T.C.
YOZGAT BOZOK ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

STM32F103C8 MİKRODENETLEYİCİSİ TEMELLİ PLC TASARIMI

Gamze ÖZTÜRK

YÜKSEK LİSANS TEZİ

Danışman: Prof. Dr. Murat UZAM

HAZİRAN – 2024

YOZGAT

T.C.
YOZGAT BOZOK ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

STM32F103C8 MİKRODENETLEYİCİSİ TEMELLİ PLC TASARIMI

Gamze ÖZTÜRK

YÜKSEK LİSANS TEZİ

Danışman: Prof. Dr. Murat UZAM

HAZİRAN – 2024

YOZGAT

	YOZGAT BOZOK ÜNİVERSİTESİ **LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ** **LİSANSÜSTÜ TEZ ONAY FORMU**

T.C.

YOZGAT BOZOK ÜNİVERSİTESİ

LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

Enstitümüzün **Elektrik-Elektronik Mühendisliği** Anabilim Dalı Tezli Yüksek Lisans Programı öğrencisi **Gamze Öztürk**'ün hazırladığı "**STM32F103C8 Mikrodenetleyicisi Temelli PLC Tasarımı**" başlıklı tezi ile ilgili tez savunma sınavı, Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliği'nin ilgili maddeleri gereğince 27 / 06 / 2024 Perşembe günü saat 10.00'da yapılmış, tezin onayına oy birliği ile karar verilmiştir.

Başkan (Danışman) **:** Prof. Dr. Murat UZAM

Jüri Üyesi **:** Dr. Öğretim Üyesi Mustafa YAZ

Jüri Üyesi **:** Dr. Öğretim Üyesi Sıtkı AKKAYA

ONAY:

Bu tezin kabulü, Enstitü Yönetim Kurulu'nun ……../……../….. tarih ve …… sayılı Enstitü Yönetim Kurulu Kararı ile onaylanmıştır.

…...../……../.....

Prof. Dr. Ümit BUDAK

Lisansüstü Eğitim Enstitüsü Müdürü

KYT-FRM-110/00

TEZ BEYANI

Tez yazım kurallarına uygun olarak hazırlanan bu tezin yazılmasında bilimsel ahlak kurallarına uyulduğunu, başkalarının eserlerinden yararlanılması durumunda bilimsel normlara uygun olarak atıfta bulunulduğunu, tezin içerdiği yenilik ve sonuçların başka bir yerden alınmadığını, kullanılan verilerde herhangi bir tahrifat yapılmadığını, tezin herhangi bir kısmının bu üniversite veya başka bir üniversitedeki başka bir tez çalışması olarak sunulmadığını beyan eder, aksi bir durumda aleyhime doğabilecek tüm hak kayıplarını kabullendiğimi beyan ederim.

Gamze ÖZTÜRK

03 / 06 / 2024

ÖN SÖZ

Programlanabilir lojik denetleyiciler (Programmable Logic Controllers – PLC) günümüz modern endüstriyel sistemlerinde geniş bir kullanım alanına sahiptir. Fabrikalarda, enerji tesislerinde, su arıtma tesislerinde, otomasyon sistemlerinde, trafik kontrol sistemlerinde, bina otomasyonunda ve pekçok alanda kullanılırlar. PLC'ler, endüstriyel süreçlerin kontrolünü, otomasyonunu ve izlenmesini sağlayarak üretim verimliliğini arttırıp insan hatasını minimize edebilmektedir. Bu tez çalışmasında STM32F103C8 Mikrodenetleyicisi kullanılarak küçük çaplı uygulamalarda hem yeterince hızlı hem de çok uygun fiyatlı bir PLC'nin tasarımı gerçekleştirilmiştir. Bu çalışmanın ilgili alanlara faydalı olmasını dilerim.

