3} değeri belirlenmiştir. $K_{p3} = 60$ için kökler reel eksende ve istenilen sonuçlar yaklaşık olarak elde edilmiştir [59].

İkinci aşamada sisteme integral denetleyici eklenip sistem performansı incelenirse;

PI transfer fonksiyonu;

$$G_{c3}(\mathrm{s}) = \mathrm{K}_{p3} + \frac{K_{i3}}{s} = K_{p3}\left(\frac{s + K_{i3}/K_{p3}}{s}\right) \tag{5.38}$$

İleri yön transfer fonksiyonu;

$$G_3(\mathrm{s}) = \left(\frac{2.9855K_{p3}(s + K_{i3}/K_{p3})}{s^2(\mathrm{s} + 28.10)}\right) \tag{5.39}$$

Kapalı çevrim transfer fonksiyonu;

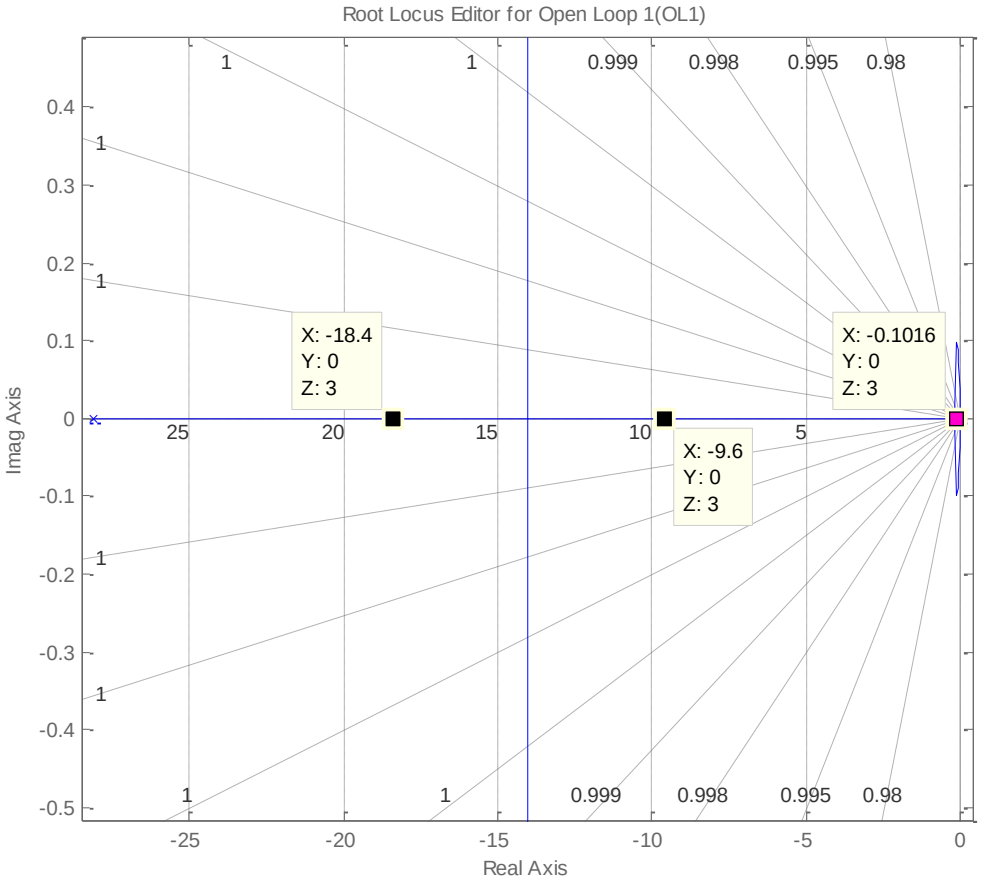
$$T_3(\mathrm{s}) = \frac{2.9855K_{p3}(s + K_{i3}/K_{p3})}{s^3 + 28.10s^2 + 2.9855K_{p3}s + 2.9855K_{i3}} \tag{5.40}$$

Matlab Siso Design Tool yardımıyla sistemin yapısını çok fazla etkilemeyecek şekilde $\frac{K_{i3}}{K_{p3}} = 0.1$ seçilirse, istenilen tasarım kriterleri sağlanmış olur.

Seçilen kontrolcü;

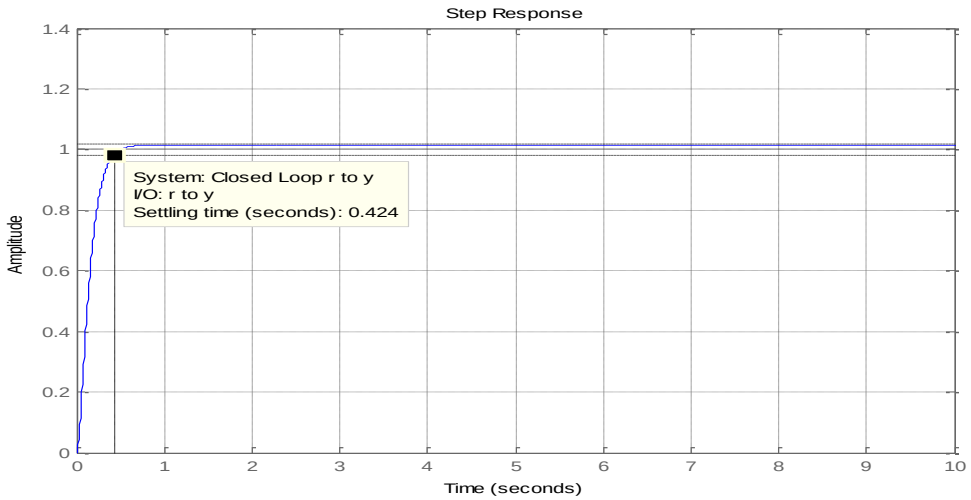
$$G_{c3}(\mathrm{s}) = 60\left(\frac{s + 0.1}{s}\right) \tag{5.41}$$

Matlab Siso Design Tool ile sistemin kökleri Şekil 5.18'deki gibi yerleştirilmiştir.

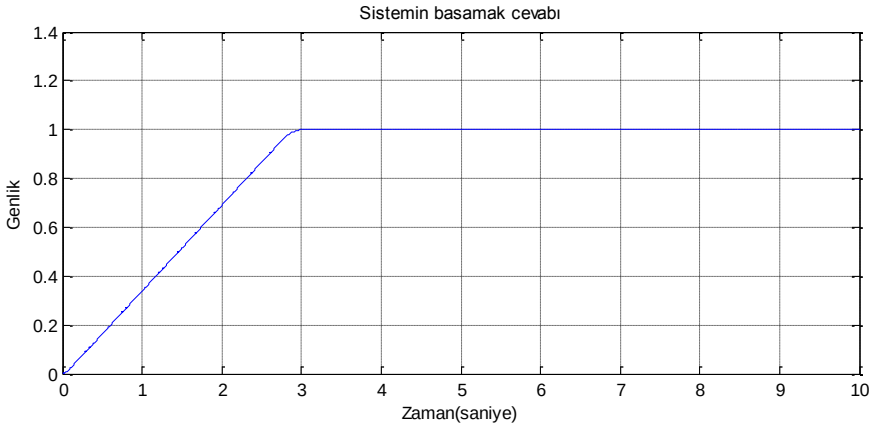


Şekil 5.18 Üçüncü eklem için Matlab/Sisotool ile köklerin yerleştirilmesi

$K_{p3} = 60$ ve $K_{i3} = 6$, sistemin kökleri; $s_1 = -18.4$, $s_2 = -9.6$, $s_3 = -0.1016$



Şekil 5.19 Sistemin basamak cevabı



Şekil 5.20 Gerçek sistemin basamak cevabı

Elde edilen kontrolcü, kapalı çevrim transfer fonksiyonunda yerine yazılırsa;

$$T_3(\mathrm{s}) = \frac{179.1s + 17.91}{s^3 + 28.1s^2 + 179.1s + 17.91} \tag{5.42}$$

Denklem 5.47, $T = 0.02\,sn$ örnekleme periyodu ile ayrık zaman uzayına çevririp kökleri incelenirse köklerinin birim çember içersinde kaldığı görülür.

$$T_3(\mathrm{z}) = \frac{0.02981z^2 - 0.005017z - 0.02468}{z^3 - 2.516z^2 + 2.086z - 0.5701} \tag{5.43}$$

Ayrık zamanlı transfer fonksiyonun kökleri;

$$z_1 = 0.99796, \quad z_2 = 0.8259, \quad z_3 = 0.692$$

<u>Dördüncü eklem için PI kontrolcü tasarımı</u>

Sistemin transfer fonksiyonu;

$$G_{p4}(\mathrm{s}) = \frac{N * 0.492}{s(\mathrm{s} + 30.9)} = \frac{1.7891}{s(\mathrm{s} + 30.9)} \tag{5.44}$$

Sisteme sadece oransal denetleyici eklenip sistem performansı incelenirse;

İleri yön transfer fonksiyonu;

$$G_4(\mathrm{s}) = G_{c4}(\mathrm{s}) * \mathrm{G}_{p4}(\mathrm{s}) = \frac{1.78.91K_{p3}}{s(\mathrm{s} + 30.9)} \tag{5.45}$$

Kapalı çevrim transfer fonksiyonu;

$$T_4(\mathrm{s}) = \frac{G_4(\mathrm{s})}{1 + G_4(\mathrm{s})} = \frac{1.7891K_{p4}}{s^2 + 30.9s + 1.7891K_{p4}} \tag{5.46}$$

Bu kriterler doğrultusunda Matlab Siso Design Tool kullanılarak sistemin kökleri ve K_{p4} değeri belirlenmiştir. $K_{p4}=130$ için kökler reel eksende ve istenilen sonuçlar yaklaşık olarak elde edilmiştir [59].

İkinci aşamada sisteme integral denetleyici eklenip sistem performansı incelenirse;

PI transfer fonksiyonu;

$$G_{c4}(\mathrm{s})=\mathrm{K}_{p4}+\frac{K_{i4}}{s}=K_{p4}\left(\frac{s+K_{i4}/K_{p4}}{s}\right) \tag{5.47}$$

İleri yön transfer fonksiyonu;

$$G_4(\mathrm{s})=\left(\frac{1.7891K_{p4}(s+K_{i4}/K_{p4})}{s^2(\mathrm{s}+30.9)}\right) \tag{5.48}$$

Kapalı çevrim transfer fonkiyonu;

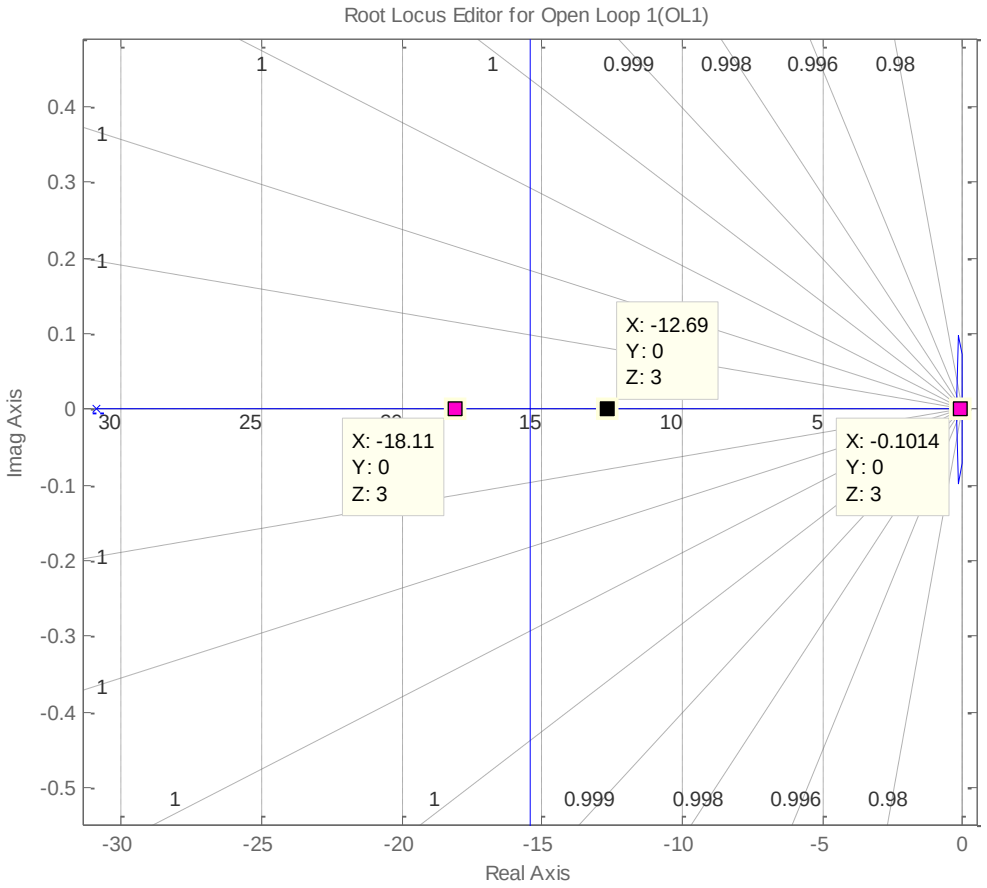
$$T_4(\mathrm{s})=\frac{1.7891K_{p4}(s+K_{i4}/K_{p4})}{s^3+30.9s^2+1.7891K_{p4}s+1.7891K_{i4}} \tag{5.49}$$

Sistemin yapısını çok fazla etkilemeyecek şekilde diğre eklemlerde olduğu gibi $\frac{K_{i4}}{K_{p4}}=0.1$ seçilirse, istenilen tasarım kriterleri sağlanmış olur.

Seçilen kontrolcü;

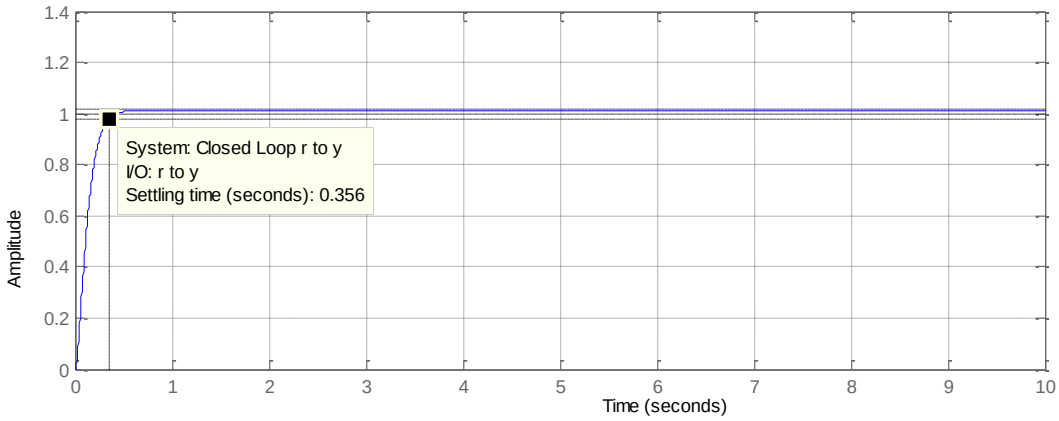
$$G_{c4}(\mathrm{s})=130\left(\frac{s+0.1}{s}\right) \tag{5.50}$$

Matlab Siso Design Tool ile sistemin kökleri Şekil 5.21'deki gibi yerleştirilmiş ve basamak cevabı Şekil 5.21'deki gibi elde edilmiştir.

