A. HARMANDA, 2011

YÜKSEK LİSANS TEZİ

NİĞDE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

T.C.
NİĞDE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

16 BİTLİK BİR PIC MİKRODENETLEYİCİSİ İLE BİR PROGRAMLANABİLİR LOJİK DENETLEYİCİ TASARIMI VE UYGULAMASI

ARİF HARMANDA

Ekim 2011

T.C.
NİĞDE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

16 BİTLİK BİR PIC MİKRODENETLEYİCİSİ İLE BİR PROGRAMLANABİLİR LOJİK DENETLEYİCİ TASARIMI VE UYGULAMASI

ARİF HARMANDA

Yüksek Lisans Tezi

Danışman

Prof. Dr. Murat UZAM

Ekim 2011

Arif HARMANDA tarafından **Prof. Dr. Murat UZAM** danışmanlığında hazırlanan "**16 Bitlik bir PIC Mikrodenetleyicisi ile bir Programlanabilir Lojik Denetleyici Tasarımı ve Uygulaması**" adlı bu çalışma jürimiz tarafından Niğde Üniversitesi Fen Bilimleri Enstitüsü **Elektrik-Elektronik Mühendisliği** Anabilim Dalında Yüksek Lisans tezi olarak kabul edilmiştir.

Başkan : **Prof. Dr. Kenan DANIŞMAN**

Erciyes Üniversitesi Müh. Fak. Elektrik-Elektronik Müh. Böl.

Üye : **Prof. Dr. Murat UZAM**

Melikşah Üniversitesi Müh. Mim. Fak. Elektrik-Elektronik Müh. Böl.

Üye : **Doç. Dr. Saffet AYASUN**

Niğde Üniversitesi Müh. Fak. Elektrik-Elektronik Müh. Böl.

ONAY:

Bu tez, Fen Bilimleri Enstitüsü Yönetim Kurulunca belirlenmiş olan yukarıdaki jüri üyeleri tarafından …./…./20.... tarihinde uygun görülmüş ve Enstitü Yönetim Kurulu'nun …./…./20.... tarih ve ..... sayılı kararıyla kabul edilmiştir.

..... /..... / 2011

Doç. Dr. Nurettin ACIR

MÜDÜR

ÖZET

16 BİTLİK BİR PIC MİKRODENETLEYİCİSİ İLE BİR PROGRAMLANABİLİR LOJİK DENETLEYİCİ TASARIMI VE UYGULAMASI

HARMANDA, Arif

Niğde Üniversitesi

Fen Bilimleri Enstitüsü

Elektrik-Elektronik Mühendisliği Anabilim Dalı

Danışman: Prof. Dr. Murat UZAM

Ekim 2011, 143 sayfa

Programlanabilir lojik denetleyiciler (PLC) günümüz endüstriyel sistemlerinde en çok kullanılan otomasyon cihazlarıdır. Bu tezin amacı 16 bitlik bir PIC mikrodenetleyicisi yardımıyla endüstride yaygın olarak kullanılan özelliklere sahip bir PLC tasarlanması ve gerçekleştirilmesidir.

Gerçekleştirilen PLC donanımında merkezi işlem birimi olarak PIC24FJ128GA010 mikrodenetleyicisi ve 16 ayrık giriş/16 ayrık çıkış bulunmaktadır. Ayrık girişler 5V DC veya 24V DC olarak uygulanabilmektedir. Ayrık çıkışlar olarak röle kontakları kullanılmıştır. PLC yazılımı PIC C dili ile MPLAB IDE yazılım geliştirme ortamı yardımıyla geliştirilmiştir.

Anahtar sözcükler: PLC, PIC, Mikrodenetleyici, Endüstriyel Otomasyon, MPLAB IDE

SUMMARY

DESIGN AND APPLICATION OF A PROGRAMMABLE LOGIC CONTROLLER WITH A 16 BIT PIC MICROCONTROLLER

HARMANDA,Arif

Nigde University

Graduate School of Natural and Applied Sciences

Department of Electrical and Electronics Engineering

Supervisor: Assoc. Prof. Dr. Murat UZAM

October 2011, 143 pages

Programmable logic controllers (PLC) are the most prefered devices in modern industrial automation systems. The aim of this thesis is to design and implement a 16 bit microcontroller based PLC with general features which are common in PLCs used in industry.

The PLC developped has a PIC24FJ128GA010 microcontroller as a central processing unit, 16 discrete inputs and 16 discrete outputs. 5V DC and 24V DC signals can be applied as discrete inputs. Discrete outputs are obtained by means of relay contacts. The PLC software is developped by means of PIC C language within MPLAB IDE software development environment.

Keywords: PLC, PIC, Microcontroller, Industrial Automation, MPLAB IDE

ÖNSÖZ

Bu çalışma değerli danışmanım Prof. Dr. Murat UZAM ve Yrd. Doç. Dr. Gökhan GELEN'in katkılarıyla gerçekleştirilmiştir. Endüstriyel otomasyonla ilgili bir çalışmadır. Ülkemizde ve dünyada endüstriyel otomasyon sektöründe PLC'lerin çok önemli bir yer teşkil ettiği bilinmektedir. Günümüz rekabet dünyasında yabancı kökenli PLC markaları piyasada önemli bir yer tutmaktadır. Ekonomik olarak çok avantajlı olmayan bu markalar, uzun yıllar otomasyon sektöründe öncelikli tercihler olmuştur. Ancak şimdilerde piyasaya yerli tasarım PLC'lerin de girmesiyle rekabet artmış ve bu markaların pazar payı giderek daralmaya başlamıştır. Aynı özellikleri sağlayan ve daha ucuz olan markalar ciddi pazar oluşturmaya başlamıştır. Bu durum yerli sermayenin yurt içinde kalması açısından da önemli bir gelişmedir.

Yapılan bu çalışma; piyasadaki benzerlerinden eksiği olmayacak şekilde, bir otomasyon sisteminin gerçekleştirilmesi için gerekli yazılım ve donanımı sağlama amacıyla tasarlanmıştır. Bu tez çalışmasının; gerek üniversitelerdeki gerekse endüstriyel AR-GE birimlerindeki çalışmalara ışık tutmasını ve ülkemiz sanayisine katkıda bulunmasını dilerim.

TEŞEKKÜR

Bu tez çalışmasının hazırlanmasında emeği geçen, maddi ve manevi yardımlarını esirgemeyen başta danışmanım Prof. Dr. Murat UZAM olmak üzere Yrd. Doç. Dr. Gökhan GELEN'e sonsuz teşekkürlerimi sunarım. Bilgi ve tecrübelerinden faydalandığım araştırma görevlisi ve proje asistanı arkadaşlarıma da teşekkür eder çalışmalarında başarılar dilerim. Ayrıca maddi ve manevi desteklerini esirgemeyen aileme teşekkürü bir borç bilirim.

İÇİNDEKİLER

ÖZET ……II

SUMMARY ……III

ÖNSÖZ ……...IV

TEŞEKKÜR ……V

İÇİNDEKİLER DİZİNİ ……..VI

ÇİZELGELER DİZİNİ ……X

ŞEKİLLER DİZİNİ ……XI

RESİM VE FOTOĞRAFLAR DİZİNİ ……XIV

BÖLÜM I. GİRİŞ …… 1

BÖLÜM II. TEKNOLOJİ VE OTOMASYON ……...4

2.1 Teknolojinin Önemi ve Gelişimi ……4

2.2 Endüstriyel Otomasyon Kavramı ……...5

2.3 Programlanabilir Lojik Denetleyiciler (PLC) ……6

2.3.1 PLC'nin yapısı ……....8

2.3.2 PLC programlama ve Merdiven (Ladder) mantığı ……..10

2.3.3 Röle ile PLC'nin karşılaştırılması ……12

2.4 Mikrodenetleyiciler ve PIC ……13

2.4.1 Mikroişlemci ve mikrodenetleyicinin karşılaştırılması ……14

2.4.2 PIC mikrodenetleyicileri ……15

2.4.3 PIC mikrodenetleyicilerinin özellikleri ……16

BÖLÜM III. PIC24FJ128GA010 AİLESİNE GENEL BAKIŞ ……17

3.1 CPU Çekirdek Yapısı ……...18

3.2 CPU Özellikleri ……....20

3.2.1 Çarpıcı ……21

3.2.2 Bölücü ……22

3.2.3 Çoklu bit kaydırma ……22

3.2.4 Program adres alanı ……...22

3.2.5 Veri adres alanı ……..22

3.2.6 Resetler ……..23

3.2.7 Zamanlayıcılar (Timers) ……..24

3.2.7.1 Timer 1 ……24

3.2.7.2 Timer 2/3 ve Timer 4/5 ……...25

BÖLÜM IV. PIC24FJ128GA010 MİKRODENETLEYİCİSİ İLE TASARLANAN 16 GİRİŞLİ/16 ÇIKIŞLI PLC DONANIMI ……..26

4.1 Explorer 16 Geliştirme ve Uygulama Kartı ……..28

4.2 UZAM_PLC I/O Genişletme Kartı ……..31

BÖLÜM V. PIC24FJ128GA010 MİKRODENETLEYİCİSİ İLE TASARLANAN 16 GİRİŞLİ/16 ÇIKIŞLI PLC YAZILIMI ……34

5.1 Explorer 16 Programlama……..34

5.1.1 Proje Oluşturma ……34

5.1.2 Project Wizard……35

5.2 Kod Oluşturma ……39

5.2.1 C dili ile programlama ……39

5.2.2 C kodu temel özellikleri ……40

5.2.3 Veri tipleri ……..40

5.2.4 Aritmetik operatörler……..43

5.2.5 Atama operatörleri……..43

5.2.6 Mantıksal operatörler……..43

5.2.7 Karşılaştırma operatörleri……...44

5.2.8 ‘if’ ve ‘else if’ yapısı……...44

5.2.9 ‘while’ döngüsü ……45

5.2.10 ‘for’ döngüsü ……45

5.2.11 Sonsuz döngü……46

5.3 Gerçekleştirilen PLC’nin Temel Yazılımı ……46

5.3.1 Temel yazılım ve tanımlama fonksiyonları ……...47

5.3.2 Ana program ……..…55

5.4 Kontak ve Röle Temelli Fonksiyonlar…….……..56

5.4.1 Kontak ve röle temelli fonksiyon örnekleri ……...64

5.5 Karşılaştırma Fonksiyonları ……...…..67

5.5.1 Karşılaştırma fonksiyonları örnekleri…….……70

5.6 Aritmetik Fonksiyonları…….....72

5.6.1 Aritmetik fonksiyon örnekleri ……...75

5.7 Lojik Fonksiyonları ...……...77

5.7.1 Lojik fonksiyonları örnekleri ...……..81

5.8 Zamanlayıcı Fonksiyonları ……...84

5.8.1 Düz zaman rölesi (On delay timer - TON) ……85

5.8.2 Ters zaman rölesi (Off delay timer) ……..87

5.8.3 Zamanlayıcı fonksiyon örnekleri ……...89

5.9 Sayıcı Fonksiyonları ……90

5.9.1 İleri sayıcı (up counter – CTU) ……..90

5.9.2 İleri/geri sayıcı (up/down counter – CTUD) ……..92

5.9.3 İleri/geri sayıcı (up/down counter – CTUD) ……..94

5.9.4 Sayıcı fonksiyonları örnekleri ……97

BÖLÜM VI. SONUÇ ……99

KAYNAKLAR ……100

EKLER ……103

ÇİZELGELER DİZİNİ

Çizelge 5.1 Aritmetik operatörler ……...……43

Çizelge 5.2 Atama operatörleri ……...43

Çizelge 5.3 Mantıksal operatörler ……..43

Çizelge 5.4 Karşılaştırma operatörleri ……44

Çizelge 5.5 16 bitlik hafıza yapısı (union) ve tanımlamaları ……47

Çizelge 5.6 16 bitlik hafıza yapılarının bit tanımlamaları ……..48

Çizelge 5.7 Kontak ve röle temelli fonksiyonlar ……58

Çizelge 5.8 "**r_edge**" fonksiyonu, sembolleri ve zaman diyagramı ……..60

Çizelge 5.9 "**f_edge**" fonksiyonu, sembolleri ve zaman diyagramı ……...61

Çizelge 5.10 "**latch**" fonksiyonu, sembolleri ve doğruluk tablosu ……..62

Çizelge 5.11 "**sr_ff**" fonksiyonu, sembolleri ve doğruluk tablosu ……...…..63

Çizelge 5.12 Karşılaştırma Fonksiyonları ……68

Çizelge 5.13 Aritmetik fonksiyonları …….……...……...73

Çizelge 5.14 Lojik fonksiyonları ……..78

Çizelge 5.15 Gerçekleştrilen düz zaman rölesinin (TON) sembolü ve C kodu ...…....86

Çizelge 5.16 Gerçekleştirilen ters zaman rölesinin (TOF) sembolü ve C kodu …..….88

Çizelge 5.17 İleri sayıcı doğruluk tablosu ……91

Çizelge 5.18 Gerçekleştirilen ileri sayıcıya (upcounter - CTU) ait sembol ve C Kodu ……...……91

Çizelge 5.19 Geri sayıcı (CTD) doğruluk tablosu ……93

Çizelge 5.20 Gerçekleştirilen geri sayıcının (downcounter - CTD) sembolü ve C Kodu ...……93

Çizelge 5.21 İleri/geri sayıcı (CTUD) doğruluk tablosu ……..95

Çizelge 5.22 Gerçekleştirilen ileri/geri sayıcıya (CTUD) ait sembol ve C kodu…….96

ŞEKİLLER DİZİNİ

Şekil 2.1 PLC genel blok şeması……...7

Şekil 2.2 PLC iç yapısı …….8

Şekil 2.3 PLC çalışma diyagramı …….9

Şekil 2.4 Merdiven (Ladder) diyagramı ……10

Şekil 2.5 Sıralı Fonsiyon Şeması ……11

Şekil 2.6 Mnemonic komut listesi yöntemi ……11

Şekil 2.7 Mikroişlemci ve mikrodenetleyicinin karşılaştırılması ……...14

Şekil 3.1 PIC24FJ128GA010 mikrodenetleyicisi pin diyagramı ……..17

Şekil 3.2 CPU çekirdeği blok diyagramı ……18

Şekil 3.3 PIC24FJ128GA010 ailesi genel blok diyagramı ……19

Şekil 3.4 Reset modülü blok diyagramı …….23

Şekil 3.5 Timer1 blok diyagramı ……24

Şekil 3.6 Timer2/3 ve timer 4/5 32 bit blok diyagramı ……..25

Şekil 4.1 Explorer 16 geliştirme ve uygulama kartı şeması……29

Şekil 4.2 Explorer 16 geliştirme kartı blok diyagramı ……...30

Şekil 4.3 Yardımcı kart bağlantı pinleri …….32

Şekil 4.4 UZAM_PLC I/O genişletme kartına ait devre şeması ……33

Şekil 5.1 Cihaz seçme adımı ……..35

Şekil 5.2 Dil ve derleyici seçenekleri adımı ……...36

Şekil 5.3 Proje dosyası oluşturma ve isim verme adımı ……36

Şekil 5.4 Kaynak (source-.c) ve başlık (header-.h) dosyaları ekleme adımı ……..37

Şekil 5.5 MPLAB Project Window ……38

Şekil 5.6 Durum bitleri ve tanımlamaları ……...49

Şekil 5.7 Kontrol sinyalleri tanımlamaları …….50

Şekil 5.8 “get_inputs.c” program kodu …….51

Şekil 5.9 send_outputs.c program kodu …….54

Şekil 5.10 main (void) programı ……..55

Şekil 5.11 Kontak ve röle temelli fonksiyon örnekleri için şematik diyagram ……64

Şekil 5.12 Kontak ve röle temelli fonksiyon örnekleri için merdiven diyagramı …65

Şekil 5.13 Kontak ve röle temelli fonksiyon örnekleri kullanıcı programı …….….66

Şekil 5.14 Veri karşılaştırma işleminin genel şekli …….….67

Şekil 5.15 Karşılaştırma fonksiyonları örneği merdiven diyagramı ……70

Şekil 5.16 Karşılaştırma fonksiyonları örneği şematik diyagramı ……...71

Şekil 5.17 Karşılaştırma fonksiyonları örneği kullanıcı programı ……..71

Şekil 5.18 Bir aritmetik toplama işlemi şeması ……72

Şekil 5.19 Aritmetik fonksiyon örnekleri merdiven diyagramı ……...75

Şekil 5.20 Aritmetik fonksiyonları örnekleri şematik diyagramı ……76

Şekil 5.21 Aritmetik fonksiyon örnekleri kullanıcı programı …….76

Şekil 5.22 Bir lojik “AND” fonksiyonu şeması …….……..77

Şekil 5.23 Lojik fonksiyon örneklerine ait merdiven diyagramı …….81

Şekil 5.24 Lojik fonksiyon örnekleri için şematik diyagram ……..….82

Şekil 5.25 Lojik fonksiyon örneklerine ait kullanıcı programı ……83

Şekil 5.26 Referans zamanlama sinyalleri ve zaman değerleri ……84

Şekil 5.27 Düz zaman rölesinin (TON) sembolü ve zamanlama diyagramı ...…….85

Şekil 5.28 Ters zaman rölesinin (TOF) sembolü ve zamanlama diyagramı ……....87

Şekil 5.29 Zamanlayıcı fonksiyon örnekleri ……89

Şekil 5.30 İleri sayıcının sembolü ……90

Şekil 5.31 Geri sayıcının (CTD) sembolü ……92

Şekil 5.32 İleri/geri sayıcının (CTUD) sembolü ……..94

Şekil 5.33 Sayıcı fonksiyonları örnekleri ……97

RESİM VE FOTOĞRAFLAR DİZİNİ

Resim 2.1 Çeşitli transistörlerden örnekler ……4

Resim 2.2 Bir otomasyon sisteminden kesit …….5

Resim 2.3 Siemens S7 300 PLC …….6

Resim 2.4 Çeşitli PIC' lerden örnekler ……15

Fotoğraf 4.1 PIC24FJ128GA010 mikrodenetleyicisi ile tasarlanan ve gerçekleştirilen 16girişli/16çıkışlı PLC ……..26

Fotoğraf 4.2 Fotograf 4.2 Explorer 16 geliştirme ve uygulama kartı …….28

Fotoğraf 4.3 UZAM_PLC I/O genişletme kartı ……...31

BÖLÜM I

GİRİŞ

İçinde bulunduğumuz çağda birçok alanda çok hızlı bilimsel ve teknolojik gelişmeler yaşanmaktadır. Şüphe yok ki bu gelişmelerin temelinde bilgisayar teknolojisi ve bu alandaki hızlı gelişmeler yatmaktadır. Ancak tüm bu gelişmeler insan hayatını kolaylaştırdığı ve insanların ihtiyaçlarını kolay ve hızlı bir şekilde giderebildiği zaman esas hedefine ulaşmış olur.

İnsan hayatını kolaylaştırma, en iyi ürün veya hizmeti ekonomik bir şekilde sunma endüstri dünyasını otomatikleşme sürecine yönlendirmiştir. İnsanların daha iyiye ve kaliteye ulaşma arzuları sonucu yaptıkları çalışmalarla her geçen gün teknolojide ilerleme kaydedilmektedir. İlerleyen bu teknolojik çalışmalar sonucunda endüstride üretim maliyeti düşmekte, kalite yükselmekte, insan gücünün kullanımı azalmaktadır. İşte endüstride teknolojinin ilerlemesi sonucunda iş makinelerinde otomasyona gidilmiş, otomasyon sistemleri geliştirilmeye başlanmıştır. Otomasyon sistemleri sayesinde el değmeden üretim gerçekleşmiş, insanlar için tehlike arz eden yerlerde robot kullanımı yaygınlaşmış, çalışan insan sayısı asgariye indirilerek üretimde ucuzluk, kalite ve miktar artmış, zaman kaybı ve mali kayıp azalmıştır.

Günümüzde programlanabilir lojik denetleyiciler (Programmable Logic Controller - PLC) endüstride otomasyon işlemlerinin gerçekleştirilmesinde çok yaygın olarak kullanılmaktadır. Bunun en önemli sebeplerinden birisi PLC'lerin endüstriyel ortamlarda (nem, sıcaklık, mekanik titreşim, v.b.) çalışmak üzere tasarlanmış olması ve programlanmalarının nispeten kolay olması gösterilebilir. Benzer işlemleri yapabilecek şekilde tasarlanabilen mikrodenetleyicilerle kıyaslandığı zaman PLC'lerin fiyatları nispeten yüksek olmaktadır [42].

PLC'ler otomasyon devrelerini gerçeklemeye yönelik özel donanım ve yazılım özellikleri taşıyan, sayısal mantıkla çalışan kontrol birimleridir. Ancak günümüz ekonomik şartları göz önünde bulundurulduğu zaman PLC'ye muadil, aynı işlevi daha az maliyetle yapabilecek sistemler ön plana çıkmaya başlamıştır. Bu konuda en çok göze çarpan sistemler mikrodenetleyici temelli sistemlerdir.

PLC tasarımı ve üretimi süreçleri, şu ana kadar PLC üreticilerinin bir sırrı olarak kalmıştır. Mikroişlemci temelli bir teknolojiye dayalı olarak gerçekleştirilen PLC'lerin yerine getirdiği işlev kullanıcılar tarafından çok iyi bilinmesine karşın mikroişlemci/ mikrodenetleyici temelli bir PLC'nin gerçekleştirilmesine dair birkaç yıl öncesine kadar rapor edilmiş ciddi bir çalışma mevcut değildir. Bu konuda son yıllarda yayınlanmış bazı çalışmalar bulunmaktadır.

[39]'daki yüksek lisans tez çalışmasında PIC16F84 mikrodenetleyicisi kullanılarak 8 girişli/8 çıkışlı bir PLC gerçekleştirilmiştir. [2-23]'te görülen "PLC with PIC16F648A Microcontroller" isimli çalışmada PIC16F648A mikrodenetleyicisi kullanılarak bir PLC gerçekleştirilmiştir. Bu PLC'nin giriş/çıkış kartları modüler olarak artırılabilir niteliktedir. Benzer bir başka proje [24-38]'de rapor edilmiştir. "PIC16F877A Mikro-Denetleyicisi Temelli PIC_PLC Donanımı" isimli bu çalışmada PIC16F877A mikrodenetleyicisi ile 16 girişli/16 çıkışlı bir PLC gerçekleştirilmiştir. Bu çalışmaların tamamında 8 bitlik mikrodenetleyiciler ve PIC Assembly dili kullanılmıştır. Gerçekleştirilen bu yüksek lisans tezinde daha önce yapılan bu çalışmalardan farklı olarak 16 bitlik bir mikrodenetleyici ve PIC C dili kullanılmıştır. Böylece elde edilen PLC çalışma hızı ve hafıza kapasitesi bakımından daha yüksek bir performansa sahiptir. Öte yandan PIC C dilinin kullanılması PIC Assembly dilinin kullanılmasına göre daha esnek bir program yapısı geliştirilmesi sağlamıştır.

Bu tez çalışmasında, günümüzde çok yaygın olarak kullanılan PIC mikrodenetleyicilerinden 16 bitlik bir mikrodenetleyici kullanılarak program hafıza kapasitesi, çalışma hızı ve diğer donanım özellikleri, küçük kapasiteli 8 bitlik ve çok yüksek kapasiteli 32 bitlik PIC mikrodenetleyicilerine göre orta büyüklükte olan bir PLC'nin tasarımı ve uygulaması gerçekleştirilmiştir. Böylelikle donanım ve yazılım özellikleri olarak orta büyüklükteki endüstriyel çözümler için uygun ve kullanımı kolay bir endüstriyel kontrol aracı elde edilmiştir.

Bu tez çalışması altı bölümden oluşmaktadır ve diğer bölümlerde ele alınan çalışmalar şu şekildedir:

İkinci bölümde teknolojik gelişmeler, otomasyon sistemleri, PLC'ler ve mikrodenetleyiciler hakkında genel bilgi sunulmuştur.

Üçüncü bölümde Microchip firmasına ait 16 bitlik bir PIC mikrodenetleyici olan PIC24FJ128GA010'un genel özellikleri anlatılmıştır.

Dördüncü bölümde PIC24FJ128GA010 mikrodenetleyicisi ile tasarlanan 16 girişli/16 çıkışlı PLC donanımı incelenmektedir.

Beşinci bölümde PIC24FJ128GA010 mikrodenetleyicisi ile tasarlanan 16 girişli/16 çıkışlı PLC için geliştirilen yazılım incelenmektedir.

Altıncı ve son bölümde ise bu çalışmadan elde edilen sonuçlar ifade edilmiştir.

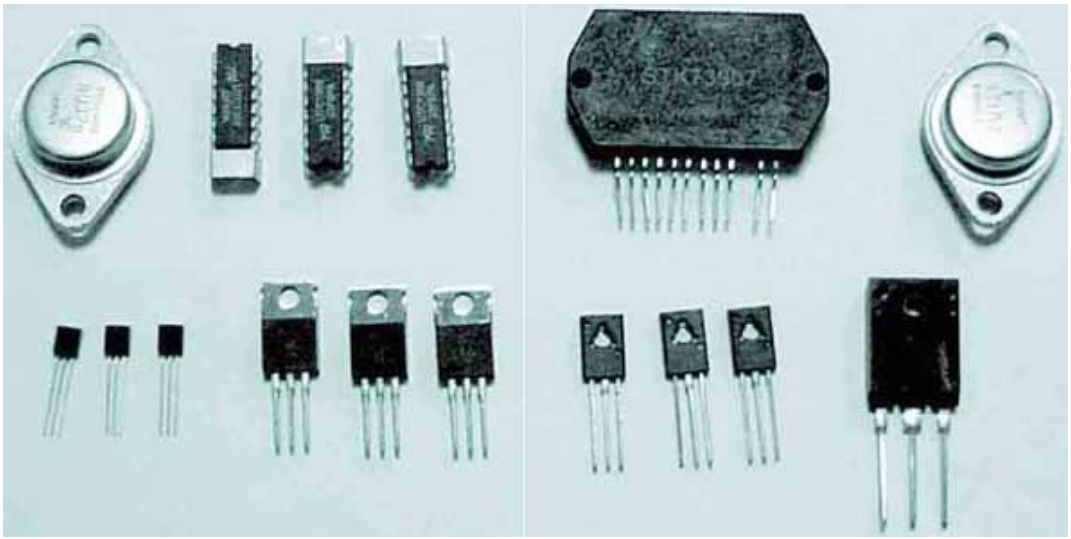
BÖLÜM II

TEKNOLOJİ VE OTOMASYON

2.1 Teknolojinin Önemi ve Gelişimi

Teknolojinin geçmişten günümüze kadar insan yaşamına birçok faydaları olmuştur. Bu yüzden ülkelerin teknoloji üretmesi gelişmişlikleriyle doğrudan ilgilidir. Teknolojinin oluşturulması ve kullanılması insan yaşamına olumlu ve olumsuz birçok etkiler yapmıştır. İnsanın kendini tanıma yolunda dur durak bilmeyen çabaları ile birçok keşif ve icatlar gerçekleşmiştir. Elde edilen bu yeni değerler ilk aşamada laboratuar ortamında kullanılırken daha sonra günlük hayatın da parçası olmuştur.

Endüstri devrimi ile bilimin tüm alanlarındaki gelişmeler de ivmelenmiştir. Yazının bulunuşundan endüstri devrimine kadar geçen süredeki gelişmeler ile son yüzyıldaki, hatta 1950 yılında elektronik ve bilgisayar teknolojisinde transistörün bulunmasından bu yana geçen süre içinde insanlığın elde ettiği gelişmeler, hızlı bir gelişme göstermektedir. Transistör, teknolojinin hızlı gelişiminde çok önemli bir paya sahiptir. Resim 2.1'de çeşitli transistörlerden örnekler görülmektedir. Transistörün geliştirilmesini izleyen yaklaşık elli yıllık bir süre içinde bilime dayalı "ileri teknolojiler" doğar. Biyoteknoloji, gen mühendisliği ve moleküler biyoloji ile üretim sistemindeki değişimler yanında ürünlerin boyutlarında da bir minyatürleşme olur ve gıda üretimi tarlalardan araştırma laboratuvarlarına doğru kaymaya başlar [44].

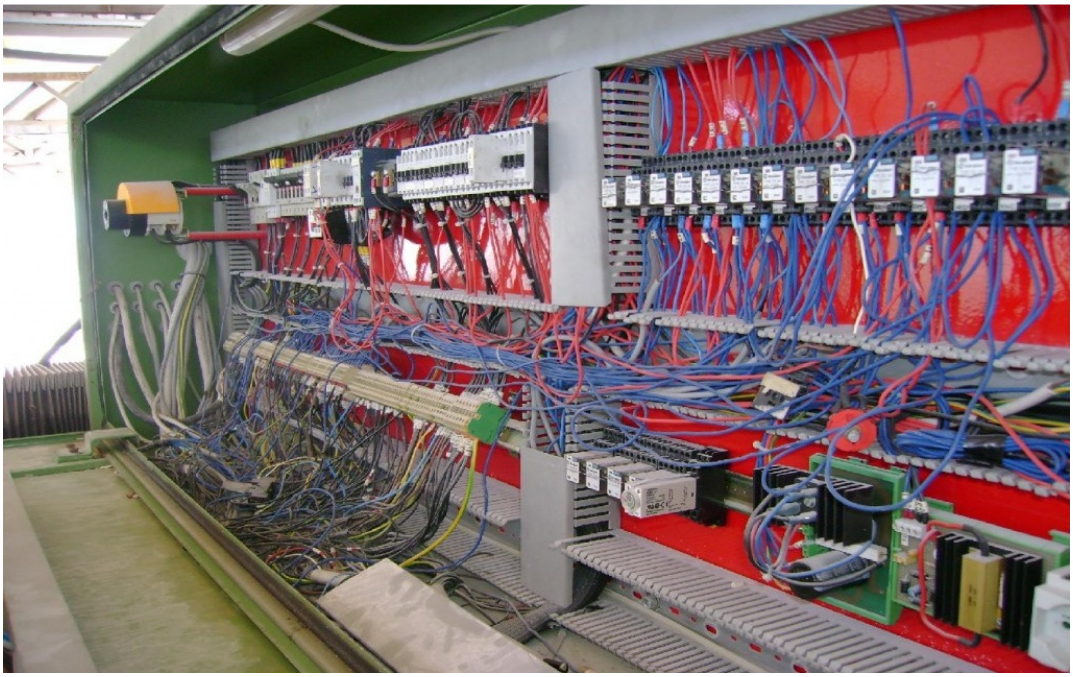


Resim 2.1 Çeşitli transistörlerden örnekler

2.2 Endüstriyel Otomasyon Kavramı

Otomasyon kısaca bir işin insan ile makine arasında paylaşılması şeklinde tanımlanabilir. Bir otomasyon sisteminde toplam işin paylaşım yüzdesi ise otomasyonun düzeyini belirler. İnsan gücünün yoğun olduğu otomasyon sitemleri yarı otomasyon, makinenin yoğun olduğu sitemler ise tam otomasyon olarak adlandırılır. Resim 2.2'de bir otomasyon sisteminden örnek görülmektedir.

Endüstride kolay ve güvenilir üretim yönetimi, temelde prosesin doğru işletilmesi ve her adımında kontrol edilmesiyle mümkündür. Bu şekilde bir prosesi planlarken, daha az insan gücü ile daha çok kontrollü ve kaliteli üretim için en iyi yardımcı otomasyon olacaktır. Otomatik kontrol endüstride ilk yerleşmeye başladığı tarihte pek çok sistem rölelerle veya özel olarak tasarlanmış elektronik kartlarla kontrol ediliyordu. Bu sistemlerin çok maliyetli olması yanında karmaşık yapısı, arıza takibinin zorluğu ve yeni teknolojik gelişmelere açık olmaması gibi daha birçok sorunu da beraberinde getirmekteydi. Aynı zamanda reçete işleme, veri toplama, değerlendirme ve raporlama özelliklerinde yoktu. İşte bu nedenlerden dolayı PLC'ler ortaya çıkmıştır [41].



Resim 2.2 Bir otomasyon sisteminden kesit

Otomatik üretim modern sanayinin temeli ve teknik ilerlemenin genel eğilimi olmaktadır. Bu da yeni fabrikasyon süreçleri, otomasyon olanaklarının daha geniş uygulanışı, otomatik işlem görücülerin, sanayi robotlarının ve otomatik kontrol sistemlerinin kullanımı demektir.

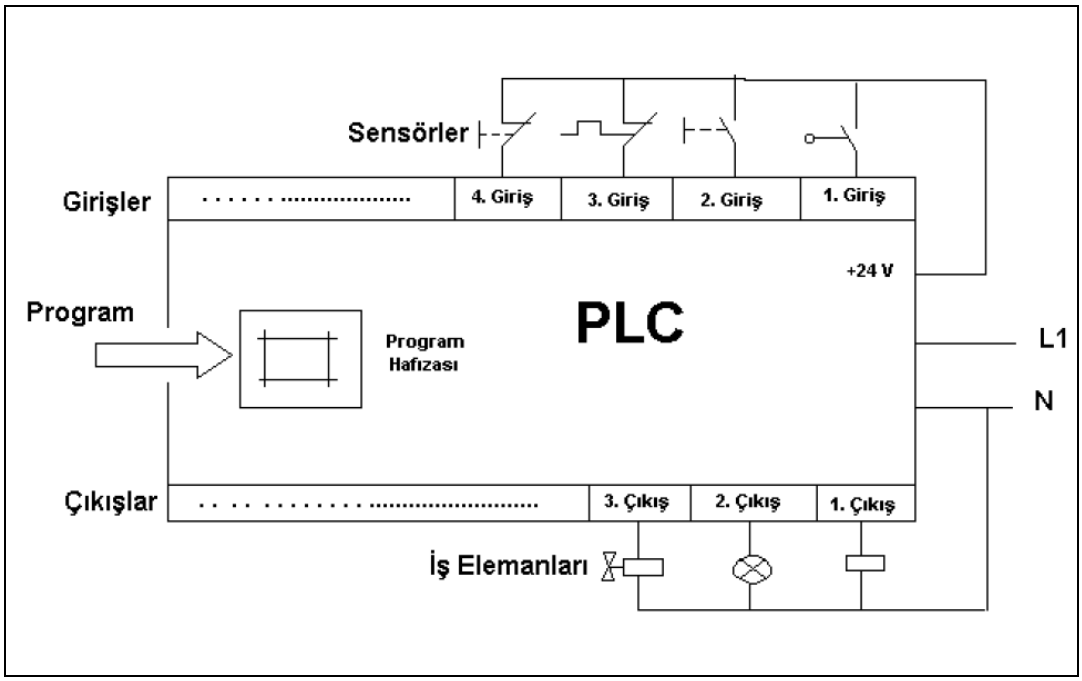
Günümüz modern üretim süreçlerinde yüksek verim ve kaliteli üretim için kaçınılmaz olan endüstriyel otomasyon sistemleri her geçen gün hızla gelişmektedir. Bu hızlı gelişimde PLC kullanımı çok önemli bir paya sahiptir. Bilindiği gibi endüstriyel otomasyon sistemleri en küçük üretim biriminin amaca uygun çalışmasını düzenlediği gibi bütün üretim sistemleri arasında veri iletişimi olanağı sağlayarak daha üst düzeyde yönetim ve planlama için gerekli bilgi tabanını oluşturur [40].

2.3 Programlanabilir Lojik Denetleyiciler (PLC)

Programlanabilir Lojik Denetleyiciler (PLC) otomasyon devrelerinde yardımcı röleler, zaman röleleri, sayıcılar gibi kumanda elemanlarının yerine kullanılan mikroişlemci temelli cihazlardır. Algılayıcılardan aldığı bilgiyi kendine verilen programa göre işlerler ve iş elemanlarına aktarırlar. Başlangıçta yalnızca basit işlemler için kullanılan PLC'ler günümüzde çeşitli firmalar tarafından geliştirilerek çok daha karmaşık sorunları çözebilecek özelliklerle donatılmış ve endüstriyel kontrol alanlarında güvenle kullanılmaya başlanmıştır. Günümüzde SIEMENS, MITSUBISHI, OMRON gibi birçok firma çeşitli özelliklerde PLC'ler üretmektedir [45]. Her firma, ürettiği PLC'ye uygun yazılımları da geliştirmektedir. PLC'lerin farklı özellikleri olsa da programlama mantığı birbirlerine çok benzemektedir. Resim 2.3'te bir Siemens PLC görülmektedir.

Resim 2.3 Bir Siemens S7 300 PLC

PLC sistemi sahada meydana gelen fiziksel olayları, değişimleri ve hareketleri çeşitli ölçüm cihazları ile belirleyerek, gelen bilgileri yazılan kullanıcı programına göre bir değerlendirmeye tabi tutar. Mantıksal işlemler sonucu ortaya çıkan sonuçları da kumanda ettiği elemanlar aracılığıyla sahaya yansıtır. Sahadan gelen bilgiler ortamda meydana gelen aksiyonların elektriksel sinyallere dönüşmüş halidir [40].



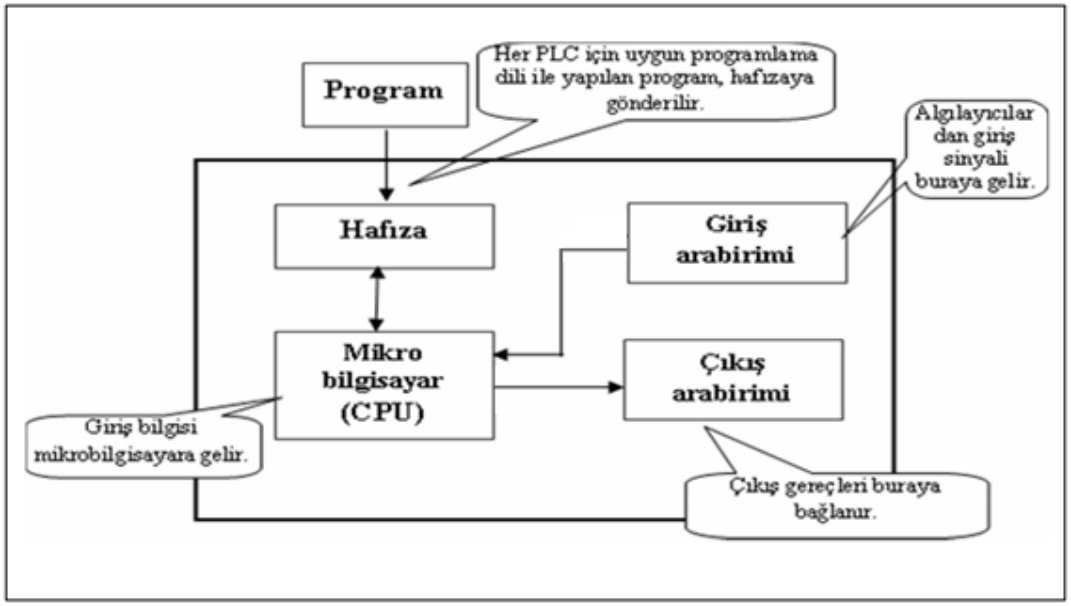
Şekil 2.1 PLC genel blok şeması

Şekil 2.1'deki blok diyagramda görüldüğü gibi PLC sensörlerden aldığı bilgiyi hafızadaki programa göre işleyen ve iş elemanlarına aktaran bir mikroişlemci sistemidir. Sensörlere örnek olarak, herhangi bir metali algılayan endüktif sensör, PLC girişine uygun gerilim vermede kullanılan buton ve anahtarlar verilebilir. İş elemanları için PLC çıkışından alınan gerilimi kullanan kontaktörler, bir cismi itme veya çekmede kullanılan pnömatik silindirleri süren elektro-valfler, lambalar örnek olarak verilebilir.

Giriş komutları buton, seçici anahtar, dijital anahtar veya sensör girişi olan anahtar, fotoelektrik anahtar gibi elemanlar yoluyla verilir. Bu elemanlarla yüklerin çalışma şartları kontrol edilir veya gözlemlenir. Giriş sinyallerine karşılık çıkış sinyallerinin iletimi PLC'de yazılı olan programa bağlıdır. Selenoid valf, sinyal lambası, röle gibi küçük yükler PLC tarafından direkt olarak sürülebilir. Fakat büyük kapasiteli selenoid valf, 3 fazlı motor gibi yükler kontaktör ve röle üzerinden sürülür.

2.3.1 PLC' nin yapısı

Şekil 2.2'den görüleceği gibi PLC; içerisinde işlemci (CPU), giriş-çıkış ara yüzleri ve hafıza bulunduran bir endüstriyel mikrobilgisayardır. Bir PLC'nin iç yapısı ve donanımı PLC mimarisi olarak adlandırılır. PLC'lerin tümü CPU, hafıza, giriş/çıkış gibi temel arayüz elemanlarına sahiptirler. PLC ile kontrolde, programlama konsolundan PLC'ye çeşitli giriş sinyalleri ve kontrol içerikleri gönderilir. Sonuç olarak da çıkış sinyalleri, çıkış elemanlarına yöneltilir.