Yüksek lisans eğitim sürecim boyunca bana bilgi, birikim ve deneyimlerini aktaran, çalışmalarımın her adımında desteğini benden esirgemeyen değerli danışman hocam Sayın Prof. Dr. Murat UZAM'a,

Çalışmalarımda motivasyonları ile beni destekleyip yanımda olan değerli meslektaşlarım Elektrik-Elektronik Mühendisi Sayın Cemal Ahmet EĞİN'e ve Makina Mühendisi Sayın Burhaneddin YILMAZ'a,

Ayrıca eğitim hayatım boyunca her koşulda yanımda olup benim bu günlere gelmemi sağlayan değerli aileme, anne ve babama sonsuz teşekkürlerimi sunarım.

Gamze ÖZTÜRK

27 / 06 / 2024

ÖZET

YÜKSEK LİSANS/ DOKTORA TEZİ

STM32F103C8 MİKRODENETLEYİCİSİ TEMELLİ PLC TASARIMI

Gamze ÖZTÜRK

YOZGAT BOZOK ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ
ANABİLİM DALI

TEZ DANIŞMANI: Prof. Dr. Murat UZAM

Bu tez çalışmasında STM32F103C8 mikrodenetleyici temelli bir PLC (programlanabilir lojik denetleyici) tasarımı gerçekleştirilmiştir. Bu PLC'de STM32F103C8 mikrodenetleyicisi merkezi işlem birimi (CPU) olarak kullanılmış olup 8 dijital giriş ve 8 dijital çıkış mevcuttur. PLC yazılımı C dili kullanılarak STM32CubeIDE yazılım geliştirme ortamında oluşturulmuştur. STM32F103C8 mikrodenetleyicisi ile gerçekleştirilen bu PLC ile giriş/çıkış sayısı fazla olmayan endüstriyel otomasyon uygulamaları için ideal bir seçenek ortaya konulmuştur. Önerilen STM32F103C8 mikrodenetleyicisi tabanlı PLC tasarımıyla düşük maliyetli ve hızlı endüstriyel otomasyon uygulamaları için cazip bir çözüm elde edilmiştir.

2024, xv + 195 Sayfa

Anahtar Kelimeler:

PLC, STM32, STM32CubeIDE, STM32F103C8 Mikrodenetleyici

ABSTRACT

MASTER THESIS

PLC DESIGN BASED ON STM32F103C8 MICROCONTROLLER

Gamze ÖZTÜRK

YOZGAT BOZOK UNIVERSITY
SCHOOL OF GRADUATE STUDIES
ELECTRICAL ELECTRONICS ENGINEERING
DEPARTMENT

SUPERVISOR: Prof. Dr. Murat UZAM

In this thesis, a PLC (programmable logic controller) design based on STM32F103C8 microcontroller was carried out. In this PLC, STM32F103C8 microcontroller is used as the central processing unit (CPU) and it has 8 digital inputs and 8 digital outputs. PLC software was created in the STM32CubeIDE software development environment using C language. This PLC, implemented with the STM32F103C8 microcontroller, is an ideal option for industrial automation applications that do not have many inputs/outputs. An attractive solution for low-cost and fast industrial automation applications has been achieved with the proposed STM32F103C8 microcontroller-based PLC design.

2024, xv + 195 Pages

Keywords:

PLC, STM32, STM32CubeIDE, STM32F103C8 Microcontroller

İÇİNDEKİLER

Sayfa

TABLOLAR LİSTESİ

Tablo **Sayfa**

ŞEKİLLER LİSTESİ

Şekil **Sayfa**

SİMGELER VE KISALTMALAR LİSTESİ

Bu yüksek lisans tez çalışmasında kullanılmış olan simgeler ve kısaltmalar açıklamaları ile birlikte aşağıda sunulmuştur.