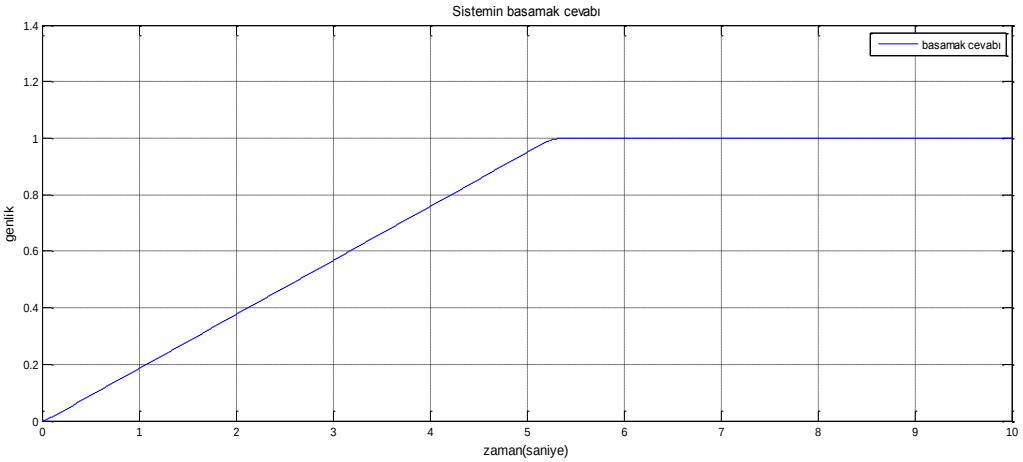


Şekil 5.21 Dördüncü eklem için Matlab/Sisotool ile köklerin yerleştirilmesi

$K_{p4} = 130$ ve $K_{i4} = 13$, sistemin kökleri; $s_1 = -18.18$, $s_2 = -12.69$, $s_3 = -0.1014$



Şekil 5.22 Sistemin basamak cevabı



Şekil 5.23 Gerçek sitemin basamak cevabı

Elde edilen kontrolcü, kapalı çevrim transfer fonksiyonunda yerine yazılırsa;

$$T_4(\mathrm{s}) = \frac{232.6s + 23.26}{s^3 + 30.9s^2 + 232.6s + 23.26} \tag{5.51}$$

Denklem 5.56, $T = 0.02\,sn$ örnekleme periyodu ile ayrık zaman uzayına çevririp kökleri incelenirse köklerinin birim çember içersinde kaldığı görülür.

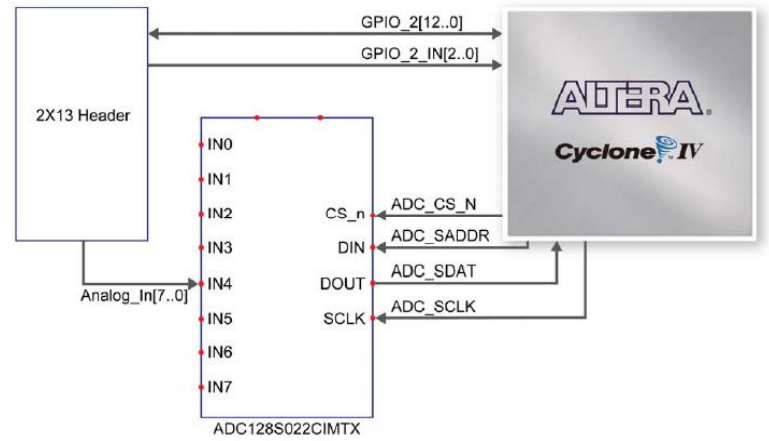
$$T_4(\mathrm{z}) = \frac{0.03799z^2 - 0.006978z - 0.03088}{z^3 - 2.47z^2 + 2.009z - 0.539} \tag{5.52}$$

Ayrık zamanlı transfer fonksiyonun kökleri;

$$z_1 = 0.99797, \quad z_2 = 0.7768, \quad z_3 = 0.6953$$

5.4 VHDL ile Analog Veri Okuma Algoritması

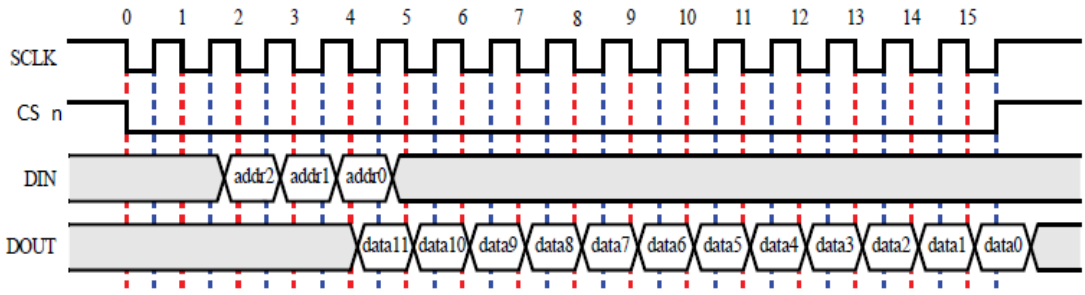
DE0 Nano kart üzerinde ADC128S022 A/D dönüştürücü bulunmaktadır. Bu dönüştürücü 8 kanal analog girişi 12 bit ayrık sinyale dönüştürülür. ADC'nin sekiz giriş kanalı FPGA kartı üzerinde bulunan 3x13 harici bağlantı girişlerine IN0'dan IN7'ye kadar yapılandırılmıştır. Şekil 5.24'de 2x13 bağlantı girişi, ADC ve FPGA birimlerinin bağlantı şeması gösterilmektedir [53].



Şekil 5.24 2x13 Bağlantı girişi, ADC ve FPGA birimlerinin bağlantı şeması

SCLK ve CS_n sinyalleri FPGA tarafından üretilir ve ADC'yi kontrol etmek için kullanılır. SCLK bağlantısı ADC biriminin saat sinyalidir. CS_n bağlantısı da ADC için düşük aktif chip seçim sinyalini oluşturur. Yani CN_n sinyali düşük değerde olduğu zaman (0 dijital) ADC seçilen kanaldan okuduğu analog veriyi dijitale çevirir. DIN ve DOUT bağlantıları seçilen kanal bilgisinin ve ayrık verinin chipler arasında transferini sağlamak için kullanılırlar. DIN bağlantısı seçilecek bir sonraki kanal bilgisini adreslemek için kullanılır. FPGA cipinin ADC_SADDR pinine bağlanmıştır ve üç bit uzunluğundaki adres bilgisi ADC'ye seri olarak SCLK saat sinyalinin her periyod döngüsünde bir bit olarak gönderilir. DOUT bağlantısı da FPGA'in ADC_SDAT pinine bağlanmıştır ve ADC'nin ürettiği 12 bit uzunluğundaki ayrık bilgiyi seri olarak SCLK'nın her periyodunda alarak istenilen adres yerine depolar [61].

ADC128S022 analog-ayrık dönüştürücüsü bir kanaldan veri okuma işleminin SCLK saat sinyalinin 16 periodunda tamamlar. Şekil 5.25'de ADC'nin zamanlama diyagramı gösterilmiştir.



Şekil 5.25 ADC zamanlama diyagramı[61]

CS_n sinyali, SCLK saat sinyalinin ilk düşen kenarında dijital 0 olur ve ADC'nin bir çevrimini başlatır. DOUT sinyali bir önceki seçilmiş kanaldan 12 bit dijital sinyal oluşturulan bilgiyi, data bitleri azalan sırada iletilir, yani en yüksek değerlikli bit ilk olarak FPGA'e iletilir. FPGA tarafından bu bitler SCLK'nın yükselen kenarında yakalanır. DIN sinyali ile bir sonraki seçilecek kanalın bilgisi azalan sırada ADC'ye gönderilir. ADC adres bilgisini SCLK'nın pozitif kenarında alacağı için veriler kullanıcı tarafından SCLK'nın negatif kenarında gönderilmelidir. SCLK frekansı ADC'nin düzgün çalışması için 0.8 ile 3.2 MHz arasında sınırlandırılmalıdır[61][62].

Deneysel calışmanın ilk uygulamasında ADC'nin sekiz kanalı da aktif olarak okunmaktadır. Bu sekiz kanaldan dört tanesi robotun taban, omuz, dirsek ve bilek bölümlerindeki DC motorların konum bilgilerini okuyan potansiyometrelere ve geri kalan dört kanal da robotun referans girişlerini belirleyecek potansiyometrelere bağlıdır. Sistem çalıştığı sürece ADC'nin sekiz kanalıda aktif olarak okunmakta ve değerler sürekli olarak güncellenmektedir. İkinci uygulamada ise referans değerler algoritma içersinde sabit olarak verilmiştir ve ADC'nin dört kanalı aktif olarak okunmaktadır.

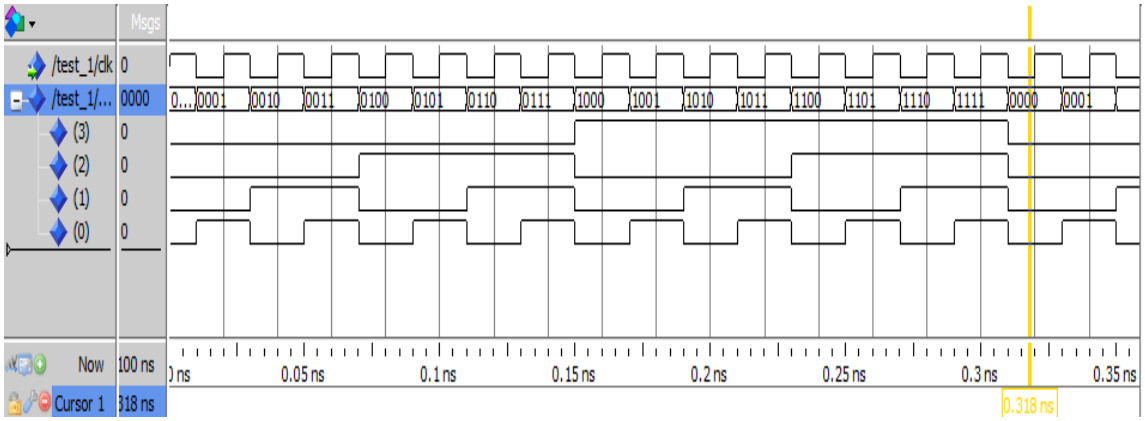
FPGA içinde bulunan bütün saatli devrelerin senkron olarak tetiklenmesi sistemin doğru çalışması açısından önemlidir. Tetiklemeler eş zamanlı olarak gerçekleşmezse ADC'de kanal kayması ya da okunan veride bit kayması gibi sonuçlar oluşur. Bu durumu engellemek için tüm sistemin tek bir saat sinyali ya da sayıcı ile kontrol edilmesi yaklaşımı düşünülmüştür. VHDL bit (bit_vector) tabanlı bir program olduğu için tanımlanan değişkenin tüm bitlerini istenildiği anda erişilmekte ve hatta bir bitlerdeki değişimler saat sinyali olarak kullanılabilmektedir. Bu yaklaşımla ana programda otuzüç bitlik bir değişken tanımlanmış (32 downto 0) ve bu otuz üç bitin ilk on iki biti ADC'ye kanal atama ve veri okuma için kullanılmıştır.

Bu yaklaşımı göstermek için aşağıda basit bir sayıcı programı yazılmıştır. Sayıcı bilinen program yapısının aksine *integer* yerine *std_logic_vector* olarak tanımlanmıştır ve bu sayede sayıcının her bir bitine birbirinden bağımsız olarak ulaşılabilmektedir. Sayıcı yapısı gereği sıralı olarak çalışacağı için *counter_system* adında bir *process* yapısının içine yazılmıştır. Programda *prosess* yapısının duyarlılık listesinde yer alan *clk* değişkeni

FPGA'in harici saat sinyalini temsil eder. Saat sinyalinin her değişiminde ilgili process içine girilir ve saat sinyalinin her düşük değerinde sayıcı (counter) bir artar. Dört bit olarak tanımlanan sayıcı ikilik sistemde "0000" başlayarak "1111"'e kadar sayar ve sayıcı maximum değerine gelince değeri kendisinin otomatik olarak sıfırlanır ve tekrar saymaya başlar.

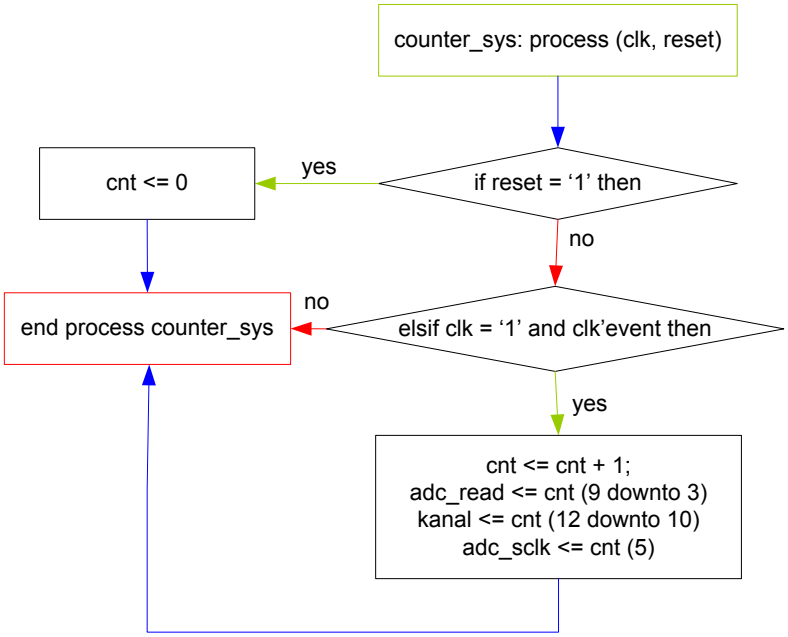
```
--4 bitlik sayıcı VHDL kodu
architecture arch of test_1 is
signal counter : std_logic_vector (3 downto 0) := "0000";
begin
   counter_system : process(clk)
   begin
      if (reset = '1') then
         counter <= "0000";
      elsif (clk = '0' and clk'event) then
         counter <= counter + 1;
      end if;
   end process counter_system;
end arch;
```

Modelsim programı ile elde edilen simülasyon sonucunda, clk sinyalini ve counter sinyalin her bitindeki değişimini Şekil 5.26'da gösterilmiştir.