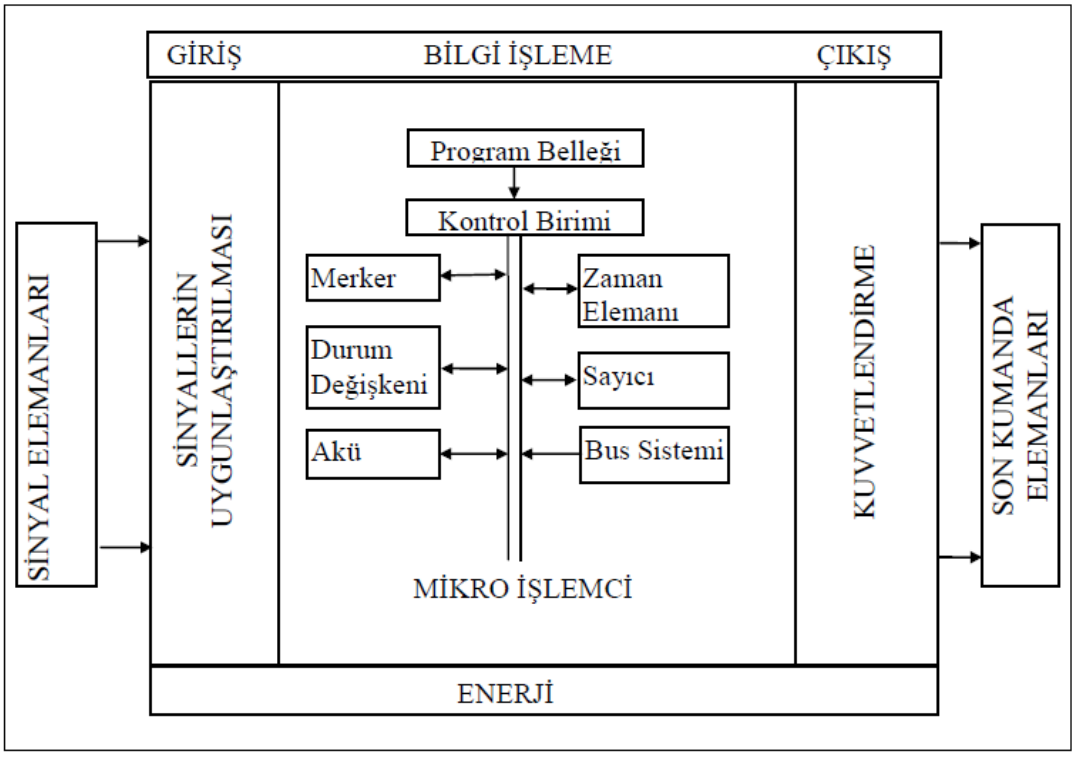
Şekil 2.2 PLC iç yapısı

Şekil 2.2 PLC ile çevre birimleri arasındaki bağlantıyı göstermektedir. PLC'ler temel prensip olarak üç bölümden oluşur. Birincisi merkezi işlem birimidir(central processing unit-CPU). İkincisi bütün sinyal elemanlarının bağlandığı giriş birimidir. Üçüncüsü ise son kumanda elemanlarının bağlandığı çıkış birimidir.

Giriş elemanlarından gelen sinyaller PLC'nin işlemcisine giriş arabirimi üzerinden gönderilir. İşlemci (CPU), hafızaya kaydedilmiş olan programın içeriğine ve giriş sinyallerine göre çıkış sinyallerini kontrol eder. Hafızaya kaydedilen program işlemci (CPU) tarafından okunur. Okunduktan sonra da programın içeriği yapılabilirliğine göre kontrol edilir [41].

Program uygulanabilir olduğunda, CPU tarafından çalıştırılır ve bir sonraki adıma geçilir. Hafızadaki tüm program satır satır işlenir. Çıkış bilgisi güncellenerek çıkış elemanlarına gönderilir.

Bir PLC; yüzlerce ayrı röle, zamanlayıcı, sayıcı ve veri saklama yerlerinden oluşmuş bir kutu gibi düşünülebilir. Bu kadar eleman aslında fiziksel olarak mevcut değildir. Sanal olarak yazılımda oluşturulmaktadır. Bu dâhili röleler register (yazmaç) denen kaydedicilerde bulunan bit konumlarında simule edilirler.



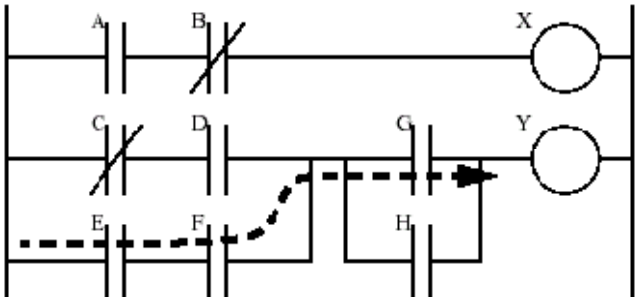
Şekil 2.3 PLC çalışma diyagramı

Şekil 2.3'teki çalışma diyagramından görüldüğü gibi her PLC'nin prizden aldığı 220 Volt'u 5 Volt'a indirgeyen bir enerji modülü bulunmaktadır. Giriş ve çıkış bölümleri için genellikle 24 Volt kullanılmakla beraber bu değer 220 Volt'a kadar değişebilmektedir.

Girişler Opto-Coupler ile merkezi işlem biriminden izole edilir. Aynı şekilde çıkışlar da Opto-Coupler veya röle ile izole edilmiştir. Ayrık girişlerden gelen sinyallerin durumu Boole (0 veya 1) şeklinde değerlendirilerek CPU'da işleme alınır. Bu aşamada zamanlayıcı, sayıcı ya da durum değişkenleri gibi elemanlar devreye girer. Hafızadaki programın işlenmesinde, CPU giriş sinyallerini dikkate alarak komutları işler. Normal şartlarda bu işlem sürekli şekilde tekrarlanır [44].

2.3.2 PLC programlama ve Merdiven (Ladder) mantığı

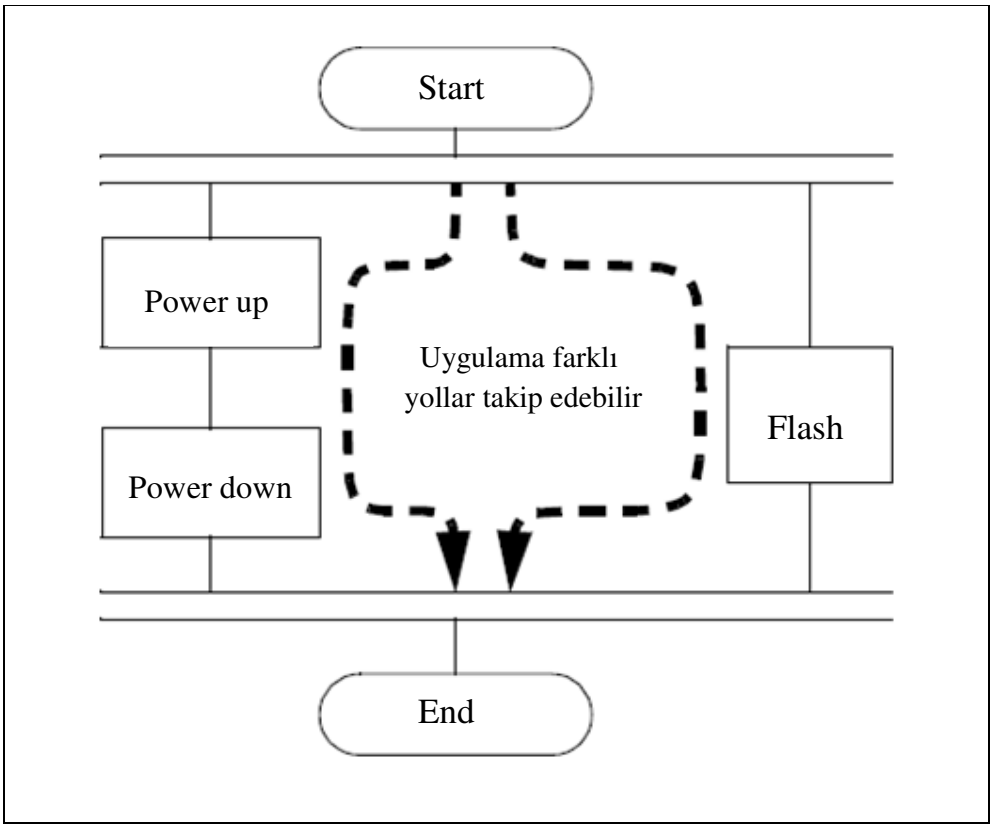
Merdiven (Ladder) metodu PLC'lerde en çok kullanılan programlama yöntemdir. PLC donanımını yazılımsal olarak gerçekleştirmek, röleleri sembolize etmek amacıyla geliştirilmiştir.



Şekil 2.4 Merdiven (Ladder) diyagramı

Şekil 2.4'te Merdiven (Ladder) diyagramına bir örnek verilmiştir. Sol taraf canlı uç, sağ taraf nötr uçtur. Enerji; giriş elemanlarından gelen sinyaller doğrultusunda kontakların alacağı pozisyona göre devresini tamamlayıp çıkış elemanlarına ulaşır.

Merdiven mantığı PLC'de en çok kullanılan yöntem olmakla beraber farklı programlama yöntemleri de vardır. "Sıralı Fonsiyon Şeması" (Sequential Function Charts-SFC) bunlardan bir tanesidir. Akış diyagramları mantığına benzemektedir. Karmaşık sistemlerin programlanmasında tercih edilir. Şekil 2.5'te bu yönteme ait bir örnek verilmiştir.



Şekil 2.5 Sıralı fonsiyon şeması

Bir başka programlama metodu "Mnemonic Komut Listesi" (Mnemonic Instruction List) yöntemidir. Merdiven diyagramındaki satırların komut satırları halinde yazılması olarak ifade edilebilir. Assembly diline benzeyen bu teknikte kontakların durumları 'LOAD, OUT, AND, OR' gibi komutlarla kontrol edilir. Şekil 2.6'da Mnemonic komut listesi yöntemine bir örnek verilmiştir.

```
0000    LD I 0.0
0001    AND I0.1
0002    OUT  Q0.0
0003    LD I0.2
0004    OR I 0.3
0005    OUT Q0.1
0006    LD 1.1
0007    OUT Q 1.1
```

Şekil 2.6 Mnemonic komut listesi yöntemi

2.3.3 Röle ile PLC'nin karşılaştırılması

PLC kullanımının röleli kontrole göre çeşitli açılardan üstünlükleri vardır. Eski sistemlerin getirdiği birtakım zorluklar bugün PLC'lerin yaygınlaşması ile aşılmıştır. PLC sistemleri önceki sistemlere göre daha az yer kaplamaktadır. Dolayısıyla kontrol sisteminin yer aldığı dolap ya da pano boyutları oldukça küçülmektedir. Sınırlı alanlarda kontrol mekanizmasının kurulması imkânı ortaya çıkmıştır. Sistem için sarfedilen kablo maliyetleri nispeten daha azdır. Ayrıca PLC sisteminin kurulmasının kolay olması ve kullanıcıya, kurulu hazır bir sistemin üzerinde değişiklik ve ilaveleri kolayca yapabilme esnekliğinin sağlanması, PLC'lerin giderek yaygınlaşmasına ve endüstride her geçen gün daha fazla kullanılmalarına neden olmuştur.

Bir makinenin ya da bir fabrikadaki prosesin gerçekleştirilmesi sırasında aynı anda birçok olay meydana gelir ve bunların bir sıra halinde olması gerekmez. Dolayısıyla bir bilgisayar programıyla bu tür prosesler kontrol edilemez. Fakat bir PLC için aynı anda gerçekleşen olayları kontrol etmek sorun değildir. Sırf kumanda işlemlerine yönelik birçok komutu da fazladan ihtiva etmesi sebebiyle, PLC ile bu tip programları yazıp çalıştırmak çok kolaydır [41].

Aşağıda PLC kullanımının röleli kontrole göre çeşitli açılardan üstünlükleri belirtilmiştir:

- PLC ile daha üst seviyede otomasyon sağlanır.
- Üretim süresinde kısalma olur.
- PLC, röleli kontrole göre çok daha küçük boyutludur. Daha az yer kaplar ve daha az enerji harcar.
- PLC'li sistem daha uzun süre bakımsız çalışır ve ortalama bakım onarım süresi daha azdır.
- PLC sisteminin kurulması kolaydır ve kullanıcıya, kurulu hazır bir sistemin üzerinde değişiklik ve ilaveleri kolayca yapabilme esnekliğini sağlar.
- Röleli sistemlerde ekleme ve değişiklik yapmak zordur ancak PLC sisteminde çalışma limitine kadar genişletme yapılabilir.
- Röleli sistemler büyük çaplı klasik kontrol açısından pahalıdır. PLC'ler böyle bir durumda daha ekonomik olur.

2.4 Mikrodenetleyiciler ve PIC

Bir bilgisayar içerisinde bulunması gereken temel bileşenlerden CPU, RAM ve I/O ünitesinin tek bir chip içerisinde üretilmiş biçimine mikrodenetleyici (microcontroller) denir. Bilgisayar teknolojisi gerektiren uygulamalarda kullanılmak üzere tasarlanmış olan mikrodenetleyiciler, mikroişlemcilere göre çok daha basit ve ucuzdur. Günümüz mikrodenetleyicileri otomobillerde, kameralarda, cep telefonlarında, fax-modem cihazlarında, fotokopi, radyo, TV, bazı oyuncaklar gibi pek çok alanda kullanılmaktadır.

Günümüz mikrodenetleyicileri birçok entegre üreticisi tarafından üretilmektedir. Her firma ürettiği entegreye farklı isimler vermektedir. Örneğin Microchip firması ürettiklerine PIC adını verirken, Intel'in ürettiği ve 1980'lerin başında piyasaya sürdüğü 8051, bazen MCS-51 olarak da adlandırılır. Neredeyse her mikroişlemci (CPU) üreticisinin ürettiği birkaç mikrodenetleyicisi bulunmaktadır. Bu denetleyicilerin mimarileri arasında çok küçük farklar olsa da aşağı yukarı aynı işlemleri yapabilmektedirler. Her firma ürettiği entegreye bir isim ve özelliklerini birbirinden ayırmak için de parça numarası vermektedir [44].

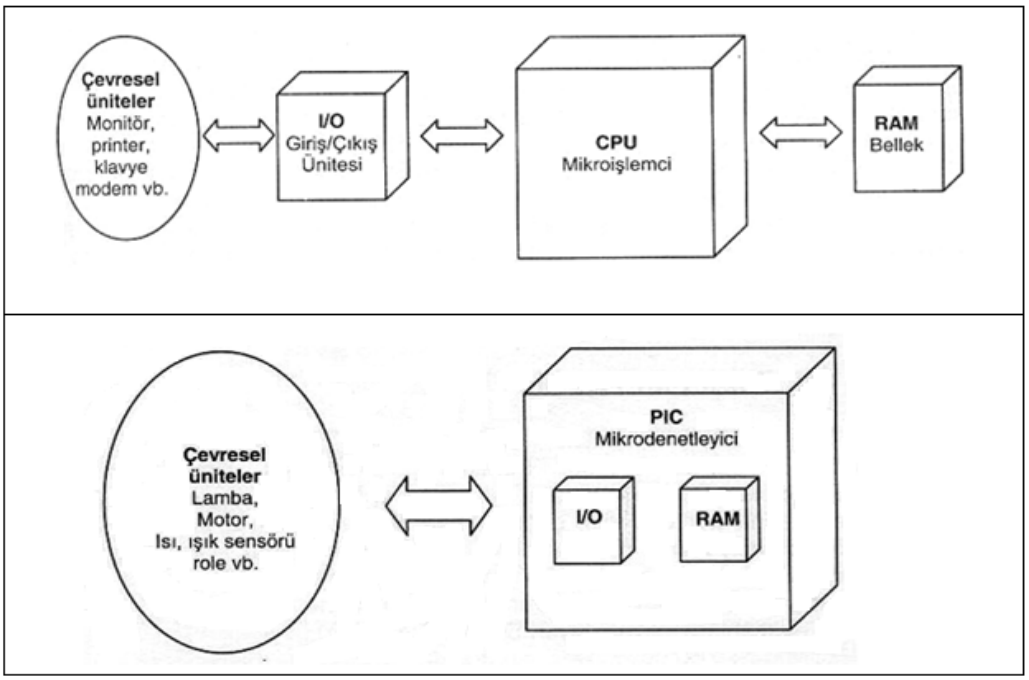
Bir uygulamaya başlamadan önce hangi firmanın ürünü kullanılacağına, daha sonra da hangi numaralı denetleyicinin kullanılacağına karar vermek gerekir. Bunun için mikrodenetleyici gerektiren uygulamada hangi özelliklerin bulunmasının gerektiği önceden bilinmesi gereklidir. Aşağıda bu özellikler sıralanmıştır:

- Programlanabilir dijital paralel giriş/çıkış
- Programlanabilir analog giriş/çıkış
- Seri giriş/çıkış (senkron, asenkron ve cihaz denetimi gibi)
- Motor veya servo kontrol için pals sinyali çıkışı
- Harici giriş vasıtasıyla kesme
- Timer vasıtasıyla ile kesme
- Harici bellek arabirimi
- Harici bus arabirimi (PC ISA gibi)
- Dâhili bellek tipi seçenekleri (ROM, EPROM, PROM ve EEPROM)
- Dâhili RAM seçeneği
- Kayan nokta hesaplaması

2.4.1 Mikroişlemci ve mikrodenetleyicinin karşılaştırılması

Günümüzde kullanılan bilgisayarların özelliklerinden bahsedilirken duyulan 80386, 80486, Pentium II, Pentium III, Pentium IV, Celeron, Centrino birer mikroişlemcidir. Mikroişlemciler bilgisayar programlarının yapmak istediği tüm işlemleri yerine getirdiği için, çoğu zaman merkezi işlem birimi (CPU) olarak da adlandırılır. PC adını verdiğimiz kişisel bilgisayarlarda kullanıldığı gibi, bilgisayarla kontrol edilen sanayi tezgâhlarında ve ev aygıtlarında da kullanılabilmektedir.

Mikroişlemci ile kontrol edilecek bir sistemi kurmak için en azından CPU, RAM, I/O üniteleri bulunmalıdır ve bu ünitelerin arasındaki veri alış verişini kurmak için DATA BUS (veri yolu) gerekmektedir. Bu üniteleri yerleştirmek için baskılı devre de olması gerekir. Mikrodenetleyici ile kontrol edilecek sistemde ise yukarıda saydığımız ünitelerin yerine geçecek tek bir entegre yani 'mikrodenetleyici' ve bir de devre seti kullanmak yeterli olur. Tek entegre kullanarak elektronik çözümler üretmenin maliyetinin daha düşük olacağı açıktır. Ayrıca kullanım ve programlama kolaylığı da ikinci bir avantajıdır. Bu nedenlerden dolayı son zamanlarda bilgisayar kontrolü gerektiren elektronik uygulamalarda mikrodenetleyici kullanmaya eğilimi artmıştır. Şekil 2.7'de mikrodenetleyicinin mikroişlemciye göre avantajı ifade edilmektedir [44].



Şekil 2.7 Mikroişlemci ve mikrodenetleyicinin karşılaştırılması

2.4.2 PIC mikrodenetleyicileri

PIC, adını İngilizcedeki Peripheral Interface Controller (Çevresel Arayüz Denetleyicisi) cümlesindeki kelimelerin baş harflerinden almış olan bir mikrodenetleyicidir. Bir giriş/çıkış işlemcisidir. PIC serisi mikrodenetleyiciler MICROCHIP firması tarafından geliştirilmiştir ve üretim amacı çok fonksiyonlu lojik uygulamalarının hızlı ve ucuz bir mikroişlemci ile yazılım yoluyla karşılanmasıdır.

Çevresel üniteler adı verilen lamba, motor, role, ısı ve ışık sensörü gibi I/0 elemanların denetimini çok hızlı olarak yapabilecek şekilde tasarlanmıştır. İlk olarak 1994 yılında 16 bitlik ve 32 bitlik büyük işlemcilerin, giriş ve çıkışlarındaki yükü azaltmak ve denetlemek amacıyla çok hızlı ve ucuz bir çözüme ihtiyaç duyulduğu için geliştirilmiştir. RISC mimarisi adı verilen bir yöntem kullanılarak üretildiklerinden bir PIC'i programlamak için kullanılacak olan komutlar oldukça basit ve sayı olarak da azdır. 1980'lerin başından itibaren uygulanan bir tasarım yöntemi olan RISC (Reduced Instruction Set Computer) mimarisindeki temel düşünce, daha basit ve daha az komut kullanılmasıdır [46].

Resim 2.4 Çeşitli PIC'lerden örnekler

PIC serisi tüm işlemciler herhangi bir ek bellek veya giriş/çıkış elemanı gerektirmeden sadece 2 adet kondansatör, 1 adet direnç ve bir kristal ile çalıştırılabilmektedir. Tek bacaktan 40 mA akım çekilebilmekte ve entegre toplamı olarak 150 mA akım akıtma kapasitesine sahiptir. Tipik bir entegrenin 4 MHz osilator frekansında çektiği akım çalışırken 2 mA, stand-by durumunda ise 20uA kadardır. Resim 2.4'te çeşitli PIC' lerden örnekler verilmiştir.

2.4.3 PIC mikrodenetleyicilerinin özellikleri

Güvenirlik: PIC komutları bellekte çok az yer kaplarlar. Dolayısıyla bu komutlar 12 veya 14 bitlik bir program bellek sözcüğüne sığarlar. Harward mimarisi teknolojisi kullanılmayan mikrodenetleyicilerde yazılım programının veri kısmına atlama yaparak bu verilerin komut gibi çalıştırılmasını sağlamaktadır. Bu da büyük hatalara yol açmaktadır. PIC'lerde bu durum engellenmiştir.

Hız: PIC oldukça hızlı bir mikrodenetleyicidir. Her bir komut döngüsü 1µsn'dir. Örneğin 5 milyon komutluk bir programın 20Mhz'lik bir kristalle işletilmesi yalnız 1sn sürer. Bu süre 386SX33 hızının yaklaşık 2 katıdır. Ayrıca RISC mimarisi işlemcisi olmasının hıza etkisi oldukça büyüktür.

Komut seti: Tüm PIC türleri RISC mimarisi yapısında olduğu için oldukça daraltılmış komut setlerine sahiplerdir. Örneğin 16CXX ailesinde bir yazılım yapmak için sadece 35 komuta ihtiyaç duyulur. Bu sayı 18XX ve 24XX ailelerine gelince 70'in üzerine çıksa da benzer mikrodenetleyicilere oranla daha azdır. PIC tarafından kullanılan komutların hepsi yazmaç (register) temellidir.

Statik İşlem: PIC tamamıyla statik bir işlemcidir. Yani saat durdurulduğunda da tüm kaydedici içerikleri korunur. Pratikte bunu tam olarak gerçekleştirebilmek mümkün değildir. PIC programı işletilmediği zaman uyuma (sleep) moduna geçirilerek mikrodenetleyicinin çok düşük akım çekmesi sağlanır. PIC uyuma moduna geçirildiğinde saat durur ve PIC uyuma işleminden önce hangi durumda olduğunu çeşitli bayraklarla (elde bayrağı, 0 (zero) bayrağı... vb.) ifade eder. PIC uyuma modunda 1µA'den küçük değerlerde akım çeker (Standby akımı).

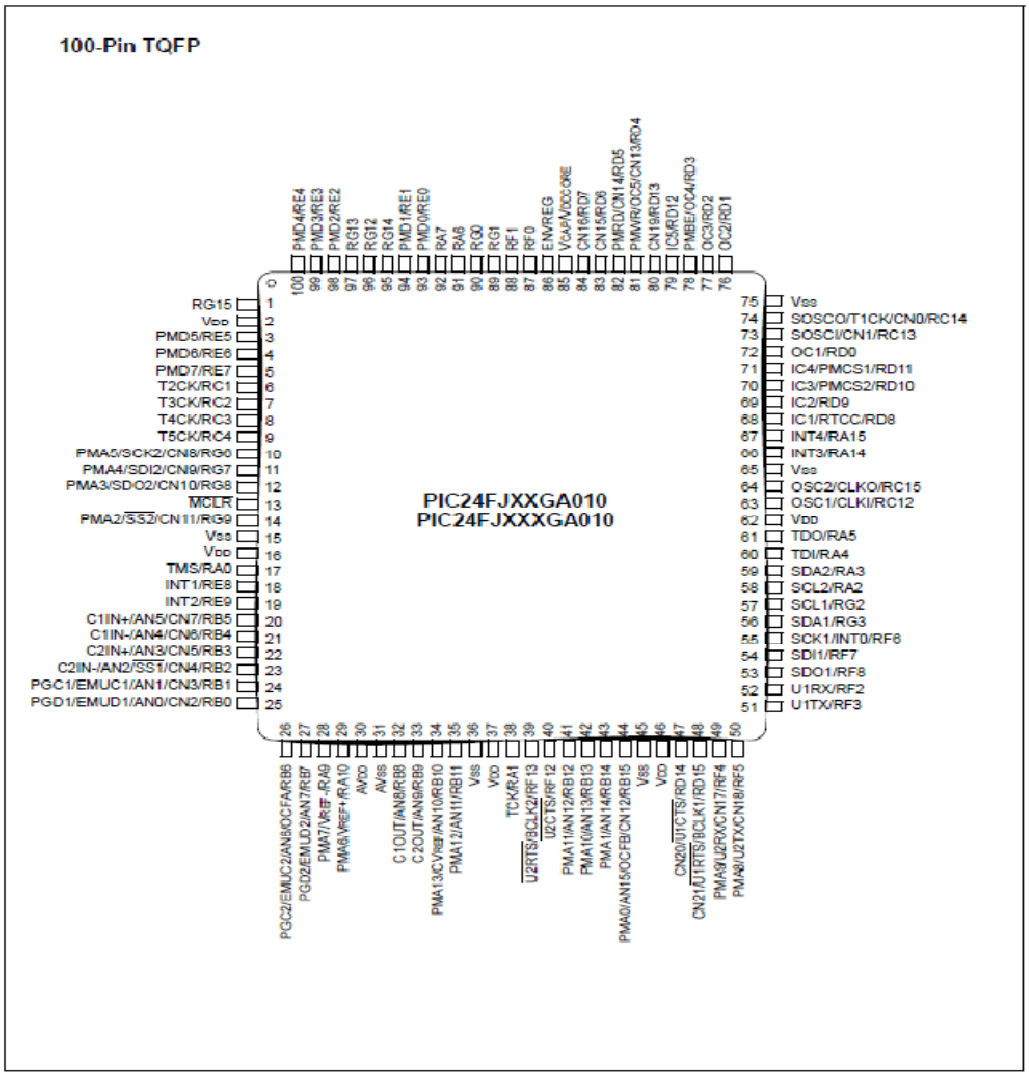
Sürme özelliği (Sürücü kapasitesi): PIC yüksek bir çıktı kapasitesine sahiptir. Tipik bir PIC tek bacaktan 40mA akım çekebilmekte ve entegre toplamı olarak 150mA akım akıtma kapasitesine çıkabilmektedir.

Seçenekler: PIC ailesinde her türlü ihtiyaçların karşılanacağı çeşitli hız, sıcaklık, kılıf, I/O hatları, zamanlama (Timer) fonksiyonları, seri iletişim portları, A/D ve bellek kapasite seçenekleri bulunur.

BÖLÜM III

PIC24FJ128GA010 AİLESİNE GENEL BAKIŞ

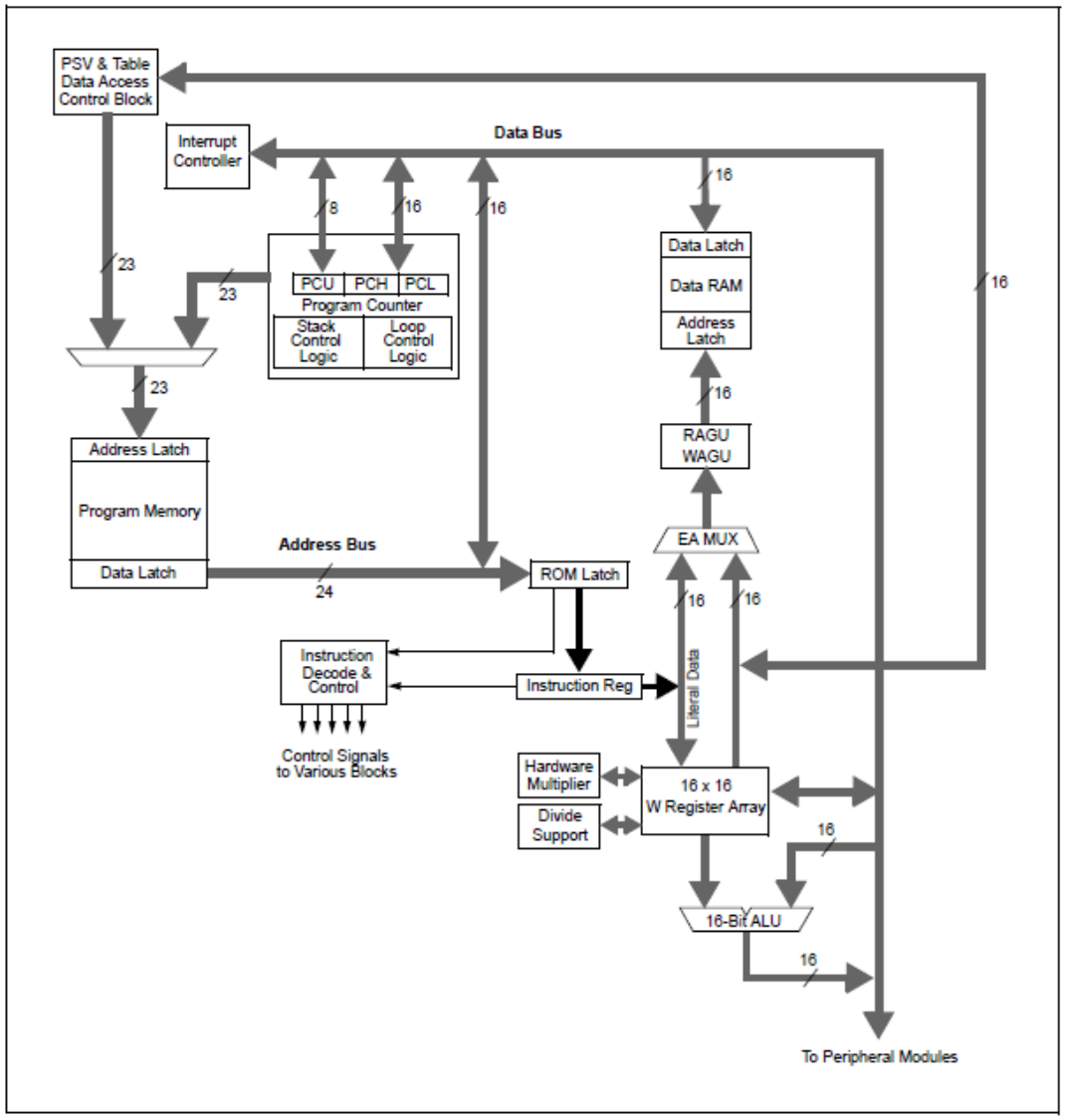
Bu bölümde gerçekleştirilen PLC donanımı merkezi işlem biriminin temeli olan PIC24FJ128GA010 mikrodenetleyicisine ait bilgiler sunulmaktadır. 24F ailesinden olan PIC24FJ128GA010; 100 pinli, 16 bitlik bir mikrodenetleyicidir. 128 Kb program hafızası, 8Kb SRAM hafızası vardır. Yüksek performanslı bir CPU'ya sahiptir. Kendine has mikrodenetleyici özellikleri, analog özellikleri ve çevresel özellikleri mevcuttur [43]. Şekil 3.1'de 100 pinli yapı diyagramı görülmektedir.



Şekil 3.1 PIC24FJ128GA010 mikrodenetleyicisi pin diyagramı

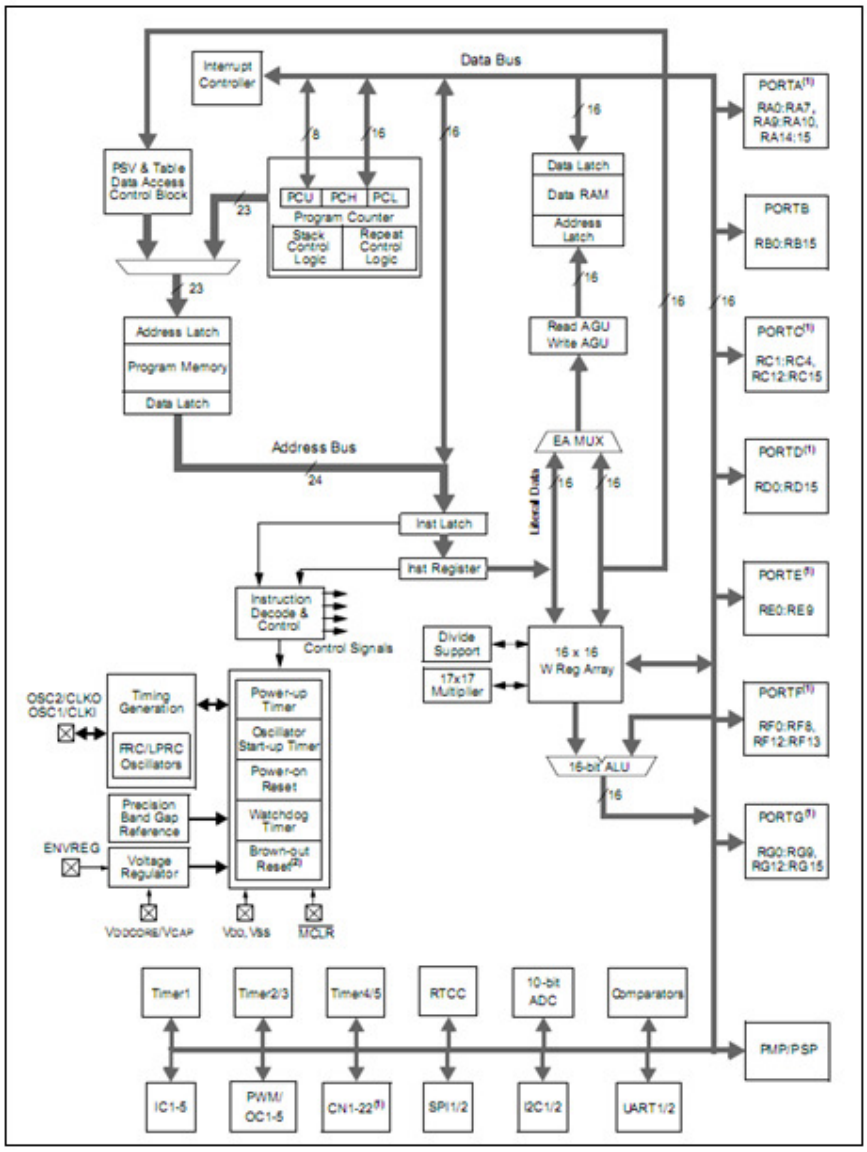
3.1 CPU Çekirdek Yapısı

PIC24F CPU'su 16 bit geliştirilmiş ve arttırılmış bir komut setine göre uyarlanmış Harvard Mimarisi yapısına sahiptir. 'Program Counter (PC)' 23 bit genişliğindedir ve 4 makine saykılına kadar işlem yapabilir [43]. Şekil 3.2'de CPU çekirdeği blok diyagramı görülmektedir.



Şekil 3.2 CPU çekirdeği blok diyagramı

PIC24F Aritmetik Lojik Birim (ALU) 16 bit genişliğindedir ve toplama, çıkarma, bit kaydırma ile lojik işlemleri yapabilmektedir. Aksi belirtilmedikçe aritmetik işlemler 2'nin tümleyeni olarak ele alınır. ALU'daki işlemlerden Carry (C), Digit Carry (DC), Zero (Z), Negative (N), Overflow (OV) durum bitleri etkilenebilir. ALU; kullanılan komut kodunun moduna göre 8 bit veya 16 bit işlemleri gerçekleştirebilir. ALU işlemleri için gereken bilgi, komutun adresleme moduna bağlı olarak W kaydedicisi (working register) dizisinden ya da veri belleğinden gelebilir. Aynı şekilde ALU'dan gelen çıkış bilgisi de W kaydedicisine yazılabilir [43]. Şekil 3.3'te PIC24FJ128GA010 ailesi genel blok diyagramı görülmektedir.



Şekil 3.3 PIC24FJ128GA010 ailesi genel blok diyagramı

3.2 CPU Özellikleri

PIC24FJ128GA010 mikrodenetleyicisine ait CPU özellikleri aşağıda verilmiştir.

16 bit mimarisi:

- 16 bitlik veri hattı genişliğine göre düzenlenmiş, gelişmiş komut setli Harward Mimarisi yapısı
- 24 bitlik komut genişliği
- 4×PLL ve Çoklu Bölme seçeneklerine sahip 8MHz dahili osilatörü
- 17 bit × 17 bit single-cycle donanımsal çarpıcı; 32 bit/16 bit donanımsal bölücü
- 16 × 16 bit working register (akümülatör) dizisi
- C ve Assembly dillerine uygun programlama yeteneği

Özel mikrodenetleyici özellikleri:

- 2.0 V' tan 3.6 V'a kadar çalışma gerilimi aralığı
- Flash program hafızası
  - En az 1000 silme/yazma döngüsü
  - 20 yıl bilgi saklayabilme
- Yazılım kontrollü kendi kendini programlayabilme özelliği
- Seçenekli güç yönetimi modları
  - Uyku modu, boşta modu, alternatif clock modu
- Dahili 2.5 V regülatör
- JTAG sınır tarama ve programlama desteği
- 'Power on Reset', 'Power-up Timer' ve 'Osilatör Start-up Timer' özellikleri
- Dahili 'Watchdog Timer' özelliği ve düşük güç 'RC Osilatör' özelliği

Analog özellikler:

- 10 bit 16 kanala kadar 'Analog – Dijital Çevirici'
- Programlanabilir giriş/çıkış konfigürasyonlu çift 'Analog Karşılaştırıcı'

Çevresel özellikler:

- 2 adet 3-hat/4-hat SPI modülü
- 2 adet I^2C (inter-integrated circuit) modülü (master/slave mod ve 7 bit/10 bit adresleme)
- 2 adet UART modülü
  - RS-232, RS-485
  - Dahili encoder/decoder
  - Start bitinde otomatik uyandırma
  - 4 seviye FIFO buffer
- Paralel 'Master/Slave' port
  - 8 bit veya 16 bit data hattı
  - 16 bit adres hattı
- Gerçek zamanlı clock
- Clock, takvim ve alarm
- 5 adet 16 bit 'Timer'
- 5 adet 'Counter'
- 5 adet 'Önbölücü (Prescaler)'
- 5 adet compare/PWM çıkışı
- 5 adet 'Capture' girişi
- Yüksek akımlı 'Sink/Source' (tüm giriş-çıkışlarda)
- 5 harici kesme (interrupt) kaynağı
- 5.5 V toleranslı giriş (sadece dijital pinlerde)

3.2.1 Çarpıcı

ALU çok hızlı, 17 bit × 17 bit çarpıcı ihtiva etmektedir. Bu çarpıcı, işaretli ve işaretsiz sayılar arası kombinasyonel çarpım durumlarını desteklemektedir.

1. 16-bit × 16-bit işaretli
2. 16-bit × 16-bit işaretsiz
3. 16-bit işaretli × 16-bit işaretsiz
4. 16-bit işaretsiz × 16-bit işaretli
5. 8-bit işaretsiz × 8 bit işaretsiz

PIC24F çarpıcısı; eski yapıdaki 16 bitlik mikrodenetleyicilere göre üstün olarak, tek makine saykılında 16-bit × 16-bit işlemi geçekleştirebilir [43].

3.2.2 Bölücü

Bölme bloğu 32-bit/16-bit ve 16-bit/16-bit işaretli ve işaretsiz bölme işlemlerini gerçekleştirir [43].

1. 32-bit işaretli/16-bit işaretli
2. 32-bit işaretsiz/16-bit işaretsiz
3. 16-bit işaretli/16-bit işaretli
4. 16-bit işaretsiz/16-bit işaretsiz

3.2.3 Çoklu bit kaydırma

PIC24F ALU'su hem tek bit, tek makine saykılı hem de çoklu bit aritmetik ve lojik kaydırma işlemlerini gerçekleştirir. Çoklu bit kaydırmaları tek makine saykılında blok olarak 15 bit sağa veya 15 bit sola gerçekleştirebilir. Tüm çoklu bit kaydırma komutları direk adresleme metodunu desteklemektedir [43].

3.2.4 Program adres alanı

PIC24FJ128GA010 mikrodenetleyicisinde program adres belleği alanı 4 makine saykılı komutlardan oluşmaktadır. Adresleme 24 bit ile yapılmaktadır. 23 bitlik Program Counter tarafından programlama esnasında adresleme sağlanır Kullanıcının program bellek alanına erişimi hafıza haritasının sadece düşük değerli yarı bölgesi ile sınırlandırılmıştır (000000h – 7FFFFFh). Program bellek alanı 'kelime adresleme'li bloklar olarak düzenlenmiştir [43].

3.2.5 Veri adres alanı

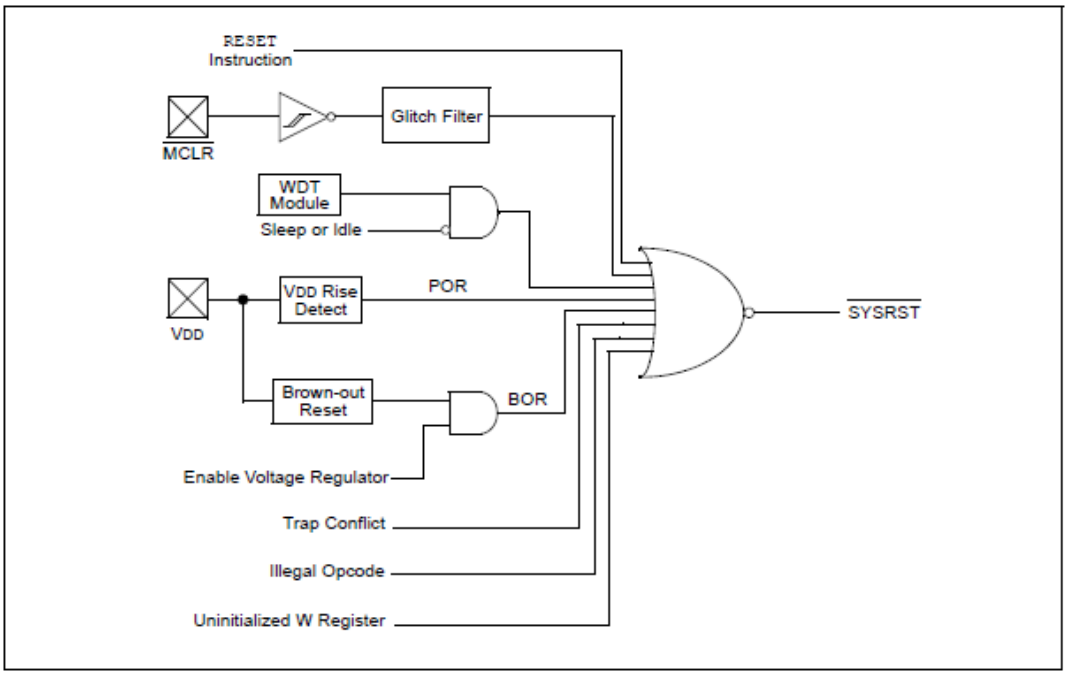
PIC24F çekirdeği 16 bit genişliğinde veri bellek alanına sahiptir. Veri alanına iki adet adres oluşum birimi (AGU) aracılığıyla erişilebilir. Tüm etkin adresler 16 bit genişliğindedir. Veri bellek alanının düşük değerlikli yarı bölgesi bellek adresleri için kullanılırken yüksek değerlikli yarı bölgesi ise program görüntü alanı için ayrılmıştır [43].