Simgeler	**Açıklamalar**
+	**:** toplama
-	**:** çıkartma
*	: çarpma
/	: bölme
%	: artık bölme
=	: atama
+=	: ekleyerek atama
-=	: eksilterek atama
/=	: bölerek atama
%=	: bölüp kalanını atama
++	: bir arttırma
--	: bir azaltma
&&	**:** mantıksal VE
‖	: mantıksal VEYA
>	: büyüktür
<	: küçüktür
==	: eşittir
>=	: büyük veya eşittir
<=	: küçük veya eşittir
!=	: eşit değil
\n	: yeni satır başına geçer

\r : satır başına geçer

%d : onluk (decimal) olarak yazar

Kısaltmalar	**Açıklamalar**
PLC	: Programlanabilir Lojik Denetleyici (Programmable Logic Controller)
CTU	: İleri Sayıcı (Up Counter)
CTD	: Geri Sayıcı (Down Counter)
CTUD	: İleri/Geri Sayıcı (Up/Down Counter)
TON	: Düz Zaman Rölesi (On Delay Timer)
TOF	: Ters Zaman Rölesi (Off Delay Timer)
ADD	: Toplama (Addition)
SUB	: Çıkartma (Subtraction)
MUL	: Çarpma (Multiplication)
DIV	: Bölme (Division)
INC	: Bir arttırma (Increment)
DEC	: Bir azaltma (Decrement)

1. GİRİŞ

Programlanabilir lojik denetleyiciler (Programmable Logic Controllers – PLC) mikroişlemci veya mikrodenetleyici temelli olarak üretilen otomasyon cihazlarıdır. PLC'ler 1970'lerin başından itibaren endüstride çok yaygın olarak kullanılmaktadır. PLC üreticileri değişik fonksiyonlara sahip, farklı hafıza kapasiteleri olan, birkaç giriş-çıkıştan binlerce giriş-çıkışa kadar değişik sayıda giriş-çıkışı bulunan, farklı büyüklüklerde pek çok PLC üretip kullanıma sunmaktadır. Otomasyon sistemlerinin en önemli parçası olan PLC'ler çoğunlukla yabancı üreticilerin ürünleri olmakla birlikte yerli üreticilerin piyasaya sunduğu PLC'ler de mevcuttur. PLC üreticileri doğal olarak üretmiş oldukları donanımı ve yazılımı koruma altına alarak özellikle yazılımın rakip firmaların eline geçmesine müsaade etmemektedirler. Dışa bağımlılığı azaltmak için Türkiye'deki yerli ve milli PLC üreticileri de alternatif PLC'leri üreterek kullanıcılara sunmaktadırlar. Bu kapsamda PLC teknolojisinin geliştirilmesi için yapılacak olan çalışmalar çok büyük önem arz etmektedir.

Yabancı PLC üretici firmalara örnek olarak şu firmalar gösterilebilir:

1. Siemens PLC (Siemens, t.y.)
2. Delta PLC (Deltaww, t.y.)
3. Omron (Omron, t.y)

Yerli ve milli PLC üretici firmalara örnek olarak şu firmalar gösterilebilir:

1. Mikrodev (Mikrodev, t.y.)
2. GMTCNT (Gmtcontrol, t.y.)

PLC tasarımı konusunda yapılan akademik çalışmalara bakıldığında, PIC16F84 mikrodenetleyicisi temelli 8 dijital girişli/8 dijital çıkışlı ve 5 MHz frekansında çalışan bir PLC (Kitiş, 2007)'da görülmektedir. Gerçekleştirilen PLC'nin yazılımında PLC'lerde kullanılmakta olan 'Boolean dili programlama' ya da komut listesi (statement list) olarak bilinen yöntemdekine benzer bir programlama ortamı oluşturulmuştur. Bunun için PIC assembly dili kullanılarak PLC komutları için makrolar yazılmıştır. Bu makroların oluşturulmasında 'PICBIT' programından faydalanılmıştır. Gerçekleştirilen PLC yardımıyla küçük ölçekli otomasyon problemlerinin çözümü için düşük maliyetli bir ürün ortaya koyulmuştur. Bu PLC tasarımda PIC mikrodenetleyicileri kullanılarak program hafızası ile giriş/çıkış sayılarının arttırılması ve daha büyük sistemlerin kumandası