Şekil 5.26 VHDL sayıcı programı

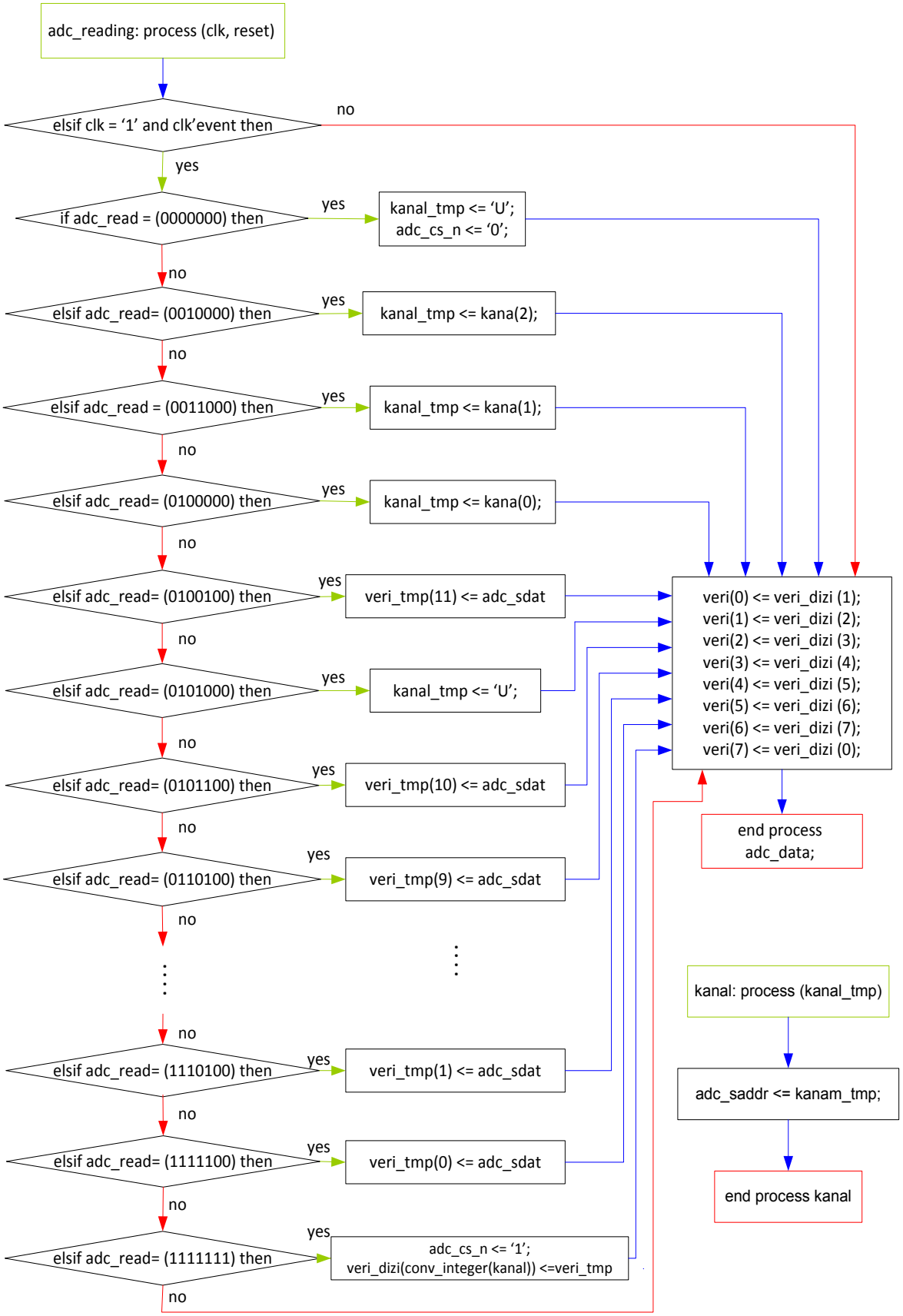
Bu yaklaşım kullanılarak hazırlanan ADC okuma algoritması aşağıda detaylı olarak anlatılmıştır. Şekil 5.26'da sistemin senkron çalışması için oluşturulan temel sayıcı için akış diyagramına yer verilmiştir. *Process* yapısının duyarlılık listesinde belirtilen *clk* FPGA'in harici saat sinyalidir. *Reset* ise sistemi sıfırlayan reset butonu için oluşturulan sinyaldir.



Şekil 5.27 Senkron saat sinyali oluşturmak için akış diyagramı

Std_logic_vector olarak tanımlanan otuz üç bitlik değişkenin (cnt), altıncı biti ADC'nin saat sinyali için (adc_sclk <= cnt (5)), üçüncü bitinden dokuzuncu bitine kadar bitleri ADC kanal seçmek ve veri okumak için (adc_read <= cnt (9 downto 3)), onuncu bitinden on ikinci bitine kadar olan bitleri de ADC kanal atamak için (kanal <= cnt (12 downto 10) kullanılmıştır. ADC'nin düzgün çalışabilmesi için saat sinyalinin frekansı 0.8 ile 3.2 MHz aralığında seçilmelidir [62]. Ana programda cnt (5) bitinin frekansı yaklaşık olarak 1.56 Mhz'dir ve ADC'nin çalışması için uygundur

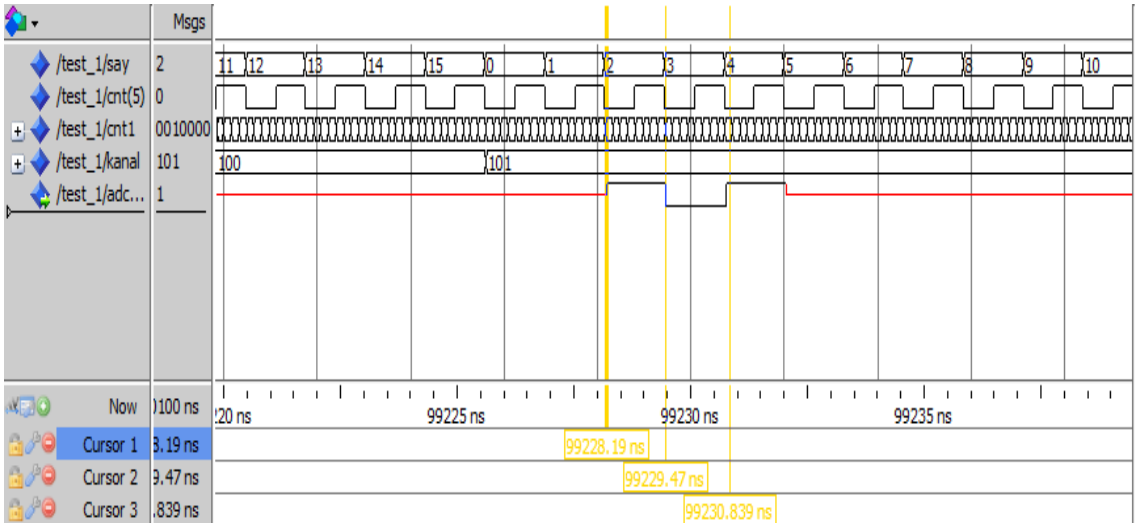
adc_read değişkeni "0000000" değerini aldığı zaman ADC'nin kanallarına 'U' değeri atanır (kanal_tmp <= U) ve ADC veri alışverişine açık hale getirilir (adc_cs_n <= 0). Sonra ADC'nin saat sinyalinin (adc_sclk) ikinci, üçüncü ve dördüncü düşen kenarlarında ADC'ye kanal bilgisi gönderilir (kanal_tmp <= kanal(3)) . ADC kanal bilgisini kaydettikten sonra beşinci düşen kenarda kanal bilgisi göndermeyi bırakır. ADC veri gönderme işlemini de saat sinyalinin dördüncü yükselen kenarından itibaren başlayarak on beşinci yükselen kenrına kadar sıra ile gönderir (veri_tmp_(x) <= adc_sdat) ve FPGA on iki bitlik seri bilgiyi ADC'den azalan sırada almış olur. Şekil 5.27'de ADC'ye kanal atama ve ADC'den veri okuma algoritmasına yer verilmiştir.



Şekil 5.28 ADC veri okuma akış diyagramı

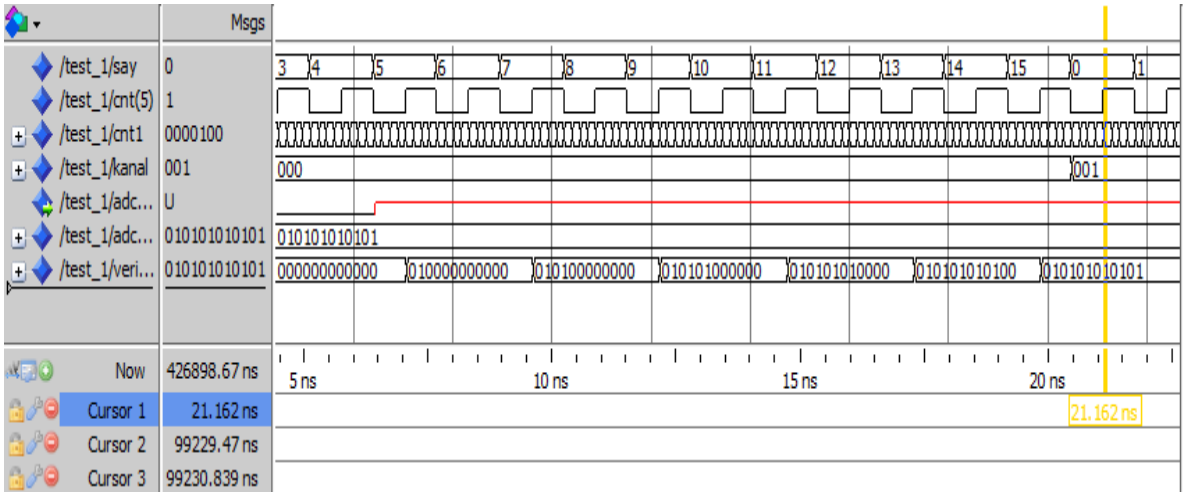
FPGA Nano kartı varsayılan olarak sıfırıncı kanalı okumaya ayarlandığı için ‘U’ değeri atanması ile ADC’nin hatalı okuma yapması engellenmiş olur. Bu değer ilgili kütüphanede “tanımlanmamış” değer anlamına gelmektedir ve ADC kanalları bu değerde iken herhangi bir okuma yapmaz. Kanal değerleri seri olarak ADC’ye kanal bilgisini gönderecek olan adc_saddr pinine gönderilir. İlgili saat sinyali geldiğinde FPGA ADC’den veri toplamaya başlar ve sıralı olarak her kanaldan aldığı veriyi std_logic_vector tipinde tanımlanmış ve her biri on iki bit uzunluğundaki değişkenlerine depolar (veri_tmp_(x) <= adc_sdat). *adc_read* değişkeni “1111111” değerini aldığı zaman ADC’den alınan datalar sıra ile *veri_dizi* adında tanımlanmış olan her biri oniki bitlik 8 x 1’lik bir diziye depolanır (veri_dizi(conv_integer(kanal)) <= veri_tmp) ve ADC veri alışveriline kapatılır (adc_cs_n <= 0). Sonra her kanal için toplanan veriler veri(0), veri(1)…veri(7) olarak isimleidirilen oniki bitlik değişkenlere aktarılır.

Şekil 5.29’de MODELSIM ile elde edilen ADC kanal atama grafiği gösterilmiştir. Grafikte ADC’nin beşinci kanalı seçilmek istenmiş ve işlem başarı ile gerçekleştirilmiştir.



Şekil 5.29 ADC kanal atama

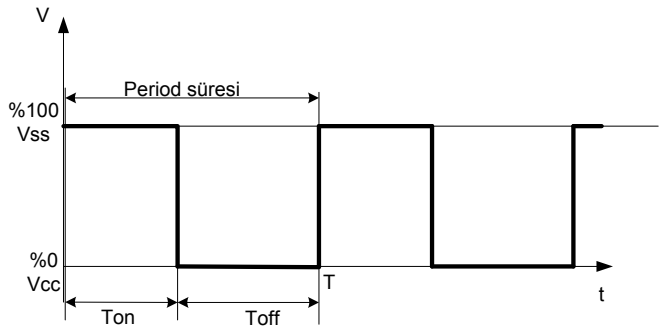
Şekil 5.29’da MODELSIM ile elde edilen ADC’den veri okuma işlemi grafiği gösterilmiştir. İşlem sonucunda ADC’deki veri seri olarak FPGA’e aktarılmıştır.



Şekil 5.30 ADC'den veri okuma grafiği

5.5 VHDL ile PWM Algoritması

PWM (Pulse-width modulation, darbe genişlik modülasyonu), üretilecek olan darbenin genişlikleri kontrol edilerek çıkışta istenilen analog değerin veya sinyalin elde etmek için kullanılır. Telekominikasyon, güç, voltaj düzenleyiciler, ses üreticiler gibi pek çok farklı uygulamada kullanılan PWM tekniği bu çalışmada, robot eklemlerindeki motorlara istenilen gerilim değeri göndermek için kullanılmıştır. Aşağıdaki şekilde bir PWM dalgası gösterilmiştir;



Şekil 5.31 PWM dalgası

PWM dalgasının tepe noktasında kaldığı bölgeye yüksekte kalma süresi(T_{on}), aşağıda kaldığı bölgeye düşükte kalma süresi (T_{off}) adı verilir. Benzer şekilde PWM dalgasının tepe değeri (tepe gerilimi) V_{ss} ve çukur değeri de V_{cc} olarak isimlendirilebilir. PWM ile kontrol mantığı ile dalganın T_{on} ve T_{off} sürelerini ayarlayarak V_{ss} ve V_{cc} arasındaki tüm

gerilim değerleri elde edilebilir. PWM tekniği ile elde edilebilecek gerilim değeri aşağıdaki gibi hesaplanır;

T_{on} : PWM daldasının yüksekte kalma süresi

T_{off} : PWM dalgasının düşükte kalma süresi

T : PWM periodu

D : Sinyal Oranı(kullanım oranı, duty cycle)

V_{ss} : En yüksek gerilim değeri

V_{cc} : En düşük gerilim değeri

V : Çıkışta elde edilen gerilim değeri

$f(\mathrm{t})$: PWM frekansı

$$T = T_{on} + T_{off}, \quad D = \frac{T_{on}}{T_{on} + T_{off}} = \frac{T_{on}}{T} \tag{5.53}$$

PWM dalgasının bir period boyunca oluşturduğu gerilim aşağıdaki gibi hesaplanır;

$$V = \frac{1}{T}\int_0^T f(\mathrm{t})\,\mathrm{dt}\ , \quad 0 < t < T_{on}, \quad T_{on} < t < T \tag{5.54}$$