3.2.6 Resetler

Reset modülü tüm reset kaynaklarının birleşimidir ve cihazın 'Master Reset Signal' bilgisini kontrol eder [43]. Aşağıda reset kaynakları belirtilmiştir:

- POR: Power on reset
- MCLR: Pin reset
- WDT: Watchdog timer reset
- BOR: Brown out reset
- CM: Configuration word mismatch reset
- TRAPR: Trap conflict reset
- IOPUWR: Illegal opcode reset
- UWR: Uninitialized W register reset

Şekil 3.4'te reset modülünün basitleştirilmiş blok diyagramı görülmektedir.



Şekil 3.4 Reset modülü blok diyagramı

3.2.7 Zamanlayıcılar (Timers)

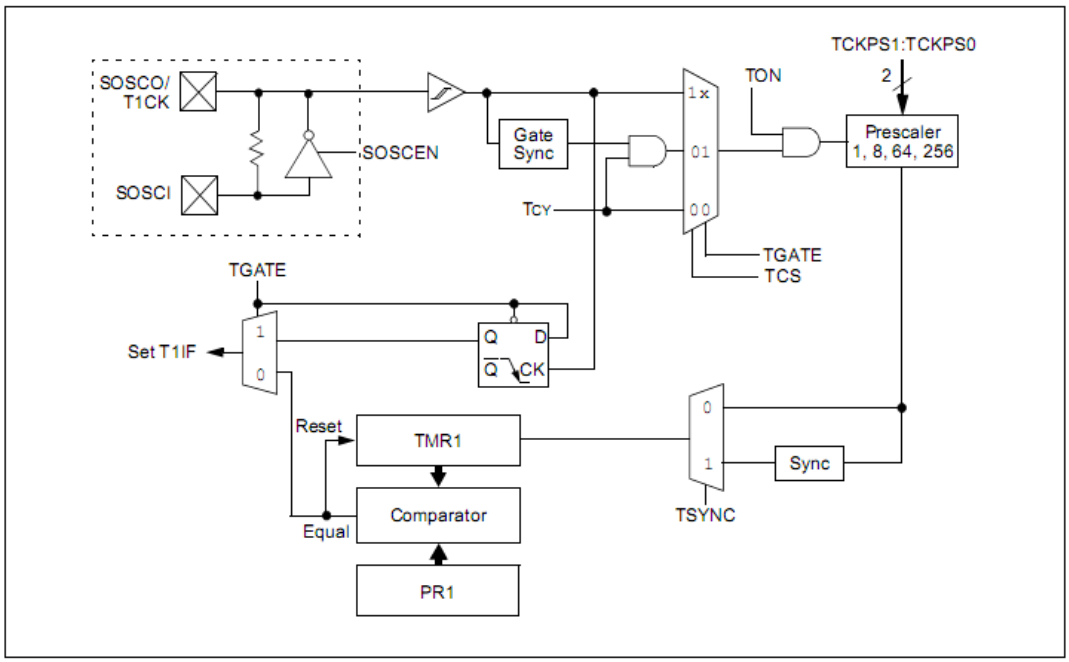
PIC24FJ128GA010 mikrodenetleyicisi beş adet 16 bitlik zamanlayıcı ihtiva eder. Bu zamanlayıcılar Timer1, Timer2/3 ve Timer4/5 olarak ifade edilir.

3.2.7.1 Timer 1

Gerçek zamanlı bir 'clock' için zaman sayıcısı olarak veya serbest çalışan bir zamanlayıcı/sayıcı olarak çalışabilen 16 bitlik bir zamanlayıcıdır. Üç modda işlem yapabilir:

- 16 bit zamanlayıcı
- 16 bit senkron sayıcı
- 16 bit asenkron sayıcı

Ayrıca ayarlanabilir önbölücü özelliklerine sahiptir. Boşta modunda ve uyku modunda da zamanlayıcı işlevi görebilmektedir. Harici bir girişe bağlı olarak kesme özelliği de gösterebilmektedir [43]. Şekil 3.5'te 16 bitlik Timer1 blok diyagramı görülmektedir.



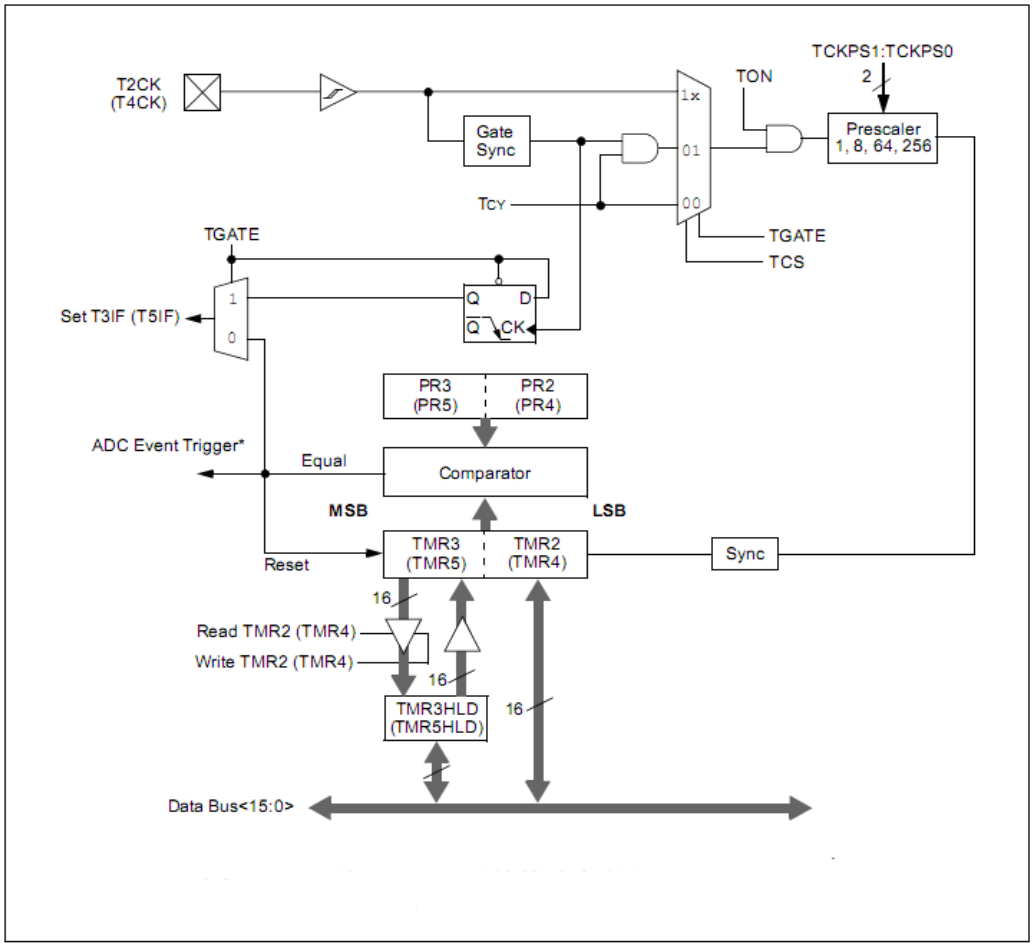
Şekil 3.5 Timer1 blok diyagramı

3.2.7.2 Timer 2/3 ve Timer 4/5

Timer 2/3 ve Timer 4/5 modülleri 32 bitlik zamanlayıcılardır. Aynı zamanda bağımsız olarak 16 bitlik zamanlayıcılar olarak da işlem yapabilirler. 32 bit zamanlayıcı olarak üç modda çalışırlar :

- 16 bitlik işlemlerde iki bağımsız 16 bitlik zamanlayıcı olarak
- 32 bit zamanlayıcı olarak
- 32 bit senkron sayıcı olarak

Ayrıca Timer1'de olduğu gibi bunlar da ayarlanabilir önbölücü özelliğine sahiptir. Boşta ve uyku modlarında çalışabilirler. Kesme işlemlerinde kullanılabilirler. 32 bitlik zamanlayıcı/sayıcı işlemlerinde Timer2 ve Timer4 düşük değerlikli, Timer3 ve Timer5 yüksek değerliklidir [43]. Şekil 3.6'da bu zamanlayıcılara ait blok diyagram görülmektedir.

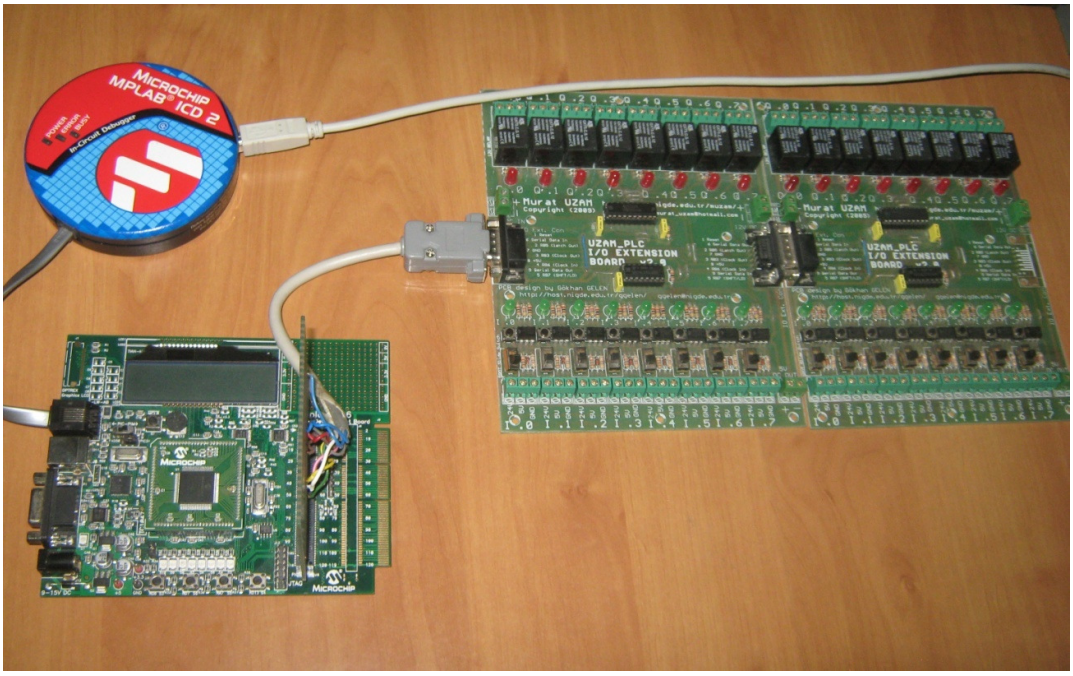


Şekil 3.6 Timer2/3 ve Timer 4/5 32 bit blok diyagramı

BÖLÜM IV

PIC24FJ128GA010 MİKRODENETLEYİCİSİ İLE TASARLANAN 16 GİRİŞLİ/16 ÇIKIŞLI PLC DONANIMI

Bu bölümde; yüksek lisans tez çalışması için tasarlanan 16 girişli/16 çıkışlı PLC donanımı incelenmektedir. Tasarlanan PLC donanımı temel olarak Explorer 16 Geliştirme ve Uygulama Kartı ile UZAM_PLC I/O genişletme kartlarından oluşturmaktadır. Bu iki donanım birbirine Explorer 16 yardımcı kart (Daughter Board) yoluyla bağlanır. Yazılımın yüklenmesi için gerekli sinyallerin giriş/çıkış ünitesine aktarılması için ilgili pinler bir seri port kablosu yoluyla yardımcı karttan dışarı alınır ve UZAM_PLC I/O genişletme kartına bağlanır. Fotoğraf 4.1'de PIC24FJ128GA010 mikrodenetleyicisi ile tasarlanan ve gerçekleştirilen 16 girişli/16 çıkışlı PLC donanımı görülmektedir.

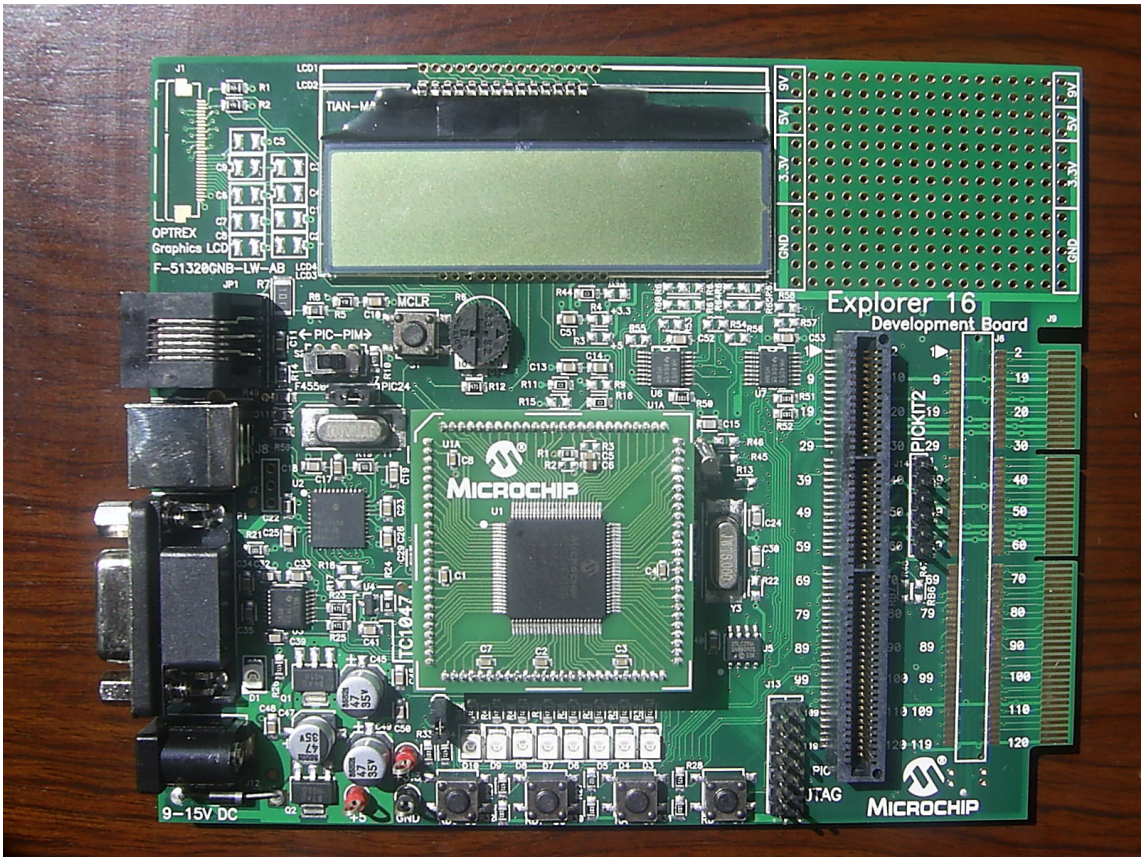


Fotoğraf 4.1 PIC24FJ128GA010 mikrodenetleyicisi ile tasarlanan ve gerçekleştirilen 16 girişli/16 çıkışlı PLC

Gerçekleştirilen bu tasarımda kullanılan Explorer 16 Geliştirme ve Uygulama Kartı, PLC'nin merkezi işlem birimi (CPU) olarak düşünülebilir. Kart üzerindeki PIC24FJ128GA010 mikrodenetleyicisi, yüksek seviyeli yazılım dillerinden olan PIC C dili ile MPLAB IDE arayüz programı ortamında oluşturulan fonksiyonlar yardımıyla programlanmıştır. Çevresel giriş/çıkış birimleri için, UZAM PLC giriş/çıkış modülü için tasarlanmış olan 8 bitlik I/O genişletme kartlarından iki adet kullanılarak 16 bitlik bir giriş/çıkış kartı elde edilmiştir.

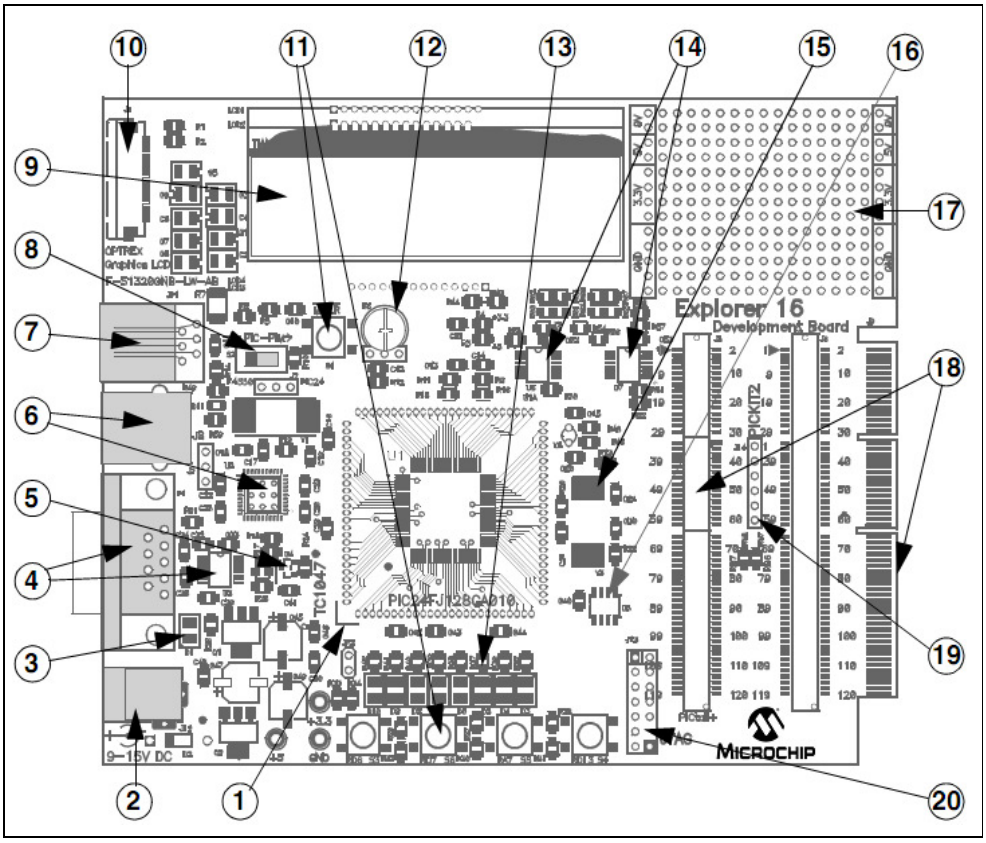
PLC'nin programlanması; merkezi işlem birimi (CPU) olan PIC24FJ128GA010 mikrodenetleyicisinin, fonksiyon oluşturmada da kullanılan MPLAB IDE arayüz programı aracılığıyla programlanması ile yapılır. Programlama dili olarak PIC C dili kullanılmıştır. C dili ile oluşturulan fonksiyon kodları, 'Merdiven (Ladder) Diyagramı Yöntemi'ndeki satırlarda kullanılan PLC kontaklarını ifade etmektedir. PLC programlamada en çok kullanılan yöntem olan 'Merdiven (Ladder) Diyagramı Yöntemi' ; bir PLC arayüz programı geliştirilmesi halinde gerçekleştirilen bu tasarımda rahatlıkla kullanılabilecektir.

4.1 Explorer 16 Geliştirme ve Uygulama Kartı



Fotograf 4.2 Explorer 16 geliştirme ve uygulama kartı

Fotoğraf 4.2'de görülmekte olan Explorer 16 Geliştirme ve Uygulama Kartı; Microchip firmasının ürettiği 100 pinli bir PIC olan PIC24FJ128GA010 mikrodenetleyicisi için bir uygulama seti niteliğindedir. Özellikle PIC24FJ128GA010 mikrodenetleyicisi için tasarlanmıştır. Bununla birlikte tüm PIC24F ve dsPIC33F ailesi için de kullanıma uygundur [43].



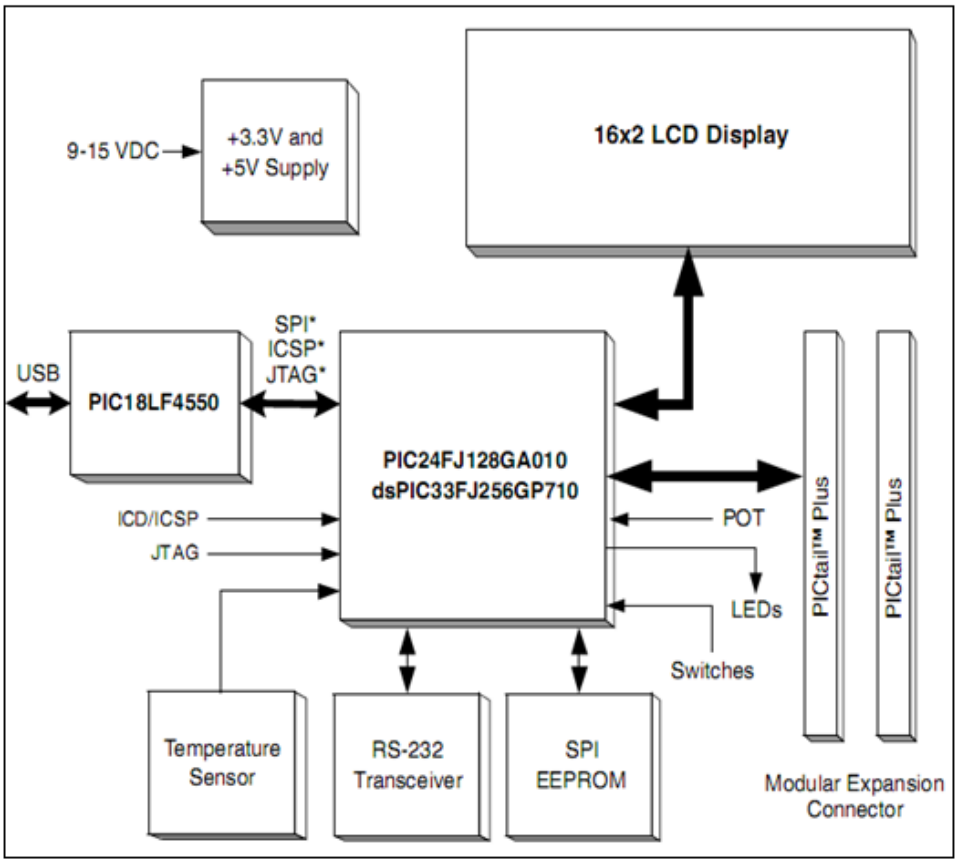
Şekil 4.1 Explorer 16 geliştirme ve uygulama kartı şeması

Şekil 4.1'de görülmekte olan Explorer 16 Geliştirme ve Uygulama Kartı devre şemasında numaralarla ifade edilen kısımlar şunlardır:

1. 100 pinli mikrodenetleyici soket yuvası (PIC24F/24H)
2. +3.3V - +5V DC doğrultulmuş gerilim sağlayan 9V DC güç girişi
3. Güç gösterge ledi
4. RS-232 seri portu
5. Tümleşik analog sıcaklık sensörü
6. USB girişi
7. Devre İçi Hata Ayıklayıcı (In Circuit Debugger - ICD) konektörü
8. İşlemci yükleme modülü (PIM) seçme anahtarı
9. 16 karakter 2 satırlı LCD ekran
10. Grafik LCD eklenti yeri
11. Cihaz reset anahtarı
12. Analog girişler için potansiyometre
13. I/O portlarına bağlı 8 adet led
14. İki adet 74HCT4053 multiplexer
15. 8 MHz kristal osilatör

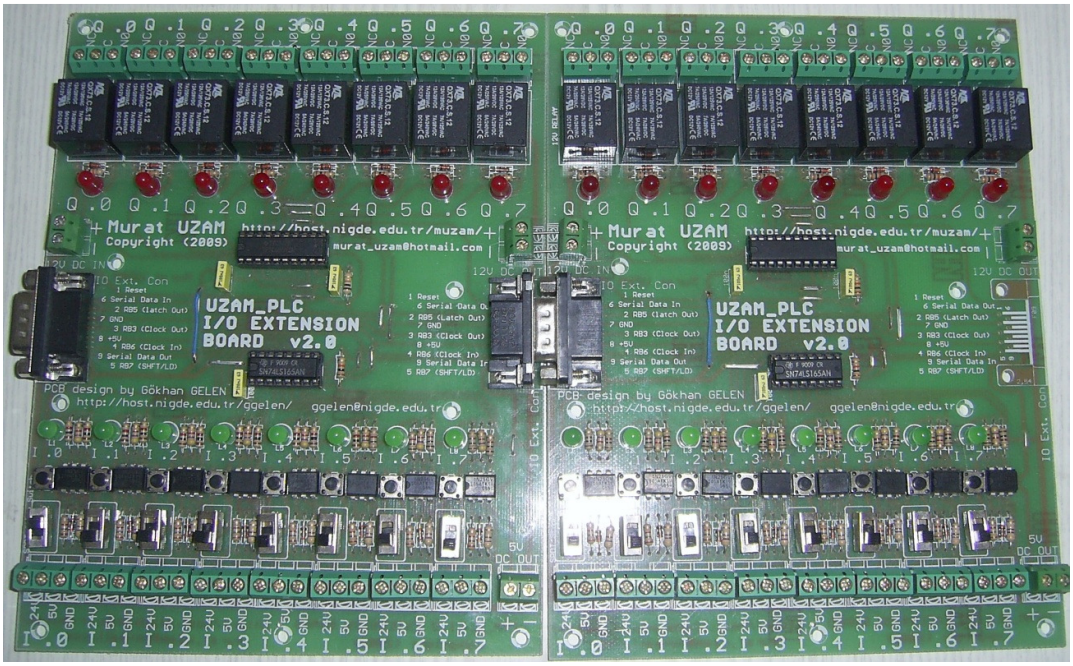
16. Seri EEPROM
17. Geliştirme uygulamaları için prototip alan
18. Yardımcı kartlar (Daughter Board) için soket girişi ve konektör
19. PICkit 2 Programlayıcısı için 6 pinli arayüz
20. JTAG konektör yastığı

Bu kartın beslemesi 9-15V arası bir DC gerilimle sağlanmaktadır. USB ve RS-232 bağlantısı ile haberleşme sağlar. Üzerinde sıcaklık sensörü ve LCD ekran mevcuttur. Program uygulamalarında giriş ve çıkış olarak kullanılabilen LED ve anahtarlar ihtiva etmektedir. Ayrıca genişletilmiş uygulamalar için modüler genişletme konnektörü bulunmaktadır. PIC24FJ128GA010 mikrodenetleyicisi kullanılarak geliştirilen yazılımlar için MPLAB IDE arayüz programı kullanılmaktadır. Mikrodenetleyicinin programlanması MPLAB ICD 2 programlayıcısı aracılığıyla yapılır. Programlayıcı, mikrodenetleyici ile USB bağlantısı yoluyla haberleşmektedir [43]. Şekil 4.2'de Explorer 16 geliştirme kartı blok diyagramı görülmektedir.



Şekil 4.2 Explorer 16 geliştirme kartı blok diyagramı

4.2 UZAM_PLC I/O Genişletme Kartı

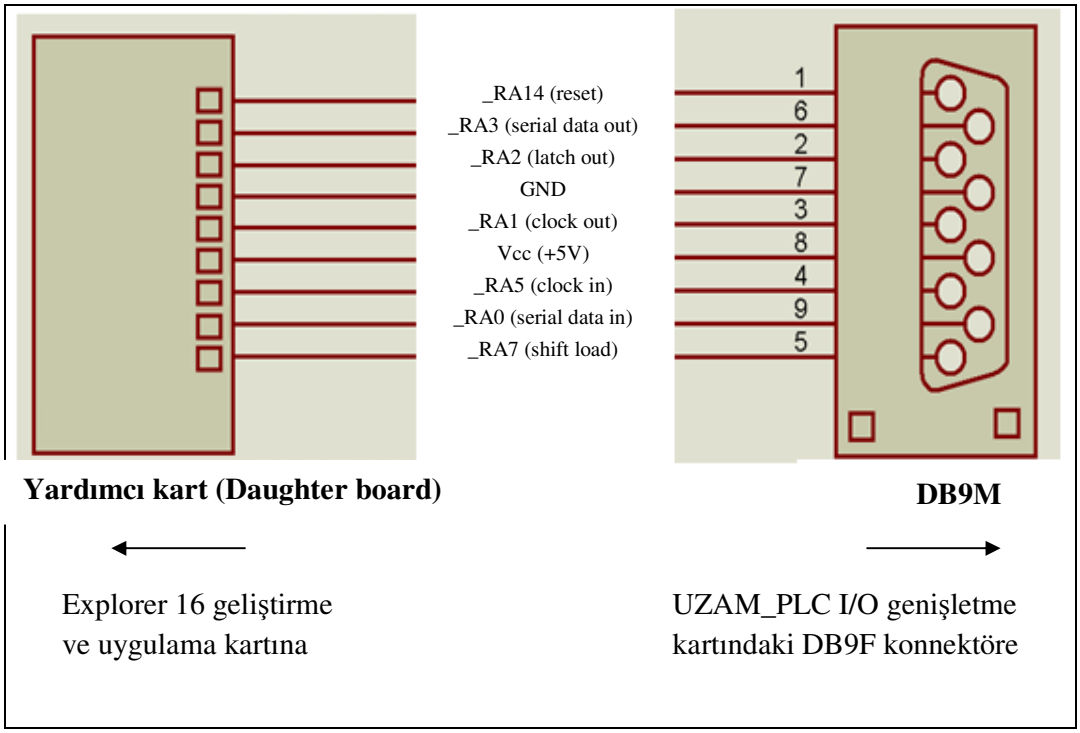


Fotoğraf 4.3 UZAM_PLC I/O genişletme kartı

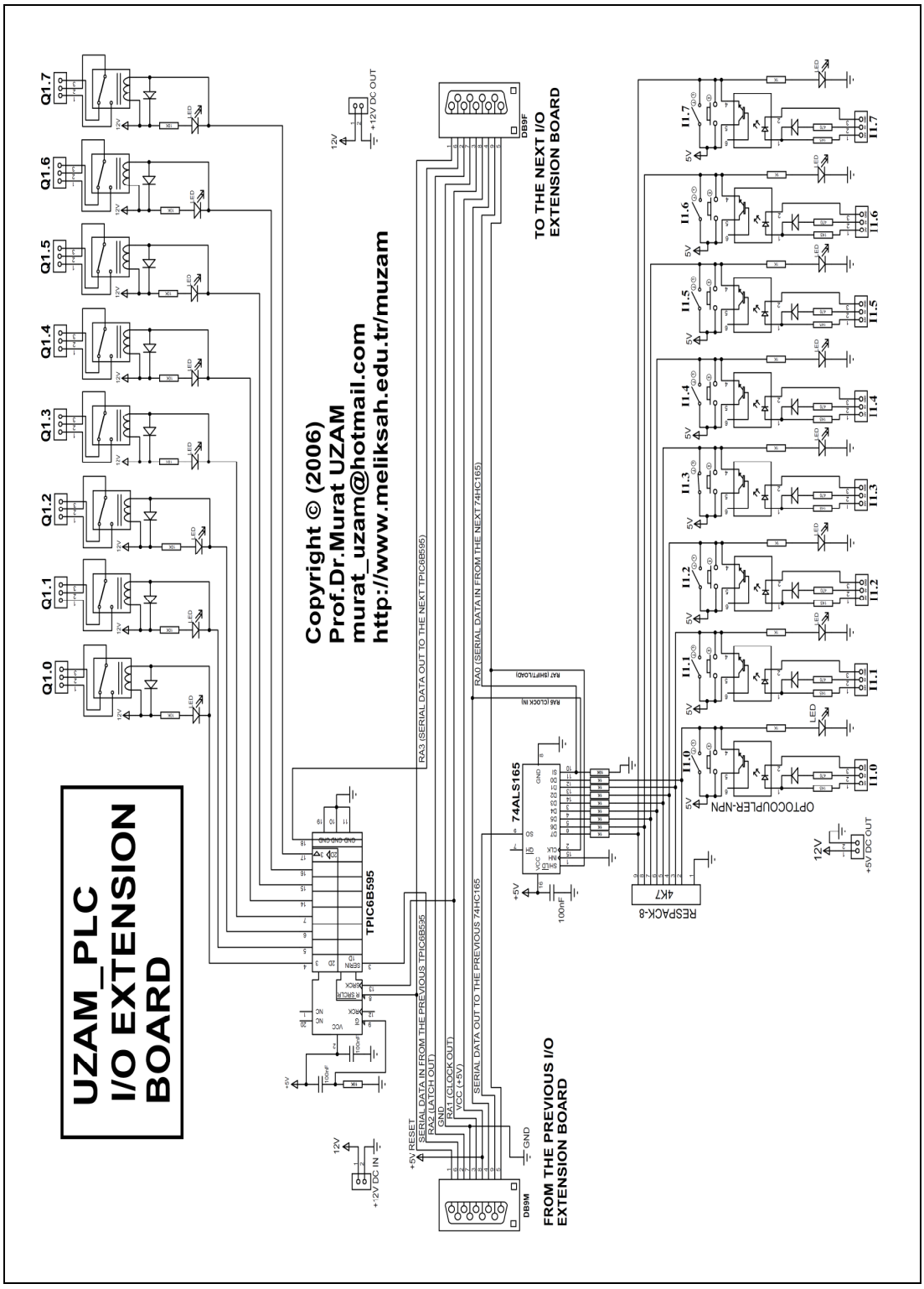
Fotoğraf 4.3'te UZAM_PLC I/O genişletme kartı görülmektedir. 8 girişli/8 çıkışlı iki parça genişletme kartı DB9 konnektörlerle birleştirilmiş ve 16 girişli/16 çıkışlı bir PLC giriş/çıkış ünitesi elde edilmiştir. 16 adet giriş için I0.0, I0.1, I0.2, I0.3, I0.4, I0.5, I0.6, I0.7, I1.0, I1.1, I1.2,I1.3, I1.4, I1.5, I1.6 ve I1.7 olarak isimlendirilmiş 16 adet giriş düzenlenmiştir. Her bir girişten 5V DC ya da 24V DC giriş sinyali uygulanabilmektedir. Harici giriş sinyalleri NPN tipi (4N25) bir optik yalıtıcı (opto-coupler) kullanılarak PLC kartındaki diğer kısımlardan elektriksel olarak izole edilmiştir. Harici girişlerin kullanılmadığı uygulamalarda, girişlerin simülasyonunu yapabilmek için kart üzerinde her bir giriş için ani temaslı bir buton ve bir sürgülü anahtar mevcuttur. Her bir girişte, giriş sinyalinin aktif olması durumunu gösteren bir LED kullanılmıştır. Her biri 8 giriş bilgisini alan iki adet paralel/seri dönüştürücü 74ALS165 entegresi aracılığıyla 16 adet girişten gelen sinyaller alınmaktadır. Girişlerden paralel yükleme yapılarak alınan bilgi paralel/seri dönüşüm yapılarak seri veri yolundan merkezi işlem birimine aktarılır.

16 adet çıkış için de Q0.0, Q0.1, Q0.2, Q0.3, Q0.4, Q0.5, Q0.6, Q0.7, Q1.0, Q1.1, Q1.2,Q1.3, Q1.4, Q1.5, Q1.6 ve Q1.7 olarak isimlendirilmiş 16 adet röle çıkışı düzenlenmiştir. Her biri 8 röleyi süren iki adet seri/paralel dönüştürücü (register/driver) TPIC6B595 entegresi yoluyla 16 adet çıkış sürülmektedir. Seri olarak gelen bilgi seri/paralel dönüşüm yapılarak çıkışlara gönderilir. Devrede, her bir rölenin çalışmakta olduğunu gösteren bir LED mevcuttur [1]. Şekil 4.4'te UZAM_PLC I/O genişletme kartı devre şeması görülmektedir.

Explorer 16 Geliştirme ve Uygulama kartındaki PIC24FJ128GA010 mikrodenetleyicisi ile UZAM_PLC I/O Genişletme Kartı arasındaki bağlantıyı sağlayan yardımcı kart (daughter board) üzerindeki bağlantıların DB9 konnektördeki pinleri Şekil 4.3'te görüldüğü gibidir. Bu bağlantılarda A portunun pinleri kullanılmıştır.



Şekil 4.3 Yardımcı kart bağlantı pinleri



Şekil 4.4 UZAM_PLC I/O genişletme kartına ait devre şeması

BÖLÜM V

PIC24FJ128GA010 MİKRODENETLEYİCİSİ İLE TASARLANAN 16 GİRİŞLİ/16 ÇIKIŞLI PLC YAZILIMI

Bu bölümde; yüksek lisans tez çalışması için tasarlanan ve bir önceki bölümde incelenen 16 girişli/16 çıkışlı PLC donanımını destekleyen PLC yazılımı incelenmektedir. Geliştirilen yazılımda yüksek seviyeli yazılım dillerinden olan 'Pic C' dili kullanılmıştır. Oluşturulan kodlarla temel yazılım fonksiyonları, kontak/röle temelli fonksiyonlar, karşılaştırma fonksiyonları, aritmetik fonksiyonlar, lojik fonksiyonlar, zamanlayıcı fonksiyonları ve sayıcı fonksiyonları gerçekleştirilmiştir. Bu fonksiyonların kullanımı Pic Assembly dilindeki komutların kullanımıyla benzerdir. Ayrıca fonksiyonların içerisinde kullanılan register, pointer ya da argümanların tanımlarının yapıldığı fonksiyonlar tanımlama dosyası olarak ilave edilmiştir.

5.1 Explorer 16 Programlama

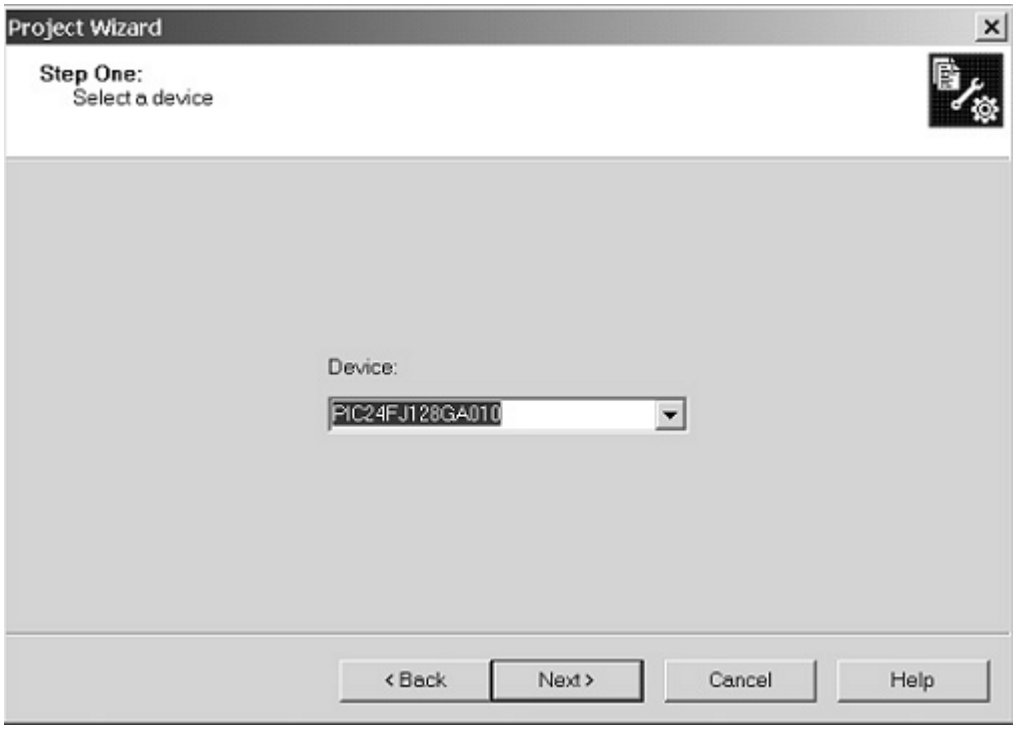
Explorer 16 Geliştirme ve Uygulama kartının ve dolayısıyla PIC24FJ128GA010 mikrodenetleyicisinin programlanması, MPLAB IDE arayüz programı yardımıyla C dili ile MPLAB C30 derleyicisi yapısına göre yapılmaktadır [43].

5.1.1 Proje oluşturma

Başlangıç adımı bir proje ve MPLAB IDE içinde bir çalışma alanı oluşturmaktır. Bir proje herhangi bir uygulama yapmak için gerekli dosyaları içerir (kaynak kod, linker dosyası vs.). Bir çalışma alanı; seçilen mikrodenetleyici ile ilgili bir ya da daha fazla proje ve bilgi, debug/program araçları ile MPLAB IDE konfigürasyon ayarları içermektedir. MPLAB IDE'de yeni bir proje oluşturmak için Project Wizard (proje sihirbazı) isimli yapı kullanılır.

5.1.2 Project wizard

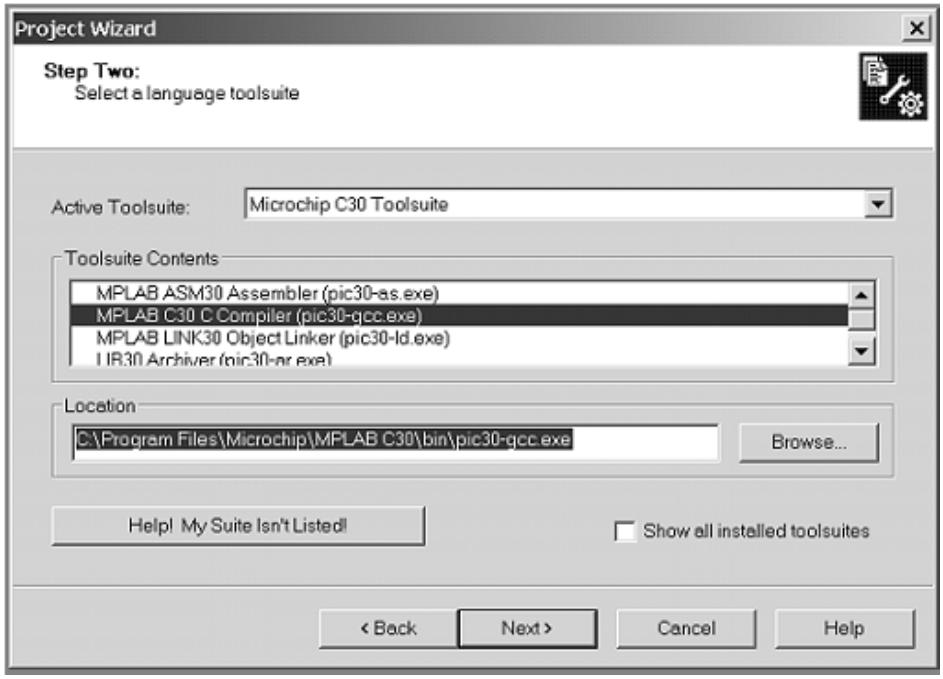
- MPLAB IDE başlatılır.
- Açık olması muhtemel çalışma alanları kapatılır (*File > Close Workspace*)
- *Project* menüsünden *Project Wizard* seçilir.
- Açılan ekranda Next'e basılır ve birinci adıma geçilir.