mümkün olabilir. PIC16F877 Mikrodenetleyicisi temelli 7 dijital girişli/6 dijital çıkışlı bir PLC (Erkol, 2008)'de tasarlanmıştır. Merdiven (ladder) diyagramlarıyla yazılım geliştirebilecek gelişime açık bir bilgisayar yazılımı ve bu yazılımla gerçekleştirilen programları çalıştırabilecek üzerinde analog giriş-çıkışları, LCD paneli, tuş takımı, dijital girişleri ve röleli çıkışları olan bir PLC gerçekleştirilmiştir. Bu PLC'nin hem donanımı hem de yazılımı geliştirmeye açıktır. Atmega128 mikrodenetleyicisi tabanlı bir PLC tasarımı (Tongur, 2008)'de mevcuttur. Tasarlanan bu PLC için kontrol devrelerinin simülasyonunu yapabilen, hex kodlar üretebilen bir arayüz yazılımı (EVRENPLC) geliştirilmiştir. Hex kodlarının çalışmasını sağlayan bir yorumlayıcı geliştirilerek mikrodenetleyicinin sabit belleğine yerleştirilmiştir. PIC16F877 Mikrodenetleyicisi temelli başka bir 6 dijital girişli/5 dijital çıkışlı bir PLC (Rafat, 2010)'te tasarlanmıştır. Yapılan bu çalışmada PIC16F877 mikrodenetleyicisi kullanarak bir PLC tasarlanmıştır. Hex kodlar üretebilen bir arayüz yazılımı geliştirilmiştir. Gerçekleştirilen arayüz yazılımında merdiven (ladder diagram) diyagramı ve komut listesine (statement list) benzer bir programlama ortamı oluşturulmuştur. 16 bitlik bir PIC mikrodenetleyicisi ile 16 dijital girişli/16 dijital çıkışlı bir PLC tasarımı (Harmanda, 2011)'da mevcuttur. 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. Bu çalışmayla donanım ve yazılım özellikleri olarak orta büyüklükteki endüstriyel çözümler için uygun bir endüstriyel kontrol aracı elde edilmiştir. Gerçekleştirilen PLC yazılımı PIC C dili ile MPLAB IDE yazılım geliştirme ortamı yardımıyla geliştirilmiştir. (Kılıçarslan, 2014)'de mikrodenetleyici temelli PLC'ler için programlama yazılımının gerçekleştirilmesi söz konusudur. Uygulama için PIC16F877A temelli PLC giriş ve çıkışları kullanılarak dağıtım istasyonu deney setine bağlanmıştır. Tasarlanan merdiven diyagramı kodu arayüz yardımıyla PIC koduna dönüştürülerek PIC-PLC'ye yüklenmiştir. (Uzam, 2013)'de PIC16F877A mikrodenetleyici temelli 16 dijital girişli/16 dijital çıkışlı bir PLC tasarımı detaylı olarak açıklanmıştır. Bu PLC'nin çalışma frekansı 20 MHz'dir. (Uzam, 2013)'deki PLC donanım ve yazılım olarak geliştirilerek (Uzam, 2022)'te sunulmuştur. PIC PLC'ye direk olarak program yükleme işlemi yapmak için MPLAB X IDE tarafından desteklenen bir programlayıcıyla gerçekleştirilmiştir. PIC16F877A mikrodenetleyicisi 20 MHz'te

çalıştırılmaktadır. Bu çalışma frekansı, zaman gecikmesi fonksiyonlarındaki hesaplamalarda kullanmıştır.