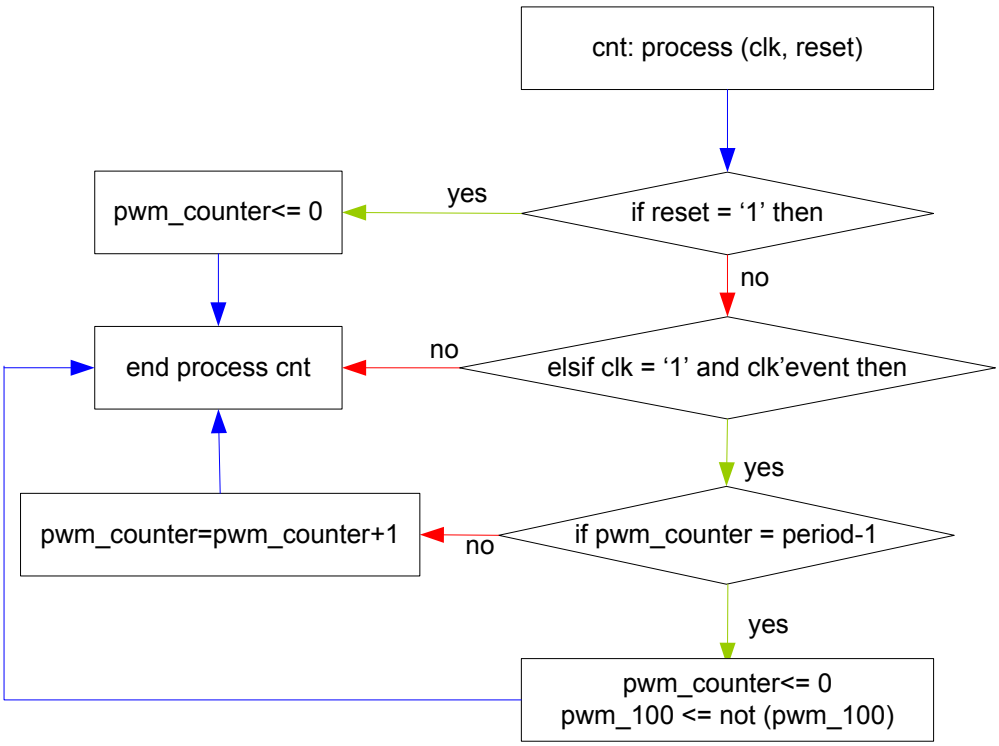
$$V = \frac{1}{T}\left(\int_0^{T_{on}} V_{ss}\,dt + \int_{T_{on}}^{T_{off}} V_{cc}\,dt\right) \tag{5.55}$$

$$V = \frac{1}{T}\left(V_{ss}T_{on} + V_{cc}(T_{off} - T_{on})\right) \tag{5.56}$$

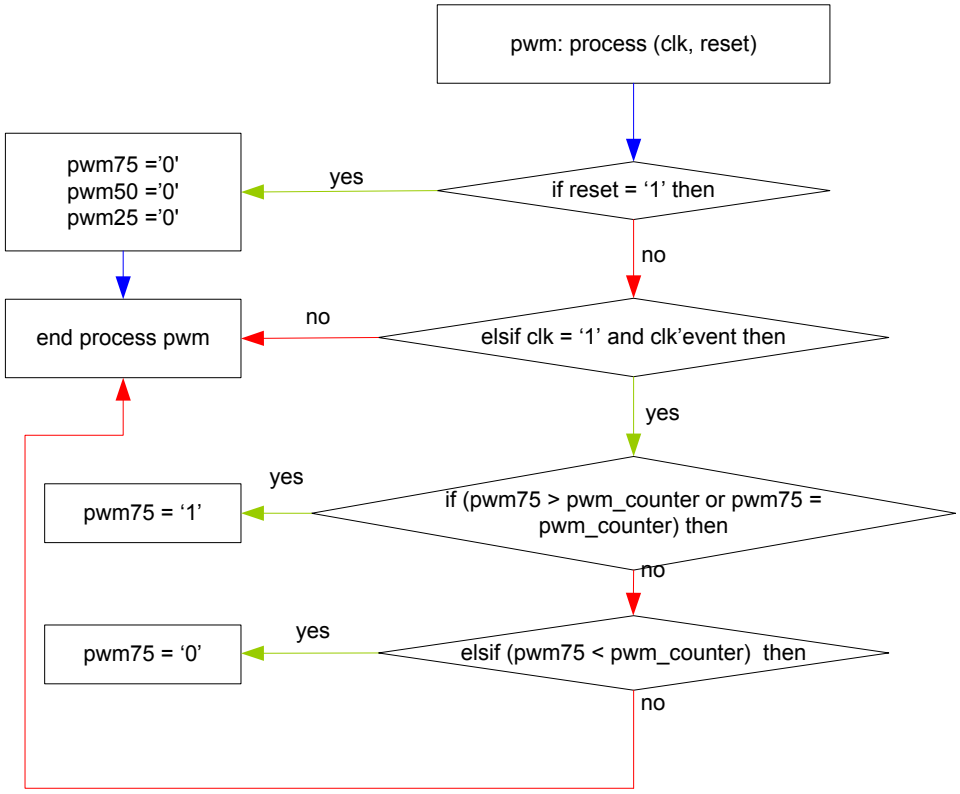
$$V = V_{ss}D + (1-\mathrm{D})V_{cc}), \qquad V_{cc} = 0 \Rightarrow \qquad V = V_{ss}D \tag{5.57}$$

Günümüzde PWM desteği olan birçok mikroişlemci bulunaktadır. Bu mikroişlemciler ile yazılımsal ve donanımsal PWM elde edilebilir. Fakat bu mikroişlemciler düşük çözünürlüklü, tek kanal PWM üretme ya da yavaş çalışma gibi sistemin çalışmasını etkileyebilecek birçok dezavantaja sahiptir. Bu uygulamada FPGA ile yazılımsal olarak beş kanal PWM üretilmiş ve bunlar eş zamanlı olarak robot eklemlerindeki motorlara uygulanmıştır. Şekil 5.31 ve 5.32'de VHDL dili ile yazılımsal PWM oluşturmak için gerekli akış diyagramları verilmiştir. Şekil 5.33'de ModelSim programı ile elde edilen %100, %75, %50, %25 pwm oranları için elde edilen grafiklere yer verilmiştir.

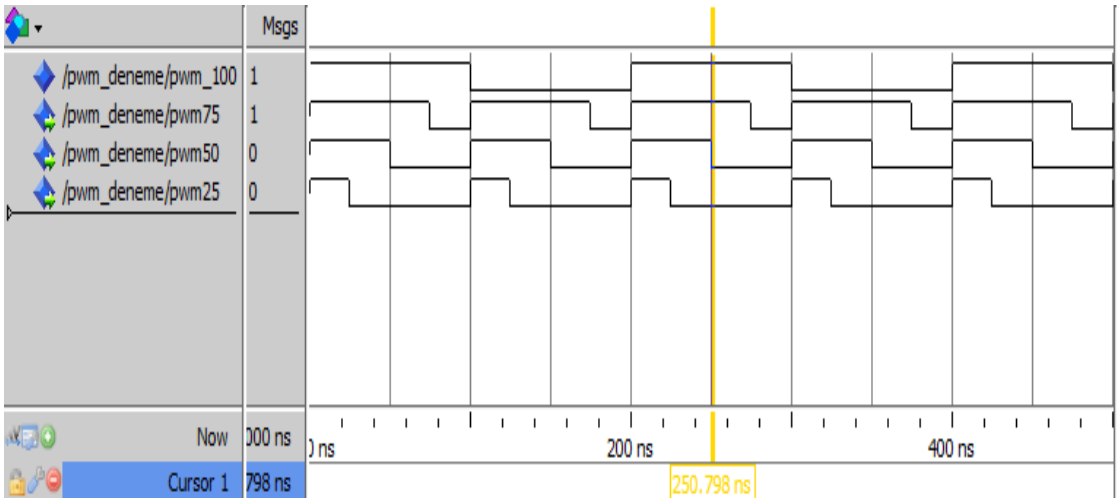
FPGA ile eş zamanlı kod yazmak için genellikle iki *process* kullanılır. Birinci *process'de* kullanılacak olan sayıcı ya da saat sinyali tanımlanırken ikinci process'de bu sayıcılara ya da sinyallere ilgili işlemler gerçekleştirilir. Bu yaklaşımla yazılan PWM kodunun ilk *processi* içinde reset işlemi, sayici ve tam period PWM (karşılaştırmak amacıyla) oluşturulmuştur. İkinci *prosess'in* içinde de istenilen doluluk oranlarında PWM dalgaları oluşturmak için işlemler gerçekleştirilmiştir.



Şekil 5.32 PWM sinyali için sayıcı tasarımı



Şekil 5.33 %75 doluluk oranında PWM sinyali oluşturma



Şekil 5.34 %75, %50 ve %25 doluluk oranında PWM dalgaları

5.6 VHDL ile PID Algoritması

Bu bölümde bölüm 5.3'te elde ettiğimiz dijital pid denklemi için olıuşturulan VHDL algoritması anlatılmıştır. Aşağıda denklem ile ilgili yapılan düzenlemeler verilmiştir.

$$u[\mathrm{k}] = (K_p + \frac{\mathrm{K}_d}{T_s})\,\mathrm{e}[\mathrm{k}] + (-\frac{\mathrm{K}_d}{T_s})\,\mathrm{e}[\mathrm{k}-1] + (K_I T_s)\,\mathrm{e}[\mathrm{k}-1] + \mathrm{u}_i[\mathrm{k}-1] \quad (5.58)$$

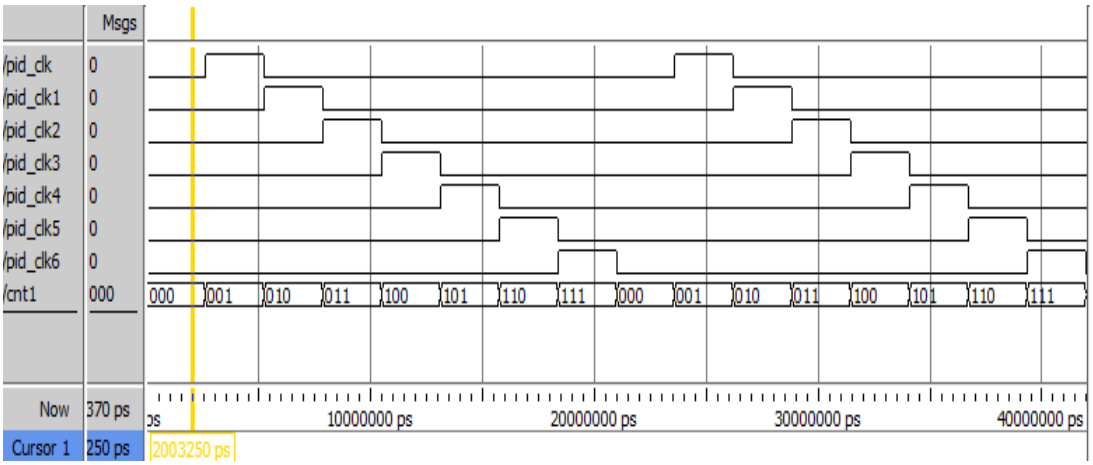
$$\alpha = K_p + \frac{\mathrm{K}_d}{T_s}, \quad \beta = -\frac{\mathrm{K}_d}{T_s}, \quad \zeta = K_I T_s \quad (5.59)$$

$$u[\mathrm{k}] = \alpha\,\mathrm{e}[\mathrm{k}] + \beta\,\mathrm{e}[\mathrm{k}-1] + \zeta\,\mathrm{e}[\mathrm{k}-1] + \mathrm{u}_i[\mathrm{k}-1] \quad (5.60)$$

Algoritmanın gerçeklenmesi için saat sinyali olarak otuz üç bitlik temel sayıcının (cnt) on sekizden yirmiye kadar olan üç bitinin değişimi kullanılmıştır. Bu üç bit değişiminin yedi farklı kombinasyonu ile sıralı olarak değişen yedi saat sinyali oluşturulmuş ve Denklem 5.60'de verilen ayrık PID algoritması yedi adımda tamamlanmıştır. Aşağıda ayrık PID algoritmasının adım adım nasıl gerçeklendiği, ilgili kodlar, akış diyagramları ve simülasyonlara yer verilmiştir.

```
--counter
counter_sys: process (clk, reset)
begin
   if reset = '1' then
     counter <= 0;
   elsif clk ='0' and clk'event then
     cnt<=cnt+1;
     pid_clk  <= (not cnt(19)) and (not cnt (18))and(cnt (17));
     pid_clk1 <= (not cnt(19)) and (cnt(18)) and (not cnt(17));
     pid_clk2 <= (not cnt(19)) and (cnt(18)) and (cnt(17));
     pid_clk3 <= (cnt(19)) and (not cnt(18)) and (not cnt(17));
     pid_clk4 <= (cnt(19)) and (not cnt(18)) and (cnt(17));
     pid_clk5 <= (cnt(19)) and (cnt(18)) and (not cnt(17));
     pid_clk6 <= (cnt(19)) and (cnt(18)) and (cnt(17));
   end if;
end process counter_sys;
```

Bu saat sinyallerinin tetiklenme zamanlarını göstermek için MODELSIM programnda elde edilen grafikler Şekil 5.35'te gösterilmiştir.



Şekil 5.35 Pid algoritması için ardışık saat sinyalleri

FPGA'in içinde PID algoritmasının gerçeklenmesi için yukarıda grafikleri verilen yedi saat sinyalinde ile tamamlanır. Denklem 6.60'ın gerçeklenmesi için VHDL programında kullanılan değişkenler ve sinyaller Çizelge 5.2'de listelenmiştir.

Çizelge 5.2 VHDL programında kullanılan ifadeler

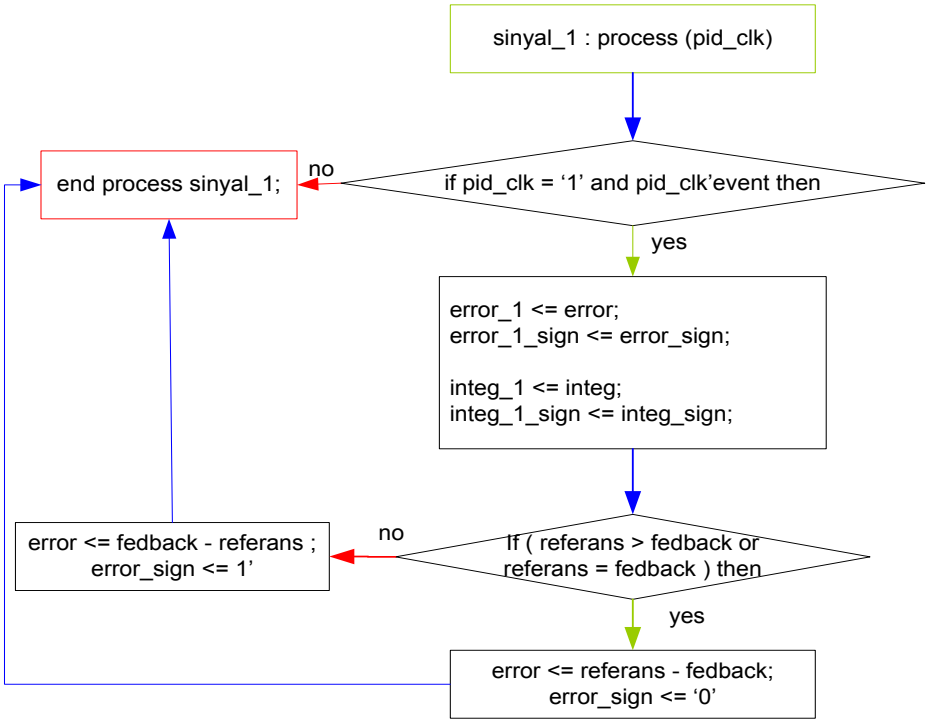
Değişken adı	**Açıklama**
error	Güncel hata değeri ($e[k]$)
error_1	Geçmiş hata değeri ($e[k-1]$)
error_sign	Güncel hata sinyalinin işareti
error _1_sign	Geçmiş hata sinyalinin işareti
integ	İntegral kontrolcünün değeri ($u_i[k]$)
integ_1	İntegral kontrolcünün önceki değeri değeri ($u_i[k-1]$)
integ_sign	İntegral kontrolcünün işareti
integ _1_sign	İntegral kontrolcünün önceki değerinin işareti
referans	Referans değer
fedback	Geri besleme değeri
alfa	$K_p + K_d / T_s$ (α)
alfa_sign	alfa'nı işareti