Şekil 5.1 Cihaz seçme adımı

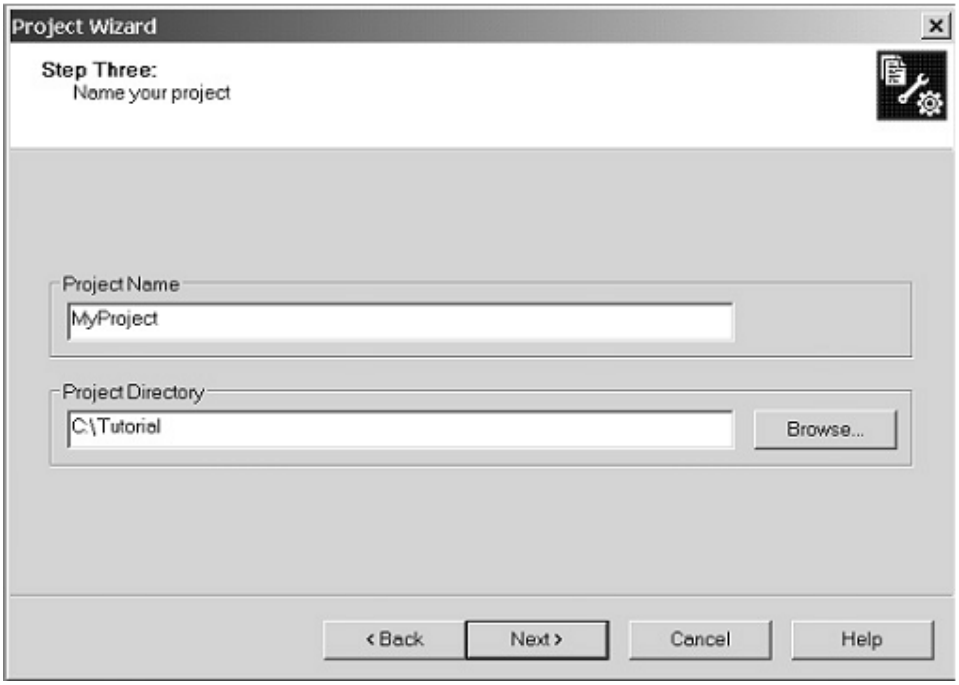
- Şekil 5.1'de görüldüğü gibi birinci adım cihaz seçme adımıdır. Bu aşamada cihaz seçenek listesinden PIC24FJ128GA010 seçilir ve bir sonraki adım için **Next**'e basılır.

- İkinci adım programlama dili ve derleyici seçenekleri kısmıdır. Burada programlama dili olarak MPLAB C30 C derleyicisi (pic30-gcc.exe) seçilir. Program Files içindeki yeri otomatik olarak seçilmiş olur. Dil ve derleyici seçenekleri adımı Şekil 5.2'de görülmektedir.



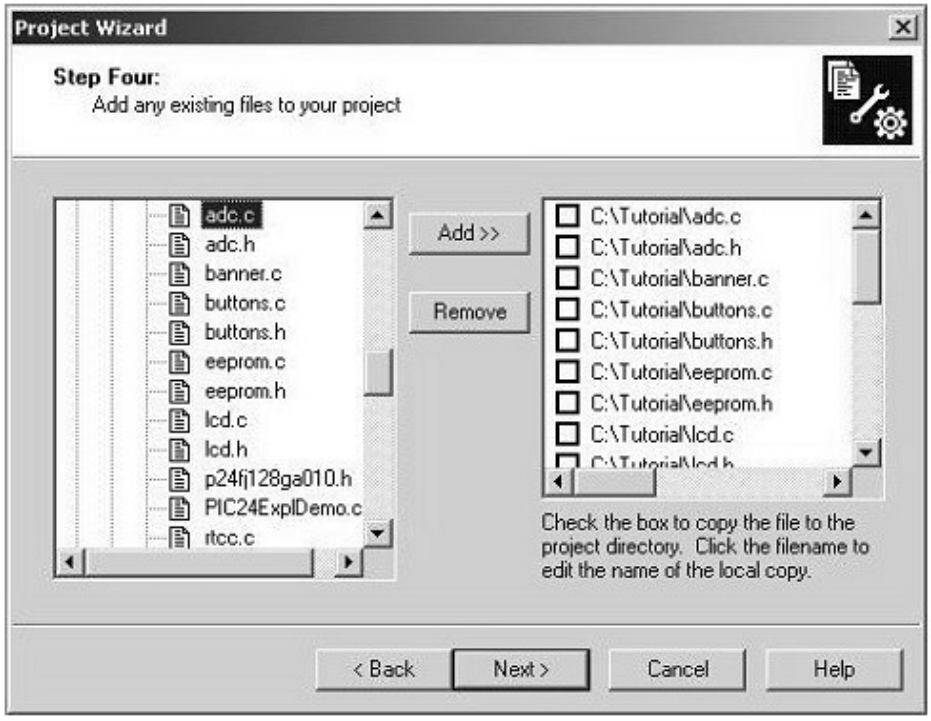
Şekil 5.2 Dil ve derleyici seçenekleri adımı

- Üçüncü adımda proje dosyası oluşturulur ve oluşturulan projeye isim verilir. Şekil 5.3'te dosya oluşturma ve isim verme adımı görülmektedir.



Şekil 5.3 Proje dosyası oluşturma ve isim verme adımı

- Dördüncü adımda projeye kaynak (source-.c) ve başlık (header-.h) dosyaları eklenir.

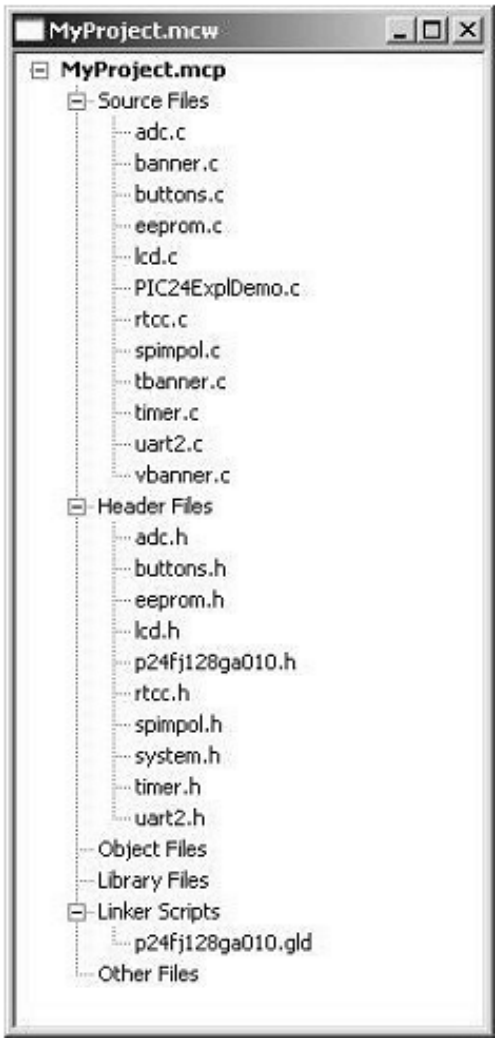


Şekil 5.4 Kaynak (source-.c) ve başlık (header-.h) dosyaları ekleme adımı

- Eklenmek istenen dosya seçilir ve 'Add'e tıklanır. Bunlarla birlikte C:\Program Files\Microchip\MPLAB 30\support\gld klasörü açılır. Buradan p24fj128ga010.gld veya p33fj256gp710ps.gld dosyasının eklenmesi gerekmektedir. Çünkü bu dosyalar 'linker' dosyalarıdır.

- Bundan sonra **Next'**e basılır ve özet ekranı geldiği zaman **Finish'**e basılır ve işlem tamamlanmış olur. Şekil 5.4'te kaynak (source-.c) ve başlık (header-.h) dosyaları ekleme adımı görülmektedir.

Project Wizard tamamlandığında ‘MPLAB Project Window’ açılır ve kaynak, başlık ve linker klasörlerinin içinde kendilerine ait dosyaları görüntülenir. Şekil 5.5’te MPLAB Project Window görülmektedir.



Şekil 5.5 MPLAB Project Window

Bu aşamada MPLAB IDE içinde proje ve çalışma alanı oluşturulmuş durumdadır. ‘MyProject.mcw’ çalışma alanı dosyası, ‘MyProject.mcp’ ise proje dosyasıdır.

5.2 Kod Oluşturma

PIC24FJ128GA010 mikrodenetleyicisi C ve Assembly dilleriyle oluşturulmuş kodlarla programlanabilmektedir. Bu çalışmada programlama dili olarak C dili kullanılmıştır.

5.2.1 C dili ile programlama

Dünyada en yaygın kullanılan programlama dillerinden biri olan C dili ile birçok yazılım etkili bir biçimde yazılabilir. Gerek işletim sistemleri, gerek sürücüler, gerekse de çoklu ortam yazılımları bunlara dâhildir. Aynı şekilde bu dil ile mikrodenetleyiciler de rahatlıkla programlanabilir.

C programlama dili; genel amaçlı orta seviyeli ve yapısal bir programlama dilidir. 1972 yılında Dennis Ritchie tarafından Bell Telefon Labaraturvarı'nda 'Unix' işletim sistemi ile kullanılmak için tasarlanmıştır. C, özellikle sistem programlamada sembolik makine dili (assembler) ile tercih edilmektedir. İşletim sistemleri, derleyiciler ve debug gibi düşük seviyeli sistem programlarının yazılımında yoğun olarak C programlama dili kullanılır.

C'nin yayılması ve gelişmesi, büyük bir bölümü C dili ile yazılan UNIX işletim sisteminin popüler olmasıyla başlamıştır. C programlama dili, hemen her alanda kullanılmaktadır. Günümüzde nesneye yönelik programlama dilleri (C++, Java) ve script dilleri (JavaScript, JavaApplet, PHP) gibi programlama dilleri C programlama dilinden esinlenmiştir. C taşınabilir (portable) bir dildir. Yani herhangi bir C programı hiçbir değişikliğe uğramadan, veya çok az bir değişimle, başka bir derleyicide veya işletim sisteminde derlenebilir. Örneğin; 'Windows' işletim sistemlerinde yazılan bir C kodu, 'Linux', 'UNIX' veya 'VAX' gibi işletim sistemlerinde de derlenebilir [46].

C programlama dilini popüler kılan önemli nedenler aşağıda listelenmiştir:

- C, güçlü ve esnek bir dildir. C ile işletim sistemi veya derleyici yazılabilir, kelime işlemciler oluşturulabilir veya grafik çizilebilir.
- C, iyi bir yazılım geliştirme ortamına sahiptir.
- C, özel komut ve veri tipi tanımlamasına izin verir.
- C, taşınabilir bir dildir. Aynı kod çoğunlukla farklı C derleyicilerinde çalışabilir.

- C, gelişimini tamamlamış ve standardı oluşmuş bir dildir.
- C, yapısal bir dildir. C kodları 'fonksiyon' olarak adlandırılan alt programlardan oluşmuştur.
- C++, Java, Java Script, Java Applet, PHP, C# gibi diller C dilinden esinlenmiştir.

C programları veya kaynak kodları (source code) uzantısı '.c' olan dosyalarda saklanır. Kaynak kod, bir C derleyicisi (C compiler) ile nesne koduna (object code) daha sonra uygun bir bağlayıcı (linker) programı ile işletim sisteminde çalıştırılabilen (executable) bir koda dönüştürülür [46].

5.2.2 C kodu temel özellikleri

Bir C programı aşağıda verilen özellikleri mutlaka taşımalıdır.

- Yazılımda kullanılacak olan her fonksiyon için ilgili başlık dosyası programın başına ilave edilmelidir.
- Her C programı main() fonksiyonunu içermelidir.
- Program içinde kullanılacak olan değişkenler ve sabitleri mutlaka tanımlanmalıdır.
- Satırın sonuna ' ; ' işareti koyulmalıdır.
- Her bloğun ve fonksiyonun başlangıcı ve bitişi sırasıyla '{' ve '}' sembolleridir.
- C dilinde yazılan kodlarda küçük-büyük harf ayrımı vardır (case sensitive).
- Açıklama operatörü '/*, */' sembolleridir.

5.2.3 Veri tipleri

Veri tipi (data type) program içinde kullanılacak değişken, sabit, fonksiyon isimleri gibi tanımlayıcıların tipini, yani bellekte ayrılacak bölgenin büyüklüğünü belirlemek için kullanılır. Bir programcı, bir programlama dilinde ilk olarak öğrenmesi gereken, o dile ait veri tipleridir. Çünkü bu, programcının kullanacağı değişkenlerin ve sabitlerin sınırlarını belirler. C programlama dilinde altı tane temel veri tipi bulunmaktadır.

Bu altı temel veri tipi şunlardır:

- bit
- char
- int
- float
- long
- double

- **'bit'** veri tipinde sayılar Boolean (0 veya 1) olabilir. Örneğin, aşağıdaki tanımda değişken flag sadece 0 veya 1 değerlerini alabilir.
  bit flag;

- **'char'** veri tipinin iki çeşidi vardır. **'unsigned char'** ile 8 bitlik sayılar tanımlanır ve bu sayılar 0 ve 255 arasında değer alabilirler. Örneğin, aşağıdaki veri tanımında değişken q, 0 ve 255 arasında herhangi bir değer alabilir ve burada q 180 olarak değerlendirilmiştir. Aynı zamanda, değişken a karakter R'ye eşitlenmiştir.
  unsigned char q,a;
  q = 180;
  a='R';

- İşaret içeren **'signed char'** veri tanımı, -128 ve +127 arasındaki sayıları tanımlamak için kullanılır. Aşağıdaki örnekte değişken p'ye '-50' değeri ve değişken z'ye '28' değeri verilmiştir.
  signed char p,z;
  p = -50;
  z = 28;

- **'int'** veri tipi de iki çeşittir. **'unsigned int'** ile 16 bit olan ve 0 ile 65535 arasında değişen sayılar tanımlanabilir. Aşağıdaki örnekte, değişken q'ya 915 ve değişken x'e 3000 değerleri verilmiştir.

  unsigned int q;
  q = 915;
  x = 3000;

- İşaret içeren ve 16 bitlik sayılar için kullanılan **'signed int'** tanımında, sayının '-32768' ve '+32767' arasında olması gerekir. Aşağıdaki örnekte değişken t'ye '-500' değeri verilmiştir.

  signed int t;
  t = -500;

- 32 bit ve işaret içeren sayılar için kullanılan **'signed long'** tanımında sayının '-2147483648' ve '+2147483647' arasında olması gerekir. Örneğin, aşağıdaki tanımda değişken 'm' long olarak tanımlanmış ve değeri '200000' yapılmıştır.

  signed long m;
  m = 200000;

- 32 bit işaretsiz sayılar için kullanılan **'unsigned long'** tanımda sayıların '0' ve '4294967295' arasında olması gerekir. Aşağıdaki örnekte değişken 'k' unsigned long olarak tanımlanmış ve değeri '2000000' yapılmıştır.

  unsigned long k;
  k =2000000;

- **'float'** tanımı 24 veya 32 bitlik kayan nokta değişkenler için kullanılır. Kayan nokta değişkenler matematik işlemlerinde çok önem taşımaktadırlar. Aşağıdaki örnekte değişken temp'e '1.23' değeri verilmiştir.

  float temp;
  temp = 1.23;

5.2.4 Aritmetik operatörler

Değişken veya sabitler üzerinde temel aritmetik işlemleri gerçekleyen operatörlerdir. Bunlar Çizelge 5.1'de listelenmiştir.

Çizelge 5.1 Aritmetik operatörler

Operatör	Açıklama	Örnek	Anlamı
+	toplama	`x + y`	x ve y nin toplamı
-	çıkarma	`x - y`	x ve y nin farkı
*	carpma	`x * y`	x ve y nin çarpımı
/	bölme	`x / y`	x ve y nin oranı
%	artık bölme	`x % y`	x / y den kalan sayı

5.2.5 Atama operatörleri

Bu operatörler bir değişkene, bir sabit veya bir aritmetik ifade atamak (eşitlemek) için kullanılır. Bu operatörler Çizelge 5.2'de listelenmiştir.

Çizelge 5.2 Atama operatörleri

Operatör	Açıklama	Örnek	Anlamı
=	atama	`x = 7;`	`x = 7;`
+=	ekleyerek atama	`x += 3`	`x = x + 3`
-=	eksilterek atama	`x -= 5`	`x = x - 5`
*=	çarparak atama	`x *= 4`	`x = x * 4`
/=	bölerek atama	`x /= 2`	`x = x / 2`
%=	bölüp, kalanını atama	`x %= 9`	`x = x % 9`
++	bir arttırma	x++ veya ++x	`x = x + 1`
--	bir azaltma	x-- veya --x	`x = x - 1`

5.2.6 Mantıksal operatörler

C dilinde, bir mantıksal işlemin sonucu tamsayı 0 (sıfır) veya başka bir değer olur. Çizelge 5.3'te mantıksal operatörler listelenmiştir.

Çizelge 5.3 Mantıksal operatörler

Operatör	Açıklama	Örnek	Anlamı
&&	mantıksal VE	`x>2 && x<y`	x, 2 den büyük VE y den küçük mü?
||	mantıksal VEYA	`x>2 || x<y`	x, 2 den büyük VEYA y den küçük mü?

5.2.7 Karşılaştırma operatörleri

Çizelge 5.4'te listelenen karşılaştırma operatörleri, sayısal değerleri veya karakterleri mukayese etmek için kullanılır.

Çizelge 5.4 Karşılaştırma operatörleri

Operatör	Açıklama	Örnek	Anlamı
>	büyüktür	`x > y`	x, y den büyük mü?
<	küçüktür	`x < y`	x, y den küçük mü?
==	eşittir	`x == y`	x, y ye eşit mi?
>=	büyük-eşittir	`x >= y`	x, y den büyük yada eşit mi?
<=	küçük-eşittir	`x <= y`	x, y den küçük yada eşit mi?
!=	eşit değil	`x != y`	x, y den farklı mı?

5.2.8 'if' ve 'else if' yapısı

Bu deyimler, koşullu işlem yapan deyimlerdir. 'if' ve 'else' tek bir karşılaştırma deyimi olup else kullanımı isteğe bağlıdır. Eğer bu koşul olumlu ise 'if' den sonraki bölüm yürütülür ve else den sonraki bölüm atlanır. Koşul olumsuz ise 'if' den sonraki küme atlanır ve eğer varsa, else den sonraki kümedeki işlemler gerçekleştirilir.

'if' deyiminin yapının genel biçimi şöyledir:

```
if (koşul)
{
  ...
   deyimler; (küme)
  ...
}
```

'if' deyimi kullanılırken kümenin başlangıcı ve bitişini gösteren, küme parantezleri kullanılmasında kullanıcıya bir esneklik sunulmuştur. Eğer 'if' deyiminden sonra icra edilecek deyimler tek satırdan oluşuyorsa, bu işaretlerin kullanılması zorunlu değildir. Yani, 'if' deyimden sonra '{' ve '}' işaretleri kullanılmamışsa, bu deyimi takip eden sadece ilk satır işleme konur.

Buna göre aşağıdaki kullanım

```
if(x == y){
  puts("x ve y esit");
}                                    ile

if(x == y)
  puts("x ve y esit")              eşdeğerdir.
```

if deyiminin else ile birlikte kullanımı şu şekildedir:

```
if (koşul) {
  ...
  deyimler; (küme1)
  ...
}

 else{
  ...
  deyimler; (küme2)
  ...
}
```

5.2.9 'while' döngüsü

Tekrarlama deyimidir. Bir küme ya da deyim **while** kullanılarak birçok kez yinelenebilir. Yinelenmesi için koşul sınaması döngüye girilmeden yapılır. Koşul olumlu olduğu sürece çevrim yinelenir. Genel yazım biçimi:

```
while(koşul)
    {
     ...
      döngüdeki deyimler; [küme]
     ...
    }
```

5.2.10 'for' döngüsü

Bu deyim, diğer döngü deyimleri gibi bir kümeyi bir çok kez tekrarlamak için kullanılır. Koşul sınaması 'while' da olduğu gibi döngüye girmeden yapılır. Bu döngü deyimin içinde diğerlerinden farklı olarak başlangıç değeri ve döngü sayacına sahip olmasıdır.

Genel yazım biçimi:

for (*başlangıç ; koşul ; artım*)
{ ...
döngüdeki deyimler;
...}

5.2.11 Sonsuz döngü

Bir döngü işlemini sonsuz kere tekrarlarsa bu döngü sonsuz döngü olarak adlandırılır. Böyle bir döngü için, koşul çok önemlidir. Örneğin while döngüsü için

```
while(1)
{
  i = i++
}
```

5.3 Gerçekleştirilen PLC'nin Temel Yazılımı

Bir PLC'de; 1- Girişleri Güncelleme, 2- Kullanıcı Programını Tarama, 3- Çıkışları Güncelleme işlemleri sonsuz bir döngü içinde sürekli gerçekleştirilmekte ve PLC çalıştığı sürece bu programlar tekrarlanmaktadır. Program sonsuz döngüye girmeden önce başlangıç şartlarının düzenlenmesi gerekir.

Temel yazılım olarak kullanılan iki ana fonksiyon "get_inputs( )" ve "send_outputs( )" fonksiyonlarıdır. "get_inputs( )" fonksiyonu girişleri güncelleme, "send_outputs( )" fonksiyonu ise çıkışları güncelleme fonksiyonudur. Kullanıcı yazacağı programı bir 'while' döngüsü içerisinde bu iki ana fonksiyon arasında gerçekleştirir.

"get_inputs( )" fonksiyonu 'get_inputs.c' isimli bir C dosyası olarak, "send_outputs( )" fonksiyonu da 'send_outputs.c' isimli bir C dosyası olarak projeye eklenmiştir.

Temel yazılımda bu fonksiyonlara ek olarak, diğer tüm fonksiyonlarda sıkça kullanılan değişkenlerin tanımlarının yapıldığı tanımlama fonksiyonları da mevcuttur. Bu tanımlamalar 'p24FJ_PLC' isimli bir C dosyasına kaydedilmiş ve projeye eklenmiştir.

5.3.1 Temel yazılım ve tanımlama fonksiyonları

Yazılımın temelini teşkil eden; giriş/çıkış birimleri, zamanlayıcılar (timers), işaretçiler (markers), durum bitleri, flip-flop temelli fonksiyonlar birer kaydedici durumunda olup program hafızasında belirli adreslerde yer almaktadır. Bunlar 16 bitlik hafıza yapıları (union) olarak tüm bitleri ayrı ayrı tanımlanacak şekilde program hafızasına yerleştirilmiştir. Çizelge 5.5'te 16 bitlik hafıza yapısı (union) ve bu yapıdaki kaydedici tanımlamaları görülmektedir. Benzer 16 bitlik hafıza yapıları ekte verilmiştir.

Çizelge 5.5 16 bitlik hafıza yapısı (union) ve tanımlamaları

```
extern union R16bit
{
    unsigned int bits;
       struct {
                unsigned n0:1;
                unsigned n1:1;
                unsigned n2:1;
                unsigned n3:1;
                unsigned n4:1;
                unsigned n5:1;
                unsigned n6:1;
                unsigned n7:1;
                unsigned n8:1;
                unsigned n9:1;
                unsigned n10:1;
                unsigned n11:1;
                unsigned n12:1;
                unsigned n13:1;
                unsigned n14:1;
                unsigned n15:1;
               };
};
```

```
extern union    R16bit   INPUTS;
extern union    R16bit   OUTPUTS;
extern union    R16bit   TIMERS;
extern union    R16bit   TEMPS;
extern union    R16bit   MARKER0;
extern union    R16bit   MARKER1;
extern union    R16bit   MARKER2;
extern union    R16bit   MARKER3;
extern union    R16bit   REDS;
extern union    R16bit   FEDS;
extern union    R16bit   LATCHS;
extern union    R16bit   SRFFS;
```

Bu 16 bitlik hafıza yapıları 'p24FJ_PLC.h' isimli header dosyasında 'extern union' olarak tanımlanmış ve program hafızasına 16 bitlik bölümler olarak kaydedilmiştir. Yine bu dosyada tüm bitleri teker teker tanımlanmıştır. p24FJ_PLC.h' dosyası ekte verilmiştir. Çizelge 5.6'da bit tanımlamaları görülmektedir.

Çizelge 5.6 16 bitlik hafıza yapılarının bit tanımlamaları

#define Q00 OUTPUTS.n0 #define Q01 OUTPUTS.n1 #define Q02 OUTPUTS.n2 #define Q03 OUTPUTS.n3 #define Q04 OUTPUTS.n4 #define Q05 OUTPUTS.n5 #define Q06 OUTPUTS.n6 #define Q07 OUTPUTS.n7 #define Q10 OUTPUTS.n8 #define Q11 OUTPUTS.n9 #define Q12 OUTPUTS.n10 #define Q13 OUTPUTS.n11 #define Q14 OUTPUTS.n12 #define Q15 OUTPUTS.n13 #define Q16 OUTPUTS.n14 #define Q17 OUTPUTS.n15 #define OUTPUT OUTPUTS.bits	#define I00 INPUTS.n0 #define I01 INPUTS.n1 #define I02 INPUTS.n2 #define I03 INPUTS.n3 #define I04 INPUTS.n4 #define I05 INPUTS.n5 #define I06 INPUTS.n6 #define I07 INPUTS.n7 #define I10 INPUTS.n8 #define I11 INPUTS.n9 #define I12 INPUTS.n10 #define I13 INPUTS.n11 #define I14 INPUTS.n12 #define I15 INPUTS.n13 #define I16 INPUTS.n14 #define I17 INPUTS.n15 #define INPUT INPUTS.bits	#define Tm00 TIMERS.n0 #define Tm01 TIMERS.n1 #define Tm02 TIMERS.n2 #define Tm03 TIMERS.n3 #define Tm04 TIMERS.n4 #define Tm05 TIMERS.n5 #define Tm06 TIMERS.n6 #define Tm07 TIMERS.n7 #define Tm10 TIMERS.n8 #define Tm11 TIMERS.n9 #define Tm12 TIMERS.n10 #define Tm13 TIMERS.n11 #define Tm14 TIMERS.n12 #define Tm15 TIMERS.n13 #define Tm16 TIMERS.n14 #define Tm17 TIMERS.n15 #define TIMER TIMERS.bits
#define RED0 REDS.n0 #define RED1 REDS.n1 #define RED2 REDS.n2 #define RED3 REDS.n3 #define RED4 REDS.n4 #define RED5 REDS.n5 #define RED6 REDS.n6 #define RED7 REDS.n7 #define RED8 REDS.n8 #define RED9 REDS.n9 #define RED10 REDS.n10 #define RED11 REDS.n11 #define RED12 REDS.n12 #define RED13 REDS.n13 #define RED14 REDS.n14 #define RED15 REDS.n15	#define FED0 FEDS.n0 #define FED1 FEDS.n1 #define FED2 FEDS.n2 #define FED3 FEDS.n3 #define FED4 FEDS.n4 #define FED5 FEDS.n5 #define FED6 FEDS.n6 #define FED7 FEDS.n7 #define FED8 FEDS.n8 #define FED9 FEDS.n9 #define FED10 FEDS.n10 #define FED11 FEDS.n11 #define FED12 FEDS.n12 #define FED13 FEDS.n13 #define FED14 FEDS.n14 #define FED15 FEDS.n15	#define LB0 LATCHS.n0 #define LB1 LATCHS.n1 #define LB2 LATCHS.n2 #define LB3 LATCHS.n3 #define LB4 LATCHS.n4 #define LB5 LATCHS.n5 #define LB6 LATCHS.n6 #define LB7 LATCHS.n7 #define LB8 LATCHS.n8 #define LB9 LATCHS.n9 #define LB10 LATCHS.n10 #define LB11 LATCHS.n11 #define LB12 LATCHS.n12 #define LB13 LATCHS.n13 #define LB14 LATCHS.n14 #define LB15 LATCHS.n15
#define SR0 SRFFS.n0 #define SR1 SRFFS.n1 #define SR2 SRFFS.n2 #define SR3 SRFFS.n3 #define SR4 SRFFS.n4 #define SR5 SRFFS.n5 #define SR6 SRFFS.n6 #define SR7 SRFFS.n7 #define SR8 SRFFS.n8 #define SR9 SRFFS.n9 #define SR10 SRFFS.n10 #define SR11 SRFFS.n11 #define SR12 SRFFS.n12 #define SR13 SRFFS.n13 #define SR14 SRFFS.n14 #define SR15 SRFFS.n15	#define M00 MARKER0.n0 #define M01 MARKER0.n1 #define M02 MARKER0.n2 #define M03 MARKER0.n3 #define M04 MARKER0.n4 #define M05 MARKER0.n5 #define M06 MARKER0.n6 #define M07 MARKER0.n7 #define M10 MARKER0.n8 #define M11 MARKER0.n9 #define M12 MARKER0.n10 #define M13 MARKER0.n11 #define M14 MARKER0.n12 #define M15 MARKER0.n13 #define M16 MARKER0.n14 #define M17 MARKER0.n15 #define M0 MARKER0.bits	#define M20 MARKER1.n0 #define M21 MARKER1.n1 #define M22 MARKER1.n2 #define M23 MARKER1.n3 #define M24 MARKER1.n4 #define M25 MARKER1.n5 #define M26 MARKER1.n6 #define M27 MARKER1.n7 #define M30 MARKER1.n8 #define M31 MARKER1.n9 #define M32 MARKER1.n10 #define M33 MARKER1.n11 #define M34 MARKER1.n12 #define M35 MARKER1.n13 #define M36 MARKER1.n14 #define M37 MARKER1.n15 #define M1 MARKER1.bits

Çizelge 5.6. 16 bitlik hafıza yapılarının bit tanımlamaları (devam)

#define M40 MARKER2.n0	#define M60 MARKER3.n0
#define M41 MARKER2.n1	#define M61 MARKER3.n1
#define M42 MARKER2.n2	#define M62 MARKER3.n2
#define M43 MARKER2.n3	#define M63 MARKER3.n3
#define M44 MARKER2.n4	#define M64 MARKER3.n4
#define M45 MARKER2.n5	#define M65 MARKER3.n5
#define M46 MARKER2.n6	#define M66 MARKER3.n6
#define M47 MARKER2.n7	#define M67 MARKER3.n7
#define M50 MARKER2.n8	#define M70 MARKER3.n8
#define M51 MARKER2.n9	#define M71 MARKER3.n9
#define M52 MARKER2.n10	#define M72 MARKER3.n10
#define M53 MARKER2.n11	#define M73 MARKER3.n11
#define M54 MARKER2.n12	#define M74 MARKER3.n12
#define M55 MARKER2.n13	#define M75 MARKER3.n13
#define M56 MARKER2.n14	#define M76 MARKER3.n14
#define M57 MARKER2.n15	#define M77 MARKER3.n15
#define M2 MARKER2.bits	#define M3 MARKER3.bits

Bunlara ek olarak, fonksiyonların çalışması esnasında anlık bilginin tutulduğu değişkenler de mevcuttur. Bu değişkenler, 16 bitlik veya 16 bitlik 64'lü gruplar şeklindedir. Program hafızasında kaydedici olarak yer alan bu değişkenlerin tanımlanması Şekil 5.6'da görülmektedir. Benzer kaydedici ve değişkenlerin tanımlandığı dosyalar ekte verilmiştir.

```
# define   TEMP        TEMPS.n0
# define   LOGIC0     0X0000
# define   LOGIC1     0X0001

 unsigned int         TEMP_REG;
 unsigned int         CV[64];
 unsigned char        CQ[64];
 unsigned char        CRED[64];
 unsigned int          TON[64];
 unsigned char        TONQ[64];
 unsigned char        TONRED[64];
 unsigned int          TOF[64];
 unsigned char         TOFQ[64];
 unsigned char        TOFRED[64];
```

Şekil 5.6 Durum bitleri ve tanımlamaları

Burada **TEMP_REG** geçici bilgi tutan 16 bitlik bir değişkendir. **CV**, **CQ** ve **CRED** sayıcılara ait; **TON, TONQ** ve **TONRED** düz zaman rölesine ait; **TOF, TOFQ** ve **TOFRED** ise ters zaman rölesine ait değişkenleri ifade etmektedir.

Temel yazılımın iki ana fonksiyonu olan "get_inputs( )" ve "send_outputs( )" fonksiyonlarının işlevi olan girişlerden bilgi alma ve çıkışlara bilgi gönderme işlemi kontrol sinyalleri yardımıyla gerçekleştirilir. Kontrol sinyalleri **RCK, SRCK, SERIN, RST, SDO, CLKIN** ve **ShLd** olup Şekil 5.7'de tanımlamaları görülmektedir.

```
//…… definition of I/O pins……..
#define RCK      _RA2     // latch out
#define SRCK     _RA1     // clk out
#define SERIN    _RA0     // serial data in
#define RST      _RA14    // reset
#define SDO      _RA3     // Serial data out
#define CLKIN    _RA5     // Clk in
#define ShLd     _RA7     // Shift load
```

Şekil 5.7 Kontrol sinyalleri tanımlamaları

Tanımlamalar için mikrodenetleyicinin A portunun bazı pinleri kullanılmıştır. Bu kontrol sinyallerinden **SDO, CLKIN** ve **ShLd** "get_inputs( )" programına; **RCK, SRCK, SERIN** ve **RST** ise "send_outputs( )" programına aittir. "get_inputs( )" ve "send_outputs( )" C kodları ekte verilmiştir. "get_inputs( )" program kodu Şekil 5.8'de görülmektedir. Bu programın amacı; giriş verisini, 16 girişin bağlı olduğu kaskad bağlı iki tane 74ALS165 paralelden seriye dönüştüren kaydedici üzerinden mikrodenetleyici içerisinde tanımlı olan INPUT isimli 16 bitlik kaydediciye aktarmaktır.

Programın başında sıfırda aktif olan **ShLd** (_RA7) girişi ile anlık bir düşen kenar tetiklemesi oluşturularak girişlerdeki 16 bitlik verinin aynı anda kaskad bağlı iki tane 74ALS165 paralelden seriye dönüştüren kaydediciye yüklenmesi sağlanır. Daha sonra **SDO** (_RA3) girişi yardımıyla 74ALS165 kaydedicilerinde yüklü 16 bitlik veriyi INPUT kaydedicisine aktarmak için kod yazılmıştır. Kaskad bağlı iki 74ALS165'in clock girişlerine **CLKIN** (_RA5) pini yardımıyla uygulanan toplam 16 düşen kenar sinyali ile 16 bitlik giriş verisi INPUT kaydedicisine aktarılmış olur.

```
#include "p24fj128ga010.h"
#include "p24fj_plc.h"
#include "stdio.h"

#define bit_get(p,m) ((p) & (m))
#define bit_set(p,m) ((p) |= (m))
#define bit_clear(p,m) ((p) &= ~(m))
#define bit_flip(p,m) ((p) ^= (m))
#define bit_write(c,p,m) (c ? bit_set(p,m) : bit_clear(p,m))
#define BIT(x) (0x01 << (x))

unsigned int get_inputs(void)
{               int k;

            ShLd=     0;
            Nop()
            ShLd=     1;

            if (SDO==0)                                 // I0.7
                        bit_clear(INPUT,BIT(7));
            else
                        bit_set(INPUT,BIT(7));

            CLKIN = 0;
                        for(k=1;k>0;k--)
            CLKIN = 1;
            if (SDO==0)                                 //I0.6
                        bit_clear(INPUT,BIT(6));
            else
                        bit_set(INPUT,BIT(6));

             CLKIN = 0;
                        for(k=1;k>0;k--)
             CLKIN = 1;

             if (SDO==0)                                //I0.5
                        bit_clear(INPUT,BIT(5));
            else
                        bit_set(INPUT,BIT(5));

             CLKIN = 0;
                        for(k=1;k>0;k--)
             CLKIN = 1;

             if (SDO==0)                                //I0.4
                        bit_clear(INPUT,BIT(4));
            else
                        bit_set(INPUT,BIT(4));

              CLKIN = 0;
                        for(k=1;k>0;k--)
              CLKIN = 1;
```

Şekil 5.8 “get_inputs.c” program kodu

```
if (SDO==0)                                  //I0.3
        bit_clear(INPUT,BIT(3));
else
        bit_set(INPUT,BIT(3));

 CLKIN = 0;
        for(k=1;k>0;k--)
 CLKIN = 1;

 if (SDO==0)                                 //I0.2
        bit_clear(INPUT,BIT(2));
else
        bit_set(INPUT,BIT(2));

 CLKIN = 0;
        for(k=1;k>0;k--)
 CLKIN = 1;

 if (SDO==0)                                 //I0.1
        bit_clear(INPUT,BIT(1));
else
        bit_set(INPUT,BIT(1));

 CLKIN = 0;
        for(k=1;k>0;k--)
 CLKIN = 1;

 if (SDO==0)                                 //I0.0
        bit_clear(INPUT,BIT(0));
else
        bit_set(INPUT,BIT(0));

 CLKIN = 0;
        for(k=1;k>0;k--)
 CLKIN = 1;

 if (SDO==0)                                 //I1.7
        bit_clear(INPUT,BIT(15));
else
        bit_set(INPUT,BIT(15));

 CLKIN = 0;
        for(k=1;k>0;k--)
 CLKIN = 1;

 if (SDO==0)                                 //I1.6
        bit_clear(INPUT,BIT(14));
else
        bit_set(INPUT,BIT(14));

 CLKIN = 0;
         for(k=1;k>0;k--)
 CLKIN = 1;
```

Şekil 5.8 “get_inputs.c” program kodu (devam)

```
        if (SDO==0)                             //I1.5
                bit_clear(INPUT,BIT(13));
        else
                bit_set(INPUT,BIT(13));

        CLKIN = 0;
                for(k=1;k>0;k--)
        CLKIN = 1;

        if (SDO==0)                             //I1.4
                bit_clear(INPUT,BIT(12));
        else
                bit_set(INPUT,BIT(12));

        CLKIN = 0;
                for(k=1;k>0;k--)
        CLKIN = 1;

        if (SDO==0)                             //I1.3
                bit_clear(INPUT,BIT(11));
        else
                bit_set(INPUT,BIT(11));

        CLKIN = 0;
                for(k=1;k>0;k--)
        CLKIN = 1;

        if (SDO==0)                             //I1.2
                bit_clear(INPUT,BIT(10));
        else
                bit_set(INPUT,BIT(10));

        CLKIN = 0;
                for(k=1;k>0;k--)
        CLKIN = 1;

        if (SDO==0)                             //I1.1
                bit_clear(INPUT,BIT(9));
        else
                bit_set(INPUT,BIT(9));

        CLKIN = 0;
                for(k=1;k>0;k--)
        CLKIN = 1;

        if (SDO==0)                             //I1.0
                bit_clear(INPUT,BIT(8));
        else
                bit_set(INPUT,BIT(8));
        return 0;
}
```

Şekil 5.8 “get_inputs.c” program kodu (devam)

```
  unsigned int send_outputs (void)
{
          int i;

          RST=0;
          Nop()
          RST=1;

          RCK = 0;
          for(i=0;i<16;i++)
            { if (( OUTPUT & 0x8000) == 0)
                SERIN = 0;
              else
                SERIN = 1;
                OUTPUT=OUTPUT<<1;
                SRCK = 1;
                Nop()
               SRCK = 0;
             }
         if (i==16)
         RCK = 1;
         return 0;
  }
```

Şekil 5.9 send_outputs.c program kodu

'send_outputs.c' program kodu Şekil 5.9'da görülmektedir. Bu programın amacı mikrodenetleyicinin içerisinde bulunan ve çıkışlara gönderilecek olan çıkış verisini tutan OUTPUT isimli 16 bitlik kaydedicideki veriyi giriş/çıkış kartı üzerinde kaskad bağlı olan iki tane TPIC6B595 seriden paralele dönüştüren kaydediciye göndermektir.

Programın başında TPIC6B595'ler **RST** (_RA14) çıkış pini üzerinden uygulanan bir lojik 0 sinyali yardımıyla sıfırlanır. OUTPUT kaydedicisinin en küçük basamağındaki bitinden başlamak üzere bitin 0 ya da 1 olduğu tespit edilir. İlgili bit 0 (1) ise **SERIN** (_RA0) çıkışındaki sinyal 0 (1) yapılır. Bu sinyal daha sonra **SRCK** pini üzerinden kaskad bağlı TPIC6B595 kaydedicilerine uygulanan clock sinyali ile dışarı gönderilir. Bu işlem benzer şekilde 16 kez tekrarlanarak çıkış verisinin OUTPUT kaydedicisinden çıkışların bağlı olduğu TPIC6B595 kaydedicilerine aktarılması sağlanır.