Gerçekleştirilen bu yüksek lisans tez çalışmasında yerli ve milli PLC geliştirilmesi çalışmalarına katkı yapılması hedeflenmiştir. STM32F103C8 mikrodenetleyicisi kullanarak özellikle küçük çaplı uygulamalar için hem yeterince hızlı hem de düşük maliyetli bir PLC tasarlanmıştır. Bu PLC'nin merkezi işlem birimi (CPU) olarak STMicroelectronics firması tarafından üretilen 72 MHz clock frekansında çalışan STM32F103C8 (32-bit ARM Cortex-M3) mikrodenetleyicisi kullanılmıştır. PLC'nin 8 dijital girişi ve 8 dijital çıkışı bulunmaktadır. Bu sayede özellikle zaman-kritik sistemler için kullanılması söz konusu olabilecek bir PLC altyapısı oluşturulması mümkün olmuştur. Bu tasarımın geliştirilmesiyle de hem çok hızlı hem de geniş işlem gücü ve çevresel uyumluluğu sayesinde düşük maliyetli olarak çok uygun bir PLC elde edilmiştir. Tasarlanan PLC'nin temel işlevlerini gerçekleştirmek için gerekli olan giriş/çıkış bağlantıları, STM32F103C8 mikrodenetleyicisi üzerindeki GPIO pinleri kullanılarak sağlanır. Bu, harici bileşenlere olan ihtiyacı azaltır ve tasarımı daha ekonomik hale getirir. Ayrıca, STM32F103 mikrodenetleyicilerinin geniş bir programlama diline sahip olması PLC yazılımını hızlı ve pratik bir şekilde geliştirmeyi mümkün kılar. Tasarlanan PLC STM32CubeIDE bütünleşik geliştirme ortamı kullanılarak kolayca kodlanabilmektedir. Bu da projenin daha da erişilebilir olmasını sağlamıştır.

Bu tez çalışması on bir bölümden oluşmaktadır. İkinci bölümünde STM32F103XX mikrodenetleyicileri ve kullanılan mikrodenetleyiciye ait temel bilgilerden bahsedilmiştir. Üçüncü bölümünde tasarlanan PLC donanımı detaylı olarak incelenmiştir. Gerçekleştirilen PLC'ye ait temel yazılım bileşenleri dördüncü bölümde anlatılmaktadır. Beşinci bölümde PLC yazılım kısmında yer alan kontak ve röle temelli fonksiyonlar detaylı olarak sunulmuştur. 6., 7., 8. ve 9. bölümlerde sırasıyla sayıcı fonksiyonları, zamanlayıcı fonksiyonları, karşılaştırma fonksiyonları ve aritmetik fonksiyonları örneklerle birlikte incelenmektedir. Lojik ve kaydırma fonksiyonları 10. bölümde yer almaktadır. On birinci bölümde ise elde edilen sonuçlar vurgulanmıştır.

2. STM32F103XX MİKRODENETLEYİCİLERİ

Bu bölümde STM32F103XX mikrodenetleyicilerinden biri olan ve bu tez konusu için seçilen STM32F103C8 mikrodenetleyicinin yapısı ve bu tez çalışması kapsamında kullanılan bazı özellikleri incelenmektedir. STM32F103XX mikrodenetleyicileri orta yoğunluklu performans hattı ailesi olarak bilinir. 72 MHz frekansında çalışan yüksek performanslı Arm® Cortex®-M3 32-bit RISC çekirdeği, yüksek hızlı gömülü bellekleri içerir. STM32F103C8 mikrodenetleyicileri;

- Motor sürücülerinde
- Uygulama kontrollerinde
- Tıbbi el ekipmanlarında
- PC ve oyun çevre birimlerinde
- GPS platformları
- Endüstriyel uygulamalar
- PLC'ler
- İnvertörler
- Yazıcılar ve tarayıcılar
- Alarm sistemleri gibi birçok alanda kullanılmaktadır

Tablo 2.1'de STM32F103XX mikrodenetleyici çeşitleri görülmektedir (RM0008, 2021).

Tablo 2.1. STM32F103XX mikrodenetleyici çeşitleri

Reference	Part number
STM32F103x8	STM32F103C8, STM32F103R8 STM32F103V8, STM32F103T8
STM32F103xB	STM32F103RB STM32F103VB, STM32F103CB, STM32F103TB