Çizelge 5.2 VHDL programında kullanılan ifadeler (devamı)

Değişken adı	Açıklama
beta	$-(\mathrm{K}_d/T_s)(\beta)$
beta_sign	beta'nın işareti
zeta	$K_I T_s$ (ζ)
zeta_sign	zeta'nın işareti
alfa_1	$\alpha_1 = \alpha e[\mathrm{k}]$
alfa_1_sign	alfa_1'in işareti
beta_1	$\beta_1 = \beta e[\mathrm{k}-1]$
beta_1_sign	beta_1'in işareti
zeta_1	$\zeta_1 = \zeta e[\mathrm{k}-1]$
zeta_1_sign	zeta_1'in işareti
teta	$\theta = \alpha_1 + \beta_1$
teta_sign	teta'nın işareti
epsilon	$\varepsilon = \zeta_1 + u_i[k-1]$
epsilon_sign	epsilon'un işareti
ucontrol	Toplam kontrol sinyali
ucontrol_sign	Toplam kontrol sinyalinin işareti
u_limit	Limitli kontrol sinyali
u_end	Kontol sinyali
u_end_sign	Kontrol sinyalinin işareti
$u[\mathrm{k}]$	Kontrol sinyali (integer)

Algoritma içinde bir bilgi için, bu bilginin değerini, işaretini, bir önceki değerini ve işaretini saklayan dört değişken bulunur. Bilgilerin işaretleri bir bit işaret değişkeni ile kaydedilmekte olup bilgi pozitifse ya da sıfırsa işaret değeri '0', negatifse '1' değerini alır. Her sinyalde gerçeklenen adımlar aşağıda açıklanmıştır.

pid_clk sinyali değişiminde, hata sinyali ve integral kontrolcü, bir önceki değerlerini saklayacak değişkene atanır ve hata bilgisi kendilerini günceller. Aynı şekilde değişkenlerin işaretleri de başka bir değişkene aktarılarak saklanır. Şekil 5.36'da *pid_clk* sinyali değişimi ile algoritmada meydana gelen değişikler ile ilgili akış diyagramı ve devamında ilgili VHDL kodu verilmiştir.

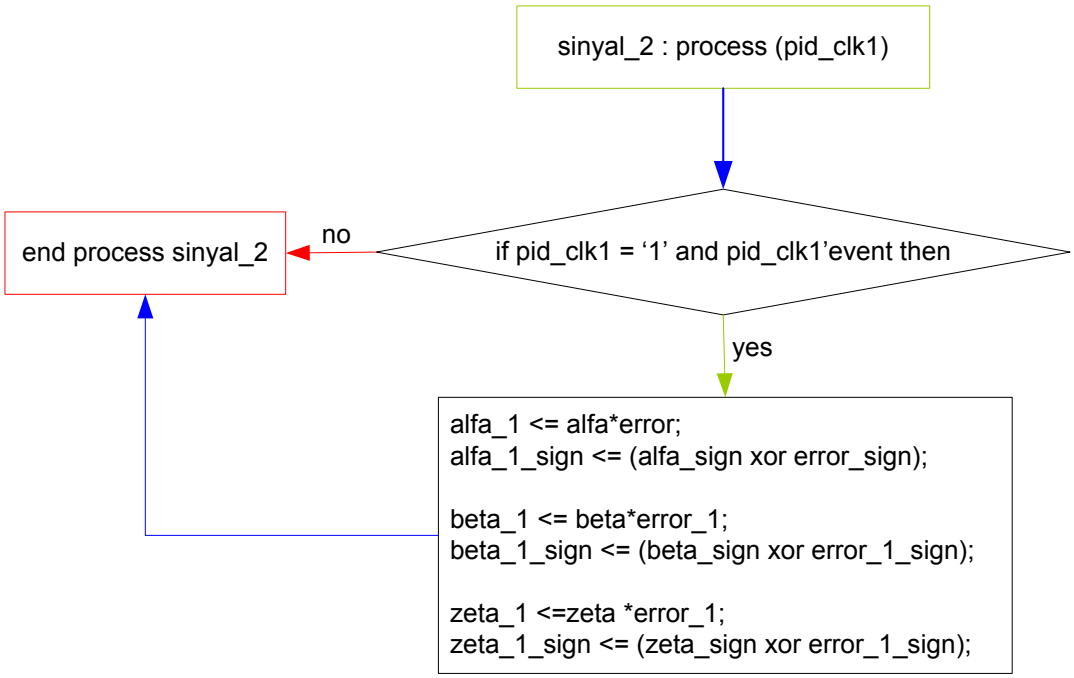


Şekil 5.36 pid_clk sinyal değişimi için akış diyagramı

```
sinyal_1: process (pid_clk)
begin
   if pid_clk ='1' and pid_clk'event then
      error_1 <= error;
      error_1_sign <= error_sign; --işaret bilgisi
      integ_1<= integ;
      integ_1_sign <= integ_sign;
      -- hata bilgisinin güncellenmesi
```

```
      if(referans > feedback or referans = feedback) then
         error <= referans - feedback;
         error_sign <='0';
      else
         error <= feedback - referans;
         error_sign <='1';
      end if;
   end if;
end process sinyal_1;
```

pid_clk1 sinyali değişiminde, $\alpha_1 = \alpha e[k]$, $\beta_1 = \beta e[k-1]$ ve $\zeta_1 = \zeta e[k-1]$ işlemleri gerçekleşir. Şekil 5.37'de ilgili akış diyagramı ve devamında kod bilgisi verilmiştir



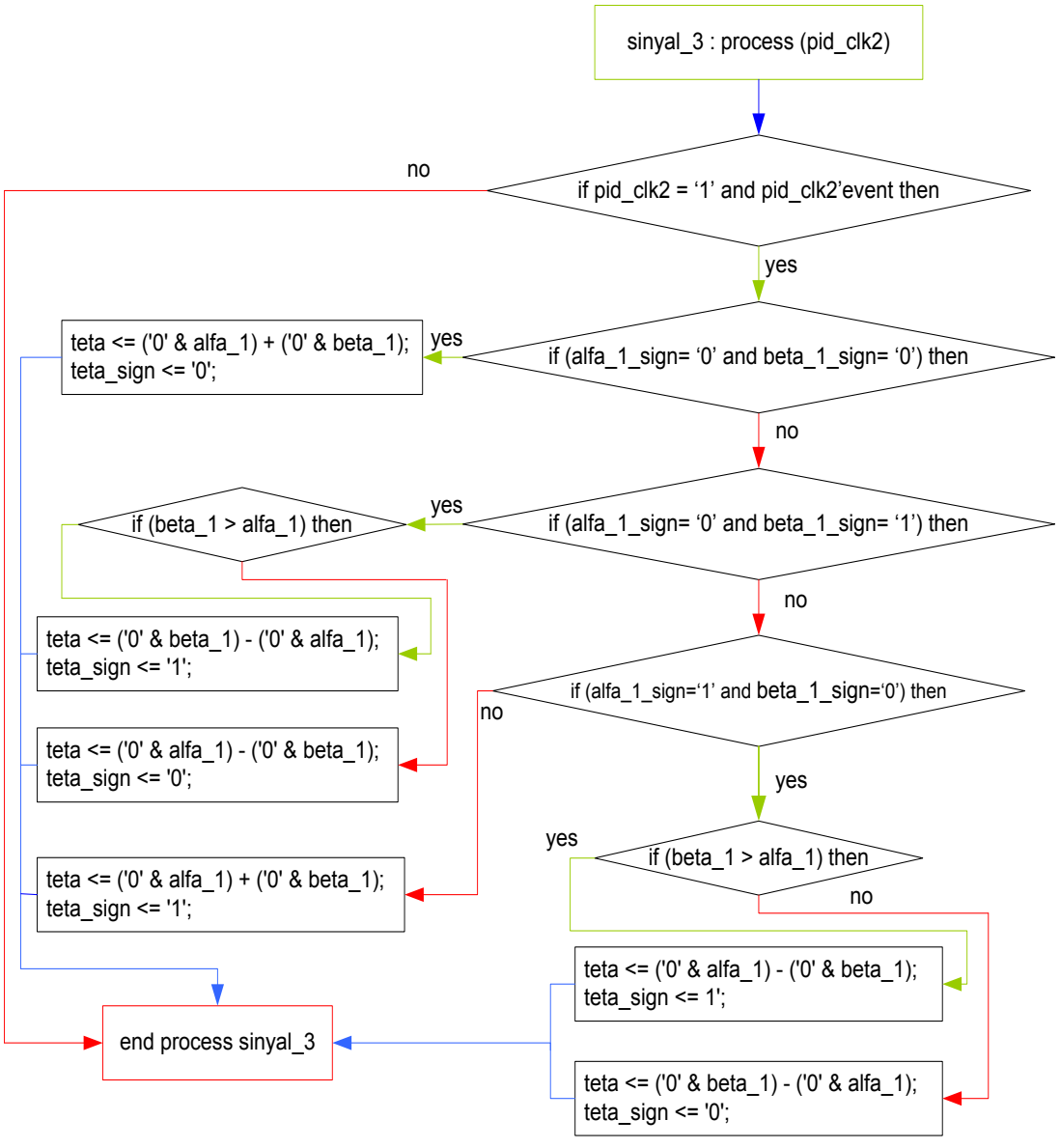
Şekil 5.37 pid_clk1 sinyal değişimi için akış diyagramı

Bilgilerin işaretleri pozitifse ya da sıfırsa işaret bitinin (alfa_sign, beta_sign, alfa_1_sign, …) değeri '0', negatifse '1' değerini almaktadır. Bu nedenle çarpma işlemi sonucunda oluşacak değerin işareti xor kapısı kullanılarak hesaplanır.

```
sinyal_2:process (pid_clk1)
begin
   if (pid_clk1'event and pid_clk1 ='1') then
      alfa_1 <= alfa*error;
```

```
        alfa_1_sign <= (alfa_sign xor error_sign);
        beta_1 <= beta*error_1;
        beta_1_sign <= (beta_sign xor error_1_sign);
        zeta_1 <= zeta*error_1;
        zeta_1_sign <= (zeta_sign xor error_1_sign);
    end if;
end process sinyal_2;
```

pid_clk2 sinyali değişiminde, $\theta = \alpha_1 + \beta_1$ ve $\varepsilon = \zeta_1 + u_i[k-1]$ işlemleri gerçekleşir. Şekil 5.38 $\theta = \alpha_1 + \beta_1$ işlemi için hazırlanan akış diyagramına ve devamında ilgili kodlara yer verilmiştir.



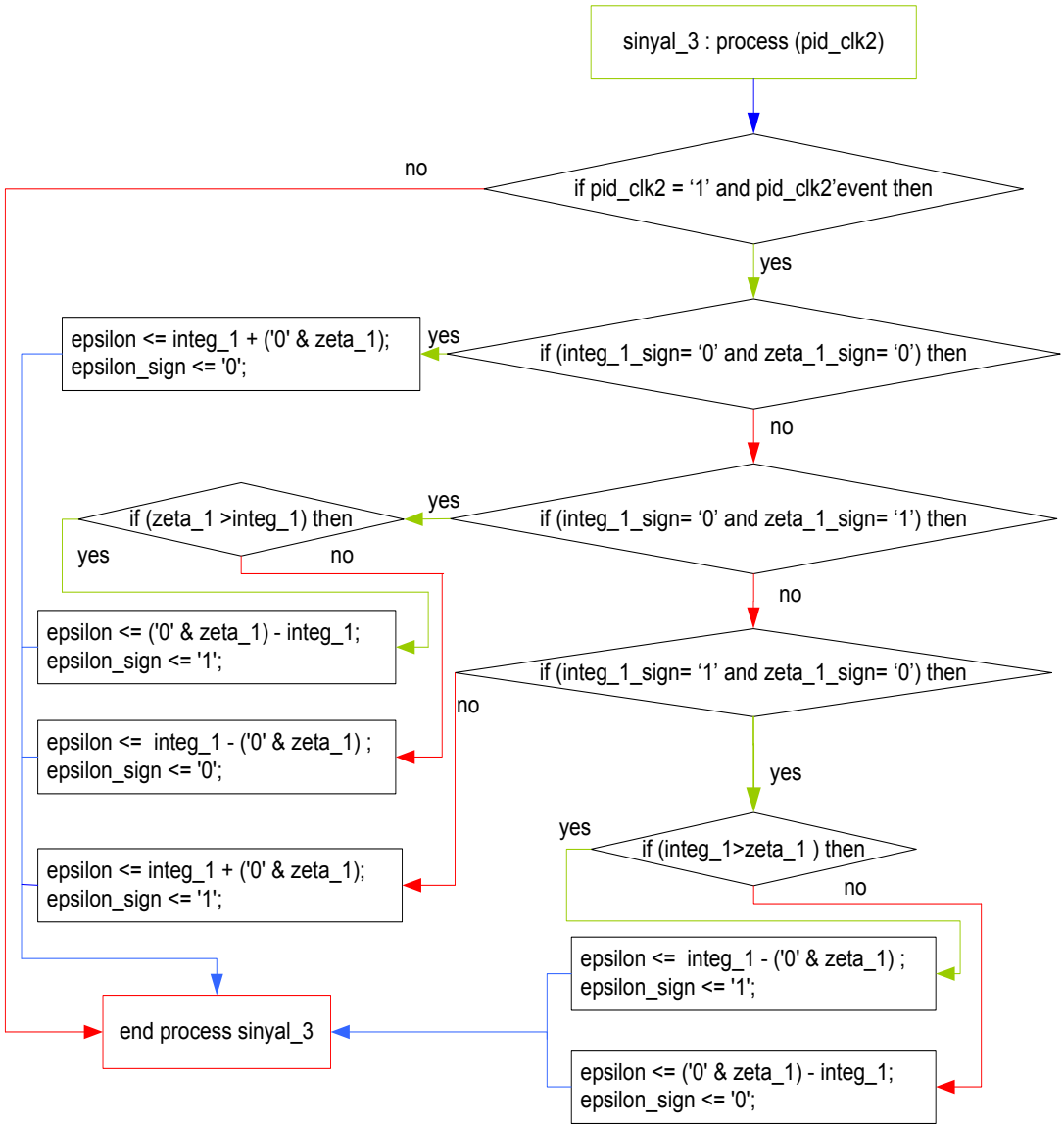
Şekil 5.38 pid_clk2 sinyal değişimi için $\theta = \alpha_1 + \beta_1$ işlemi

Teta değerini bulmak için yapılan işlem algoritmik olarak düşünüldüğü zaman alfa ve beta değişkenlerinin işaretlerinin dört ayrı kombinasyonuna göre teta değeri ayrı ayrı hesaplanır. Bu hesaplama işlemi için ilgili akış diyagramı Şekil 5.38'de verilmiştir.