5.3.2 Ana program

Bu yazılımın ana programı ‘main (void)’dir. Portların giriş/çıkış tanımlamaları ile zamanlayıcı, zamanlayıcı kesme, zamanlama önbölücü ve periyod konfigürasyonları burada tanımlanmaktadır. ‘get_inputs ( )’ ve ‘send_outputs ( )’ programları burada kullanılır. Kullanıcı esas PLC programını ‘get_inputs ( )’ ve ‘send_outputs ( )’ fonksiyonları arasındaki kısıma yazar. Şekil 5.10’da ana fonksiyon ‘main (void)’ görülmektedir. ‘main (void)’ C kodu ekte kullanıcı örnekleriyle birlikte verilmiştir.

```
#include "p24FJ128GA010.h"
#include "p24FJ_PLC.h"
#include "macro.h"


int main(void)
{
      TRISA = 0X0008;
       _T1IP = 4;                        // this is the default value anyway
       TMR1 = 0;                         // clear the timer
       PR1 = 0xffff-1;//25000-1;         // set the period register

       // 2.1 configure Timer1 module
       T1CON = 0x8030;                   // enabled, prescaler 1:64, internal clock

       // 2.2 init the Timer 1 Interrupt, clear the flag, enable the source
       _T1IF = 0;
       _T1IE = 1;

       // 2.3 init the processor priority level
       _IPL = 0;                         // this is the default value anyway

RST=0;
Nop()
RST=1;

while(1)  {
get_inputs();
TIMER = TMR1;

//---------- KULLANICI PROGRAMI BURADAN BAŞLAR----------




//---------- KULLANICI PROGRAMI BURADA BİTER----------
send_outputs();
                }
}
```

Şekil 5.10 main (void) programı

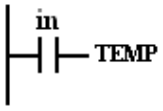
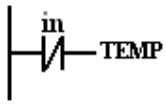
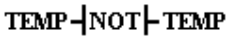
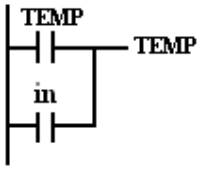
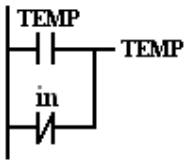
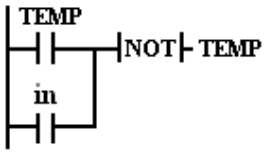
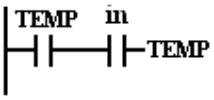
5.4 Kontak ve Röle Temelli Fonksiyonlar

Kontak ve röle temelli fonksiyonlar bir bitlik (ikili-binary) değişkenler için tanımlanmıştır. İşlemler **in** ve **TEMP** adında iki değişken yardımıyla gerçekleştirilir. İlgili fonksiyonun argümanına karşılık gelen **in** kaydedicisinden gelen bilgi **TEMP** kaydedicisi yoluyla aktarılır. **sr_ff** (set-reset flip-flop) fonksiyonları dışındaki tüm fonksiyonlar; yine bir bitlik olan TEMP kaydedicisi üzerinden bilgi transferi yapar [26]. Gerçekleştirilen 18 adet kontak ve röle temelli fonksiyon Çizelge 5.7, 5.8, 5.9, 5.10 ve 5.11'de doğruluk tabloları ve sembolleriyle birlikte verilmiştir. Kontak ve röle temelli fonksiyonlar şunlardır:

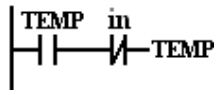
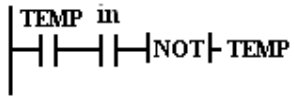
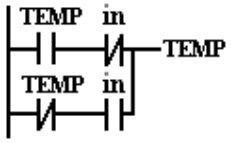
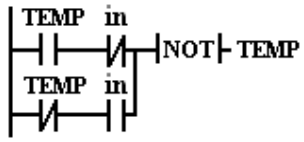
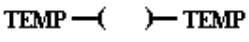
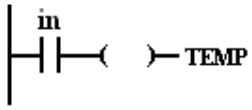
1. **ld** – Merdiven diyagramında normalde açık kontak (normally open - NO) ile ifade edilen bu fonksiyonda, bilgi 'in' argümanından alınır ve TEMP üzerinden dışarı gönderilir. Gelen bilgi 0 ise çıkış (TEMP) 0, bilgi 1 ise çıkış (TEMP) 1 yapılır.
2. **ld_not** – Merdiven diyagramında normalde kapalı kontak (normally closed - NC) ile ifade edilen bu fonksiyonda, bilgi 'in' argümanından alınır ve TEMP üzerinden dışarı gönderilir. Gelen bilgi 0 ise çıkış (TEMP) 1, bilgi 1 ise çıkış (TEMP) 0 yapılır.
3. **not** – Merdiven diyagramında lojik DEĞİL (NOT) kapısı olarak kullanılan bu fonksiyonda, kendinden önceki fonksiyondan gelen giriş bilgisi TEMP üzerinden alınır. Aynı şekilde çıkışa TEMP üzerinden verilir. Gelen bilgi 0 ise çıkış (TEMP) 1, bilgi 1 ise çıkış (TEMP) 0 yapılır.
4. **or** – Merdiven diyagramında iki girişli bir lojik VEYA (OR) kapısı olarak kullanılan bu fonksiyonda; girişlerden biri fonksiyon argümanı olan 'in' değişkeninden, diğeri ise bir önceki fonksiyondaki bilgiyi tutan TEMP'ten alınır. İşlem sonucu tekrar TEMP ile dışarı gönderilir.
5. **or_not** – Girişlerden birisinin değili alınmış iki girişli bir lojik VEYA (OR) kapısı olarak kullanılan bu fonksiyonda; girişlerden biri 'in' değişkeninden, diğeri ise TEMP'ten alınır. TEMP'ten alınan bilginin değili alınarak işlem yapılır. İşlem sonucu TEMP ile dışarı gönderilir.
6. **nor** – Merdiven diyagramında iki girişli bir lojik VEYA DEĞİL (NOR) kapısı olarak kullanılan bu fonksiyonda; girişlerden biri 'in' değişkeninden, diğeri ise TEMP'ten alınır. İşlem sonucu TEMP ile dışarı gönderilir.

7. **and** – Merdiven diyagramında iki girişli bir lojik VE (AND) kapısı olarak kullanılan bu fonksiyonda; girişlerden biri ‘in’ değişkeninden, diğeri ise TEMP’ten alınır. İşlem sonucu TEMP ile dışarı gönderilir.
8. **and_not** – Girişlerden birisinin değili alınmış iki girişli bir lojik VE (AND) kapısı olarak kullanılan bu fonksiyonda; girişlerden biri ‘in’ değişkeninden, diğeri ise TEMP’ten alınır. TEMP’ten alınan bilginin değili alınarak işlem yapılır. İşlem sonucu TEMP ile dışarı gönderilir.
9. **nand** – İki girişli bir VE DEĞİL (NAND) kapısı olarak kullanılan bu fonksiyonda; girişlerden biri ‘in’ değişkeninden, diğeri ise TEMP’ten alınır. İşlem sonucu TEMP kaydedicisiyle dışarı gönderilir.
10. **xor** – İki girişli bir ÖZEL VEYA (EX-OR) kapısı olarak kullanılan bu fonksiyonda; girişlerden biri ‘in’ değişkeninden, diğeri ise TEMP’ten alınır. İşlem sonucu TEMP ile dışarı gönderilir.
11. **xnor** – İki girişli bir ÖZEL VEYA DEĞİL (EX-NOR) kapısı olarak kullanılan bu fonksiyonda girişlerden biri ‘in’ değişkeninden, diğeri ise TEMP’ten alınır. İşlem sonucu TEMP ile dışarı gönderilir.
12. **out** – Bu fonksiyonda ikili bilgi TEMP ile aktarılır ve çıkış yine TEMP ile dışarı gönderilir. Merdiven diyagramında bu fonksiyon röle (dâhili ya da harici) çıkışı olarak ifade edilir. Giriş değeri 0 ise, çıkış (TEMP) 0 yapılır; eğer giriş değeri 1 ise, çıkış (TEMP) 1 yapılır.
13. **out_not** – Bu fonksiyonda ikili bilgi TEMP ile aktarılır ve çıkış yine TEMP ile dışarı gönderilir. Merdiven diyagramında bu fonksiyon değillenmiş röle (dâhili ya da harici) çıkışı olarak ifade edilir. Giriş değeri 0 ise, çıkış (TEMP) 1 yapılır; eğer giriş değeri 1 ise, çıkış (TEMP) 0 yapılır.
14. **in_out** – Bu fonksiyonda ikili bilgi ‘in’ değişkeni ile aktarılır ve çıkış TEMP ile dışarı gönderilir. Giriş bilgisi 1 ise çıkış (TEMP) 1, 0 ise çıkış (TEMP) 0 yapılır.

Çizelge 5.7 Kontak ve röle temelli fonksiyonlar

<table>
<tr><th>C Kodu</th><th>Doğruluk tablosu</th><th>Merdiven diyagramı sembolü</th></tr>
<tr>
<td>void ld(unsigned int in)
{
if (in==0) TEMP=0;
else TEMP=1;
}</td>
<td><table><tr><th>GİRİŞ</th><th>ÇIKIŞ</th></tr><tr><th>in</th><th>TEMP</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table></td>
<td> 
</td>
</tr>
<tr>
<td>void ld_not(unsigned int in)
{
if (in==0) TEMP=1;
else TEMP=0;
}</td>
<td><table><tr><th>GİRİŞ</th><th>ÇIKIŞ</th></tr><tr><th>in</th><th>TEMP</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table></td>
<td> 
</td>
</tr>
<tr>
<td>unsigned int not (void)
{
if (TEMP) TEMP=0;
else TEMP=1;
return TEMP;
}</td>
<td><table><tr><th>GİRİŞ</th><th>ÇIKIŞ</th></tr><tr><th>TEMP</th><th>TEMP</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table></td>
<td> 
</td>
</tr>
<tr>
<td>unsigned int or (unsigned int in)
{
if (TEMP || in) TEMP=1;
else TEMP=0;
return TEMP;
}</td>
<td><table><tr><th>GİRİŞ1</th><th>GİRİŞ2</th><th>ÇIKIŞ</th></tr><tr><th>TEMP</th><th>in</th><th>TEMP</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table></td>
<td> 
</td>
</tr>
<tr>
<td>unsigned int or_not (unsigned int in)
{
if (TEMP==1 || in==0)
TEMP=1;
else TEMP=0;
return TEMP;
}</td>
<td><table><tr><th>GİRİŞ1</th><th>GİRİŞ2</th><th>ÇIKIŞ</th></tr><tr><th>TEMP</th><th>in</th><th>TEMP</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table></td>
<td> 
</td>
</tr>
<tr>
<td>unsigned int nor (unsigned int in)
{
if (TEMP==0 && in==0)
TEMP=1;
else TEMP=0;
return TEMP;
}</td>
<td><table><tr><th>GİRİŞ1</th><th>GİRİŞ2</th><th>ÇIKIŞ</th></tr><tr><th>TEMP</th><th>in</th><th>TEMP</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table></td>
<td> 
</td>
</tr>
<tr>
<td>unsigned int and (unsigned int in)
{ if (TEMP && in) TEMP=1;
else TEMP=0;
return TEMP;
}</td>
<td><table><tr><th>GİRİŞ1</th><th>GİRİŞ2</th><th>ÇIKIŞ</th></tr><tr><th>TEMP</th><th>in</th><th>TEMP</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table></td>
<td> 
</td>
</tr>
</table>

Çizelge 5.7 Kontak ve röle temelli fonksiyonlar (devam)

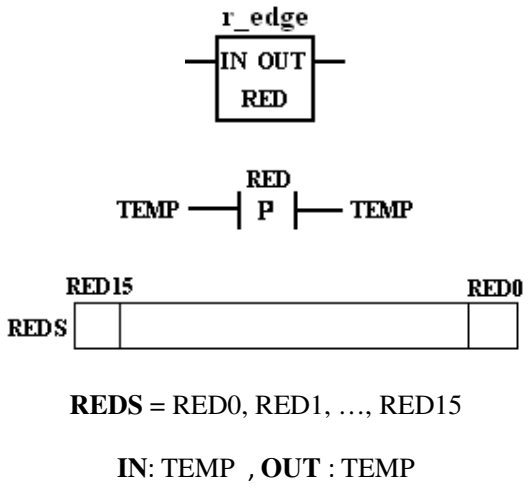
unsigned int and_not (unsigned int in) { if (TEMP==1 && in==0) TEMP=1; else TEMP=0; return TEMP;}	GİRİŞ1 (TEMP) / GİRİŞ2 (in) / ÇIKIŞ (TEMP) 0 0 0 0 1 0 1 0 1 1 1 0	
unsigned int nand (unsigned int in) { if (TEMP && in) TEMP=0; else TEMP=1; return TEMP;}	GİRİŞ1 (TEMP) / GİRİŞ2 (in) / ÇIKIŞ (TEMP) 0 0 1 0 1 0 1 0 0 1 1 0	
unsigned int xor (unsigned int in) { if (TEMP != in) TEMP=1; else TEMP=0; return TEMP;}	GİRİŞ1 (TEMP) / GİRİŞ2 (in) / ÇIKIŞ (TEMP) 0 0 0 0 1 1 1 0 1 1 1 0	
unsigned int xnor (unsigned int in) { if (TEMP == in) TEMP=1; else TEMP=0; return TEMP;}	GİRİŞ1 (TEMP) / GİRİŞ2 (in) / ÇIKIŞ (TEMP) 0 0 1 0 1 0 1 0 0 1 1 1	
unsigned int out (void) { if (TEMP==0) TEMP=0; else TEMP=1; return TEMP;}	GİRİŞ (TEMP) / ÇIKIŞ (TEMP) 0 0 1 1	
unsigned int out_not (void) { if (TEMP==0) TEMP=1; else TEMP=0; return TEMP;}	GİRİŞ (TEMP) / ÇIKIŞ (TEMP) 0 1 1 0	
unsigned int in_out(unsigned int in) { if (in) TEMP=1; else TEMP=0; return TEMP;}	GİRİŞ (in) / ÇIKIŞ (TEMP) 0 0 1 1	

15. **r_edge** – Bu fonksiyon yükselen kenar algılayıcı olarak işlem yapmaktadır. **IN** ile ifade edilen ikili değişken TEMP kaydedicisi ile alınır ve dışarıya yine TEMP üzerinden gönderilir. 'r_edge' fonksiyonu çalıştırılmadan önce **IN** giriş sinyali TEMP kaydedicisine yüklenmiş olmalıdır. Fonksiyonun sonunda OUT çıkış sinyali TEMP'e yüklenmiş olacaktır. Merdiven diyagramında pozitif geçiş algılama kontağı (positive transition-sensing contact) olarak isimlendirilen bu fonksiyon **--| P |--** sembolüyle ifade edilmektedir. **IN** giriş değişkeni 0'dan 1'e değiştiği anda **OUT** çıkış değişkeni sadece bir PLC tarama süresi için 1 yapılır. Diğer tüm durumlarda **OUT** 0 olarak kalır. Çizelge 5.8'de "**r_edge**" fonksiyonu ve bu fonksiyon için kullanılan zamanlama diyagramı ile iki farklı sembol görülmektedir.

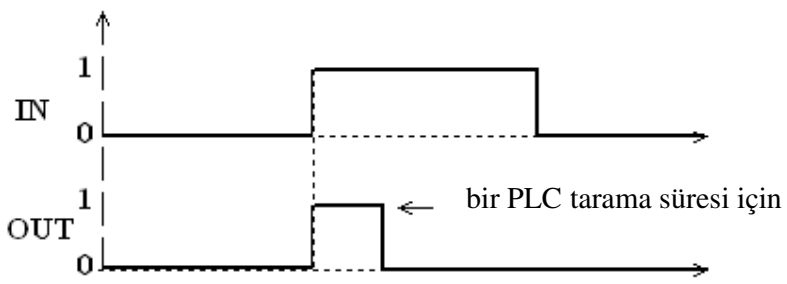
Çizelge 5.8 "**r_edge**" fonksiyonu, sembolleri ve zaman diyagramı

<table>
<tr><th>C Kodu</th><th>Sembol</th></tr>
<tr><td>

```
unsigned int r_edge (void)
{
        if(TEMP==0)
        {  RED=1;  }
        else if (RED)
            {  TEMP=1;
               RED=0; }
            else
                {
                TEMP=0;
                }
        return TEMP;
}
```

</td><td>


REDS = RED0, RED1, …, RED15

IN: TEMP , OUT : TEMP
</td></tr>
<tr><td colspan="2">


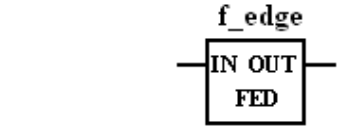
</td></tr>
</table>

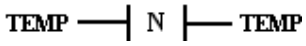
16. **f_edge** – Bu fonksiyon düşen kenar algılayıcı olarak işlem yapmaktadır. **IN** ile ifade edilen ikili değişken TEMP kaydedicisi ile alınır ve dışarıya yine TEMP üzerinden gönderilir. 'f_edge' fonksiyonu çalıştırılmadan önce **IN** giriş sinyali TEMP'e yüklenmiş olmalıdır. Fonksiyonun sonunda **OUT** çıkış sinyali TEMP'e yüklenmiş olacaktır. Merdiven diyagramında negatif geçiş algılama kontağı (negative transition-sensing contact) olarak isimlendirilen bu fonksiyon **--| N |--** sembolüyle ifade edilmektedir. **IN** giriş değişkeni 1'den 0'a değiştiği anda **OUT** çıkış değişkeni sadece bir PLC tarama süresi için 1 yapılır. Diğer tüm durumlarda **OUT** 0 olarak kalır. Çizelge 5.9'da "**f_edge**" fonksiyonu ve bu fonksiyon için kullanılan zaman diyagramı ile iki farklı sembol görülmektedir.

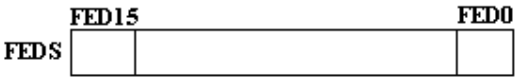
Çizelge 5.9 "**f_edge**" fonksiyonu, sembolleri ve zaman diyagramı

<table>
<tr><th>C Kodu</th><th>Sembol</th></tr>
<tr><td>

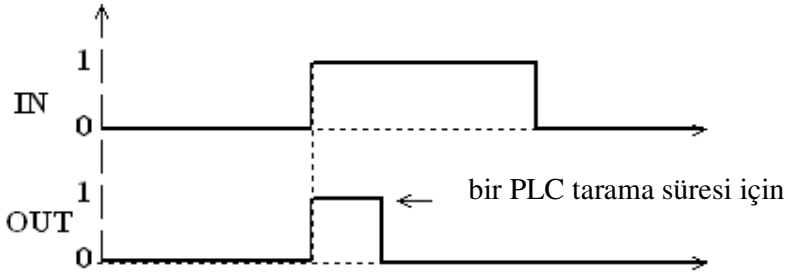
```
unsigned int f_edge (void)
{
        if(TEMP)
        {
        FED=1;
        TEMP=0;
        }
        else if (FED)
             {  TEMP=1;
                FED=0;}
             else
             {TEMP=0;}
             return TEMP;
}
```

</td><td>






FEDS = FED0, FED1, …, FED15

IN:TEMP , OUT : TEMP
</td></tr>
<tr><td colspan="2">


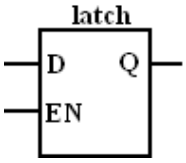
</td></tr>
</table>

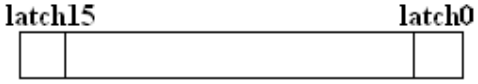
17. **latch** – Bu fonksiyon aktif 1 yetkileme girişli bir D mandalı tanımlamaktadır. Fonksiyonda iki giriş ve bir çıkış değişkeni mevcuttur. Aktif 1 yetkileme girişi (active high enable input) **EN** fonksiyon çalıştırılmadan önce TEMP'e yüklenir. Aktif 1 yetkileme girişi **EN** = 0 olduğunda **Q** çıkışında bir değişim olmaz ve **Q** o anki değerini korur. Aktif 1 yetkileme girişi **EN** = 1 olduğunda ise **D** girişindeki lojik değer **Q** çıkışına yüklenir. Çizelge 5.10'da "**latch**" fonksiyonu, bu fonksiyon için kullanılan iki farklı sembol ve bu fonksiyonun nasıl çalıştığını ifade eden doğruluk tablosu görülmektedir.

Çizelge 5.10 "**latch**" fonksiyonu, sembolleri ve doğruluk tablosu

<table>
<tr><th>C Kodu</th><th>Sembol</th></tr>
<tr><td>

```
unsigned int latch (unsigned   int D)
{
      if (TEMP) LB=D;
      return LB;
}
```

</td><td>


EN: TEMP
D: LB (0…15)
Q: LB (0….15)


</td></tr>
<tr><td colspan="2">Doğruluk tablosu</td></tr>
<tr><td colspan="2">

EN	D	Q_t	Q_{t+1}	YORUM
0	×	Q_t	Q_t	Değişim yok
1	0	×	0	Reset
1	1	×	1	Set

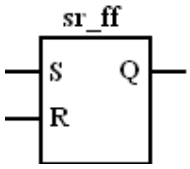
× : ne olursa olsun (don't care)
</td></tr>
</table>

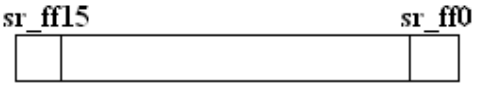
18. **sr_ff** – Bu fonksiyon bir set-reset flip-flop tanımlamaktadır. Fonksiyonda iki giriş ve bir çıkış değişkeni mevcuttur. Giriş değişkenlerinin durumuna göre çıkış değişkeni 0 ya da 1 yapılır. S = 1 ve R = 0 durumunda çıkış değişkeni 1 yapılır. S = 0 ve R = 1 durumunda ise çıkış değişkeni 0 yapılır. Her ikisinin de 0 olması durumunda çıkış değişkeni bir önceki değerini korur. Her ikisinin de 1 olması durumu geçersiz durumdur. Bu durumda çıkış 0 olur. Çizelge 5.11'de "**sr_ff**" fonksiyonu, bu fonksiyon için kullanılan iki farklı sembol ve bu fonksiyonun nasıl çalıştığını ifade eden doğruluk tablosu görülmektedir.

Çizelge 5.11 "**sr_ff**" fonksiyonu, sembolleri ve doğruluk tablosu

<table>
<tr><th>C Kodu</th><th>Sembol</th></tr>
<tr><td>

```
unsigned int sr_ff (unsigned int S, unsigned int
R)

{
      if ((S)&& (R==0))     SR=1;
      if ((R)&& (S==0))     SR=0;
      return SR;
}
```

</td><td>


S: 0 veya 1(Boole giriş)
R: 0 veya 1(Boole giriş)
Q: SR(0…15) (Boole çıkış)


</td></tr>
<tr><td colspan="2">Doğruluk tablosu</td></tr>
<tr><td colspan="2">

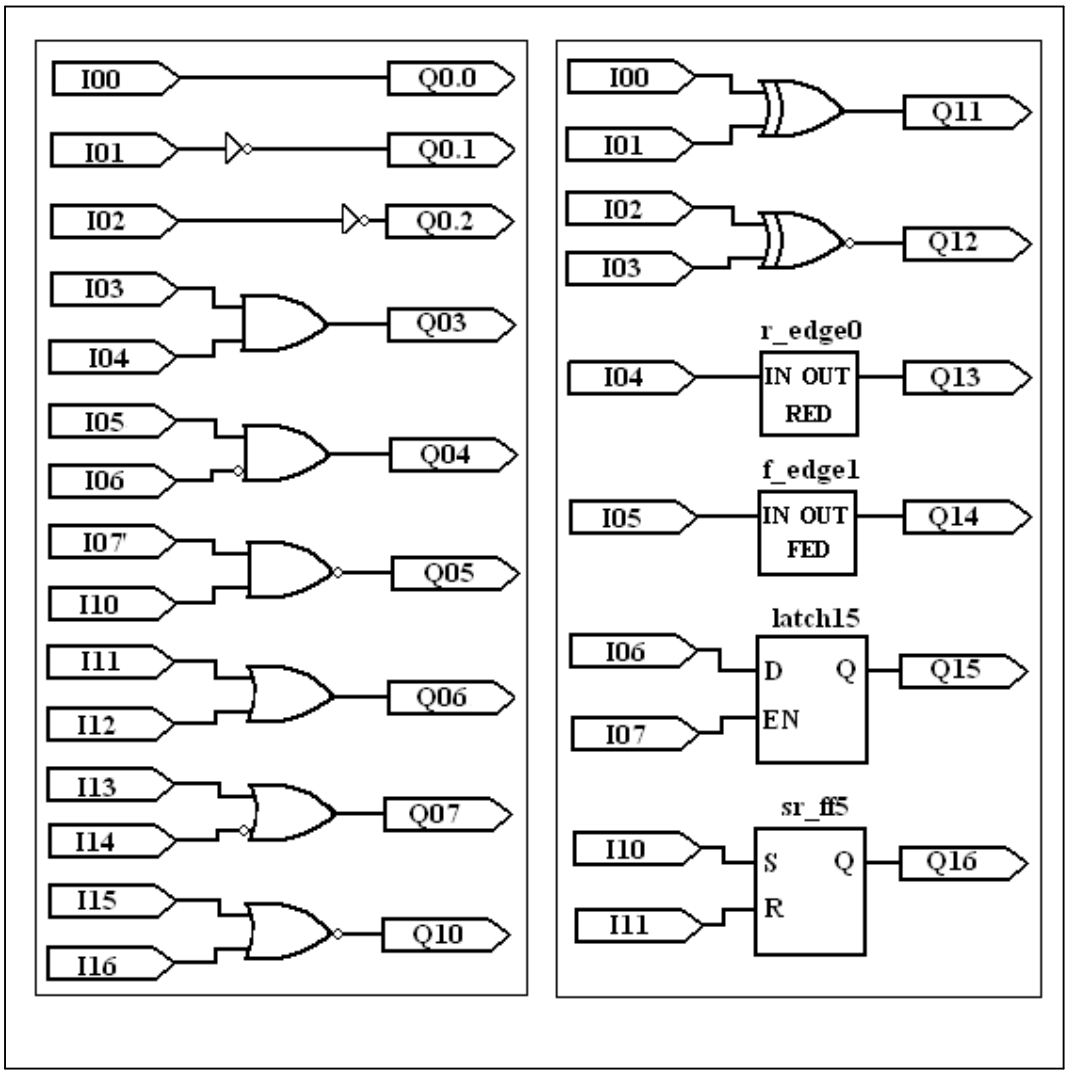
S	R	Q_t	Q_{t+1}	Yorum
0	0	Q_t	Q_t	Değişim yok
1	0	×	1	Set
0	1	×	0	Reset
1	1	×	×	Geçersiz

× : ne olursa olsun(don't care)
</td></tr>
</table>

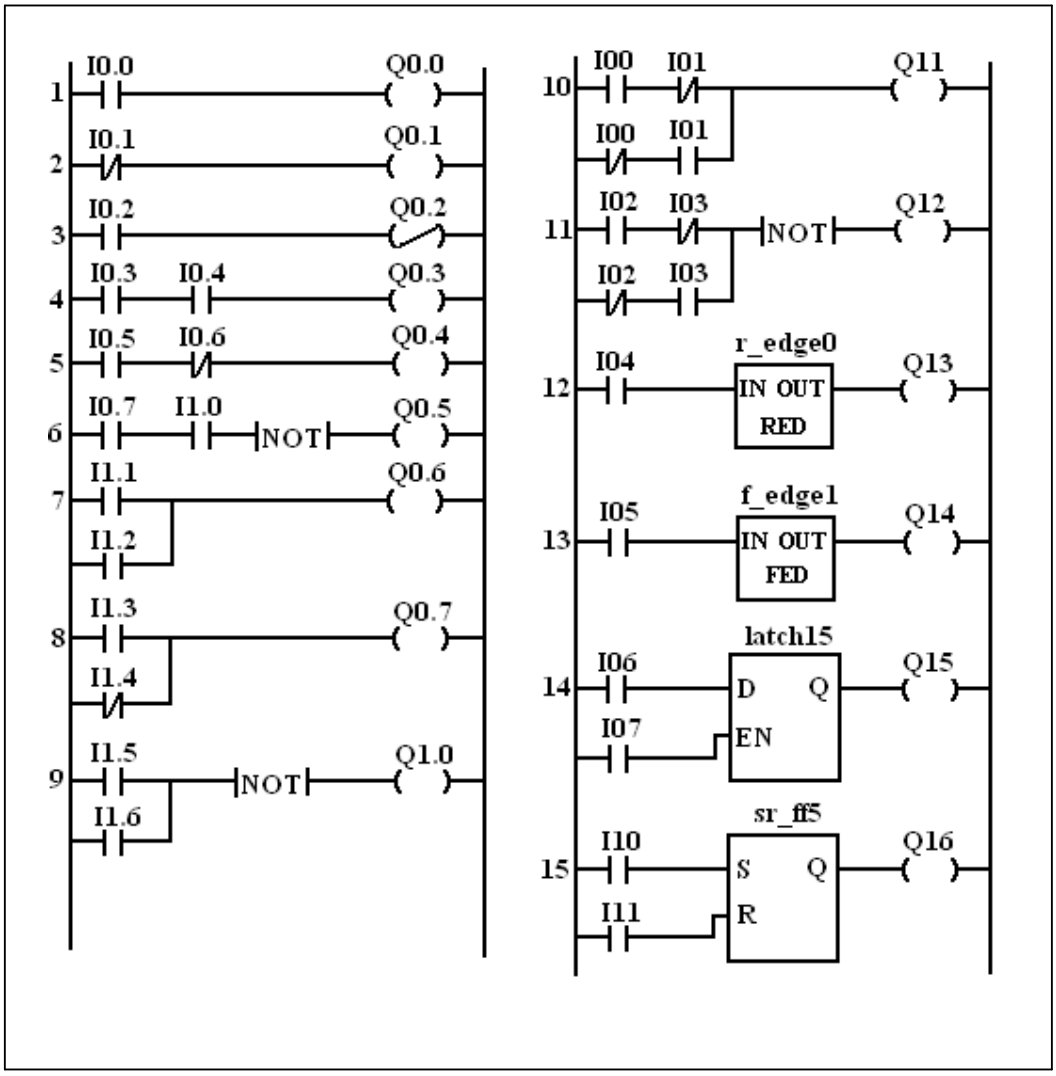
5.4.1 Kontak ve röle temelli fonksiyon örnekleri

Kontak ve röle temelli fonksiyonların örnekleri bu kısımda verilmiştir. Şekil 5.11'de şematik diyagramı, Şekil 5.12'de merdiven diyagramı ve Şekil 5.13'te ise kullanıcı programı görülmektedir.

Bu örnekte tüm satırlarda ayrı bir kontak röle temelli fonksiyon işlemi yapılmaktadır. Birinci basamaktan onbeşinci basamağa kadar, tüm kontak röle temelli fonksiyonlara ait birer örnek verilmiştir.



Şekil 5.11 Kontak ve röle temelli fonksiyon örnekleri için şematik diyagram



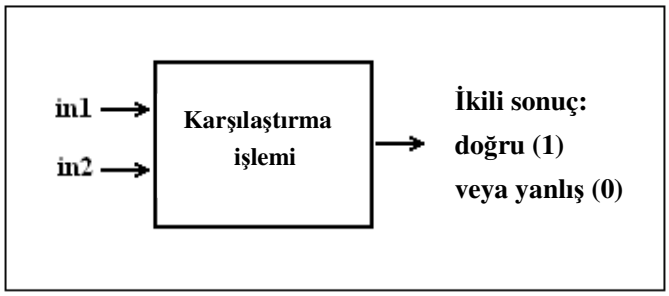
Şekil 5.12 Kontak ve röle temelli fonksiyon örnekleri için merdiven diyagramı

```
//--------------

ld(I00);                        //basamak1
Q00=out();

ld_not(I01);                    //basamak2
Q01=out();

ld(I02);                        //basamak3
Q02=out_not();

ld(I03);                        //basamak4
Q03=and(I04);

ld(I05);                        //basamak5
Q04=and_not(I06);

ld(I07);                        //basamak6
Q05=nand(I10);

ld(I11);                        //basamak7
Q06=or(I12);

ld(I13);                        //basamak8
Q07=or_not(I14);

ld(I15);                        //basamak9
Q10=nor(I16);

ld(I00);                        //basamak10
Q11=xor(I01);

ld(I02);                        //basamak11
Q12=xnor(I03);

ld(I04);                        //basamak12
Q13=r_edge0();

ld(I05);                        //basamak13
Q14=f_edge1();

ld(I07);                        //basamak14
Q15=latch15(I06);

Q16=sr_ff5(I10,I11);            //basamak15

//--------------
```

Şekil 5.13 Kontak ve röle temelli fonksiyon örneklerine ait kullanıcı programı

5.5 Karşılaştırma Fonksiyonları

Karşılaştırma işlemleri PLC programlarında sıklıkla yapılan işlemlerdir. Şekil 5.14'te veri karşılaştırma işleminin genel şekli görülmektedir. Sayısal verilerin karşılaştırılması **in1** ve **in2** olarak tanımlanan iki kaydedici içeriğinin karşılaştırılması ya da bir kaydediciyle sabit bir sayının karşılaştırılması şeklinde olur. Yetkilendirme girişi olan **TEMP** kaydedicisinden yetki sinyali geldiğinde **in1** ve **in2** değişkenleri işlem türüne göre karşılaştırılır. İşlem sonucuna göre Boole çıkış değeri 0 ya da 1 değerini alır. İşlem sonucu doğru ise 1, yanlış ise 0 ile ifade edilir. Sonuç **TEMP** ile dışarı gönderilir [30].



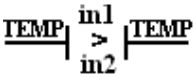
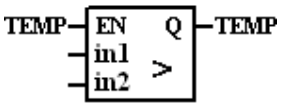
Şekil 5.14 Veri karşılaştırma işleminin genel şekli

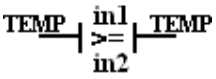
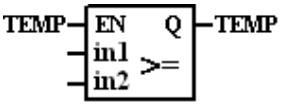
Karşılaştırma işlemleri altı farklı şekilde yapılır. Bunlar; **GT** (**G**reater **T**han – büyük ">"), **GE** (**G**reater than or **E**qual to – büyük veya eşit ">="), **EQ** (**EQ**ual to – eşit "="), **LT** (**L**ess **T**han – küçük "<"), **LE** (**L**ess than or **E**qual to – küçük veya eşit "<="), **NE** (**N**ot **E**qual to – eşit değil "< >") işlemleridir. Çizelge 5.12'de karşılaştırma fonksiyonları ve sembolleri görülmektedir. Bu altı tür karşılaştırma işlemi aşağıdaki durumları ifade etmektedir:

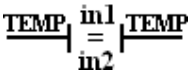
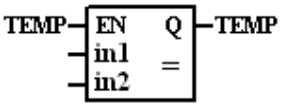
GT	=>	in1, in2'den büyük	(in1 > in2)
GE	=>	in1, in2'den büyük veya eşit	(in1 > = in2)
EQ	=>	in1, in2'ye eşit	(in1 = in2)
LT	=>	in1, in2'den küçük	(in1 < in2)
LE	=>	in1, in2'den küçük veya eşit	(in1 < = in2)
NE	=>	in1, in2'ye eşit değil	(in1 < > in2)

Çizelge 5.12 Karşılaştırma fonksiyonları

İşlem	C Kodu	Merdiven Diyagramı Sembolü	Şematik Sembol

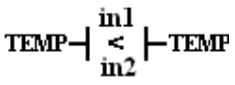
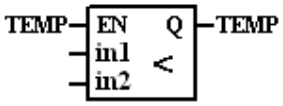
GT
(**G**reater **T**han)
	unsigned int GT (unsigned int in1,unsigned int in2)
	{
	if (TEMP && (in1 > in2))
	TEMP=1;
	else TEMP=0;
	return TEMP;
	}
			
			in1, in2 (16 bitlik kaydedici/sabit sayı)
			EN (TEMP) = 0 veya 1
			Q (TEMP) = 0 veya 1

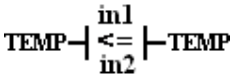
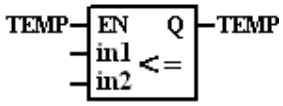
GE
(**G**reater than or **E**qual to)
	unsigned int GE (unsigned int in1, unsigned int in2)
	{
	if (TEMP && (in1 >= in2))
	TEMP=1;
	else TEMP=0;
	return TEMP;
	}
			
			in1, in2 (16 bitlik kaydedici/sabit sayı)
			EN (TEMP) = 0 veya 1
			Q (TEMP) = 0 veya 1

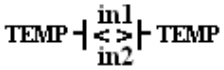
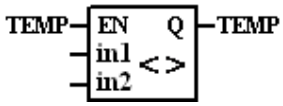
EQ
(**EQ**ual to)
	unsigned int EQ (unsigned int in1, unsigned int in2)
	{
	if (TEMP && (in1 == in2))
	TEMP=1;
	else TEMP=0;
	return TEMP;
	}
			
			in1, in2 (16 bitlik kaydedici /sabit sayı)
			EN (TEMP) = 0 veya 1
			Q (TEMP) = 0 veya 1

Çizelge 5.12. Karşılaştırma fonksiyonları (devam)

İşlem	C Kodu	Merdiven diyagramı sembolü	Şematik sembol

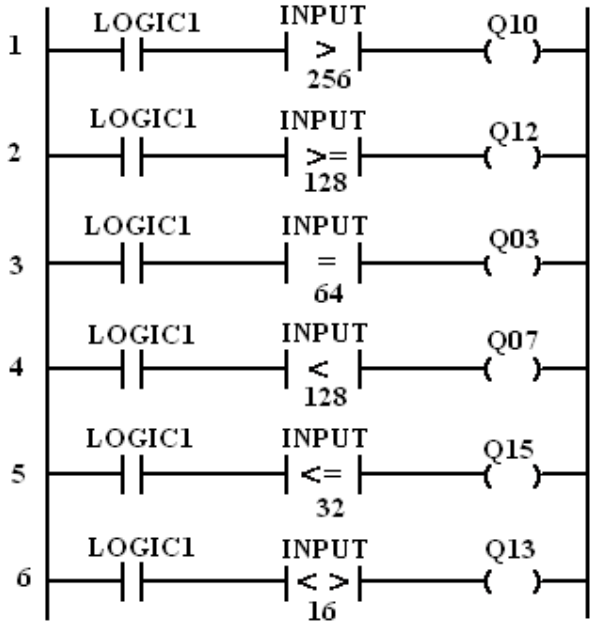
LT
(**L**ess **T**han)
	unsigned int LT (unsigned int in1, unsigned int in2)
	{
	if (TEMP && (in1 <in2))
	TEMP=1;
	else TEMP=0;
	return TEMP;
	}
			
			in1, in2 (16 bitlik kaydedici/sabit sayı)
			EN (TEMP) = 0 veya 1
			Q (TEMP) = 0 veya 1

LE
(**L**ess than or **E**qual to)
	unsigned int LE (unsigned int in1, unsigned int in2)
	{
	if (TEMP && (in1 <=in2))
	TEMP=1;
	else TEMP=0;
	return TEMP;
	}
			
			in1, in2 (16 bitlik kaydedici/sabit sayı)
			EN (TEMP) = 0 veya 1
			Q (TEMP) = 0 veya 1

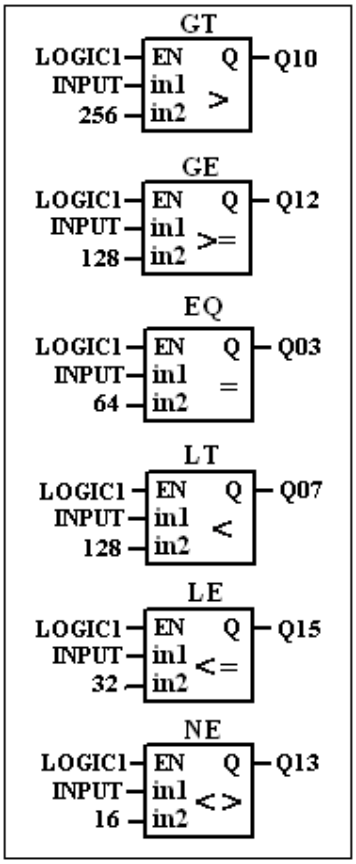
NE
(**N**ot **E**qual to)
	unsigned int NE (unsigned int in1, unsigned int in2)
	{
	if (TEMP && (in1 != in2))
	TEMP=1;
	else TEMP=0;
	return TEMP;
	}
			
			in1, in2 (16 bitlik kaydedici/sabit sayı)
			EN (TEMP) = 0 veya 1
			Q (TEMP) = 0 veya 1

5.5.1 Karşılaştırma fonksiyonları örnekleri

Karşılaştırma fonksiyonlarına ait örnekler bu kısımda verilmiştir. Şekil 5.15'te merdiven diyagramı, Şekil 5.16'da şematik diyagramı ve Şekil 5.17'de kullanıcı programı görülmektedir. Birinci satırdan itibaren sırasıyla "büyüktür", "büyük veya eşittir", "eşittir", "küçüktür", "küçük veya eşittir", "eşit değildir" işlemleri yapılmaktadır. Fonksiyonların girişlerine lojik 1 girişi verilmiştir. Ayrıca **in1** girişleri INPUT kaydedicisine atanmıştır. Dolayısıyla 16 girişten INPUT kaydedicisine alınan veri ile **in2** girişlerindeki verinin karşılaştırmasının sonucu lojik 1 ise ilgili çıkış aktif olur [30].



Şekil 5.15 Karşılaştırma fonksiyonları örneği merdiven diyagramı



Şekil 5.16 Karşılaştırma fonksiyonları örneği şematik diyagramı

```
//……..

ld(LOGIC1);                    // basamak 1
Q10=GT(INPUT,256);

 ld(LOGIC1);                   // basamak 2
 Q12=GE(INPUT,128);

 ld(LOGIC1);                   // basamak 3
 Q03=EQ(INPUT,64);

 ld(LOGIC1);                   // basamak 4
Q07=LT(INPUT,128);

 ld(LOGIC1);                   // basamak 5
 Q15=LE(INPUT,32);

 ld(LOGIC1);                   // basamak 6
 Q13=NE(INPUT,16);
 //……..
```

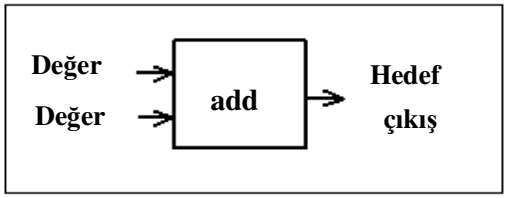
Şekil 5.17 Karşılaştırma fonksiyonları örneği kullanıcı programı

5.6 Aritmetik Fonksiyonları

Aritmetik fonksiyonlar bir veya daha fazla değeri alıp ilgili işlemi gerçekleştirir ve sonucu hafızaya kaydederler. Hemen hemen tüm PLC'lerde toplama (add), çıkarma (subtract), çarpma (multiply) ve bölme (divide) gibi bazı aritmetik işlemler mevcuttur. Bu aritmetik işlemler 16 bitlik iki kaydedicinin içeriğine uygulanabileceği gibi 16 bitlik bir değişken ve bir sabit sayıya da uygulanabilir. Benzer şekilde 16 bitlik tek bir kaydedici içeriğine uygulanan fonksiyonlar da mevcuttur.