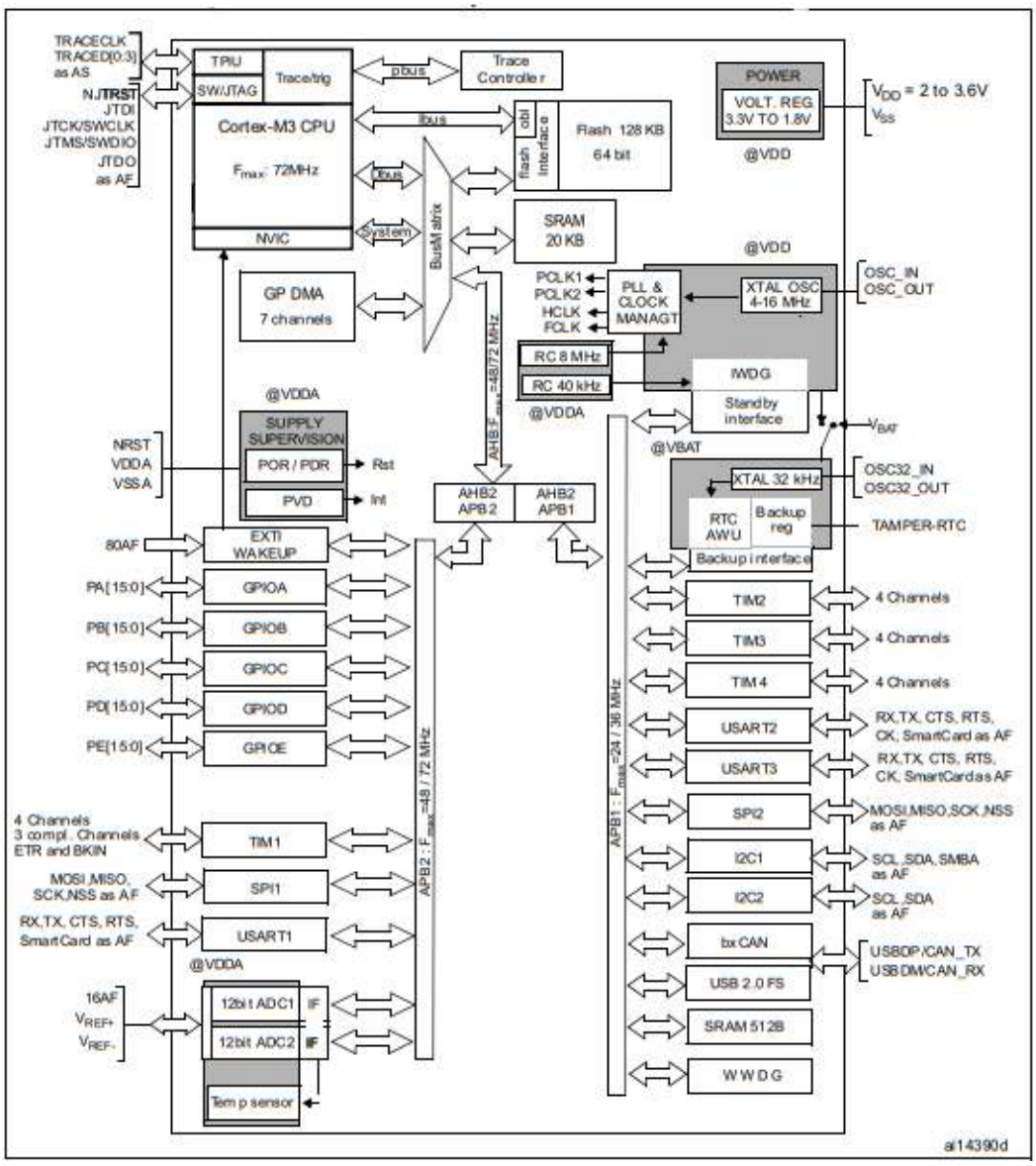
Tablo 2.1'deki tüm mikrodenetleyiciler iki adet 12-bit ADC, üç genel amaçlı 16-bit zamanlayıcı ve bir PWM zamanlayıcının yanı sıra standart ve gelişmiş iletişim arabirimleri sunar: iki adede kadar I2C ve SPI, üç USART, bir USB ve bir CAN. Bu

mikrodenetleyiciler 2.0 ila 3.6V DC güç kaynağı ile çalışır. Hem –40 ila +85 °C sıcaklık aralığında hem de – 40 ila +105 °C genişletilmiş sıcaklık aralığında çalışan türleri mevcuttur. Kapsamlı bir güç tasarrufu modu seti, düşük güçlü uygulamalara imkân sunar (RM0008, 2021).