Bu işlem için hazılanan VHDL kodları da aşağıda verilmiştir.

```
sinyal_3: process (pid_clk2)
begin
   if (pid_clk2'event and pid_clk2 ='1') then
      if(alfa_1_sign = '0' and beta_1_sign = '0') then
         teta <= ('0' & alfa_1) + ('0' & beta_1);
         teta_sign <= '0';
      elsif(alfa_1_sign = '0' and beta_1_sign = '1') then
         if(beta_1>alfa_1) then
            teta <= ('0' & beta_1) - ('0' & alfa_1);
            teta_sign <= '1';
         else
            teta <= ('0' & alfa_1) - ('0' & beta_1);
            teta_sign <= '0';
         end if;
      elsif(alfa_1_sign = '1' and beta_1_sign = '0') then
         if(alfa_1>beta_1) then
            teta <= ('0' & alfa_1) - ('0' & beta_1);
            teta_sign <= '1';
         else
            teta <= ('0' & beta_1) - ('0' & alfa_1);
            teta_sign <= '0';
         end if;
      else
         teta <= ('0' & alfa_1) + ('0' & beta_1);
         teta_sign <= '1';
      end if;
   end if
end process sinyal_3;
```

$\varepsilon = \zeta_1 + u_i[k-1]$ işlemi için hazırlanan akış diyagramına ve devamında ilgili koda yer verilmiştir.

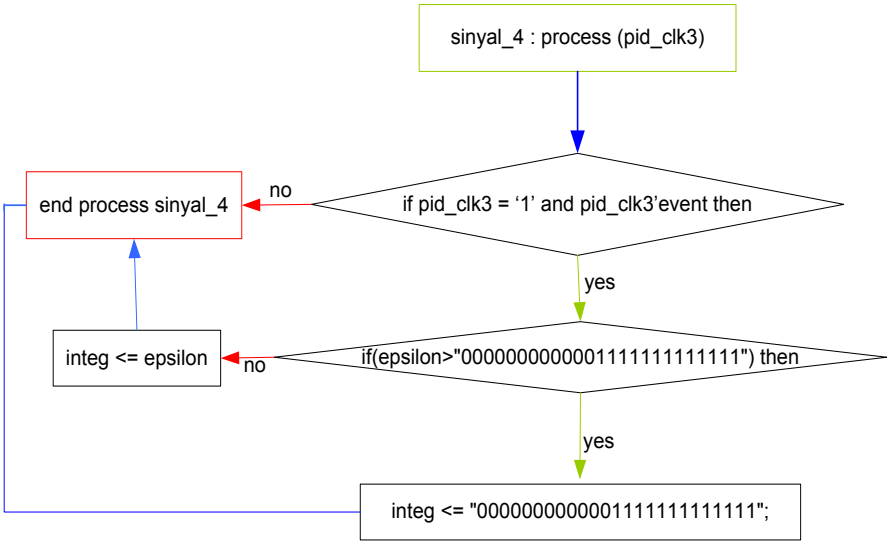


Şekil 5.39 pid_clk2 sinyal değişimi için $\varepsilon = \zeta_1 + u_i[k-1]$ işlemi

```
sinyal_3: process (pid_clk2)
begin
   if (pid_clk2'event and pid_clk2 ='1') then
      if(integ_1_sign = '0' and zeta_1_sign = '0') then
         epsilon <= integ_1 + ('0' & zeta_1);
         epsilon_sign <= '0';
      elsif(integ _1_sign = '0' and zeta_1_sign = '1') then
```

```
         if(zeta_1>integ_1) then
            epsilon <= ('0' & zeta_1)- integ_1;
            epsilon_sign <= '1';
         else
            epsilon <=  integ_1 - ('0' & zeta_1);
            epsilon_sign <= '0';
         end if;
      elsif(integ _1_sign = '1' and zeta_1_sign = '0') then
         if(integ_1> zeta_1) then
            epsilon <=  integ_1 - ('0' & zeta_1) ;
            epsilon_sign <= '1';
         else
            epsilon <= ('0' & zeta_1) - integ_1;
            epsilon_sign <= '0';
         end if;
      else
         epsilon <= integ_1 + ('0' & zeta_1);
         epsilon_sign <= '1';
      end if;
   end if
end process sinyal_3;
```

pid_clk3 sinyali değişiminde, pid_clk2 sinyali sonucunda elde edilen epsilon değeri ile ilgili sınırlama yapılır. Epsilon değerinin maksimum değerinden büyük çıkması durumunda maksimum değere eşitlenerek sınırlandırılır. Bu işlem için ilgili akış diyagramı Şekil 5.39'da verilmiştir.

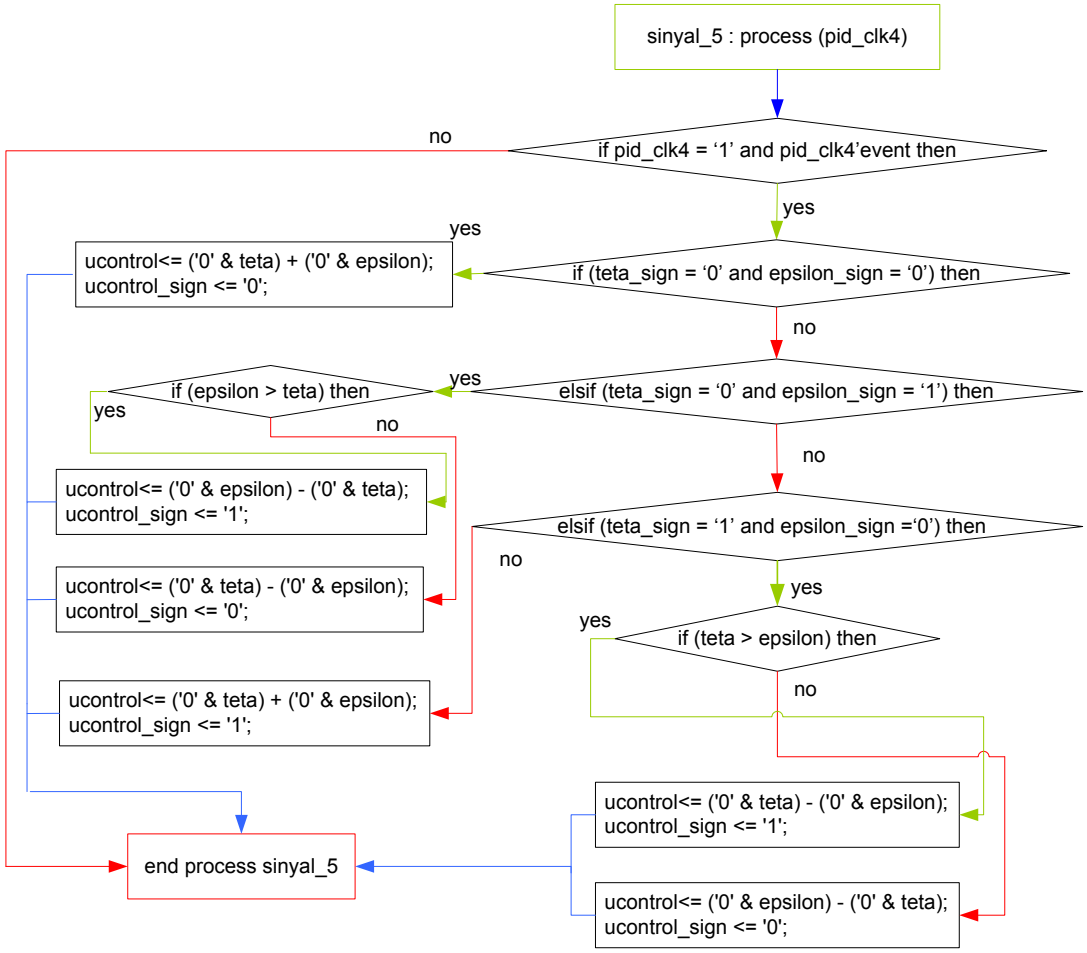


Şekil 5.40 pid_clk3 sinyal değişimi için akış diyagramı

Bu işlem için hazılanan VHDL kodları da aşağıda verilmiştir.

```
process (pid_clk3)
begin
   if (pid_clk3'event and pid_clk3 ='1') then
      if(epsilon >"00000000000001111111111111") then
         integ <= "00000000000001111111111111";
      else
         integ <= epsilon;
      end if;
   end if;
end process pid_clk3;
```

pid_clk4 sinyali değişiminde; $u[k]=\theta+\varepsilon$ işlemleri gerçekleşir. Bu işlem için hazırlanan akış diyagramı Şekil 5.41'da verilmiştir.



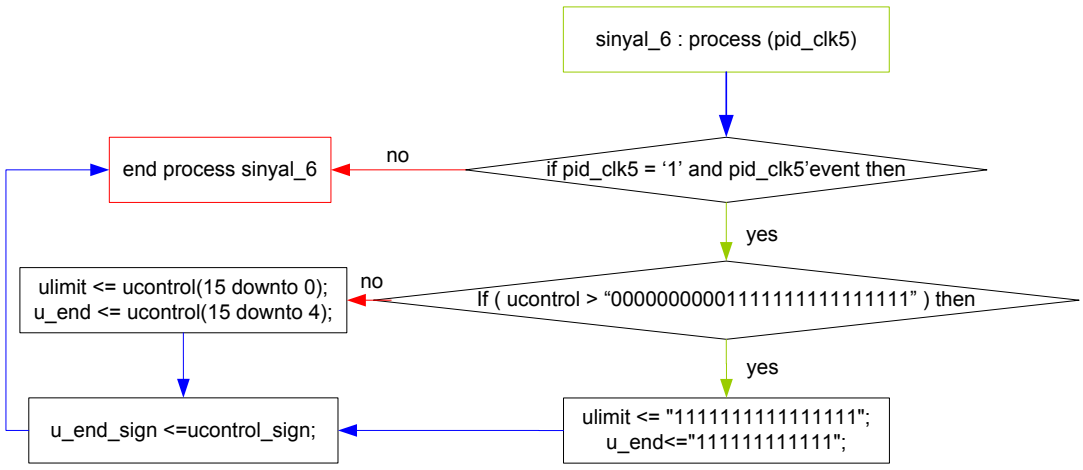
Şekil 5.41 pid_clk4 sinyal değişimi için akış diyagramı

Bu işlem sonunda kontrolcü çıkışı ($u[k]$) elde edilir ve bir sonraki aşamada u değeri maksimum limit değerinini aşmaması için sınırlandırılır. Bu işlem için hazırlanılan VHDL kodu aşağıda verilmiştir.

```
sinyal_5: process (pid_clk4)
begin
   if (pid_clk4'event and pid_clk4 ='1') then
      if(teta_sign = '0' and epsilon_sign = '0') then
         ucontrol <= ('0' & teta) + ('0' & epsilon);
         ucontrol _sign <= '0';
      elsif(teta_sign = '0' and epsilon _sign = '1') then
         if(epsilon >teta) then
            ucontrol <= ('0' & epsilon) - ('0' & teta);
            ucontrol _sign <= '1';
         else
```

```
            ucontrol <= ('0' & teta) - ('0' & epsilon);
            ucontrol _sign <= '0';
         end if;
      elsif(teta_sign = '1' and epsilon _sign = '0') then
         if(teta> epsilon) then
            ucontrol <= ('0' & teta) - ('0' & epsilon);
            ucontrol _sign <= '1';
         else
            ucontrol <= ('0' & epsilon) - ('0' & teta);
            ucontrol _sign <= '0';
         end if;
      else
         ucontrol <= ('0' & teta) + ('0' & epsilon);
         ucontrol _sign <= '1';
      end if;
   end if;
end process sinyal_5;
```

pid_clk5 sinyali değişiminde; kontrolcü çıkışı olan *ucontrol* sinyali ile ilgili sınırlamalar yaplır. İlgili akış diyagramı Şekil 5.42'de verilmiştir.



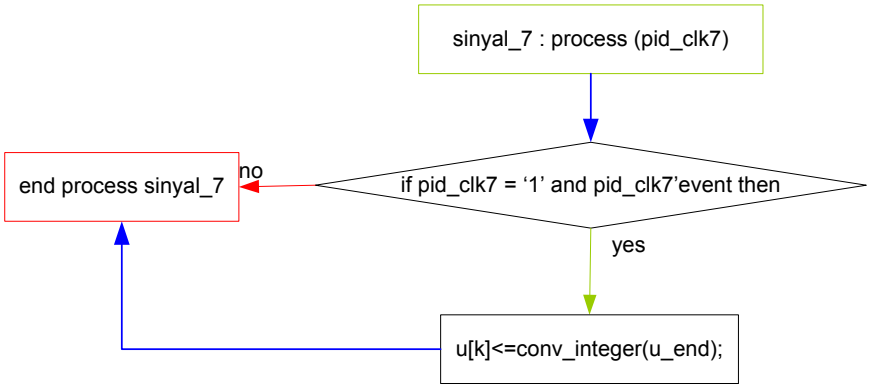
Şekil 5.42 pid_clk5 sinyal değişimi için akış diyagramı

Bu işlem için hazırlanılan VHDL kodu aşağıda verilmiştir.

```
sinyal_6: process (pid_clk5)
begin
   if (pid_clk5'event and pid_clk5 ='1') then
```

```
      if(ucontrol>"00000000001111111111111111") then
         ulimit <= "1111111111111111";
         u_end<="111111111111";
      else
         ulimit <= ucontrol(15 downto 0);
         u_end <= ucontrol(15 downto 4);
      end if;
      u_end_sign <= ucontrol _sign;
   end if;
end process sinyal_6;
```

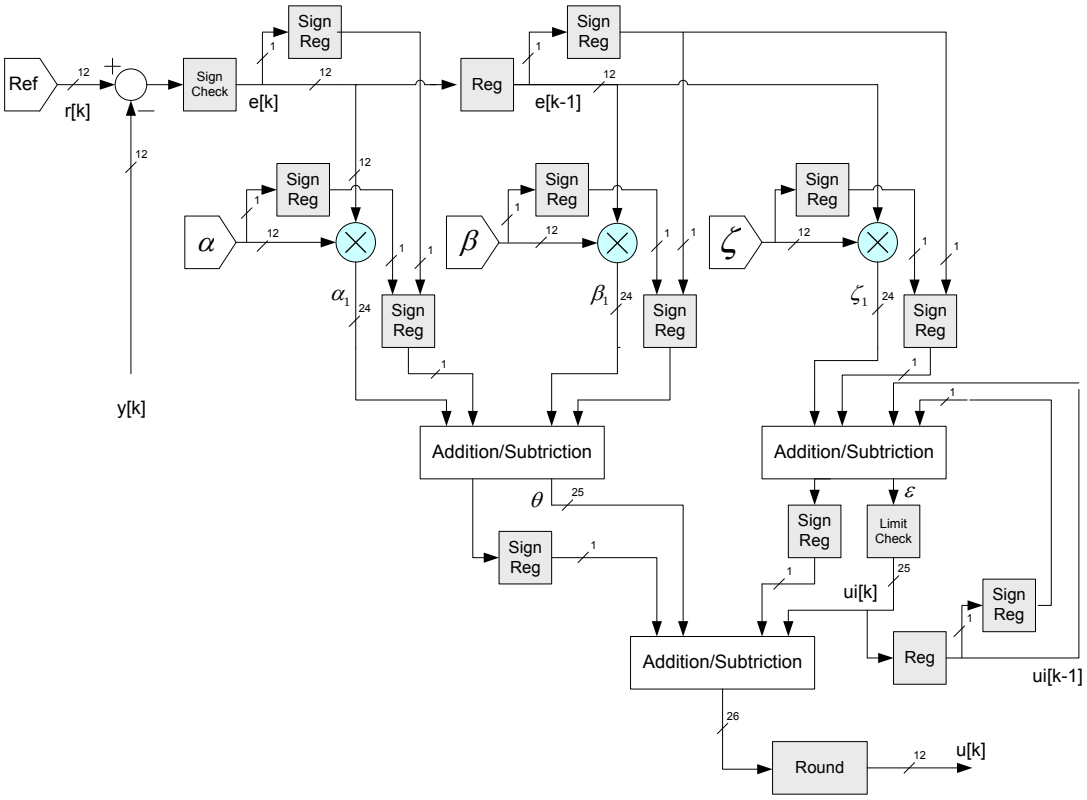
pid_clk6 sinyali değişiminde; kontrol sinyali (u_end) std_logic_vector veri tipinden integer veri tipine çevirilir. Böylece bir çevrimlik pid algoritması tamamlanmış olur. Bu işlemin devamında kontrol sinyali PWM algoritmasının olduğu bölüme aktarılarak uygun PWM değeri üretilip motor sürücülere gönderilir. Bu işlem için hazırlanan akış şeması Şekil 5.42'de verilmiştir.