Aritmetik işlemleri gerçekleştirmek için altı adet fonksiyon tanımlanmıştır. Bunlar **add** (topla), **sub** (çıkar), **mul** (çarp), **div** (böl), **inc** (bir artır) ve **dec** (bir azalt) fonksiyonlarıdır. Bir artırma ve bir azaltma fonksiyonları bir kaynak değerli, diğerleri iki kaynak değerli fonksiyonlardır. Aritmetik fonksiyonları Çizelge 5.13'te sembolleriyle birlikte görülmektedir. Bu fonksiyonlardan **add**, **sub**, **mul** ve **div** fonksiyonlarında işleme tabi tutulan **in1** ve **in2** şeklinde iki giriş değişkeni ve **TEMP_REG** şeklinde bir çıkış (out) değişkeni bulunmaktadır. **inc** ve **dec** fonksiyonlarında ise bir **in** giriş değişkeni ve bir **TEMP_REG** çıkış değişkeni bulunur. Bunlara ek olarak bir de **TEMP** yetkilendirme girişi mevcuttur. Yetkilendirme girişi olan **TEMP** değişkeni aktif olduğunda, giriş değişkenlerinden gelen bilgiye göre işlem yapılır ve sonuç çıkış değişkenine kaydedilir [31]. Şekil 5.18'de örnek olarak bir aritmetik toplama fonksiyonu şematik olarak gösterilmektedir.

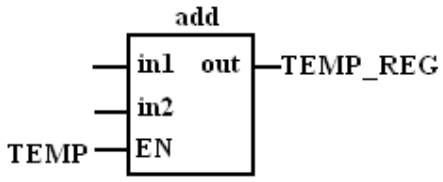
- **add** (değer1, değer2, hedef çıkış) – İki değeri topla, sonucu çıkışa kaydet
- **sub** (değer1, değer2, hedef çıkış) – İkinci değeri birinciden çıkar, sonucu çıkışa kaydet
- **mul** (değer1, değer2, hedef çıkış) – İki değeri çarp, sonucu çıkışa kaydet
- **div** (değer1, değer2, hedef çıkış) – Birinci değeri ikinci değere böl, sonucu çıkışa kaydet
- **inc** (değer, hedef çıkış) – Değeri bir artır, sonucu çıkışa kaydet
- **dec** (değer, hedef çıkış) – Değeri bir azalt, sonucu çıkışa kaydet

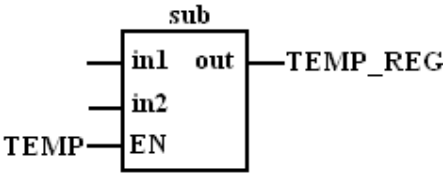


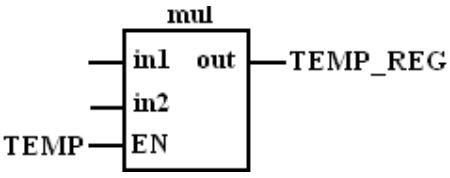
Şekil 5.18 Bir aritmetik toplama işlemi şeması

Çizelge 5.13 Aritmetik fonksiyonları

İşlem	C Kodu	Sembol

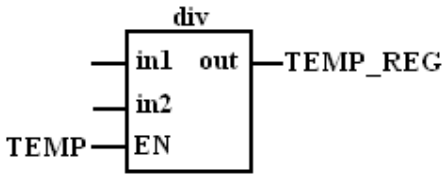
add	unsigned int add (unsigned int in1, unsigned int in2)
	{
	if (TEMP)
	{TEMP_REG=in1+in2;}
	return TEMP_REG;
	}
		
		in1, in2 (16 bitlik kaydedici/sabit sayı)
		EN = Yetkilendirme girişi (0 veya 1)
		out = 16 bitlik kaydedici

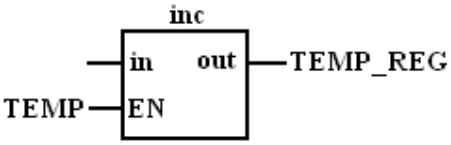
sub	unsigned int sub (unsigned int in1, unsigned int in2)
	{
	if (TEMP)
	{ TEMP_REG=in1-in2;}
	return TEMP_REG;
	}
		
		in1, in2 (16 bitlik kaydedici/sabit sayı)
		EN = Yetkilendirme girişi (0 veya 1)
		out = 16 bitlik kaydedici

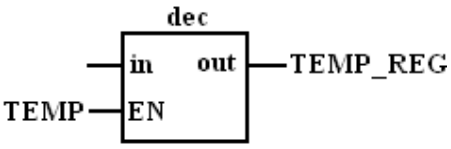
mul	unsigned int mul (unsigned int in1, unsigned int in2)
	{
	if (TEMP)
	{ TEMP_REG=in1*in2;}
	return TEMP_REG;
	}
		
		in1, in2 (16 bitlik kaydedici/sabit sayı)
		EN = Yetkilendirme girişi (0 veya 1)
		out = 16 bitlik kaydedici

Çizelge 5.13 Aritmetik fonksiyonları (devam)

İşlem	C Kodu	Sembol

div	unsigned int add (unsigned int in1, unsigned int in2)
	{
	if (TEMP)
	{TEMP_REG=in1/in2;}
	return TEMP_REG;
	}
		
		in1, in2 (16 bitlik kaydedici/sabit sayı)
		EN = Yetkilendirme girişi (0 veya 1)
		out = 16 bitlik kaydedici

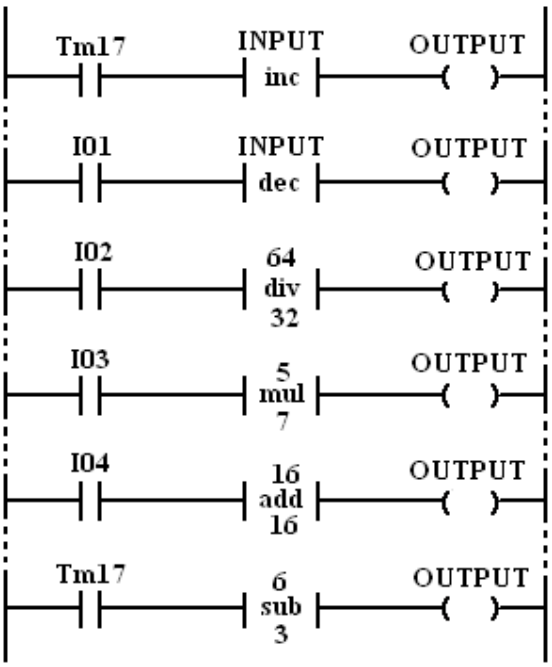
inc	unsigned int inc (unsigned int in)
	{
	if (TEMP) TEMP_REG= in + 1;
	return TEMP_REG;
	}
		
		in (16 bitlik kaydedici/sabit sayı)
		EN = Yetkilendirme girişi (0 veya 1)
		out = 16 bitlik kaydedici

dec	unsigned int dec (unsigned int in)
	{
	if (TEMP) TEMP_REG= in - 1;
	return TEMP_REG;
	}
		
		in1, in2 (16 bitlik kaydedici/sabit sayı)
		EN = Yetkilendirme girişi (0 veya 1)
		out = 16 bitlik kaydedici

5.6.1 Aritmetik fonksiyon örnekleri

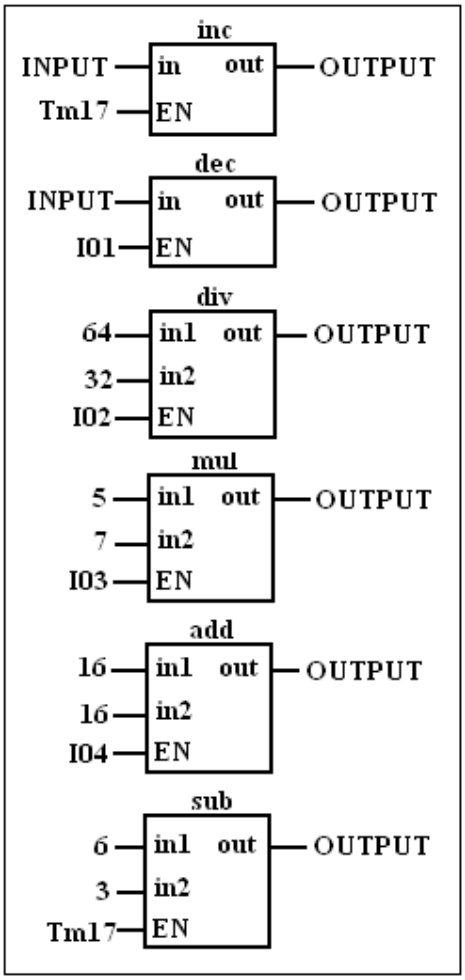
Aritmetik fonksiyonlarına ait örnekler bu kısımda verilmiştir. Şekil 5.19'da merdiven diyagramı, Şekil 5.20'de şematik diyagram ve Şekil 5.21'de kullanıcı programı görülmektedir. Birinci basamaktan itibaren sırasıyla "bir artırma", "bir azaltma", "bölme", "çarpma", "toplama" ve "çıkarma" işlemleri yapılmaktadır. Bu program her hangi bir anda sadece bir basamak aktif olacak şekilde çalıştırılmaktadır. Bir başka deyişle birden fazla basamağın aynı anda çalışması yanlış sonuçlara sebep olur.

Bu örnekte altı farklı aritmetik fonksiyona ait işlemler verilmiştir. Yetkilendirme girişinden lojik 1 sinyali gelmesi halinde "bir arttırma" ve "bir azaltma" fonksiyonları için INPUT kaydedicisindeki 16 bitlik veri, diğer fonksiyonlar için aritmetik işlem sonucu OUTPUT kaydedicisine gönderilir ve sonuç çıkışlarda görülür.



Şekil 5.19 Aritmetik fonksiyon örnekleri merdiven diyagramı

(NOT: Herbir satır ayrı ayrı çalıştırılmıştır)



Şekil 5.20 Aritmetik fonksiyonları örnekleri şematik diyagramı

(NOT: Herbir satır ayrı ayrı çalıştırılmıştır)

```
//……
//ld(Tm17);
//OUTPUT=inc(INPUT);
//ld(I10);
//OUTPUT=dec(INPUT);
//ld(I02);
//OUTPUT=div(64,32);
//ld(I03);
//OUTPUT=mul(5,7);
//ld(I04);
//OUTPUT=add(16,16);
//ld(Tm17);
//OUTPUT=sub(6,3);
//……
```

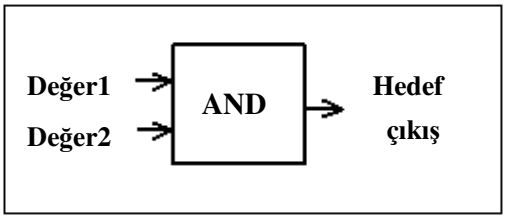
Şekil 5.21 Aritmetik fonksiyon örnekleri kullanıcı programı

(NOT: Herbir satır ayrı ayrı çalıştırılmıştır)

5.7 Lojik Fonksiyonları

Lojik fonksiyonlar; AND, OR, NAND, NOR, exclusive-OR (XOR), exclusive-NOR (XNOR) , NOT, SHIFT_L ve SHIFT_R lojik işlemlerini gerçekleştiren fonksiyonlardır. Şekil 5.22'de bir lojik fonksiyonu işleminin yapısı şematik olarak görülmektedir. Bu fonksiyonlar 16 bitlik bir kaydedici içeriği ile yine 16 bitlik bir başka kaydedicinin içeriğine ya da 16 bitlik bir sabit sayıya uygulanır**. in1** ve **in2** olarak ifade edilen bu kaydedici ya da sabit sayı üzerinde işlem yapılması **TEMP** yetkilendirme girişinden yetki sinyali gelmesine bağlıdır. **TEMP** kaydedicisinden yetki sinyali geldiğinde lojik işlem gerçekleştirilir ve Boole çıkış değeri **TEMP_REG** çıkış kaydedicisi yoluyla dışarı aktarılır. Çizelge 5.14'te lojik fonksiyonları sembolleriyle birlikte görülmektedir.

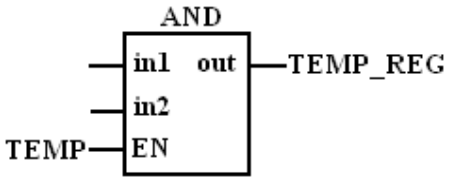
Lojik fonksiyonlardan AND, OR, NAND, NOR, XOR VE XNOR fonksiyonları iki kaynak girişe sahiptir. Bu fonksiyonlar iki farklı kaynaktan gelen bilgiyi işleme alırlar. NOT, SHIFT_L ve SHIFT_R fonksiyonları ise tek kaynak girişe sahiptir [32].

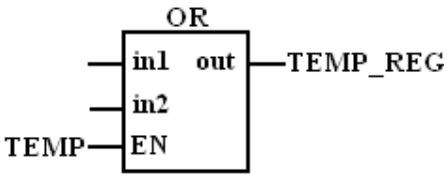


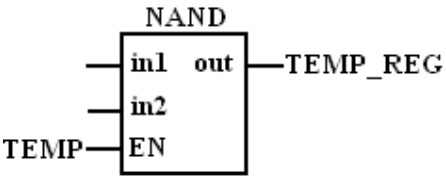
Şekil 5.22 Bir lojik "AND" fonksiyonu şeması

Çizelge 5.14 Lojik fonksiyonları

İşlem	C Kodu	Sembol

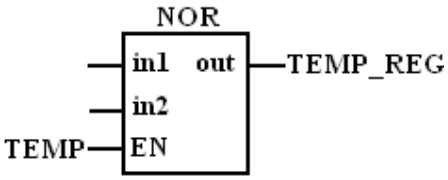
ADD	unsigned int AND (unsigned int in1, unsigned int in2)
	{
	if (TEMP) TEMP_REG=in1&in2;
	return TEMP_REG;
	}
		
		in1, in2 (16 bitlik kaydedici/sabit sayı)
		EN = Yetkilendirme girişi (0 veya 1)
		out = 16 bitlik kaydedici

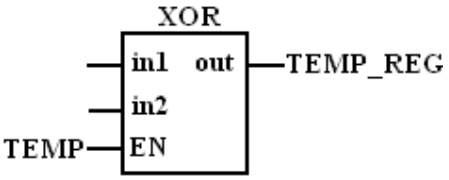
OR	unsigned int OR (unsigned int in1, unsigned int in2)
	{
	if (TEMP) TEMP_REG=in1|in2;
	return TEMP_REG;
	}
		
		in1, in2 (16 bitlik kaydedici/sabit sayı)
		EN = Yetkilendirme girişi (0 veya 1)
		out = 16 bitlik kaydedici

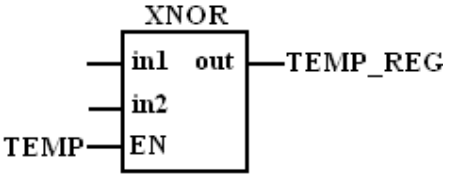
NAND	unsigned int NAND (unsigned int in1, unsigned int in2)
	{
	if (TEMP)
	TEMP_REG=~(in1&in2);
	return TEMP_REG;
	}
		
		in1, in2 (16 bitlik kaydedici/sabit sayı)
		EN = Yetkilendirme girişi (0 veya 1)
		out = 16 bitlik kaydedici

Çizelge 5.14 Lojik fonksiyonları (devam)

İşlem	C Kodu	Sembol

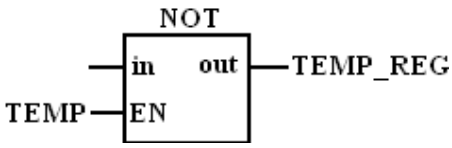
NOR	unsigned int NOR (unsigned int in1, unsigned int in2)
	{
	if (TEMP) TEMP_REG=~(in1|in2);
	return TEMP_REG;
	}
		
		in1, in2 (16 bitlik kaydedici/sabit sayı)
		EN = Yetkilendirme girişi (0 veya 1)
		out = 16 bitlik kaydedici

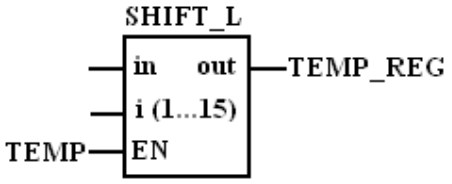
XOR	unsigned int XOR (unsigned int in1, unsigned int in2)
	{
	if (TEMP) TEMP_REG=in1^in2;
	return TEMP_REG;
	}
		
		in1, in2 (16 bitlik kaydedici/sabit sayı)
		EN = Yetkilendirme girişi (0 veya 1)
		out = 16 bitlik kaydedici

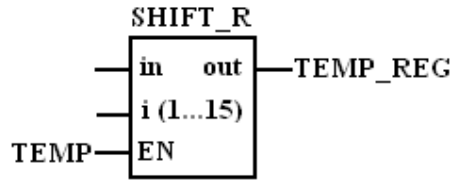
XNOR	unsigned int XNOR (unsigned int in1, unsigned int in2)
	{
	if (TEMP) TEMP_REG=~(in1^in2);
	return TEMP_REG;
	}
		
		in1, in2 (16 bitlik kaydedici/sabit sayı)
		EN = Yetkilendirme girişi (0 veya 1)
		out = 16 bitlik kaydedici

Çizelge 5.14 Lojik fonksiyonları (devam)

İşlem	C Kodu	Sembol

NOT	unsigned int NOT (unsigned int in)
	{
	if (TEMP) TEMP_REG=~in;
	return TEMP_REG;
	}
		
		in (16 bitlik kaydedici/sabit sayı)
		EN = Yetkilendirme girişi (0 veya 1)
		out = 16 bitlik kaydedici

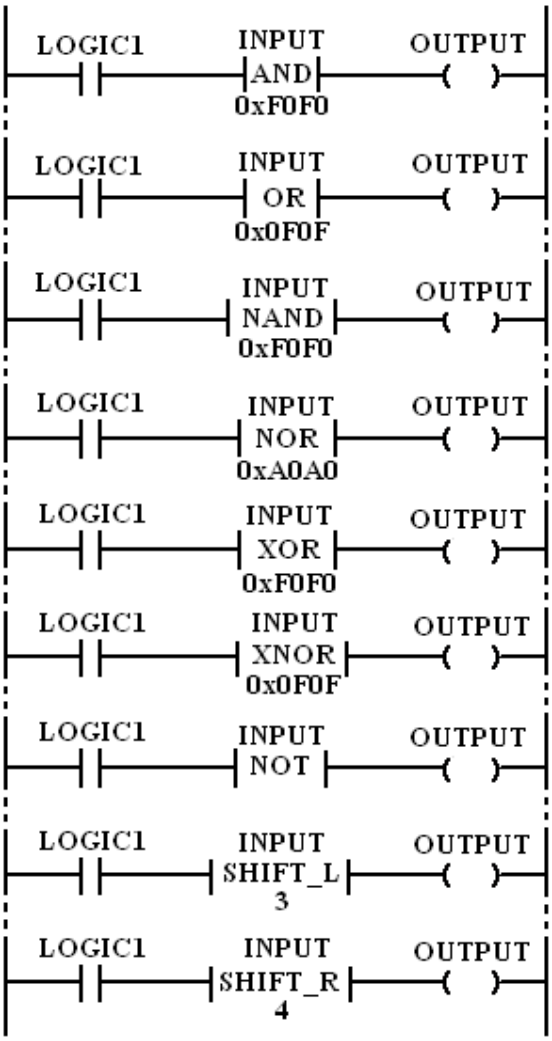
SHIFT_L	unsigned int SHIFT_L (unsigned int in, unsigned int i)
	{
	if ((i<16)&& (TEMP))
	TEMP_REG = (in << i);
	return TEMP_REG;
	}
		
		in (16 bitlik kaydedici/sabit sayı)
		i (1…15)
		EN = Yetkilendirme girişi (0 veya 1)
		out = 16 bitlik kaydedici

SHIFT_R	unsigned int SHIFT_R (unsigned int in, unsigned int i)
	{
	if ((i<16)&& (TEMP))
	TEMP_REG = (in >> i);
	return TEMP_REG;
	}
		
		in (16 bitlik kaydedici/sabit sayı)
		i (1…15)
		EN = Yetkilendirme girişi (0 veya 1)
		out = 16 bitlik kaydedici

5.7.1 Lojik fonksiyonları örnekleri

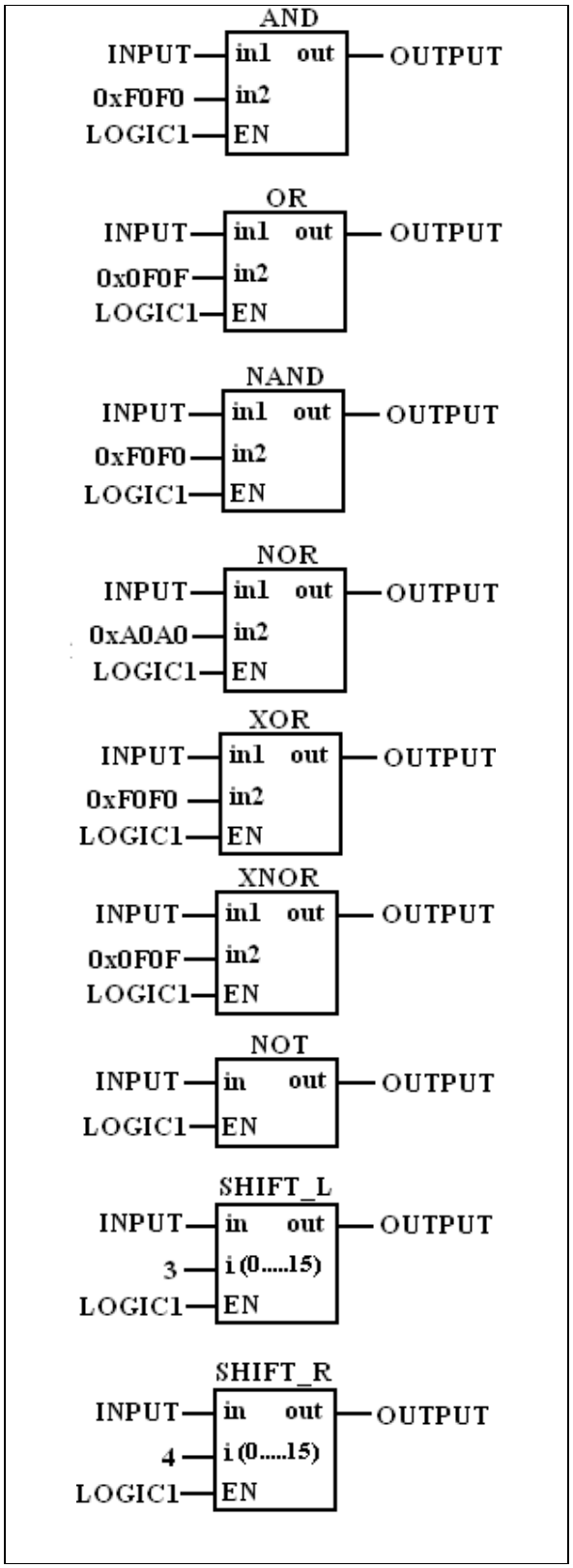
Lojik fonksiyonlarına ait örnekler bu kısımda verilmiştir. Şekil 5.23'de merdiven diyagramı, Şekil 5.24'te şematik diyagramı ve Şekil 5.25'te kullanıcı programı görülmektedir. Dokuz farklı lojik fonksiyon için örnek bulunmaktadır. Program herhangi bir anda sadece bir basamak aktif olacak şekilde çalıştırılmaktadır. Bir başka deyişle birden fazla basamağın aynı anda çalışması yanlış sonuçlara sebep verir.

Bu örnekte tüm lojik fonksiyonların girişine "lojik 1" verilmiştir. Ayrıca **in1** girişleri INPUT kaydedicisine atanmıştır. Dolayısıyla 16 ayrık girişten girilen veri **in2**'lerdeki veri ile lojik işleme tabi tutulur ve sonuç OUTPUT kaydedicisine aktarılır.



Şekil 5.23 Lojik fonksiyon örneklerine ait merdiven diyagramı

(NOT: Her bir satır ayrı ayrı çalıştırılmıştır)



Şekil 5.24 Lojik fonksiyon örnekleri için şematik diyagram

(NOT: Herbir satır ayrı ayrı çalıştırılmaktadır)

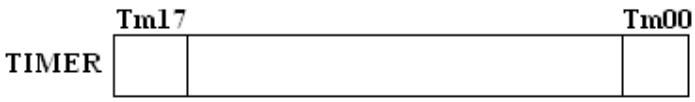
```
//ld(LOGIC1);
//OUTPUT=AND(INPUT,0xF0F0);

//ld(LOGIC1);
//OUTPUT=OR(INPUT,0x0F0F);

//ld(LOGIC1);
//OUTPUT=NAND(0xF0F0,INPUT);

//ld(LOGIC1);
//OUTPUT=NOR(0xA0A0,INPUT);

//ld(LOGIC1);
//OUTPUT= XOR(0xF0F0,INPUT);

//ld(LOGIC1);
//OUTPUT= XNOR(INPUT,0x0F0F);

//ld(LOGIC1);
//OUTPUT=NOT(INPUT);

//ld(LOGIC1);
//OUTPUT= SHIFT_L(INPUT,3);

//ld(LOGIC1);
//OUTPUT= SHIFT_R(INPUT,4);
```

Şekil 5.25 Lojik fonksiyon örneklerine ait kullanıcı programı

(NOT: Her bir satır ayrı ayrı çalıştırılmıştır)

5.8 Zamanlayıcı Fonksiyonları

Belirli bir zaman gecikmesiyle bir işlemin yapılmasının gerekli olduğu uygulamalarda zamanlayıcılar sıklıkla kullanılmaktadır. Zamanlayıcı fonksiyonları; bir giriş sinyaline bağlı olarak istenen sürede zaman gecikmesi oluşturulup, bu süre sonunda bir çıkış sinyali verilen fonksiyonlardır. İstenen zaman gecikmesinin yapısına göre 'düz zaman rölesi' ve 'ters zaman rölesi' olarak iki zamanlayıcı fonksiyonu tanımlanmıştır. Bu fonksiyonlara ait zaman gecikmesinin sağlanması, referans **clock (clk)** sinyali olan ve mikrodenetleyicinin dâhili zamanlayıcısı olan 16 bitlik **Timer1 (TMR1)** modülü ile gerçekleştirilmektedir. Her bir biti kendi içinde bir zaman sayıcısı olarak çalışan Timer1 modülü program hafızasında yer alan **TIMER** kaydedicisine atanmıştır. Her bir biti ayrı bir zaman sayıcısı olarak hafızada tanımlanmıştır. Bu sayede 16 adet referans clock sinyali elde edilmiştir. Bu sinyaller ve zaman değerleri Şekil 5.26'da görülmektedir.



TIMER: Tm17 ……..Tm00

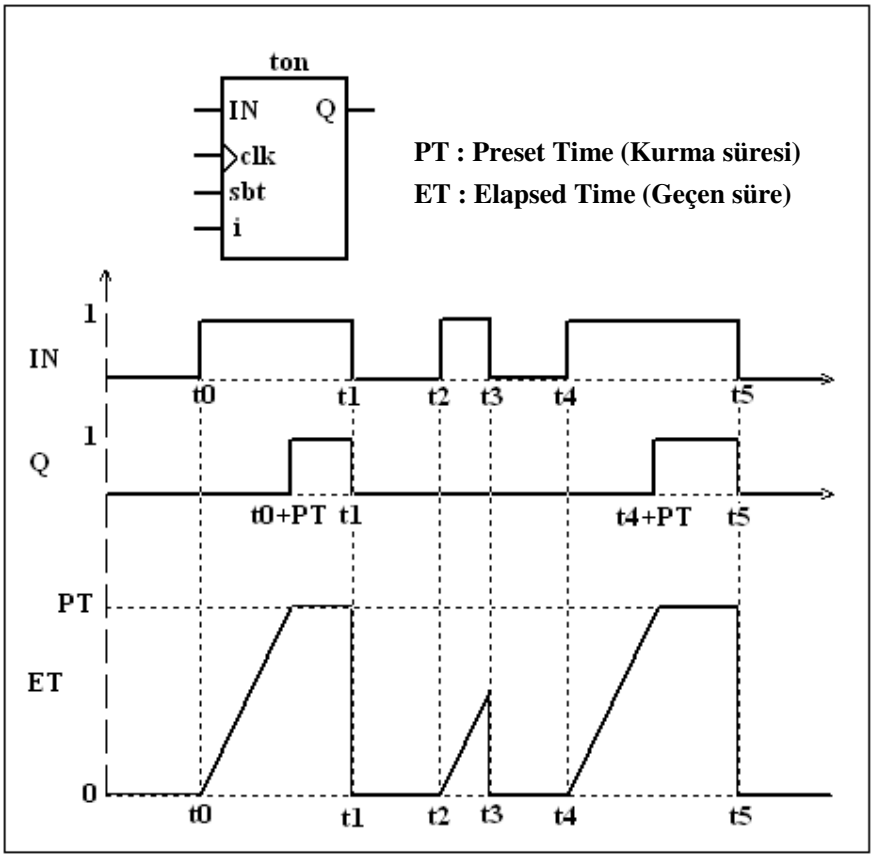
CLK	Zaman değeri
Tm17	1.04856 sn
Tm16	0.52428 sn
Tm15	0.26214 sn
Tm14	0.13107 sn
Tm13	0.065535 sn
Tm12	0.032767 sn
Tm11	0.016384 sn
Tm10	0.008192 sn
Tm07	0.004096 sn
Tm06	0.002048 sn
Tm05	0.001024 sn
Tm04	0.000512 sn
Tm03	0.000256 sn
Tm02	0.000128 sn
Tm01	0.000064 sn
Tm00	0.000032 sn

Şekil 5.26 Referans zamanlama sinyalleri ve zaman değerleri

Bu referans clock sinyalleri zamanlayıcı fonksiyonlarında kullanılmaktadır. Örneğin Tm13 sinyali kullanıldığında 0,065535 saniye değerinde olan bu zamanlayıcı süresi sabit bir sayı ile çarpılarak istenilen sürede zaman gecikmesi elde edilebilir. Buna göre 5 saniyelik bir gecikme elde etmek için (0,065535×77=5,046) 0,065535 süresinin 77 ile çarpılması sonucunda yaklaşık 5 saniyelik zaman gecikmesi elde edilir. Zaman gecikmesi algoritmaları zaman rölesi fonksiyonlarında açıklanmıştır.

5.8.1 Düz zaman rölesi (On delay timer - TON)

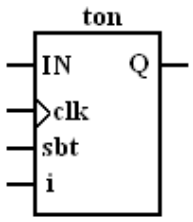
Giriş sinyali (**IN**) öngörülen süre kadar uygulandıktan sonra çıkış sinyalinin (**Q**) lojik 0 seviyesinden lojik 1 seviyesine konum değiştirdiği zaman rölesine düz zaman rölesi denir. Şekil 5.27'de düz zaman rölesi (**TON**) sembolü ve zamanlama diyagramı görülmektedir. Düz zaman rölesinde çıkışın belirli bir gecikmeyle lojik 0 seviyesinden lojik 1 seviyesine konum değiştirebilmesi için giriş sinyalinin 'kurma süresi' (**PT**) denilen ayarlanmış süre kadar uygulanması gerekir. Kurma süresi PT=clk × sbt formülü ile ifade edilir [28].



Şekil 5.27 Düz zaman rölesinin (TON) sembolü ve zamanlama diyagramı

Çizelge 5.15 Gerçekleştirilen düz zaman rölesinin (TON) sembolü ve C kodu

C Kodu	Sembol

void ton (unsigned int clk, unsigned int sbt, unsigned int i)
{
if (i<64)
{
if (TEMP==0)
{
TON[i]=0;
TONQ[i]=0;
}
else if (TONQ[i]==0)
{
if(clk==0) TONRED[i]=1;
else if(clk==1)
if (TONRED[i]==1)
{
TONRED[i]=0;
TON[i]++;
if (TON[i]==sbt)
TONQ[i]=1;
}
}
}
}
	
	IN : TEMP (giriş) = 0 veya 1
	Q : Output (çıkış)
	clk : Tm00, Tm01, …,Tm07, Tm10,….Tm17
	sbt : Sabit sayı (1,2,….65535)
	i : Düz zaman rölesi numarası (0,1…63)

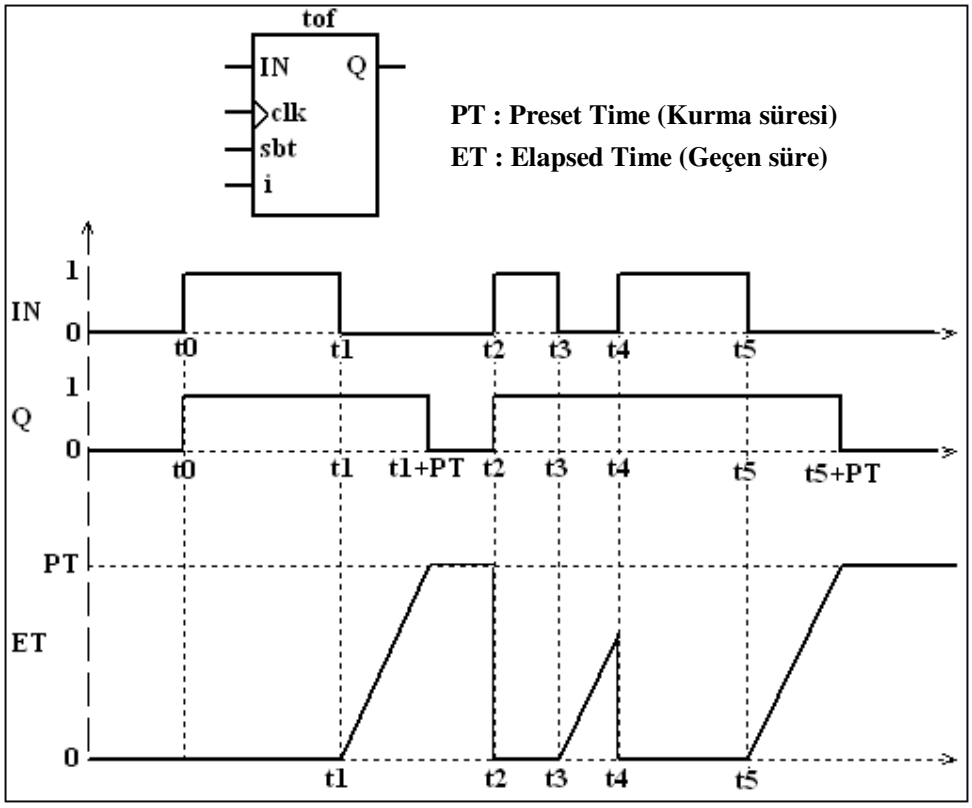
Düz zaman rölesinin sembolü ve C kodu Çizelge 5.15'te görülmektedir. '**ton**' düz zaman rölesi fonksiyonu **clk, sbt** ve **i** argümanlarından oluşmaktadır. '**clk**'; 16 bitlik TIMER kaydedicisinin herhangi bitinden alınan bir referans zamanlayıcısıdır. '**sbt**'; istenen zaman gecikmesini elde etmek için kullanılan bir sabit sayıdır (1,2…65535). '**i**'; 0'dan 63'e kadar olan 64 adet farklı kaydediciyi ifade eden indeks argümanıdır. Bir başka deyişle toplam 64 tane düz zaman rölesi tanımlanmıştır. Düz zaman rölesi fonksiyonunun çalışması şu şekilde olur:

İlk olarak **TEMP** kaydedicisinden giriş (**IN**) aktifleştirme sinyali gelip gelmediği kontrol edilir. Sinyal yoksa sayma kaydedicisi **TON[i]** ve çıkış (**Q**) kaydedicisi **TONQ[i]** içerikleri 0 yapılır. Sinyal gelmişse ve çıkış kaydedicisi **TONQ[i]**'nin içeriği 0 ise; **clk** sinyalinin her 0 değerinde yükselen kenar algılama kaydedicisi **TONRED[i]** 1 olur, her 1 değerinde ise 0 olur. Bu şekilde; yükselen kenar algılama kaydedicisi **TONRED[i]**'nin 1 değerinden 0 değerine her durum değiştirmesinde sayma kaydedicisi

TON[i] bir artırılır. **TON[i]**, istenen zaman gecikmesi süresine göre belirlenen referans sabiti **sbt** değerine ulaştığında çıkış kaydedicisi **TONQ[i]** 1 yapılır. İndeks argümanı olan **i** yerine 0'dan 63'e kadar yazılan sayı değeri neyse Boole değeri aynı numaralı kaydediciye aktarılır. Örneğin **i**=15 seçilmesi durumunda sayma kaydedicisi **TON[15]**, çıkış kaydedicisi **TONQ[15]**, yükselen kenar kaydedicisi **TONRED[15]** olur. Bu fonksiyon işleminin sonucunda bilgi, **TEMP** kaydedicisi tarafından giriş sinyali geldiği andan itibaren belirli bir gecikme ile çıkış kaydedicisi **TONQ[i]**'ye aktarılmış olur. **TEMP** kaydedicisinden gelen sinyal kesildiği zaman çıkışa giden bilgi de kesilmiş olur.

5.8.2 Ters zaman rölesi (Off delay timer)

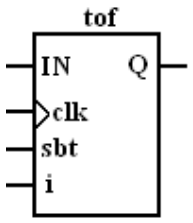
Giriş sinyali (**IN**) lojik 1 seviyesinden lojik 0 seviyesine düştükten sonra, çıkış sinyalini (**Q**) belirli bir süre (**PT**) geciktirerek lojik 0 seviyesine düşürme işlemi yapan zamanlayıcıya ters zaman rölesi denilmektedir. Giriş sinyalinin lojik 1 olmasıyla birlikte çıkış sinyali de lojik 1 olur. Giriş sinyali lojik 0 olduğunda, çıkış PT kurma süresi dolduktan sonra lojik 0 olur. Şekil 5.28'de ters zaman rölesinin (TOF) sembolü ve zamanlama diyagramı görülmektedir.



Şekil 5.28 Ters zaman rölesinin (TOF) sembolü ve zamanlama diyagramı

Çizelge 5.16 Gerçekleştirilen ters zaman rölesinin (TOF) sembolü ve C kodu

C Kodu	Sembol

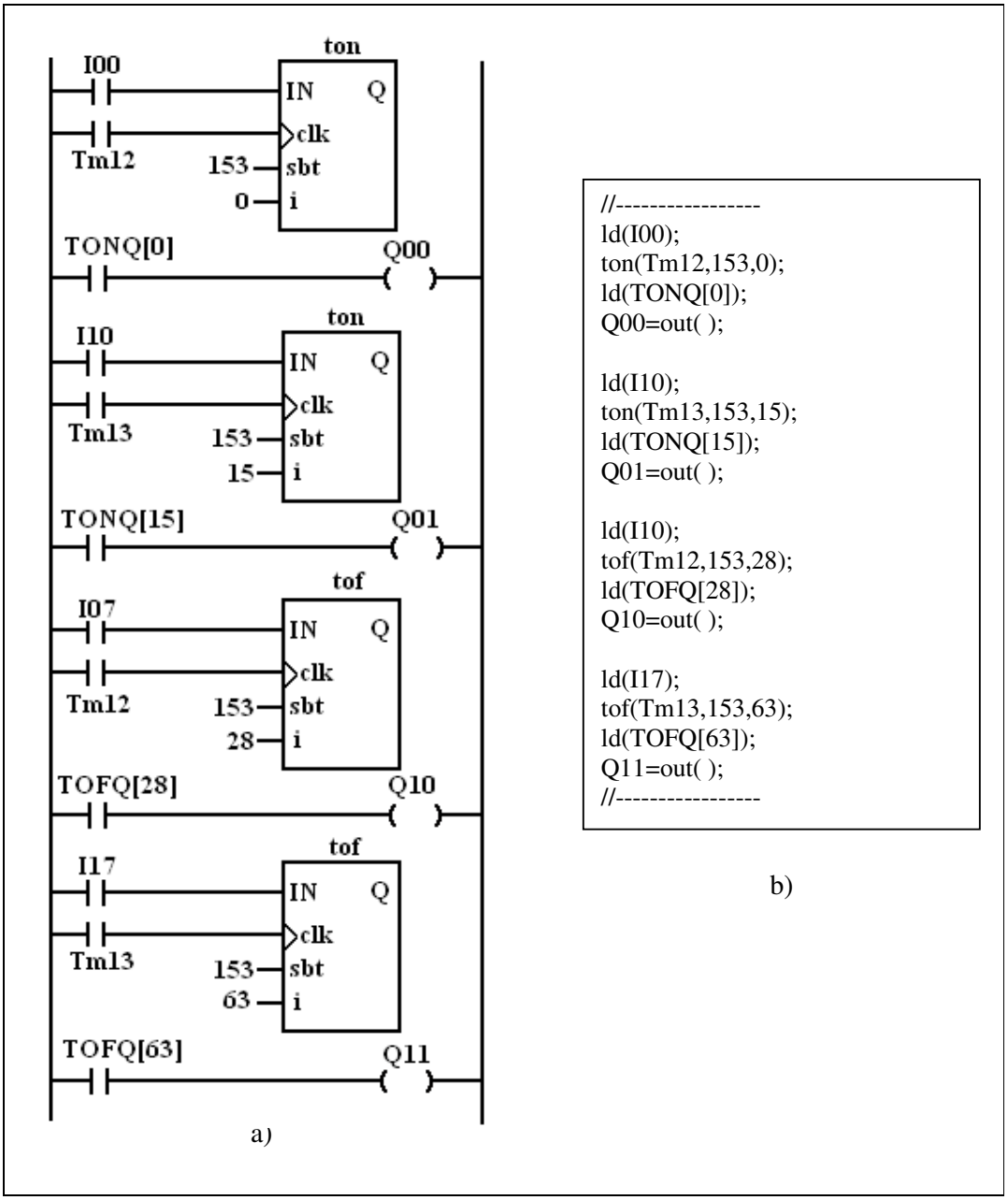
void tof (unsigned int clk, unsigned int sbt,unsigned int i)
{
if (i<64)
{
if (TOFQ[i]==0 && (TEMP))
TOFQ[i]=1;
if (TOFQ[i]==1 && (TEMP))
{TOFQ[i]=1;TOF[i]=0;}
if (TOFQ[i]==1 && TEMP==0)
{
if(clk==0) TOFRED[i]=1;
else if(clk==1)
if (TOFRED[i]==1)
{ TOFRED[i]=0;
TOF[i]++;
if (TOF[i]==sbt)
TOFQ[i]=0;
}
}
}
}
	
	IN : TEMP (giriş) = 0, 1
	Q : Output (çıkış)
	clk : Tm00, Tm01, …,Tm07, Tm10,….Tm17
	sbt : Sabit sayı (1,2,….65535)
	i : Ters zaman rölesi numarası (0,1…63)

Ters zaman rölesinin (TOF) sembolü ve C kodu Çizelge 5.16'da görülmektedir. **tof** ters zaman rölesi fonksiyonu, ton düz zaman rölesi fonksiyonunda tanımlandığı gibi **clk, sbt** ve **i** argümanlarından oluşmaktadır. Burada **clk** referans zamanlama sinyalini, **sbt** referans sabitini ve **i** indeks numarasını ifade etmektedir. Ters zaman rölesi fonksiyonunun çalışması şu şekildedir:

İlk olarak **TOFQ[i]** çıkış (**Q**) kaydedicisinin içeriği kontrol edilir. Eğer 0 ise ve **TEMP** kaydedicisinden giriş (**IN**) aktifleştirme sinyali gelmişse **TOFQ[i]** 1 olur ve sayma kaydedicisi **TOF[i]**'nin içeriği sıfırlanır. Eğer çıkış aktif iken giriş bilgisi kesilirse bu durumda **clk** referans zamanlama sinyaline bakılır. **clk** 0 ise yükselen kenar algılama kaydedicisi **TOFRED[i]** 1 olur. **clk** 1 olduğunda **TOFRED[i]** 0 olur ve sayma kaydedicisi **TOF[i]** bir artırılır. **TOF[i]**'nin bir arttırılması işlemi program döngüsünde devam eder. **TOF[i]**'nin içeriği **sbt** referans sabitine eşit olduğunda çıkış kaydedicisi **TOFQ[i]** 0 olur [28].

5.8.3 Zamanlayıcı fonksiyon örnekleri

Zamanlayıcı fonksiyonlarına ait örnekler bu kısımda verilmiştir. İki adet düz ve iki adet ters zaman rölesi örneğine ait merdiven diyagramı Şekil 5.29 (a)'da ve kullanıcı programı da Şekil 5.29 (b)'de görülmektedir. IN girişine bağlı kontaktan sinyal geldiği anda düz zaman rölesi zaman sayıcısı saymaya başlar. PT=clk × sbt formülüne göre ayarlanan süre sonunda zaman rölesi çıkış kaydedicisi aktif olur ve çıkışı lojik 1 yapar.



Şekil 5.29 Zamanlayıcı fonksiyon örnekleri
a) Merdiven diyagramı b) Kullanıcı programı

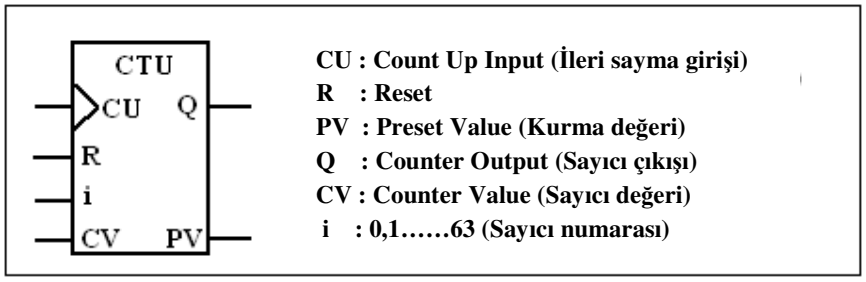
5.9 Sayıcı Fonksiyonları

Sayıcılar kumanda sistemlerinde olayların sayısına bağlı işaretlerin üretilmesinde kullanılır. Uygulanan işaretin sayıcı içeriğini arttırması ya da azaltması şeklinde işlem yaparlar. Genelde PLC'lerde tanımlı ileri sayıcı (up counter - CTU), geri sayıcı (down counter - CTD) ve ileri/geri sayıcı (up-down counter - CTUD) isimli üç temel sayıcı fonksiyonu mevcuttur. Bu fonksiyonlar sayıcı içeriklerini arttıran/azaltan ve sayıcı içeriği belirli bir değere ulaştığında sayıcı durum kaydedicisine veri gönderecek şekilde çalışmaktadır. Bu kısımda gerçekleştirilen bu üç tip sayıcı fonksiyonu açıklanmaktadır.

5.9.1 İleri sayıcı (up counter – CTU)

İleri sayıcı (up counter-CTU) bir sayma değerinin maksimum değere ulaştığını göstermek için kullanılan sayıcı türüdür. Şekil 5.30'da ileri sayıcının sembolü ve Çizelge 5.17'de doğruluk tablosu görülmektedir.

İleri sayıcı **CU** ileri sayma girişinde tespit edilen yükselen kenarları (↑) sayar. **PV** sayılacak maksimum değeri ifade etmektedir. **CU** sayma girişinden gelen her bir yükselen kenarda **CV** sayma değeri 1 artırılır. Sayıcı **PV** değerine ulaştığında sayıcı çıkışı Q set (1) yapılır ve sayma işlemi durur. **R** reset girişi istenilen anda sayıcı çıkışını off (0) yapmak ve sayma değerini sıfırlamak için kullanılır. Reset 1 iken **CU** girişi ne olursa olsun sayıcı çıkışı ve sayma değeri sıfırlanır. Reset 0 iken **CU** girişinin 0 değeri, 1 değeri ve düşen kenar değerlerinde hiçbir işlem yapılmaz. Sadece yükselen kenarda işlem yapılır. **CV** değeri **PV** değerine eşit olana kadar bir artırma işlemi yapılır. Eşit olduğu anda sayıcı çıkışı set olur ve sayma durur .



Şekil 5.30 İleri sayıcının sembolü

Çizelge 5.17 İleri sayıcının doğruluk tablosu

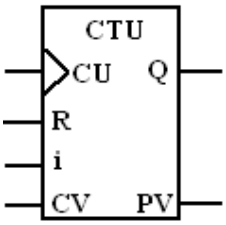
CU	R	İşlem
×	1	1. sayıcı çıkışı Q'yu resetle (OFF – 0) 2. sayma değeri CV'yi sıfır yap
0	0	NOP (No Operation – hiç bir işlem yapma)
1	0	NOP
↓	0	NOP
↑	0	Eğer CV < PV ise CV'yi bir arttır (CV = CV + 1). Eğer CV = PV ise CV'yi değiştirmeden tut ve sayıcı çıkışı Q'yu set et (ON – 1).

× : (don't care) ne olursa olsun

İleri sayıcı fonksiyonu **ctu**; **cu, reset, pv** ve **i** argümanlarından oluşmaktadır. Burada **cu** sayma girişini, **reset** sayma değeri ve çıkışı sıfırlama girişini, **pv** kurma değerini, **i** ise 0'dan 63'e kadar olan sayıcı numarasını ifade eder. Toplamda 64 farklı ileri sayıcı kullanılabilir. Çizelge 5.18'de ileri sayıcı (up counter-CTU) sembol ve C kodu görülmektedir.

Çizelge 5.18 Gerçekleştirilen ileri sayıcıya (up counter - CTU) ait sembol ve C kodu

C Kodu	Sembol

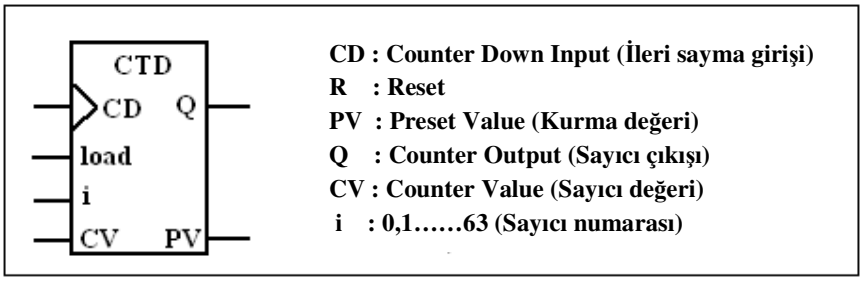
void ctu (unsigned int cu,unsigned int
reset,unsigned int pv,unsigned int i)
{
if (i<64)
{
if(reset==1)
{ CQ[i]=0;
CV[i]=0;}
else if(cu==0) CRED[i]=1;
else if(CRED[i]==1 && (CV[i]< pv))
{ CRED[i]=0;
CV[i]++;
if (CV[i]==pv) CQ[i]=1;
}
}
}
	
	CU =İleri sayma girişi (0 veya 1)
	R = Reset girişi (0 veya 1)
	PV- Kurma değeri (16 bitlik sabit sayı) = 1, 2, …65535
	CV – Sayma kaydedicisi (16 bitlik sabit sayı) = 1, 2,…65535
	Q = Sayıcı çıkışı (0 veya 1)
	i = Sayıcı numarası (0,1,…63)

İleri sayıcı (up counter - CTU) fonksiyonunun çalışması şu şekildedir: İlk olarak **reset** girişinde sinyal olup olmadığına bakılır. **reset** 1 ise sayma değeri **CV[i]** ve sayıcı çıkışı **CQ[i]** sıfırlanır. **reset** 0 ise sayma girişi **cu** kontrol edilir. Sayma girişi 0 iken yükselen kenar algılayıcı kaydedicisi **CRED[i]** 1 olur. Yükselen kenar algılayıcısı 1 iken sayma değeri **CV[i]** kurma değeri **pv**'den küçükse yükselen kenar algılayıcı kaydedicisi **CRED[i]** 0 olur ve sayma değeri **CV[i]** bir arttırılır. 'cu' ileri sayma girişindeki sinyalin her yükselen kenarında bir arttırılan **CV[i]** kurma değeri **pv**'ye eşit olduğunda sayma durur ve sayıcı çıkışı **CQ[i]** 1 yapılır. İndeks argümanı olan **i** değeri neyse bilgi aynı numaralı kaydedicilere aktarılır. Kurma ve sayma değerleri 1– 65535 arasında sayılar olabilir.

5.9.2 Geri sayıcı (down counter – CTD)

Geri sayıcı (down counter - CTD) bir sayma değerinin minimum değere ulaştığını göstermek için kullanılan sayıcı türüdür. Şekil 5.31'de geri sayıcının sembolü ve Çizelge 5.19'da doğruluk tablosu görülmektedir.

CD sayma girişinden gelen sinyalin her bir yükselen (↑) kenarında **CV** sayma değeri 1 azaltılır. **load** yükleme girişi çıkışın 0 yapılmasını ve kurma değeri **PV**'nin **CV**'ye yüklenmesini sağlar. **CD** sayma girişinin her yükselen kenarında bir azaltılan **CV** sayma değeri sıfıra eşit olduğu anda geri sayma durur ve sayıcı çıkışı set (1) yapılır [29].



Şekil 5.31 Geri sayıcının (CTD) sembolü

Çizelge 5.19 Geri sayıcının (CTD) doğruluk tablosu

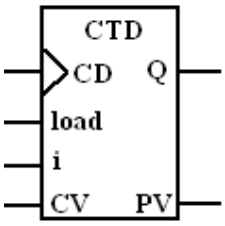
CD	LD	İşlem
×	1	1. sayıcı çıkışı Q' yu resetle (OFF – 0) 2. Sayma değeri CV'ye kurma değeri PV'yi yükle
0	0	NOP (No Operation – hiç bir işlem yapma)
1	0	NOP
↓	0	NOP
↑	0	Eğer CV > 0 ise CV'yi bir azalt. (CV = CV - 1). Eğer CV = 0 ise CV'yi değiştirmeden tut ve sayıcı çıkışı Q'yu set et (ON – 1).

× : (don't care) ne olursa olsun

Geri sayıcı fonksiyonu **cd, load, pv** ve **i** argümanlarından oluşur. Burada **cd** geri sayma girişini, **load** yükleme girişini, **pv** kurma değerini ve **i** de 0'dan 63'e kadar olan sayıcı numarasını ifade eder. Toplamda 64 farklı geri sayıcı kullanılabilir. Çizelge 5.20'de gerçekleştirilen geri sayıcının (down counter - CTD) sembolü ve C kodu görülmektedir.

Çizelge 5.20 Gerçekleştirilen geri sayıcının (down counter - CTD) sembolü ve C kodu

C Kodu	Sembol

void ctd (unsigned int cd,unsigned int load,unsigned int pv,unsigned int i)
{
if (i<64)
{
if(load==1)
{ CQ[i]=0;
CV[i]=pv;}
else if(cd==0) CRED[i]=1;
else if(CRED[i]==1 && (CV[i]>0))
{ CRED[i]=0;
CV[i]--;
if (CV[i]==0) CQ[i]=1;
}
}
}
	
	CD = Geri sayma girişi (0 veya 1)
	load =Yükleme girişi (0 veya 1)
	PV – Kurma değeri (16 bitlik sabit sayı)
	= 1, 2, …65535
	CV (16 bitlik sabit sayı) = 1, 2,…65535
	Q = Sayıcı çıkışı (0 veya 1)
	i = Sayıcı numarası (0,1,…63)

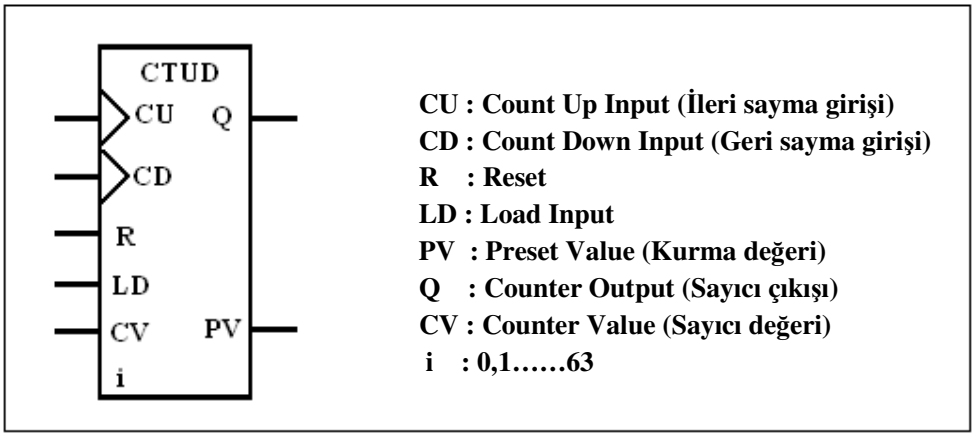
Geri sayıcı (down counter - CTD) fonksiyonunun çalışması şu şekildedir: İlk olarak **load** yükleme girişi kontrol edilir. Eğer **load** girişindeki sinyal 1 ise çıkış kaydedicisi **CQ[i]** sıfırlanır ve kurma değeri **pv** sayma kaydedicisi **CV[i]**'ye yüklenir. Geri sayıcı sayma giriş sinyali 0 iken yükselen kenar algılama kaydedicisi 1 olur. Yükselen kenar algılayıcı 1 iken sayma değeri de sıfırdan büyük ise sayma değeri **CV[i]** bir azaltılır. Yükselen kenar algılama kaydedicisi de 0 yapılır. Sayma değeri **CV[i]** sıfıra düştüğü anda çıkış kaydedicisi **CQ[i]** 1 yapılır. İndeks argümanı olan **i**'nin değeri neyse bilgi aynı numaralı kaydediciye aktarılır. Kurma ve sayma değerleri 1– 65535 arasında sayılar olabilir. Toplam 64 tane geri sayıcı tanımlanmıştır.

5.9.3 İleri/geri sayıcı (up/down counter – CTUD)

İleri/geri sayıcıların (up/down counter - CTUD), ileri sayma için kullanılan **CU** ve geri sayma için kullanılan **CD** olmak üzere iki sayma girişi vardır. İleri/geri sayıcı, **CU** girişinde tespit edilen yükselen kenarlarda (↑) ileri sayar, **CD** girişinde tespit edilen yükselen kenarlarda (↑) ise geri sayar. **PV** sayıcının saymaya başlayacağı sayı değerini belirler. Sayıcı **PV** değerine ulaştığında sayıcı çıkışı Q set (1) yapılır ve sayma işlemi durur.

R reset girişi sayıcı çıkışını 0 (OFF) yapmak ve **CV** sayma değerini sıfırlamak için kullanılabilir. **LD** yükleme girişi, **CV**'ye **PV** kurma değerinin yüklenmesi için kullanılır. Sayıcı sıfır değerine ulaşınca sayma durur [29].

İleri/geri sayıcı (CTUD) sembolü Şekil 5.32'de ve doğruluk tablosu Çizelge 5.21'de görülmektedir.



Şekil 5.32 İleri/geri sayıcının (CTUD) sembolü

Çizelge 5.21 İleri/geri sayıcı (CTUD) doğruluk tablosu

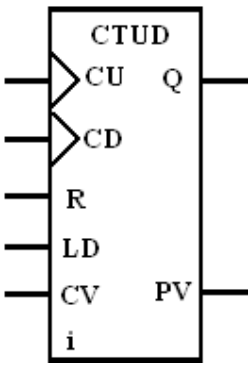
CU	CD	R	LD	İşlem
×	×	1	×	1. sayıcı çıkışı Q'yu resetle (OFF – 0) 2. sayma değeri CV'yi sıfır yap
×	×	0	1	Sayma değeri CV'ye kurma değeri PV'yi yükle
0	0	0	0	NOP (No Operation – hiç bir işlem yapma)
0	1	0	0	NOP
1	0	0	0	NOP
1	1	0	0	NOP
1	↑	0	0	NOP
↑	1	0	0	NOP
×	↓	0	0	NOP
↓	×	0	0	NOP
↑	0	0	0	Eğer CV < PV ise CV'yi bir arttır. Eğer CV = PV ise CV'yi değiştirmeden tut ve sayıcı çıkışı Q'yu set et (ON – 1).
0	↑	0	0	Eğer CV > 0 ise CV'yi bir azalt.

× : (don't care) ne olursa olsun

İleri/geri sayıcı fonksiyonu; **cu**, **cd, reset, load, pv** ve **i** argümanlarından oluşmaktadır. Burada **cu** ileri sayma girişini, **cd** geri sayma girişini, **reset** çıkış ve sayma değerini sıfırlama girişini, **load** yükleme girişini, **pv** yükleme değerini, **i** 0'dan 63'e kadar olan sayıcı numarasını ifade eder. Çizelge 5.22'de ileri/geri sayıcıya (CTUD) ait sembol ve C kodu görülmektedir.

Çizelge 5.22 İleri/geri sayıcıya (CTUD) ait sembol ve C kodu

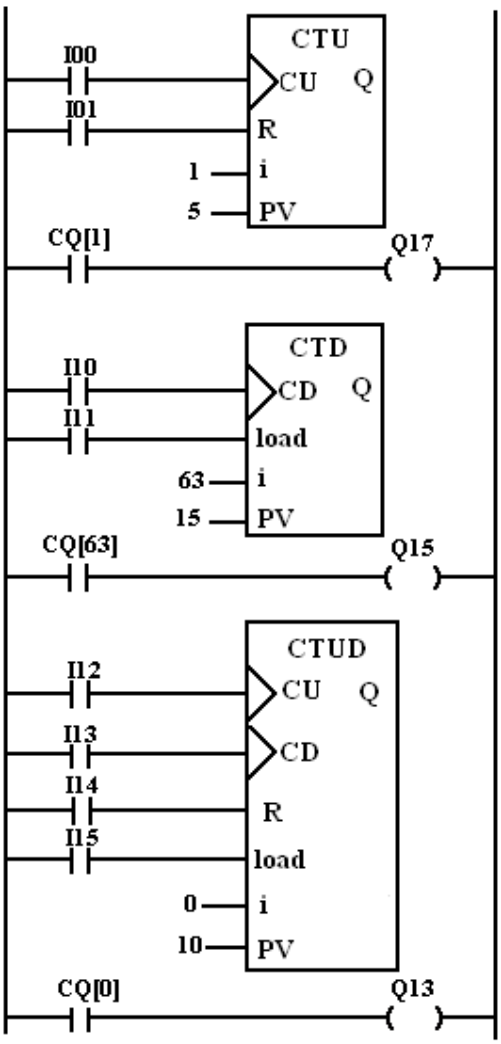
C Kodu	Sembol

void ctud (unsigned int cu,unsigned int cd,unsigned int reset,unsigned int load,unsigned int pv, unsigned int i)
{
if (i<64)
{
if(reset==1)
{ CQ[i]=0;
CV[i]=0;}
else if(load==1) {CV[i]=pv;}
elseif(cu==0 && cd==0) CRED[i]=1;
else if((CRED[i]==1 && cu==1 && cd==0)&& (CV[i]< pv))
{
CRED[i]=0;
CV[i]++;
}
else if((CRED[i]==1 && cu==0 && cd==1)&& (CV[i]> 0))
{
CRED[i]=0;
CV[i]--;
}
if (CV[i]==pv) CQ[i]=1;
if (CV[i]< pv) CQ[i]=0;
}
}
	
	CU = İleri sayma girişi (0 veya 1)
	CD= Geri sayma girişi (0 veya 1)
	load =Yükleme girişi (0 veya 1)
	R =Reset girişi (0 veya 1)
	PV – Kurma değeri (16 bitlik sabit sayı) = 1, 2, …65535
	CV- 16 bitlik Sayma kaydedicisi = 1, 2, …65535
	Q = Sayıcı çıkışı (0 veya 1)
	i = Sayıcı numarası (0,1…63)

İleri/geri sayıcı (CTUD) fonksiyonunun çalışması şu şekildedir: İlk olarak **reset** girişine bakılır. **reset** sinyali aktifse (reset=1 ise) sayma değeri kaydedicisi **CV[i]** ve çıkış kaydedicisi **CQ[i]** sıfırlanır. Eğer **reset** sinyali 0 ise **load** yükleme sinyali 1 olduğu anda kurma değeri **pv**, sayma değeri kaydedicisi **CV[i]**'ye aktarılır. Yükleme yapılmamışsa ileri ve geri sayma girişleri kontrol edilir. Sayma girişlerinde sinyal yoksa yükselen kenar algılama kaydedicisi CRED[i] 1 yapılır. CRED[i] 1 iken sayma girişlerinden herhangi birinden sinyal gelmişse CRED[i] tekrar sıfırlanır. İleri sayma sinyali gelmişse sayma değeri CV[i] bir artırılır, geri sayma sinyali gelmişse bir azaltılır. Sayma değeri CV[i], kurma değeri **pv**'ye eşit olduğunda çıkış kaydedicisi CQ[i] 1 olur. Sayma değeri kurma değerinden küçük olduğu müddetçe çıkış 0'da kalmaya devam eder.

5.9.4 Sayıcı fonksiyonları örnekleri

Sayıcı fonksiyonlarına ait örnekler bu kısımda verilmiştir. Şekil 5.33 (a)'da ileri sayıcı, geri sayıcı ve ileri/geri sayıcı örnekleri merdiven diyagramı ve Şekil 5.33 (b)'de kullanıcı programı görülmektedir.



a)

```
//---------------------------

OUTPUT=CV[1];
ctu(I00,I01,5,1);
ld(CQ[1]);
Q17=out();

OUTPUT=CV[63];
ctd(I10,I11,15,63);
ld(CQ[63]);
Q15=out();

OUTPUT=CV[0];
ctud(I12,I13,I14,I15,10,0);
ld(CQ[0]);
Q13=out();

//---------------------------
```

b)

Şekil 5.33 Sayıcı fonksiyonları örnekleri

a) Merdiven diyagramı b) Kullanıcı programı

Örneklerden de görüldüğü gibi ileri veya geri sayma girişlerinin bağlı olduğu girişlerden sinyal geldiğinde sayıcı ileri veya geri saymaya başlar. İleri sayıcıda, CU ileri sayma girişinden sinyal geldikçe sayıcı değeri arttırılır. Sayılan değer kurma değeri PV'ye ulaşınca sayıcı çıkışı CQ aktif olur ve bağlı olduğu çıkışı 1 yapar. Reset girişinden verilecek sinyal ile istenilen anda sayıcı sıfırlanabilir. Geri sayıcıda load yükleme girişi yardımıyla sayıcıya bir sabit sayı yüklenir. CD geri sayma girişinden sinyal geldikçe sayıcı değeri azaltılır. Sayılan değer kurma değeri PV'ye ulaşınca sayıcı çıkışı CQ aktif olur ve bağlı olduğu çıkışı 1 yapar. Benzer şekilde ileri/geri sayıcıda load girişiyle sayıcıya bir sabir değer yüklenir. Hangi sayma girişinden sinyal gelirse sayıcı o yöne doğru sayar. Kurma değerine ulaşılınca sayıcı çıkışı CQ aktif olur ve bağlı olduğu çıkışı 1 yapar. Reset girişinden verilen sinyal ile sayıcı sıfırlanabilmektedir.

BÖLÜM VI

SONUÇ

Bu tez çalışmasında 16 bitlik bir PIC mikrodenetleyicisi ile bir programlanabilir lojik denetleyici (PLC) tasarımı ve gerçekleştirilmesi yapılmıştır. Çalışma sırasında PIC ve PLC'ler hakkında araştırma yapılmış, bu teknolojilerin gündelik hayattaki yeri ve önemi kavranmaya çalışılmıştır. Yine bu çalışmada Microchip firmasının üretmiş olduğu Explorer 16 Geliştirme ve Uygulama seti incelenmiştir. Bu set üzerindeki PIC24FJ128GA010 mikrodenetleyicisi MPLAB IDE arayüz programı aracılığıyla yüksek seviyeli yazılım dillerinden olan C dili ile programlanarak PLC yazılımı geliştirilmiştir. Çevresel giriş/çıkış birimleri için, UZAM_PLC giriş/çıkış modülü için tasarlanmış olan 8 bitlik I/O genişletme kartlarından ikisi kullanılarak 16 bitlik bir giriş/çıkış kartı elde edilmiştir. Bu giriş/çıkış kartları ve ana modül kartı olan Explorer 16 Geliştirme ve Uygulama Seti, Explorer 16 'Daughter Board' (yardımcı kart) yoluyla birleştirilerek tasarlanan PLC gerçekleştirilmiştir.

Bu tez çalışması sonucunda gerçekleştirilen PLC; program hafıza kapasitesi, çalışma hızı ve diğer donanım özellikleri bakımından düşük kapasiteli 8 bitlik ve yüksek kapasiteli 32 bitlik benzerlerine göre orta büyüklükte olan bir PLC tasarımıdır.

Bu çalışma; piyasadaki benzerlerinden eksiği olmayacak şekilde, bir otomasyon sisteminin gerçekleştirilmesi için gerekli yazılım ve donanımı sağlama amacıyla tasarlanmıştır. Yapılan çalışma sonucunda elde edilen bu tasarım; merkezi işlem birimiyle (CPU), giriş/çıkış (I/O) birimiyle ve yazılımıyla bir PLC'den beklenen işlevlere cevap verebilecek niteliktedir. Teknik ve donanım olarak 16 giriş/16 çıkış kapasitesine kadar otomasyon işlemlerinde rahatlıkla kullanılabilecek düzeyde bir çalışmadır. 16 girişli/16 çıkışlı PLC donanımının anlatıldığı Bölüm IV'te de ifade edildiği gibi bir PLC arayüz programı geliştirilmesi halinde gerçekleştirilen bu tasarımda "Merdiven Diyagramı" rahatlıkla kullanılabilecektir. Bu yüksek lisans tez çalışmasının ileriki zamanlarda yapılabilecek olan benzer çalışmalara ışık tutması ümit edilmektedir.

KAYNAKLAR

[1] M. Uzam " PICBIT PLC ile Lojik Tasarım" , Birsen Yayınevi , 2008

[2] M. Uzam, "PLC with PIC16F648A Microcontroller Part 1," Electronics World, volume 114, issue 1871, November 2008, pp. 21-25.

[3] M. Uzam, "PLC with PIC16F648A Microcontroller Part 2," Electronics World, volume 114, issue 1872, December 2008, pp. 29-35.

[4] M. Uzam, "PLC with PIC16F648A Microcontroller Part 3," Electronics World, volume 115, issue 1873, pp. 30-34, January 2009.

[5] M. Uzam, "PLC with PIC16F648A Microcontroller Part 4," Electronics World, volume 115, issue 1874, pp. 34-40, February 2009.

[6] M. Uzam, "PLC with PIC16F648A Microcontroller - Part 5," Electronics World, volume 115, issue 1875, pp. 30-33, March 2009.

[7] M. Uzam, "PLC with PIC16F648A Microcontroller - Part 6," Electronics World, volume 115, issue 1876, pp. 26-30, April 2009.

[8] M. Uzam, "PLC with PIC16F648A Microcontroller - Part 7," Electronics World, volume 115, issue 1877, pp. 30-32, May 2009.

[9] M. Uzam, "PLC with PIC16F648A Microcontroller - Part 8," Electronics World, volume 115, issue 1878, pp. 30-32, June 2009.

[10] M. Uzam, "PLC with PIC16F648A Microcontroller - Part 9," Electronics World, volume 115, issue 1879, pp. 29-34, July 2009.

[11] M. Uzam, "PLC with PIC16F648A Microcontroller-Part 10," Electronics World, volume 115, issue 1880, pp. 29-34, August 2009.

[12] M. Uzam, "PLC with PIC16F648A Microcontroller-Part 11," Electronics World, volume 115, issue 1881, pp. 38-42, September 2009.

[13] M. Uzam, "PLC with PIC16F648A Microcontroller-Part 12," Electronics World, volume 115, issue 1882, pp. 36-41, October 2009.

[14] M. Uzam, "PLC with PIC16F648A Microcontroller-Part 13," Electronics World, volume 115, issue 1883, pp. 42-44, November 2009.

[15] M. Uzam, "PLC with PIC16F648A Microcontroller-Part 14," Electronics World, volume 115, issue 1884, pp. 40-42, December 2009.

[16] M. Uzam, "PLC with PIC16F648A Microcontroller-Part 15," Electronics World, volume 116, issue 1885, pp. 35-39, January 2010.

[17] M. Uzam, “PLC with PIC16F648A Microcontroller Part 16,” Electronics World, volume 116, issue 1886, pp. 41-42, February 2010.

[18] M. Uzam, “PLC with PIC16F648A Microcontroller Part 17,” Electronics World, volume 116, issue 1887, pp. 41-43, March 2010.

[19] M. Uzam, “PLC with PIC16F648A Microcontroller Part 18,” Electronics World, volume 116, issue 1888, pp. 41-43, April 2010.

[20] M. Uzam, “PLC with PIC16F648A Microcontroller Part 19,” Electronics World, volume 116, issue 1889, pp. 39-43, May 2010.

[21] M. Uzam, “PLC with PIC16F648A Microcontroller Part 20,” Electronics World, volume 116, issue 1890, pp. 38, 40, June 2010.

[22] M. Uzam, “PLC with PIC16F648A Microcontroller Part 21,” Electronics World, volume 116, issue 1891, pp. 40-41, July 2010.

[23] M. Uzam, “PLC with PIC16F648A Microcontroller Part 22,” Electronics World, volume 116, issue 1892, pp. 40-42, August 2010.

[24] M. Uzam, “PIC16F877A Mikro-Denetleyicisi Temelli PIC PLC Donanım”, sayı 153, sayfa 22-24, Aralık 2009.

[25] M. Uzam, “PIC16F877A Mikro-Denetleyicisi Temelli PIC PLC - 2 Temel Yazılım”, sayı 154, sayfa 18-26, Ocak 2010.

[26] M. Uzam, “PIC16F877A Mikro-Denetleyicisi Temelli PIC PLC - 3 Kontak ve Röle Temelli Makrolar”, Endüstri & Otomasyon, sayı 155, sayfa 30-33, Şubat 2010.

[27] M. Uzam, “PIC16F877A Mikro-Denetleyicisi Temelli PIC PLC - 4 Flip-Flop Makroları”, Endüstri & Otomasyon, sayı 156, sayfa 16-22, Mart 2010.

[28] M. Uzam, “PIC16F877A Mikro-Denetleyicisi Temelli PIC PLC - 5 Zamanlayıcı Makroları”, Endüstri & Otomasyon, sayı 157, sayfa 30-37, Nisan 2010.

[29] M. Uzam, “PIC16F877A Mikro-Denetleyicisi Temelli PIC PLC - 6 Sayıcı Makroları”, Endüstri & Otomasyon, sayı 158, sayfa 16-22, Mayıs 2010.

[30] M. Uzam, “PIC16F877A Mikro-Denetleyicisi Temelli PIC PLC - 7 Karşılaştırma Makroları”, Endüstri & Otomasyon, sayı 159, sayfa 16-19, Haziran 2010.

[31] M. Uzam, “PIC16F877A Mikro-Denetleyicisi Temelli PIC PLC - 8 Aritmetik Makroları”, Endüstri & Otomasyon, sayı 160, sayfa 28-31, Temmuz 2010.

[32] M. Uzam, “PIC16F877A Mikro-Denetleyicisi Temelli PIC PLC - 9 Lojik Makroları”, Endüstri & Otomasyon, sayı 161, sayfa 28-31, Ağustos 2010.

[33] M. Uzam, “PIC16F877A Mikro-Denetleyicisi Temelli PIC PLC - 10 Kaydırma ve Döndürme Makroları”, Endüstri & Otomasyon, sayı 162, sayfa 24-29, Eylül 2010.

[34] M. Uzam, “PIC16F877A Mikro-Denetleyicisi Temelli PIC PLC - 11 Veri Seçici Makrolar”, Endüstri & Otomasyon, sayı 163, sayfa 30-34, Ekim 2010.

[35] M. Uzam, “PIC16F877A Mikro-Denetleyicisi Temelli PIC PLC - 12 Veri Dağıtıcı Makrolar”, Endüstri & Otomasyon, sayı 164, sayfa 16-20, Kasım 2010.

[36] M. Uzam, “PIC16F877A Mikro-Denetleyicisi Temelli PIC PLC - 13 Kod Çözücü Makroları”, Endüstri & Otomasyon, sayı 165, sayfa 16-22, Aralık 2010.

[37] M. Uzam, “PIC16F877A Mikro-Denetleyicisi Temelli PIC PLC - 14 Öncelikli Kodlayıcı Makroları”, Endüstri & Otomasyon, sayı 166, sayfa 16-21, Ocak 2011.

[38] M. Uzam, “PIC16F877A Mikro-Denetleyicisi Temelli PIC PLC - 15 Bir Örnek: Uzaktan Kumandalı Model Kapı”, Endüstri & Otomasyon, sayı 167, sayfa 18-24, Şubat 2011.

[39] Şükrü KİTİŞ, PIC 16F84 Mikrodenetleyicisi ile Bir Programlanabilir Lojik Denetleyici Tasarımı ve Uygulaması, Niğde Üniversitesi, Fen Bilimleri Enstitüsü, Elektrik-Elektronik Mühendisliği Anabilim Dalı, Yüksek Lisans Tezi, Ağustos 2007.

[40] S. Kurtulan, “PLC ile Endüstriyel Otomasyon”, Birsen Yayınevi, 2001

[41] S. Aksoy, “Programlanabilir Lojik Denetleyiciler ve Mühendislik Uygulamaları, Değişim Yayınları, 2004

[42] A. R. Kara, “İleri Kumanda Teknikleri”, Rebel Ofset, 2000

[43] http://www.microchip.com

[44] http://www.elektrotekno.com

[45] http://www.mühendisim.com

[46] http://www.picproje.com

get_inputs.c

```
#include "p24fj128ga010.h"
#include "p24fj_plc.h"
#include "stdio.h"

#define bit_get(p,m) ((p) & (m))
#define bit_set(p,m) ((p) |= (m))
#define bit_clear(p,m) ((p) &= ~(m))
#define bit_flip(p,m) ((p) ^= (m))
#define bit_write(c,p,m) (c ? bit_set(p,m) : bit_clear(p,m))
#define BIT(x) (0x01 << (x))


unsigned int get_inputs(void)
{
        int k;

        ShLd=   0;
        Nop()
        ShLd=   1;

        if (SDO==0)                             // I0.7
                bit_clear(INPUT,BIT(7));
        else
                bit_set(INPUT,BIT(7));

        CLKIN = 0;
        for(k=1;k>0;k--)
        CLKIN = 1;

        if (SDO==0)                             //I0.6
                bit_clear(INPUT,BIT(6));
        else
                bit_set(INPUT,BIT(6));

         CLKIN = 0;
         for(k=1;k>0;k--)
         CLKIN = 1;

         if (SDO==0)                            //I0.5
                bit_clear(INPUT,BIT(5));
        else
                bit_set(INPUT,BIT(5));

         CLKIN = 0;
                for(k=1;k>0;k--)
         CLKIN = 1;

         if (SDO==0)                            //I0.4
                bit_clear(INPUT,BIT(4));
        else
                bit_set(INPUT,BIT(4));
```

```
 CLKIN = 0;
        for(k=1;k>0;k--)
 CLKIN = 1;

if (SDO==0)                                     //I0.3
        bit_clear(INPUT,BIT(3));
else
        bit_set(INPUT,BIT(3));

 CLKIN = 0;
        for(k=1;k>0;k--)
 CLKIN = 1;

 if (SDO==0)                                    //I0.2
        bit_clear(INPUT,BIT(2));
else
        bit_set(INPUT,BIT(2));

 CLKIN = 0;
        for(k=1;k>0;k--)
 CLKIN = 1;

 if (SDO==0)                                    //I0.1
        bit_clear(INPUT,BIT(1));
else
        bit_set(INPUT,BIT(1));

 CLKIN = 0;
        for(k=1;k>0;k--)
 CLKIN = 1;

 if (SDO==0)                                    //I0.0
        bit_clear(INPUT,BIT(0));
else
        bit_set(INPUT,BIT(0));

 CLKIN = 0;
        for(k=1;k>0;k--)
 CLKIN = 1;

 if (SDO==0)                                    //I1.7
        bit_clear(INPUT,BIT(15));
else
        bit_set(INPUT,BIT(15));

 CLKIN = 0;
        for(k=1;k>0;k--)
 CLKIN = 1;

 if (SDO==0)                                    //I1.6
        bit_clear(INPUT,BIT(14));
else
        bit_set(INPUT,BIT(14));

 CLKIN = 0;
for(k=1;k>0;k--)
```

```
	CLKIN = 1;

	if (SDO==0)                          //I1.5
		bit_clear(INPUT,BIT(13));
	else
		bit_set(INPUT,BIT(13));

	CLKIN = 0;
		for(k=1;k>0;k--)
	CLKIN = 1;

	if (SDO==0)                          //I1.4
		bit_clear(INPUT,BIT(12));
	else
		bit_set(INPUT,BIT(12));

	CLKIN = 0;
		for(k=1;k>0;k--)
	CLKIN = 1;

	if (SDO==0)                          //I1.3
		bit_clear(INPUT,BIT(11));
	else
		bit_set(INPUT,BIT(11));

	CLKIN = 0;
		for(k=1;k>0;k--)
	CLKIN = 1;

	if (SDO==0)                          //I1.2
		bit_clear(INPUT,BIT(10));
	else
		bit_set(INPUT,BIT(10));

	CLKIN = 0;
		for(k=1;k>0;k--)
	CLKIN = 1;

	if (SDO==0)                          //I1.1
		bit_clear(INPUT,BIT(9));
	else
		bit_set(INPUT,BIT(9));

	CLKIN = 0;
		for(k=1;k>0;k--)
	CLKIN = 1;

	if (SDO==0)                          //I1.0
		bit_clear(INPUT,BIT(8));
	else
		bit_set(INPUT,BIT(8));
	return 0;
}
```

send_outputs.c

```
#include "p24FJ128GA010.h"
#include "p24FJ_plc.h"

unsigned int send_outputs (void)
{
            int i;

            RST=0;
            Nop()
            RST=1;

            RCK = 0;
            for(i=0;i<16;i++)
    { if (( OUTPUT & 0x8000) == 0)
       SERIN = 0;
      else
       SERIN = 1;
      OUTPUT=OUTPUT<<1;
       SRCK = 1;
             Nop()
       SRCK = 0;
                        }
    if (i==16)
     RCK = 1;
return 0;
}
```