Şekil 5.43 pid_clk6 sinyal değişimi için akış diyagramı

```
sinyal_7 : process (pid_clk6)
begin
   if (pid_clk6'event and pid_clk6 ='1') then
      u[k]<=conv_integer(u_end);
   end if;
end process sinyal_7;
```

FPGA'in içinde gerçekleşen dijital PID için toplam blok diyagramı aşağıda ifade edilmiştir.



Şekil 5.44 FPGA tabanlı PID kontrolcü blok diyagramı

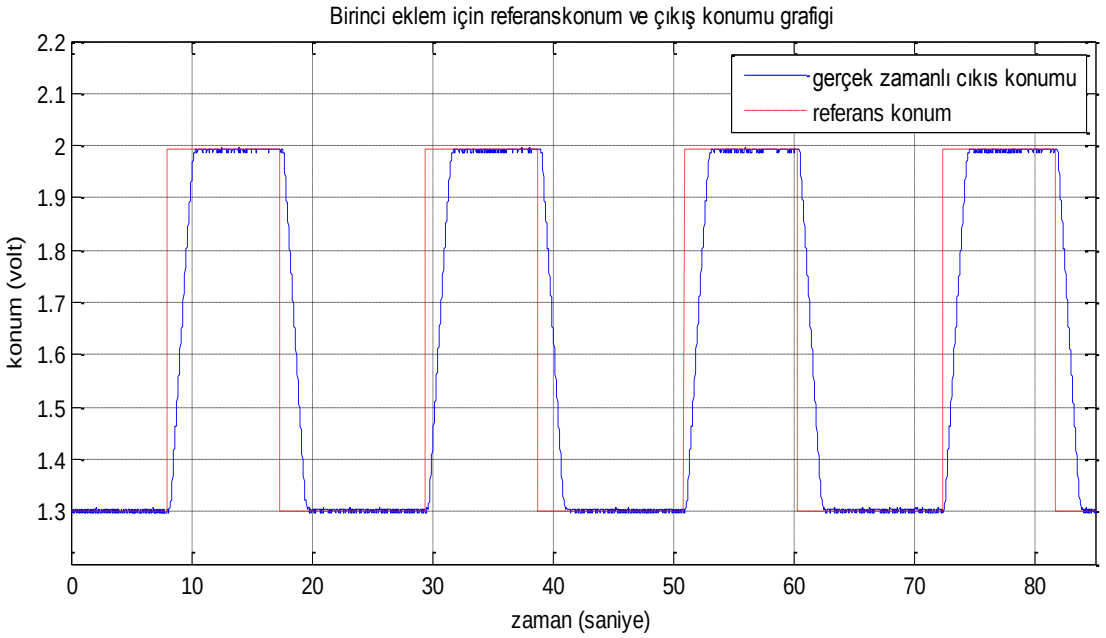
BÖLÜM 6

SONUÇ VE ÖNERİLER

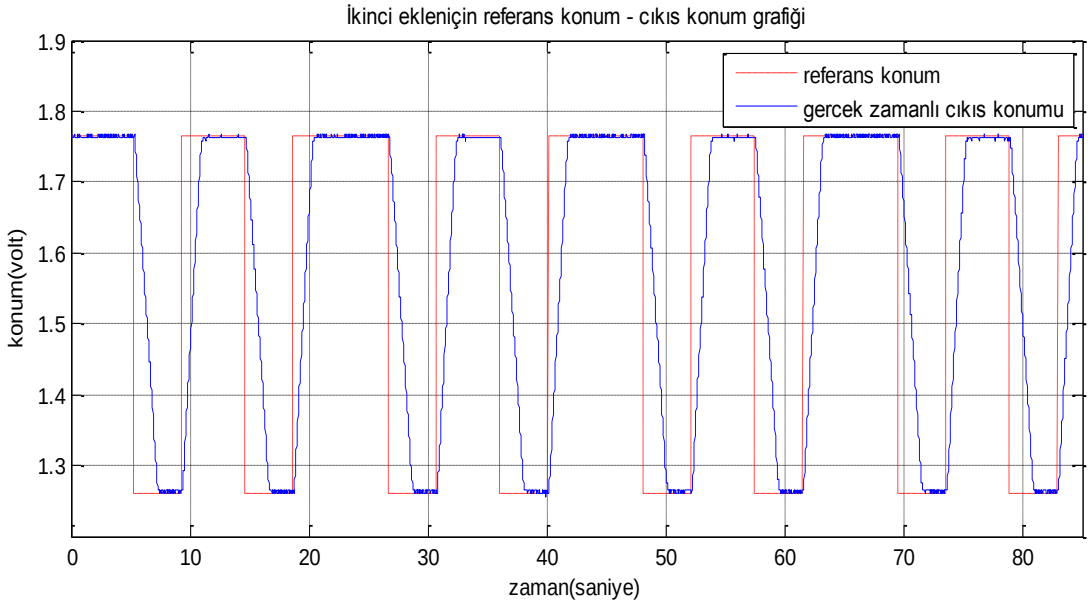
Bölüm 5.6'da anlatılan PID algoritması beş eklemli robot kolunun ilk dört eklemi için aynı anda hesaplanmaktadır. Son eklem olan tutucu ise robot istenilen noktalara geldiği zaman yeterli PWM değeri gönderilerek açılıp kapanma işlemini gerçekleştirmektedir. Uygulamada robot kolun beş eklemi için, çalışma uzayı içersinde belirli koordinatlar hesaplanarak uygun bir yörünge belirlenmiş ve robotun bu yörüngeyi takip edip edemediği gözlemlenmiştir. Referans değerler algoritma içerinde yazılmıştır. Bölüm 5.3'de elde edilen kontrolcü katsayıları kullanılmıştır.

Robot kolunun çıkışı Quanser Q2 veri toplama modülü ile gerçek zamanlı olarak Matlab platformuna aktarılmış ve çıkışın yörüngesi elde edilmiştir. Bu fonksiyon, referans fonksiyonu ile karşılaştırıldığı zaman robot kolunun istenilen referansı uygun bir şekilde takip ettiği gözlemlenmiştir.

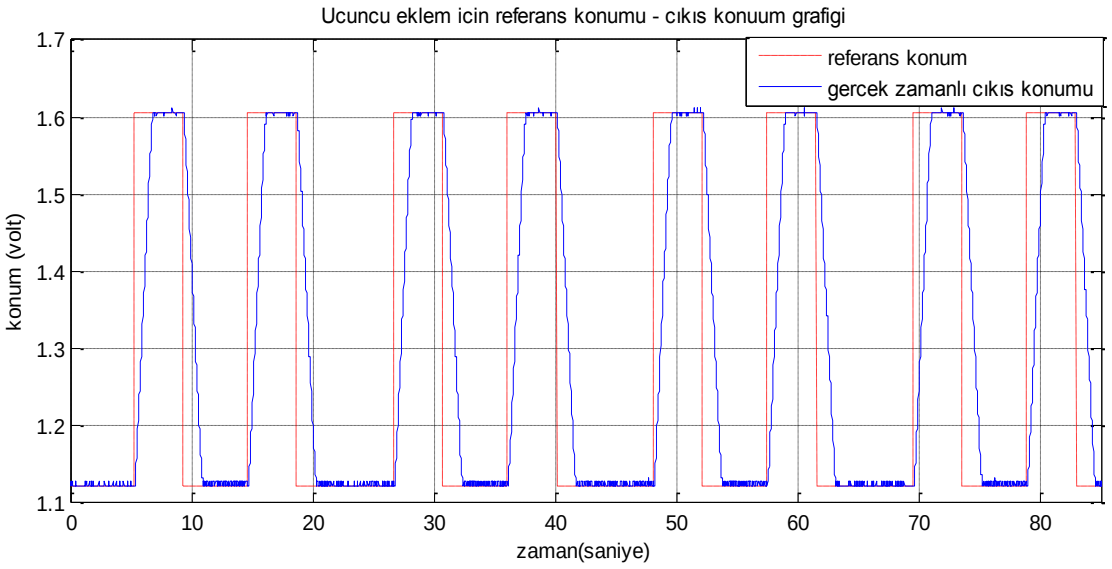
Aşağıda her bir eksenin referans konumunun ve çıkış konumunun grafikleri verilmiştir. Grafiklerde zaman ekseni birimi saniyedir. Konum ekseni ise motorlara verilen PWM değerleri karşılığında potansiyometrelerin dönme miktarları sonucunda üzerlerine düşen gerilim miktarıdır. Potansiyometreler üzerine düşen gerilim değeri 0 - 3.3 Volt aralığındadır. Quanser veri toplama modülünün analog girişleri potansiyometrelerin orta bacaklarına bağlanmıştır ve USB kablo yardımı ile Matlab simulink ile hazırlanan bloklara veri aktarımı gerçekleştirilmiştir. Grafiklerden kesik çizgi ile belirtilenler referans yörüngeyi ifade ederken düz çizgili grafikler çıkışı gösterir.



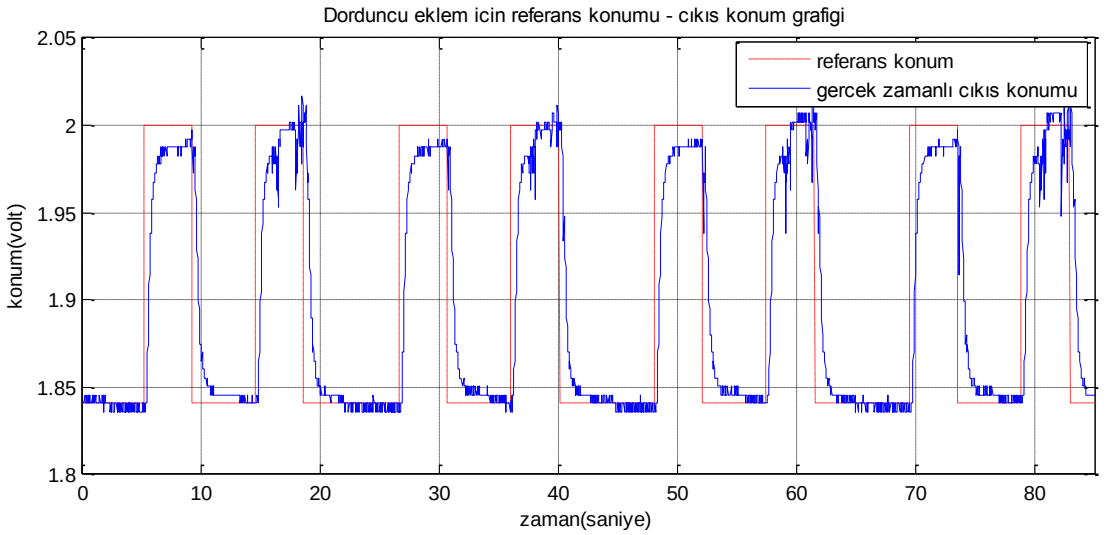
Şekil 6.1 Birinci eklem için referans konumu ve çıkış konumu grafiği



Şekil 6.2 İkinci eklem için referans konumu ve çıkış konumu grafiği

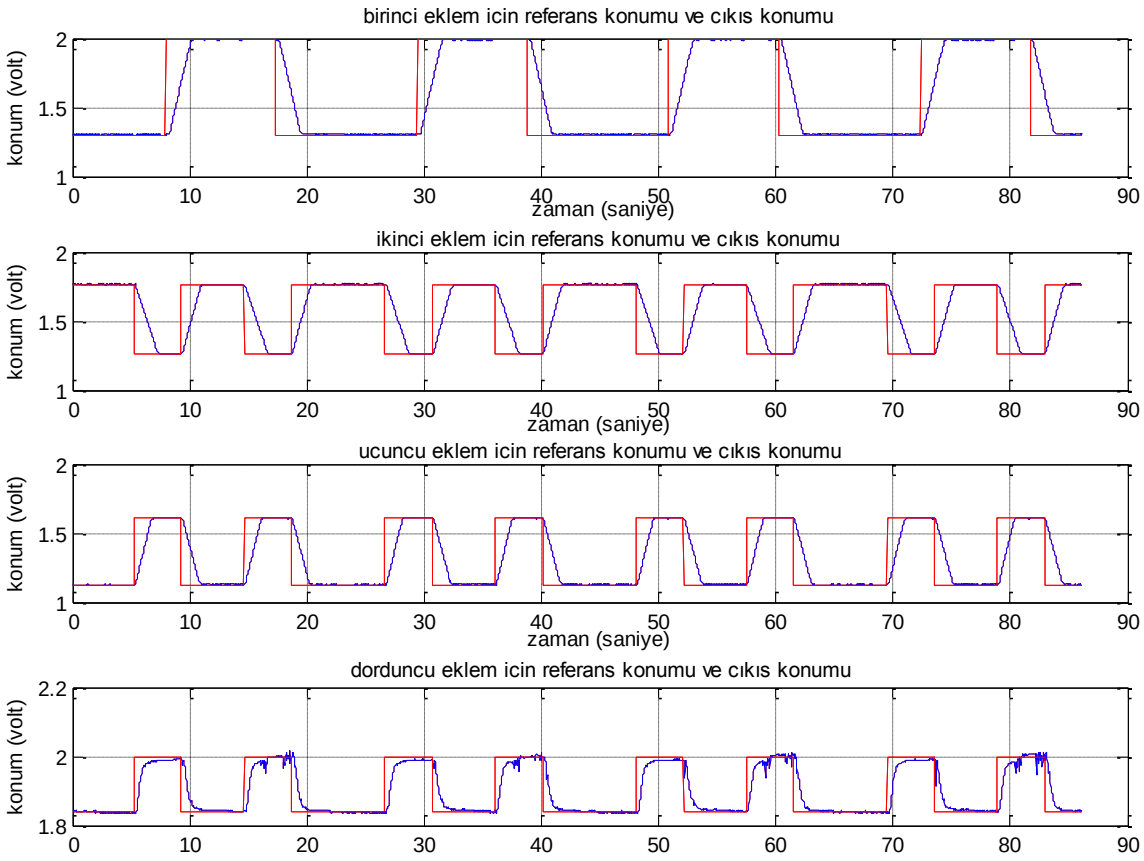


Şekil 6.3 Üçüncü eklem için referans konumu ve çıkış konumu grafiği



Şekil 6.4 Dördüncü eklem için referans konumu ve çıkış konumu grafiği

Sekil 6.5’de bütün grafikler alt alta getirilerek yeniden gösterilmiştir



Şekil 6.5 Tüm eksenler için konum zaman grafikleri

Sonuç olarak beş eksenli robot kolun eklemlerinin yaklaşık modellemeleri gerçekleştirilmiş, bu modellere dayanarak Matlab SisoTool yardımıyla PI katsayıları hesaplanmış ve FPGA ile beş eklemli robot kolunun kontrolü başarılı bir şekilde gerçekleştirilmiştir. Hareket kontrol sistemlerinde en çok karşılaşılan konularla ilgili VHDL algoritmalarının nasıl hazırlanılacağına deyinilmiş ve deney düzeneği elemanlar detaylı olarak anlatılmıştır. Bunlara ek olarak bu çalışmada FPGA’in gelişimi ve VHDL programının temel yapısı anlatılarak kaynak niteliğinde bir çalışma hazırlanmıştır.