main.c

```
#include "p24FJ128GA010.h"
#include "p24FJ_PLC.h"
#include "macro.h"

int main(void)
{

        TRISA = 0X0008;

        _T1IP = 4;                  // this is the default value anyway
        TMR1 = 0;                   // clear the timer
        PR1 = 0xffff-1;//25000-1;   // set the period register

                // 2.1 configure Timer1 module
        T1CON = 0x8030; // enabled, prescaler 1:64, internal clock

        // 2.2 init the Timer 1 Interrupt, clear the flag, enable the source
        _T1IF = 0;
        _T1IE = 1;

        // 2.3 init the processor priority level
        _IPL = 0;// this is the default value anyway

RST=0;
Nop()
RST=1;

while(1){
get_inputs();
TIMER = TMR1;

//-------------- KULLANICI PROGRAMI BURADAN BAŞLAR----------

//----------------------------örnek12---------------------------------------

//OUTPUT=CV[1];
//ctu(I00,I01,5,1);
//ld(CQ[1]);
//Q17=out();

//OUTPUT=CV[63];
//ctd(I10,I11,15,63);
//ld(CQ[63]);
//Q15=out();

//OUTPUT=CV[0];
//ctud(I12,I13,I14,I15,10,0);
//ld(CQ[0]);
//Q13=out();
```

```
//----------------------------örnek11---------------------------------------
//ld(I10);
//tof(Tm12,153,0);
//ld(TOFQ[0]);
//Q10=out();
//
//ld(I17);
//tof(Tm13,153,63);
//ld(TOFQ[63]);
//Q17=out();
//
//----------------------------örnek10---------------------------------------
//ld(I10);
//ton(Tm12,153,0);
//ld(TONQ[0]);
//Q10=out();
//
//ld(I17);
//ton(Tm13,153,63);
//ld(TONQ[63]);
//Q17=out();

//----------------------------örnek9----------------------------------------

//ld(LOGIC1);
//OUTPUT=SHIFT_R(INPUT,3);

//ld(LOGIC1);
//OUTPUT=SHIFT_L(INPUT,3);

//ld(LOGIC1);
//OUTPUT=XNOR(0xf0f0,INPUT);

//ld(LOGIC1);
//OUTPUT=NOR(0xf0f0,INPUT);

//ld(LOGIC1);
//OUTPUT=NAND(0xf0f0,INPUT);

//ld(LOGIC1);
//OUTPUT=NOT(INPUT);

//ld(LOGIC1);
//OUTPUT=XOR(0b1111000011110000,INPUT);

//ld(LOGIC1);
//OUTPUT=OR(0b1111000011110000,INPUT);

//ld(LOGIC1);
//OUTPUT=AND(0b1111000011110000,INPUT);
```

```
//----------------------------örnek8---------------------------------------

//ld(Tm17);
//OUTPUT=dec(INPUT);
//
//
//ld(Tm17);
//OUTPUT=inc(INPUT);

//ld(I00);
//OUTPUT=div(6,3);

//ld(I00);
//OUTPUT=mul(5,6);

//ld(I00);
//OUTPUT=sub(32,16);

//ld(I00);
//OUTPUT=add(16,16);
//
//----------------------------örnek7---------------------------------------

//ld(LOGIC1);
//Q10=NE(INPUT,64);
//
//ld(LOGIC1);
//Q11=LE(INPUT,64);
//
//ld(LOGIC1);
//Q12=LT(INPUT,256);
//
//ld(LOGIC1);
//Q13=EQ(INPUT,256);
//
//ld(LOGIC1);
//Q14=GE(INPUT,128);
//
//ld(LOGIC1);
//Q15=GT(INPUT,15);

//----------------------------örnek6---------------------------------------

//ld(I00);
//Q00=r_edge0();
//
//ld(I01);
//Q01=r_edge1();
//
//ld(I02);
//Q02=r_edge2();
//
//
//ld(I10);
//Q10=f_edge0();
//
```

```
//ld(I11);
//Q11=f_edge1();
//
//ld(I12);
//Q12=f_edge2();
//
//----------------------------örnek5---------------------------------------

//Q01=sr_ff0(I00,I01);
//
//Q07=sr_ff7(I06,I07);
//
//Q10=sr_ff8(I10,I11);
//
//Q17=sr_ff15(I16,I17);

//----------------------------örnek4---------------------------------------

//ld(I00);
//Q00=latch0(I01);
//
//ld(I10);
//Q10=latch15(I11);

//----------------------------örnek3---------------------------------------

//Q00=in_out(I00);
//
//ld(I00);
//Q02=and(I01);
//
//ld(I03);
//not();
//Q03=out();
//
//ld(I03);
//Q04=or(I04);
//
//ld(I05);
//Q06=xor(I06);
//
//ld(I10);
//Q10=nor(I11);
//
//ld(I13);
//Q14=nand(I14);
//
//ld(I15);
//Q17=xnor(I16);
```

```
//----------------------------örnek2---------------------------------------

//ld(I10);
//or_not(I11);
//or(I12);
//Q17=or_not(I13);
//
//ld(I00);
//and(I01);
//and_not(I02);
//Q13=and_not(I03);
//
//ld_not(I10);
//Q10=out();

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

ld(I00);
//_RA6=TEMP;
Q03=out();
//Q02=(Tm14);
//
//ld(LOGIC1);
//Q04=out();
//
//ld(I13);
//Q13=out_not();
//
//ld_not(I17);
//Q17=out();

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

//ld(I10);
//ton0(Tm13,153);
//Q10 = TONQ0;
//Q07 = Tm16;
//Q17 = Tm17;
//OUTPUT=TIMER;

//ld (I00);
//Q00=xnor(I01);

//ld (I00);
//Q00=xor(I01);

//Q00=nand5(I00,I01,I02,I03,I04);

//Q00=and5(I00,I01,I02,I03,I04);

//Q00=nor5(I00,I01,I02,I03,I04);

//Q00=or5(I00,I01,I02,I03,I04);

//OUTPUT=INPUT;
```

```
//M0=0x0f0f;
//OUTPUT=M0;

//if (bit_get(INPUT,BIT(15)))
//bit_set(OUTPUT,BIT(15));
//else
//bit_clear(OUTPUT,BIT(15));

//if (I07)
//bit_set(PORTA, BIT(4));
//else
//bit_clear(PORTA, BIT(4));

//if (I07)
//bit_set(PORTA, BIT(4));
//
//if (I06)
//bit_clear(PORTA, BIT(4));


//---------- KULLANICI PROGRAMI BURADA BİTER----------
  send_outputs();
            }
}
```

macro.h

```
#ifndef macro_H_
#define macro_H_

//---------------------------------------------------------------------------
#define bit_get(p,m) ((p) & (m))
#define bit_set(p,m) ((p) |= (m))
#define bit_clear(p,m) ((p) &= ~(m))
#define bit_flip(p,m) ((p) ^= (m))
#define bit_write(c,p,m) (c ? bit_set(p,m) : bit_clear(p,m))
#define BIT(x) (0x01 << (x))

//---------------------------------------------------------------------------
void                _ISR _T1Interrupt           (void);
unsigned int        get_inputs                  (void);
unsigned int        send_outputs        (void);
void                ld              (unsigned int in);
unsigned int        out                 (void);
unsigned int         not          (void);
unsigned int        in_out          (unsigned int in);

unsigned int        r_edge0   (void);
unsigned int        f_edge0   (void);
unsigned int        latch0    (unsigned int D);
unsigned int        sr_ff0    (unsigned int S, unsigned int R);
unsigned int        jkff_r0   (unsigned int J, unsigned int K);
unsigned int        tff_r0    (unsigned int T);
unsigned int        inc       (unsigned int in);

unsigned int          or    (unsigned int in);
unsigned int        or2       (unsigned int in1, unsigned int in2);
unsigned int        or3       (unsigned int in1, unsigned int in2, unsigned int in3);
unsigned int        or4       (unsigned int in1, unsigned int in2, unsigned int in3, unsigned int in4);
unsigned int        or5       (unsigned int in1, unsigned int in2, unsigned int in3, unsigned int in4, unsigned int in5);

unsigned int         and    (unsigned int in);
unsigned int         and2   (unsigned int in1, unsigned int in2);
unsigned int        and3      (unsigned int in1, unsigned int in2, unsigned int in3);
unsigned int        and4      (unsigned int in1, unsigned int in2, unsigned int in3, unsigned int in4);
unsigned int        and5      (unsigned int in1, unsigned int in2, unsigned int in3, unsigned int in4, unsigned int in5);


unsigned int         or_not   (unsigned int in);
unsigned int         and_not  (unsigned int in);
unsigned int         xor     (unsigned int in);
unsigned int        xnor     (unsigned int in);

unsigned int         nor     (unsigned int in);
unsigned int         nor2    (unsigned int in1, unsigned int in2);
unsigned int         nor3    (unsigned int in1, unsigned int in2, unsigned int in3);
unsigned int         nor4    (unsigned int in1, unsigned int in2, unsigned int in3, unsigned int in4);
unsigned int        nor5    (unsigned int in1, unsigned int in2, unsigned int in3, unsigned int in4, unsigned int in5);

unsigned int         nand    (unsigned int in);
```

```
unsigned int          nand2     (unsigned int in1, unsigned int in2);
unsigned int          nand3     (unsigned int in1, unsigned int in2, unsigned int in3);
unsigned int          nand4     (unsigned int in1, unsigned int in2, unsigned int in3, unsigned int in4);
unsigned int          nand5     (unsigned int in1, unsigned int in2, unsigned int in3, unsigned int in4, unsigned int in5);



void      ton (unsigned int clk, unsigned int sbt,unsigned int i);
void      tof (unsigned int clk, unsigned int sbt,unsigned int i);
void      ctu (unsigned int cu,unsigned int reset,unsigned int pv,unsigned int i);
void      ctud (unsigned int cu,unsigned int cd,unsigned int reset,unsigned int load,unsigned int pv, unsigned int i);

//------------------------------------------------------------------------
union     R16bit    INPUTS;
union     R16bit    TIMERS;
union     R16bit  TEMPS;
union     R16bit    OUTPUTS;
union     R16bit    MARKER0;
union     R16bit    MARKER1;
union     R16bit    MARKER2;
union     R16bit    MARKER3;
union     R16bit    REDS;
union     R16bit    FEDS;
union     R16bit    LATCHS;
union     R16bit    SRFFS;


unsigned int          TEMP_REG;
unsigned int          CV[64];
unsigned char         CQ[64];
unsigned char         CRED[64];
unsigned int          TON[64];
unsigned char         TONQ[64];
unsigned char         TONRED[64];
unsigned int          TOF[64];
unsigned char         TOFQ[64];
unsigned char         TOFRED[64];


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

void _ISR _T1Interrupt( void)
{
          _T1IF = 0;
}
//------------------------------------------------------------------------
```

```
//----------------- Kontak ve röle temelli fonksiyonlar -------------------------

void ld(unsigned int in)
{
if (in==0) TEMP=0;
else TEMP=1;
}

void ld_not(unsigned int in)
{
if (in==0) TEMP=1;
else TEMP=0;
}

unsigned int out (void)
{
if (TEMP==0) TEMP=0;
else TEMP=1;
return TEMP;
}

unsigned int out_not (void)
{
if (TEMP==0) TEMP=1;
else TEMP=0;
return TEMP;
}

//---------------------------------
unsigned int          and (unsigned int in)
{
if (TEMP && in) TEMP=1;
else TEMP=0;
return TEMP;
}

unsigned int          and2 (unsigned int in1, unsigned int in2)
{
if (in1 && in2) TEMP=1;
else TEMP=0;
return TEMP;
}

unsigned int          and3       (unsigned int in1, unsigned int in2, unsigned int in3)
{
if (in1 && in2 && in3) TEMP=1;
else TEMP=0;
return TEMP;
}

unsigned int          and4       (unsigned int in1, unsigned int in2, unsigned int in3, unsigned int in4)
{
if (in1 && in2 && in3 && in4) TEMP=1;
else TEMP=0;
return TEMP;
}
```

```
unsigned int		and5	(unsigned int in1, unsigned int in2, unsigned int in3, unsigned int in4, unsigned int in5)
{
if (in1 && in2 && in3 && in4 && in5) TEMP=1;
else TEMP=0;
return TEMP;
}

unsigned int		and_not (unsigned int in)
{
if (TEMP==1 && in==0) TEMP=1;
else TEMP=0;
return TEMP;
}

//---------------------------------
unsigned int		or (unsigned int in)
{
if (TEMP || in) TEMP=1;
else TEMP=0;
return TEMP;
}

unsigned int		or2 (unsigned int in1, unsigned int in2)
{
if (in1 || in2) TEMP=1;
else TEMP=0;
return TEMP;
}

unsigned int		or3	(unsigned int in1, unsigned int in2, unsigned int in3)
{
if (in1 || in2 || in3) TEMP=1;
else TEMP=0;
return TEMP;
}

unsigned int		or4	(unsigned int in1, unsigned int in2, unsigned int in3, unsigned int in4)
{
if (in1 || in2 || in3 || in4) TEMP=1;
else TEMP=0;
return TEMP;
}

unsigned int		or5	(unsigned int in1, unsigned int in2, unsigned int in3, unsigned int in4, unsigned int in5)
{
if (in1 || in2 || in3 || in4 || in5) TEMP=1;
else TEMP=0;
return TEMP;
}

unsigned int		xor (unsigned int in)
{
if (TEMP != in) TEMP=1;
else TEMP=0;
return TEMP;
}
```

```
unsigned int            xnor (unsigned int in)
{
if (TEMP == in) TEMP=1;
else TEMP=0;
return TEMP;
}


unsigned int            or_not (unsigned int in)
{
if (TEMP==1 || in==0) TEMP=1;
else TEMP=0;
return TEMP;
}


unsigned int             not (void)
{
if (TEMP) TEMP=0;
else TEMP=1;
return TEMP;
}

//---------------------------------
unsigned int            nor (unsigned int in)
{
if (TEMP==0 && in==0) TEMP=1;
else TEMP=0;
return TEMP;
}

unsigned int            nor2 (unsigned int in1, unsigned int in2)
{
if (in1==0 && in2==0) TEMP=1;
else TEMP=0;
return TEMP;
}

unsigned int            nor3        (unsigned int in1, unsigned int in2, unsigned int in3)
{
if (in1==0 && in2==0 && in3==0) TEMP=1;
else TEMP=0;
return TEMP;
}

unsigned int            nor4        (unsigned int in1, unsigned int in2, unsigned int in3, unsigned int in4)
{
if (in1==0 && in2==0 && in3==0 && in4==0) TEMP=1;
else TEMP=0;
return TEMP;
}
```

```
unsigned int          nor5       (unsigned int in1, unsigned int in2, unsigned int in3, unsigned int in4, unsigned int in5)
{
if (in1==0 && in2==0 && in3==0 && in4==0 && in5==0) TEMP=1;
else TEMP=0;
return TEMP;
}
//---------------------------------

unsigned int          nand (unsigned int in)
{
if (TEMP && in) TEMP=0;
else TEMP=1;
return TEMP;
}

unsigned int          nand2      (unsigned int in1, unsigned int in2)
{
if (in1 && in2) TEMP=0;
else TEMP=1;
return TEMP;
}

unsigned int          nand3      (unsigned int in1, unsigned int in2, unsigned int in3)
{
if (in1 && in2 && in3) TEMP=0;
else TEMP=1;
return TEMP;
}

unsigned int          nand4      (unsigned int in1, unsigned int in2, unsigned int in3, unsigned int in4)
{
if (in1 && in2 && in3 && in4) TEMP=0;
else TEMP=1;
return TEMP;
}

unsigned int          nand5      (unsigned int in1, unsigned int in2, unsigned int in3, unsigned int in4, unsigned int in5)
{
if (in1 && in2 && in3 && in4 && in5) TEMP=0;
else TEMP=1;
return TEMP;
}

unsigned int           in_out     (unsigned int in)
{
if (in) TEMP=1;
else TEMP=0;
return TEMP;
}
```

```
//----------------- latch fonksiyonları

unsigned int latch0 (unsigned int D)
{
if (TEMP) LB0=D;
return LB0;
}

unsigned int latch1 (unsigned int D)
{
if (TEMP) LB1=D;
return LB1;
}

unsigned int latch2 (unsigned int D)
{
if (TEMP)  LB2=D;
return LB2;
}

unsigned int latch3 (unsigned int D)
{
if (TEMP) LB3=D;
return LB3;
}
unsigned int latch4 (unsigned int D)
{
if (TEMP) LB4=D;
return LB4;
}

unsigned int latch5 (unsigned int D)
{
if (TEMP) LB5=D;
return LB5;
}
unsigned int latch6 (unsigned int D)
{
if (TEMP) LB6=D;
return LB6;
}

unsigned int latch7 (unsigned int D)
{
if (TEMP) LB7=D;
return LB7;
}
unsigned int latch8 (unsigned int D)
{
if (TEMP) LB8=D;
return LB8;
}

unsigned int latch9 (unsigned int D)
{
if (TEMP) LB9=D;
```

```
return LB9;
}
unsigned int latch10 (unsigned int D)
{
if (TEMP) LB10=D;
return LB10;
}

unsigned int latch11 (unsigned int D)
{
if (TEMP)
        LB11=D;
return LB11;
}
unsigned int latch12 (unsigned int D)
{
if (TEMP) LB12=D;
return LB12;
}

unsigned int latch13 (unsigned int D)
{
if (TEMP) LB13=D;
return LB13;
}

unsigned int latch14 (unsigned int D)
{
if (TEMP) LB14=D;
return LB14;
}

unsigned int latch15 (unsigned int D)
{
if (TEMP) LB15=D;
return LB15;
}

//---------- sr flip-flop fonksiyonları
unsigned int sr_ff0 (unsigned int S, unsigned int R)
{
if ((S)&& (R==0)) SR0=1;
if ((R)&& (S==0)) SR0=0;
return SR0;
}

unsigned int sr_ff1 (unsigned int S, unsigned int R)
{
if ((S)&& (R==0)) SR1=1;
if ((R)&& (S==0)) SR1=0;
return SR1;
}
```

```
unsigned int sr_ff2 (unsigned int S, unsigned int R)
{
if ((S)&& (R==0)) SR2=1;
if ((R)&& (S==0)) SR2=0;
return SR2;
}

unsigned int sr_ff3 (unsigned int S, unsigned int R)
{
if ((S)&& (R==0)) SR3=1;
if ((R)&& (S==0)) SR3=0;
return SR3;
}

unsigned int sr_ff4 (unsigned int S, unsigned int R)
{
if ((S)&& (R==0)) SR4=1;
if ((R)&& (S==0)) SR4=0;
return SR4;
}

unsigned int sr_ff5 (unsigned int S, unsigned int R)
{
if ((S)&& (R==0)) SR5=1;
if ((R)&& (S==0)) SR5=0;
return SR5;
}

unsigned int sr_ff6 (unsigned int S, unsigned int R)
{
if ((S)&& (R==0)) SR6=1;
if ((R)&& (S==0)) SR6=0;
return SR6;
}

unsigned int sr_ff7 (unsigned int S, unsigned int R)
{
if ((S)&& (R==0)) SR7=1;
if ((R)&& (S==0)) SR7=0;
return SR7;
}

unsigned int sr_ff8 (unsigned int S, unsigned int R)
{
if ((S)&& (R==0)) SR8=1;
if ((R)&& (S==0)) SR8=0;
return SR8;
}

unsigned int sr_ff9 (unsigned int S, unsigned int R)
{
if ((S)&& (R==0)) SR9=1;
if ((R)&& (S==0)) SR9=0;
return SR9;
}
```

```
unsigned int sr_ff10 (unsigned int S, unsigned int R)
{
if ((S)&& (R==0)) SR10=1;
if ((R)&& (S==0)) SR10=0;
return SR10;
}

unsigned int sr_ff11 (unsigned int S, unsigned int R)
{
if ((S)&& (R==0)) SR11=1;
if ((R)&& (S==0)) SR11=0;
return SR11;
}

unsigned int sr_ff12 (unsigned int S, unsigned int R)
{
if ((S)&& (R==0)) SR12=1;
if ((R)&& (S==0)) SR12=0;
return SR12;
}

unsigned int sr_ff13 (unsigned int S, unsigned int R)
{
if ((S)&& (R==0)) SR13=1;
if ((R)&& (S==0)) SR13=0;
return SR13;
}

unsigned int sr_ff14 (unsigned int S, unsigned int R)
{
if ((S)&& (R==0)) SR14=1;
if ((R)&& (S==0)) SR14=0;
return SR14;
}

unsigned int sr_ff15 (unsigned int S, unsigned int R)
{
if ((S)&& (R==0)) SR15=1;
if ((R)&& (S==0)) SR15=0;
return SR15;
}
```

```
//----------------- Yükselen Kenar Algılama fonksiyonları ----------

unsigned int r_edge0 (void)
{
                        if(TEMP==0)
                        {
                        RED0=1;
                        }
                        else if (RED0)
                                {
                                TEMP=1;
                                RED0=0;
                                }
                                else
                                {
                                TEMP=0;
                                }
                        return TEMP;
}

unsigned int r_edge1 (void)
{
                        if(TEMP==0)
                        {
                        RED1=1;
                        }
                        else if (RED1)
                                {
                                TEMP=1;
                                RED1=0;
                                }
                                else
                                {
                                TEMP=0;
                                }
                        return TEMP;
}

unsigned int r_edge2 (void)
{
                        if(TEMP==0)
                        {
                        RED2=1;
                        }
                        else if (RED2)
                                {
                                TEMP=1;
                                RED2=0;
                                }
                                else
                                {
                                TEMP=0;
                                }
                        return TEMP;
}
```

```
unsigned int r_edge3 (void)
{
                    if(TEMP==0)
                    {
                    RED3=1;
                    }
                    else if (RED3)
                              {
                              TEMP=1;
                              RED3=0;
                              }
                              else
                              {
                              TEMP=0;
                              }
                    return TEMP;
}

unsigned int r_edge4 (void)
{
                    if(TEMP==0)
                    {
                    RED4=1;
                    }
                    else if (RED4)
                              {
                              TEMP=1;
                              RED4=0;
                              }
                              else
                              {
                              TEMP=0;
                              }
                    return TEMP;
}

unsigned int r_edge5 (void)
{
                    if(TEMP==0)
                    {
                    RED5=1;
                    }
                    else if (RED5)
                              {
                              TEMP=1;
                              RED5=0;
                              }
                              else
                              {
                              TEMP=0;
                              }
                    return TEMP;
}
```

```
unsigned int r_edge6 (void)
{
                    if(TEMP==0)
                    {
                    RED6=1;
                    }
                    else if (RED6)
                              {
                              TEMP=1;
                              RED6=0;
                              }
                              else
                              {
                              TEMP=0;
                              }
                    return TEMP;
}

unsigned int r_edge7 (void)
{
                    if(TEMP==0)
                    {
                    RED7=1;
                    }
                    else if (RED7)
                              {
                              TEMP=1;
                              RED7=0;
                              }
                              else
                              {
                              TEMP=0;
                              }
                    return TEMP;
}

unsigned int r_edge8 (void)
{
                    if(TEMP==0)
                    {
                    RED8=1;
                    }
                    else if (RED8)
                              {
                              TEMP=1;
                              RED8=0;
                              }
                              else
                              {
                              TEMP=0;
                              }
                    return TEMP;
}
```

```
unsigned int r_edge9 (void)
{
                        if(TEMP==0)
                        {
                        RED9=1;
                        }
                        else if (RED9)
                                {
                                TEMP=1;
                                RED9=0;
                                }
                                else
                                {
                                TEMP=0;
                                }
                        return TEMP;
}

unsigned int r_edge10 (void)
{
                        if(TEMP==0)
                        {
                        RED10=1;
                        }
                        else if (RED10)
                                {
                                TEMP=1;
                                RED10=0;
                                }
                                else
                                {
                                TEMP=0;
                                }
                        return TEMP;
}

unsigned int r_edge11 (void)
{
                        if(TEMP==0)
                        {
                        RED11=1;
                        }
                        else if (RED11)
                                {
                                TEMP=1;
                                RED11=0;
                                }
                                else
                                {
                                TEMP=0;
                                }
                        return TEMP;
}
```

```
unsigned int r_edge12 (void)
{
                    if(TEMP==0)
                    {
                    RED12=1;
                    }
                    else if (RED12)
                              {
                              TEMP=1;
                              RED12=0;
                              }
                              else
                              {
                              TEMP=0;
                              }
                    return TEMP;
}

unsigned int r_edge13 (void)
{
                    if(TEMP==0)
                    {
                    RED13=1;
                    }
                    else if (RED13)
                              {
                              TEMP=1;
                              RED13=0;
                              }
                              else
                              {
                              TEMP=0;
                              }
                    return TEMP;
}

unsigned int r_edge14 (void)
{
                    if(TEMP==0)
                    {
                    RED14=1;
                    }
                    else if (RED14)
                              {
                              TEMP=1;
                              RED14=0;
                              }
                              else
                              {
                              TEMP=0;
                              }
                    return TEMP;
}
```

```
unsigned int r_edge15 (void)
{
                        if(TEMP==0)
                        {
                        RED15=1;
                        }
                        else if (RED15)
                                {
                                TEMP=1;
                                RED15=0;
                                }
                                else
                                {
                                TEMP=0;
                                }
                        return TEMP;
}

//----------------- Düşen Kenar Algılama fonksiyonları ----------------
unsigned int f_edge0 (void)
{
                        if(TEMP)
                        {
                        FED0=1;
                        TEMP=0;
                        }
                        else if (FED0)
                                {
                                TEMP=1;
                                FED0=0;
                                }
                                else
                                {
                                TEMP=0;
                                }
                        return TEMP;
}

unsigned int f_edge1 (void)
{
                        if(TEMP)
                        {
                        FED1=1;
                        TEMP=0;
                        }
                        else if (FED1)
                                {
                                TEMP=1;
                                FED1=0;
                                }
                                else
                                {
                                TEMP=0;
                                }
                        return TEMP;
}
```

```
unsigned int f_edge2 (void)
{
                    if(TEMP)
                    {
                    FED2=1;
                    TEMP=0;
                    }
                    else if (FED2)
                              {
                              TEMP=1;
                              FED2=0;
                              }
                              else
                              {
                              TEMP=0;
                              }
                    return TEMP;
}

unsigned int f_edge3 (void)
{
                    if(TEMP)
                    {
                    FED3=1;
                    TEMP=0;
                    }
                    else if (FED3)
                              {
                              TEMP=1;
                              FED3=0;
                              }
                              else
                              {
                              TEMP=0;
                              }
                    return TEMP;
}

unsigned int f_edge4 (void)
{
                    if(TEMP)
                    {
                    FED4=1;
                    TEMP=0;
                    }
                    else if (FED4)
                              {
                              TEMP=1;
                              FED4=0;
                              }
                              else
                              {
                              TEMP=0;
                              }
                    return TEMP;
}
```

```
unsigned int f_edge5 (void)
{
                    if(TEMP)
                    {
                    FED5=1;
                    TEMP=0;
                    }
                    else if (FED5)
                              {
                              TEMP=1;
                              FED5=0;
                              }
                              else
                              {
                              TEMP=0;
                              }
                    return TEMP;
}

unsigned int f_edge6 (void)
{
                    if(TEMP)
                    {
                    FED6=1;
                    TEMP=0;
                    }
                    else if (FED6)
                              {
                              TEMP=1;
                              FED6=0;
                              }
                              else
                              {
                              TEMP=0;
                              }
                    return TEMP;
}

unsigned int f_edge7 (void)
{
                    if(TEMP)
                    {
                    FED7=1;
                    TEMP=0;
                    }
                    else if (FED7)
                              {
                              TEMP=1;
                              FED7=0;
                              }
                              else
                              {
                              TEMP=0;
                              }
                    return TEMP;
}
```

```
unsigned int f_edge8 (void)
{
                    if(TEMP)
                    {
                    FED8=1;
                    TEMP=0;
                    }
                    else if (FED8)
                              {
                              TEMP=1;
                              FED8=0;
                              }
                              else
                              {
                              TEMP=0;
                              }
                    return TEMP;
}

unsigned int f_edge9 (void)
{
                    if(TEMP)
                    {
                    FED9=1;
                    TEMP=0;
                    }
                    else if (FED9)
                              {
                              TEMP=1;
                              FED9=0;
                              }
                              else
                              {
                              TEMP=0;
                              }
                    return TEMP;
}

unsigned int f_edge10 (void)
{
                    if(TEMP)
                    {
                    FED10=1;
                    TEMP=0;
                    }
                    else if (FED10)
                              {
                              TEMP=1;
                              FED10=0;
                              }
                              else
                              {
                              TEMP=0;
                              }
                    return TEMP;
}
```

```
unsigned int f_edge11 (void)
{
                    if(TEMP)
                    {
                    FED11=1;
                    TEMP=0;
                    }
                    else if (FED11)
                              {
                              TEMP=1;
                              FED11=0;
                              }
                              else
                              {
                              TEMP=0;
                              }
                    return TEMP;
}

unsigned int f_edge12 (void)
{
                    if(TEMP)
                    {
                    FED12=1;
                    TEMP=0;
                    }
                    else if (FED12)
                              {
                              TEMP=1;
                              FED12=0;
                              }
                              else
                              {
                              TEMP=0;
                              }
                    return TEMP;
}

unsigned int f_edge13 (void)
{
                    if(TEMP)
                    {
                    FED13=1;
                    TEMP=0;
                    }
                    else if (FED13)
                              {
                              TEMP=1;
                              FED13=0;
                              }
                              else
                              {
                              TEMP=0;
                              }
                    return TEMP;
}
```

```
unsigned int f_edge14 (void)
{
                    if(TEMP)
                    {
                    FED14=1;
                    TEMP=0;
                    }
                    else if (FED14)
                              {
                              TEMP=1;
                              FED14=0;
                              }
                              else
                              {
                              TEMP=0;
                              }
                    return TEMP;
}

unsigned int f_edge15 (void)
{
                    if(TEMP)
                    {
                    FED15=1;
                    TEMP=0;
                    }
                    else if (FED15)
                              {
                              TEMP=1;
                              FED15=0;
                              }
                              else
                              {
                              TEMP=0;
                              }
                    return TEMP;
}
```

```
//----------------- Karşılaştırma fonksiyonları -------------------------

unsigned int GT (unsigned int in1, unsigned int in2)
{
if (TEMP && (in1 > in2)) TEMP=1;
else TEMP=0;
return TEMP;
}

unsigned int GE (unsigned int in1, unsigned int in2)
{
if (TEMP && (in1 >= in2)) TEMP=1;
else TEMP=0;
return TEMP;
}

unsigned int EQ (unsigned int in1, unsigned int in2)
{
if (TEMP && (in1 == in2)) TEMP=1;
else TEMP=0;
return TEMP;
}

unsigned int LT (unsigned int in1, unsigned int in2)
{
if (TEMP && (in1 < in2)) TEMP=1;
else TEMP=0;
return TEMP;
}

unsigned int LE (unsigned int in1, unsigned int in2)
{
if (TEMP && (in1 <= in2)) TEMP=1;
else TEMP=0;
return TEMP;
}

unsigned int NE (unsigned int in1, unsigned int in2)
{
if (TEMP && (in1 != in2)) TEMP=1;
else TEMP=0;
return TEMP;
}

//----------------- Aritmetik fonksiyonları -------------------------
unsigned int add (unsigned int in1, unsigned int in2)
{
if (TEMP) TEMP_REG=in1+in2;
return TEMP_REG;
}

unsigned int sub (unsigned int in1, unsigned int in2)
{
if (TEMP) TEMP_REG=in1-in2;
return TEMP_REG;
}
```

```
unsigned int mul (unsigned int in1, unsigned int in2)
{
if (TEMP) TEMP_REG=in1*in2;
return TEMP_REG;
}

unsigned int  div (unsigned int in1, unsigned int in2)
{
if (TEMP) TEMP_REG=in1/in2;
return TEMP_REG;
}

unsigned int inc (unsigned int in)
{
if (TEMP) TEMP_REG= in + 1;
return TEMP_REG;
}

unsigned int dec (unsigned int in)
{
if (TEMP) TEMP_REG= in - 1;
return TEMP_REG;
}


//----------------- Lojik fonksiyonları -------------------------

unsigned int AND (unsigned int in1, unsigned int in2)
{
if (TEMP) TEMP_REG=in1&in2;
return TEMP_REG;
}

unsigned int OR (unsigned int in1, unsigned int in2)
{
if (TEMP) TEMP_REG=in1|in2;
return TEMP_REG;
}

unsigned int XOR (unsigned int in1, unsigned int in2)
{
if (TEMP) TEMP_REG=in1^in2;
return TEMP_REG;
}

unsigned int NOT (unsigned int in)
{
if (TEMP) TEMP_REG=~in;
return TEMP_REG;
}
```

```
unsigned int NAND (unsigned int in1, unsigned int in2)
{
if (TEMP)
TEMP_REG=~(in1&in2);
return TEMP_REG;
}

unsigned int NOR (unsigned int in1, unsigned int in2)
{
if (TEMP) TEMP_REG=~(in1|in2);
return TEMP_REG;
}

unsigned int XNOR (unsigned int in1, unsigned int in2)
{
if (TEMP) TEMP_REG=~(in1^in2);
return TEMP_REG;
}

unsigned int SHIFT_L (unsigned int in, unsigned int i)
{
if ((i<16)&& (TEMP)) TEMP_REG = (in << i);
return TEMP_REG;
}

unsigned int SHIFT_R (unsigned int in, unsigned int i)
{
if ((i<16)&& (TEMP)) TEMP_REG = (in >> i);
return TEMP_REG;
}


//-------------------- Zamanlayıcı fonksiyonları ----------------------
void ton (unsigned int clk, unsigned int sbt, unsigned int i)
{
 if (i<64)
            {
                        if (TEMP==0)
              {
               TON[i]=0;
                           TONQ[i]=0;
              }
                        else if (TONQ[i]==0)
            {
                        if(clk==0)          TONRED[i]=1;
                        else if(clk==1)
                                                if (TONRED[i]==1)
                                                            {
                                                            TONRED[i]=0;
                                                            TON[i]++;
                                                            if (TON[i]==sbt)
                                                                        TONQ[i]=1;
                                                            }
             }
            }
}
```

```
void tof (unsigned int clk, unsigned int sbt,unsigned int i)
{
 if (i<64)
        {
                if (TOFQ[i]==0 && (TEMP)) TOFQ[i]=1;
                if (TOFQ[i]==1 && (TEMP))
                {TOFQ[i]=1;TOF[i]=0;}
          if (TOFQ[i]==1 && TEMP==0)
        {
                if(clk==0)          TOFRED[i]=1;
                else if(clk==1)
                                    if (TOFRED[i]==1)
                                            {
                                            TOFRED[i]=0;
                                            TOF[i]++;
                                            if (TOF[i]==sbt)
                                                    TOFQ[i]=0;
                                            }
         }
        }
}

//------------------ Sayıcı fonksiyonları ----------------------------

void ctu (unsigned int cu,unsigned int reset,unsigned int pv,unsigned int i)
{
   if (i<64)
        {
        if(reset==1)
                { CQ[i]=0;
                CV[i]=0;}
   else if(cu==0) CRED[i]=1;
                  else if(CRED[i]==1 && (CV[i]< pv))
                  {       CRED[i]=0;
                          CV[i]++;
                          if (CV[i]==pv)      CQ[i]=1;
                  }
        }
}

void ctd (unsigned int cd,unsigned int load,unsigned int pv,unsigned int i)
{
   if (i<64)
        {
        if(load==1)
                { CQ[i]=0;
                CV[i]=pv;}
    else if(cd==0) CRED[i]=1;
                  else if(CRED[i]==1 && (CV[i]>0))
                  {       CRED[i]=0;
                          CV[i]--;
                          if (CV[i]==0)       CQ[i]=1;
                  }
        }
}
```

```
void ctud (unsigned int cu,unsigned int cd,unsigned int reset,unsigned int load,unsigned int pv, unsigned int i)
{
   if (i<64)
            {
                     if(reset==1)
                     { CQ[i]=0; CV[i]=0;}
                     else      if(load==1)          {CV[i]=pv;}
                               else
                                        if(cu==0 && cd==0) CRED[i]=1;
                                        else if((CRED[i]==1 && cu==1 && cd==0)&& (CV[i]< pv))
                                                 {          CRED[i]=0;
                                                            CV[i]++;
                                                 }
                                        else if((CRED[i]==1 && cu==0 && cd==1)&& (CV[i]> 0))
                                                 {          CRED[i]=0;
                                                            CV[i]--;
                                                 }
            if (CV[i]==pv)      CQ[i]=1;
            if (CV[i]< pv)      CQ[i]=0;
            }
}


#endif
```

P24FJ_PLC.h

```
#ifndef p24J_PLC_H_
#define p24J_PLC_H_
/*------------------------------------------------------------------------------
 *------------------------------------------------------------------------------*/
// definition of I/O pins
#define RCK     _RA2   // latch out
#define SRCK    _RA1   // clk out
#define SERIN   _RA0   // serial data in
#define RST     _RA14  // reset
#define SDO     _RA3   // Serial data out
#define CLKIN   _RA5   // Clk in
#define ShLd    _RA7   // Shift load


extern union R16bit
{
        unsigned int bits;
        struct {
                unsigned n0:1;
                unsigned n1:1;
                unsigned n2:1;
                unsigned n3:1;
                unsigned n4:1;
                unsigned n5:1;
                unsigned n6:1;
                unsigned n7:1;
                unsigned n8:1;
                unsigned n9:1;
                unsigned n10:1;
                unsigned n11:1;
                unsigned n12:1;
                unsigned n13:1;
                unsigned n14:1;
                unsigned n15:1;
        };
};

extern union R16bit INPUTS;
extern union R16bit OUTPUTS;
extern union R16bit TIMERS;
extern union R16bit TEMPS;
extern union R16bit MARKER0;
extern union R16bit MARKER1;
extern union R16bit MARKER2;
extern union R16bit MARKER3;
extern union R16bit REDS;
extern union R16bit FEDS;
extern union R16bit LATCHS;
extern union R16bit SRFFS;
```

```
#define Q00  OUTPUTS.n0
#define Q01  OUTPUTS.n1
#define Q02  OUTPUTS.n2
#define Q03  OUTPUTS.n3
#define Q04  OUTPUTS.n4
#define Q05  OUTPUTS.n5
#define Q06  OUTPUTS.n6
#define Q07  OUTPUTS.n7
#define Q10  OUTPUTS.n8
#define Q11  OUTPUTS.n9
#define Q12  OUTPUTS.n10
#define Q13  OUTPUTS.n11
#define Q14  OUTPUTS.n12
#define Q15  OUTPUTS.n13
#define Q16  OUTPUTS.n14
#define Q17  OUTPUTS.n15
#define OUTPUT OUTPUTS.bits

#define I00  INPUTS.n0
#define I01  INPUTS.n1
#define I02  INPUTS.n2
#define I03  INPUTS.n3
#define I04  INPUTS.n4
#define I05  INPUTS.n5
#define I06  INPUTS.n6
#define I07  INPUTS.n7
#define I10  INPUTS.n8
#define I11  INPUTS.n9
#define I12  INPUTS.n10
#define I13  INPUTS.n11
#define I14  INPUTS.n12
#define I15  INPUTS.n13
#define I16  INPUTS.n14
#define I17  INPUTS.n15
#define INPUT INPUTS.bits

#define Tm00  TIMERS.n0
#define Tm01  TIMERS.n1
#define Tm02  TIMERS.n2
#define Tm03  TIMERS.n3
#define Tm04  TIMERS.n4
#define Tm05  TIMERS.n5
#define Tm06  TIMERS.n6
#define Tm07  TIMERS.n7
#define Tm10  TIMERS.n8
#define Tm11  TIMERS.n9
#define Tm12  TIMERS.n10
#define Tm13  TIMERS.n11
#define Tm14  TIMERS.n12
#define Tm15  TIMERS.n13
#define Tm16  TIMERS.n14
#define Tm17  TIMERS.n15
#define TIMER TIMERS.bits

#define TEMP     TEMPS.n0
```

```
#define LOGIC0   0x0000
#define LOGIC1   0x0001

#define M00  MARKER0.n0
#define M01  MARKER0.n1
#define M02  MARKER0.n2
#define M03  MARKER0.n3
#define M04  MARKER0.n4
#define M05  MARKER0.n5
#define M06  MARKER0.n6
#define M07  MARKER0.n7

#define M10  MARKER0.n8
#define M11  MARKER0.n9
#define M12  MARKER0.n10
#define M13  MARKER0.n11
#define M14  MARKER0.n12
#define M15  MARKER0.n13
#define M16  MARKER0.n14
#define M17  MARKER0.n15

#define M0   MARKER0.bits

#define M20  MARKER1.n0
#define M21  MARKER1.n1
#define M22  MARKER1.n2
#define M23  MARKER1.n3
#define M24  MARKER1.n4
#define M25  MARKER1.n5
#define M26  MARKER1.n6
#define M27  MARKER1.n7

#define M30  MARKER1.n8
#define M31  MARKER1.n9
#define M32  MARKER1.n10
#define M33  MARKER1.n11
#define M34  MARKER1.n12
#define M35  MARKER1.n13
#define M36  MARKER1.n14
#define M37  MARKER1.n15

#define M1 MARKER1.bits

#define M40  MARKER2.n0
#define M41  MARKER2.n1
#define M42  MARKER2.n2
#define M43  MARKER2.n3
#define M44  MARKER2.n4
#define M45  MARKER2.n5
#define M46  MARKER2.n6
#define M47  MARKER2.n7
```

```
#define M50  MARKER2.n8
#define M51  MARKER2.n9
#define M52  MARKER2.n10
#define M53  MARKER2.n11
#define M54  MARKER2.n12
#define M55  MARKER2.n13
#define M56  MARKER2.n14
#define M57  MARKER2.n15

#define M2 MARKER2.bits

#define M60  MARKER3.n0
#define M61  MARKER3.n1
#define M62  MARKER3.n2
#define M63  MARKER3.n3
#define M64  MARKER3.n4
#define M65  MARKER3.n5
#define M66  MARKER3.n6
#define M67  MARKER3.n7

#define M70  MARKER3.n8
#define M71  MARKER3.n9
#define M72  MARKER3.n10
#define M73  MARKER3.n11
#define M74  MARKER3.n12
#define M75  MARKER3.n13
#define M76  MARKER3.n14
#define M77  MARKER3.n15

#define M3 MARKER3.bits

#define RED0  REDS.n0
#define RED1  REDS.n1
#define RED2  REDS.n2
#define RED3  REDS.n3
#define RED4  REDS.n4
#define RED5  REDS.n5
#define RED6  REDS.n6
#define RED7  REDS.n7
#define RED8  REDS.n8
#define RED9  REDS.n9
#define RED10  REDS.n10
#define RED11  REDS.n11
#define RED12  REDS.n12
#define RED13  REDS.n13
#define RED14  REDS.n14
#define RED15  REDS.n15
```

```
#define FED0  FEDS.n0
#define FED1  FEDS.n1
#define FED2  FEDS.n2
#define FED3  FEDS.n3
#define FED4  FEDS.n4
#define FED5  FEDS.n5
#define FED6  FEDS.n6
#define FED7  FEDS.n7
#define FED8  FEDS.n8
#define FED9  FEDS.n9
#define FED10  FEDS.n10
#define FED11  FEDS.n11
#define FED12  FEDS.n12
#define FED13  FEDS.n13
#define FED14  FEDS.n14
#define FED15  FEDS.n15

#define LB0          LATCHS.n0
#define LB1  LATCHS.n1
#define LB2  LATCHS.n2
#define LB3  LATCHS.n3
#define LB4  LATCHS.n4
#define LB5  LATCHS.n5
#define LB6  LATCHS.n6
#define LB7  LATCHS.n7
#define LB8  LATCHS.n8
#define LB9  LATCHS.n9
#define LB10  LATCHS.n10
#define LB11  LATCHS.n11
#define LB12  LATCHS.n12
#define LB13  LATCHS.n13
#define LB14  LATCHS.n14
#define LB15  LATCHS.n15

#define SR0          SRFFS.n0
#define SR1  SRFFS.n1
#define SR2  SRFFS.n2
#define SR3  SRFFS.n3
#define SR4  SRFFS.n4
#define SR5  SRFFS.n5
#define SR6  SRFFS.n6
#define SR7  SRFFS.n7
#define SR8  SRFFS.n8
#define SR9  SRFFS.n9
#define SR10  SRFFS.n10
#define SR11  SRFFS.n11
#define SR12  SRFFS.n12
#define SR13  SRFFS.n13
#define SR14  SRFFS.n14
#define SR15  SRFFS.n15


#endif
```