KAYNAKLAR

[1] Dick, C. ve Harris, F., (2000). "FPGA Signal Processing Using Sigma-Delta Modulation", IEEE Signal Processing Magazine, 17(1).

[2] Goslin, G. R., (1996). "Guide to Using FPGAs for Application-Specific Digital Signal Processing Performance", Photonics East'96, .

[3] Petersen, R. J. ve Hutchings, B. L., (1995)."An Assessment of the Suitability of FPGA-Based Systems for use in Digital Signal Processing", Field-Programmable Logic and Applications, Springer .

[4] Uzun, I. S., Amira, A. ve Bouridane, A., (2005). "FPGA Implementations of Fast Fourier Transforms for Real-Time Signal and Image Processing", IEEE Vision Image and Signal Processing, 152(3).

[5] Batle, J., Marti, J., Ridao, P. ve Amad, J., (2002). "A New FPGA/DSP-Based Parallel Architecture for Real-Time Image Processing", Elsevier Science Ltd, Real-Time Imaging, 345–356.

[6] Johnston, C. T., Gribbon, K. T. ve Bailey, D. G., (2004, November). "Implementing Image Processing Algorithms on FPGAs", In Proceedings of the Eleventh Electronics New Zealand Conference, ENZCon'04, 118-123.

[7] Melnikoff, S. J., Quigley, S. F. ve Russell, M. J., (2002). "Implementing a Simple Continuous Speech Recognition System on an FPGA", In Field-Programmable Custom Computing Machines, 2002. Proceedings 10th Annual IEEE Symposium, 275-276.

[8] Melnikoff, S. J. ve Russell, M. J., (2002). "Speech Recognition on an FPGA Using Discrete and Continuous Hidden Markov models", In Field-Programmable Logic and Applications, Springer Berlin Heidelberg, 202-211.

[9] Vargas, F. L., Fagundes, R. D. R. ve Junior, D. B., (2001). "A FPGA-Based Viterbi Algorithm Implementation for Speech Recognition Systems", In Acoustics, Speech, and Signal Processing, 2001. Proceedings, IEEE International Conference, 2: 1217-1220.

[10] Coric, S., Leeser, M., Miller, E. ve Trepanier, M. (2002). "Parallel-Beam Backprojection: An FPGA Implementation Optimized for Medical Imaging", In

Proceedings of the 2002 ACM/SIGDA tenth International Symposium on Field-programmable Gate Arrays, 217-226.

[11] Yokota, T., Nagafuchi, M., Mekada, Y., Yoshinaga, T., Ootsu, K. ve Baba, T., (2002). “A Scalable FPGA-based Custom Computing Machine for a Medical Image Processing”, In Field-Programmable Custom Computing Machines, Proceedings 10th Annual IEEE Symposium, 307-308.

[12] Coric, S., Leeser, M., Miller, E. ve Trepanier, M., (2002). “Parallel-Beam Backprojection: an FPGA Implementation Optimized for Medical Imaging”, In Proceedings, ACM/SIGDA tenth International Symposium on Field-Programmable Gate Arrays, 217-226.

[13] Andraka, R., (1998). “A Survey of CORDIC Slgorithms for FPGA Based Computers”, In Proceedings of the ACM/SIGDA sixth International Symposium on Field Programmable Gate Arrays, 191-200.

[14] Tiri, K. ve Verbauwhede, I., (2004). “A logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation”, In Proceedings of the Vonference on Design, Automation and Test in Europe-Volume 1 (p. 10246), IEEE Computer Society.

[15] Brown, S. ve Rose, J., (1996). “FPGA and CPLD architectures: A tutorial: Design ve Test of Computers”, IEEE, 13(2): 42-57.

[16] Gokhale, M., Stone, J., Arnold, J. ve Kalinowski, M. (2000). “Stream-Oriented FPGA Computing in the Streams-C High Level Language”, In Field-Programmable Custom Computing Machines, 2000 IEEE Symposium, 49-56.

[17] Cesur, E. ,(2013). Gerçek Zamanlı bir Hücresel Sinir Ağı Yapısının Tasarımı ve bu Yapıyla Gabor Filtrelerinin FPGA Üzerinde Gerçeklenmesi, Doktora Tezi, Yıldız Teknik Universitesi, Fen Bilimleri Enstitüsü, İstanbul Turkiye.

[18] Yılmaz, N., (2013). Design of a Cellular Neural Network Emulator and Its Iimplementation on an FPGA Device, Doktora Tezi, Yıldız Teknik Universitesi, Fen Bilimleri Enstitüsü, İstanbul Turkiye.

[19] Peker, Z., (2013). FPGA ile Veri Gizleme Uygulamaları, Uludağ Üniversitesi, Fen Bilimleri Enstitüsü, Bursa, Türkiye.

[20] Toker, K. A., (2012). Değişken Hızlı Rüzgâr Enerji Çevrim Sisteminin Yenilikçi FPGA Kontrol Uygulması, Doktora Tezi, Ege Üniversitesi, Fen Bilimleri Enstitüsü, İzmir, Türkiye.

[21] Isakova, N., (2012). Gerçek Zamanlı Stereo Görme Sisteminin FPGA'de Tasarımı ve Uygulaması, Yüksek Lisans Tezi, Yıldız Teknik Universitesi, Fen Bilimleri Enstitüsü, İstanbul Turkiye.

[22] Özçelik M. F., (2012). Görüntü İşleme Algoritmalarının FPGA Üzerinde Gerçeklenmesi, Yüksek Lisans Tezi, Gazi Universitesi, Bilişim Enstitüsü, Ankara Turkiye.

[23] Taş, T., (2010). Bulanık Petri Ağlarıyla bir Demiryolu Trafik Sisteminin Modellenmesi ve FPGA üzerinde Gerçeklenmesi, Yüksek Lisans Tezi, İstanbul Teknik Universitesi, Fen Bilimleri Enstitüsü, İstanbul Turkiye.

[24] Kayaer K., (2008). Gerçek Zamanlı Video İşleyen Yeni bir Hücresel Sinir Ağı Emülatörü Tasarımı, Doktora Tezi, Yıldız Teknik Universitesi, Fen Bilimleri Enstitüsü, İstanbul Turkiye.

[25] Yazkurt U., (2002.) Design and FPGA Implementation of an STM-1 Transceiver System Containing the Advanced Encryption Standard Algorithm, Yüksek Lisans Tezi, Boğaziçi Üniversitesi, Fen Bilimleri Enstitüsü, İstanbul Turkiye.

[26] Çayır T., (2010). FPGA üzerinde Görüntü İyileştirme Uygulamaları, Yüksek Lisans Tezi, Yıldız Teknik Universitesi, Fen Bilimleri Enstitüsü, İstanbul Turkiye.

[27] Yılmaz N., (2008). Alan Programlamalı Kapı Dizileri (FPGA) Üzerinde bir Yapay Sinir Ağları (YSA)'nın Tasarlanması ve Donanım Olarak Gerçekleştirilmesi, Yüksek Lisans tezi, Selçuk Üniversitesi, Fen Bilimleri Enstitüsü, Konya, Türkiye.

[28] Alaer, E., (2006). 16-bitlik Mikroişlemcinin FPGA Mimarileri Kullanılarak Gerçekleştirilmsi, Yüksek Lisans Tezi, Kocaeli Üniversitesi, Fen Bilimleri Enstitüsü, Kocaeli, Türkiye.

[29] Öztürk, E., (2010). FPGA Kullanarak 16-bitlik Mikroişlemci Tasarımı, Yüksek Lisans Tezi, Yıldız Teknik Üniversitesi, Fen Bilimleri Enstitüsü, İstanbul Turkiye.

[30] Monmasson, E. ve Cirstea, M. N., (2007). “FPGA Design Methodology for Industrial Control Systems”, Industrial Electronics, IEEE Transactions, 54(4), 1824-1842.

[31] Piltan, F., Sulaiman, N., Marhaban, M. H., Nowzary, A. ve Tohidian, M., (2011). “Design of FPGA-based Sliding Mode Controller for Robot Manipulator”, International Journal of Robotics and Automation (IJRA), 173.

[32] Uygur S., (2012). FPGA Tabanlı Fırçasız Doğru Akım Motor Kontrol Sistemi, Yüksek Lisans Tezi, Orta Doğu Teknik Üniversitesi, Fen Bilimleri Enstitüsü, Ankara, Türkiye.

[33] Mamur A., (2012). Fpga Denetimli Düşürücü da-da Dönüştürücünün Tasarımı ve Gerçekleştirilmesi, Yüksek Lisans Tezi, Fırat Üniversitesi, Fen Bilimleri Enstitüsü, Elazığ, Türkiye.

[34] Meshram, Urmila D. ve Harkare R., (2010). "FPGA Based Five Axis Robot Arm Controller", International Journal of Electronics Engineering 2(1): 209-211.

[35] Mansoor, A.Z., Khalil, M. R. ve Jasim, O. A., (2011). “Position Control of DC Servo Motors Using Soft-core Processor on FPGA to Move Robot Arm”, Journal of Theoretical and Applied Information Technology, 32(1).

[36] Xu, M., Zhu, W. ve Zou, Y., (2008, October). “Design of a Reconfigurable Robot Controller Based on Fpga”, In Embedded Computing, 2008, SEC'08. Fifth IEEE International Symposium, 216-222.

[37] Sugahara, K., Oida, S. ve Yokoyama, T., (2009, May). “High Performance FPGA Controller for Digital Control of Power Electronics Applications”, In Power

Electronics and Motion Control Conference, IPEMC'09. IEEE 6th Internationa,l 1425-1429.

[38] Meshram, U., Bande, P., Dwaramwar, P. A. ve Harkare, R. R., (2009, March). "Robot Arm Controller Using FPGA", Signal Processing and Communication Technologies, 2009. IMPACT'09. International, 8-11.

[39] Tsai, C., Tai, F. ve Hsies S., (2010). "Unified Motion Controller Design and FPGA-Based Implementation for Nonholonomic Mobile Robots", SICE Annual Conference, Taipei, Taiwan.

[40] Siddiqui, M. S. M., Sajid, A. H. ve Chougule, D. G., (2009). "FPGA Based Efficient Implementation of PID Control Algorithm", International Conference of Control, Automation, Communucation and Energy Conservation, 4(6).

[41] Mutlu, B. R., Yaman, U., Dölen, M. ve Koku, A. B. "Kayan Kipli DC Motor Konum Denetiminin FPGA ile Gerçekleştirilmesi".

[42] Abdelati, M. (2005). FPGA-Based PID Controller Implementation, Master Thesis, The Islamic University Of Gaza, Electrical Engineering Department,

[43] Sarıtaş, E. ve Karataş, S., (2013). Her yönüyle FPGA ve VHDL, Palme Yayıncılık, Ankara.

[44] Ekiz, H., (2005). Mantık devreleri, Değişim Yayınevi.

[45] Pedroi, V. A., (2004). Circuit Design with VHDL, MIT Press, Cambridge, Massachusetts.

[46] Asheden, P. J. ve Lewis, J., The Designer's Guide to VHDL, Academic Press, USA

[47] Pedroi, V. A., (2008). Digital Electronics and Design with VHDL, Elsevier, Morgan KAufmann Publishers.

[49] Jazar, R.N., (2007). Teory of Applied Robotics: Kinematics, Dynamics and Control, Springer.

[50] Spong, M. W., Hutchingson, S. Ve Vidyasagar, M. Robot Modeling and Control, John Wiley ve Sons, Inc.

[51] Nise, N. S., (2011). Control System Engineering, John Wiley ve Sons, Inc, Pomona.

[52] Görgün, H. "Kontrol Sistem Tasarımı Ders Notları, Frekans Analiz Yöntemleri, Bode Eğrileri".

[53] altera.com, DE0 Nano User Manual, Altera Copp. http://www.altera.com/education/univ/materials/boards/de0-nano/unv-de0-nano-board.html, 10/09/2013.

[54] Ogata, S. K., (1995). Discreate-Time Control Systems, Prentece-Hall International, New Jersey.

[55] Chin, H. H., (2006). All Digital Design and Implementation of PID Controllers, Lexington, Kentuck.

[56] mathworks.com, Siso Design Tool http://www.mathworks.com/help/control/getstart/siso-design-tool.html, 09/2013.

[57] Ang K.H, Chong G., (2005). "PID Control System Analysis, Design and Technology", IEEE Transactions on Control Systems Technology, 13(4).

[58] Altera Univercity Program, (2012). Using the DE0-Nano ADC Controller,

[59] ADC128S022 8-Channel, 50 kSPS to 200 kSPS, 12-Bit A/D Converter Datasheet, Texas Instruments, 2013 (revised).

EK A

Quartsus II programlama platformu kullanılarak VHDL dili ile yazılan, beş eksenli robot kolunun pid kontrol algoritması program kodları teze eklenmemiştir. İhtiyaç duyulduğu takdirde srnakkaya@gmail.com adresinden istenilebilir.

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı : Şirin AKKAYA

Doğum Tarihi ve Yeri : 03.05.1988-Keşap/Giresun

Yabancı Dili : İngilizce

E-posta : srnakkaya@gmail.com

ÖĞRENİM DURUMU

Derece	Alan	Okul/Üniversite	Mezuniyet Yılı
Y. Lisans	Kont. Ve Oto. Müh.	Yıldız Teknik Üniversitesi	--
Y. Lisans	Mekatronik Müh.	İstanbul Teknik Üniversitesi	--
Lisans	Mekatronik. Müh.	Kocaeli Üniversitesi	2010
Lise	Fen Bilimleri	Giresun Lisesi (YDA)	2005

İŞ TECRÜBESİ

Yıl	Firma/Kurum	Görevi
2013 -	İstanbul Teknik Üniversitesi	Araştırma Görevlisi
2011-2013	Konya N. Erbakan Üniversitesi	Araştırma Görevlisi