Tablo 2.2'de STM32F103XX orta yoğunluklu cihaz özellikleri ve çevre birimi sayıları ifade edilmiştir (RM0008, 2021). Burada STM32F103C8 Mikrodenetleyicisi 72 MHz'lik işlemci frekansına, 2 tane olmak üzere I2C ve SPI portlarına sahiptir. İşlemci mimarisinde ARM Cortex M3 32 bit ailesindendir. 64 KB flaş belleğine ve 20Kb RAM'a sahiptir. Aynı zamanda 3 tane USART girişine sahiptir.

Tablo 2.2. STM32F103XX cihaz özellikleri ve çevre birimi sayıları

Peripheral		STM32F103Tx		STM32F103Cx		STM32F103Rx		STM32F103Vx	
Flash - Kbytes		64	128	64	128	64	128	64	128
SRAM - Kbytes		20		20		20		20	
Timers	General-purpose	3		3		3		3	
	Advanced-control	1		1		1		1	
Communication	SPI	1		2		2		2	
	I^2C	1		2		2		2	
	USART	2		3		3		3	
	USB	1		1		1		1	
	CAN	1		1		1		1	
GPIOs		26		37		51		80	
12-bit synchronized ADC Number of channels		2 10 channels		2 10 channels		2 16 channels(1)		2 16 channels	
CPU frequency		72 MHz							
Operating voltage		2.0 to 3.6 V							
Operating temperatures		Ambient temperatures: -40 to +85 °C / -40 to +105 °C Junction temperature: -40 to + 125 °C							



Şekil 2.1. STM32F103XX performans hattı blok diyagramı (RM0008, 2021)

Şekil 2.1'de görülen STM32F103XX mikrodenetleyicinin performans hattı blok diyagramına göre F0, F1, F2, F3, F4, F7 gibi serileri bulunmaktadır.

2.1. STM32F103C8 Mikrodenetleyicisindeki Zamanlayıcılar

Bu kısımda STM32F103C8 mikrodenetleyicisinde bulunan zamanlayıcılar incelenmektedir. Zamanlayıcılar, mikrodenetleyicilerde veya diğer programlanabilir cihazlarda donanımsal olarak uygulanır ve birçok programlama dilinde zamanlama

işlevlerini gerçekleştirmek için kullanılır. Tablo 2.3'ten görüleceği üzere STM32F103C8 mikrodenetleyicisinde temel (TIM6 ve TIM7), genel amaçlı (16 bitlik TIM2, TIM3, TIM4 ve TIM5) ve gelişmiş (TIM1 ve TIM8) olmak üzere üç tip zamanlayıcı (timer) bulunmaktadır (RM0008, 2021). Gelişmiş kontrol zamanlayıcıları (TIM1 ve TIM8), 16 bitlik programlanabilir otomatik ön ölçekleyici tarafından yeniden yükleme sayacıyla çalıştırılan bir sayaçtan oluşur. Girişin darbe uzunluklarını ölçmek de dahil olmak üzere çeşitli amaçlar için kullanılabilir. Darbe uzunlukları ve dalga biçimi periyotları birkaç mikrosaniyeden birkaç milisaniyeye kadar modüle edilebilir. Bu zamanlayıcılarda ön ölçekleyici ve RCC saat denetleyicisi ön ölçekleyicileri kullanılır. Gelişmiş kontrol (TIM1 ve TIM8) ve genel amaçlı (TIMx) zamanlayıcılar tamamen bağımsızdır. Birlikte senkronize edilebilirler (RM0008, 2021).

STM32F103C8 veya benzeri STM32 mikrodenetleyicilerindeki donanım zamanlayıcıları (timerlar), genellikle çeşitli kontrol işlemlerinde zamanlama işlemlerini gerçekleştirmek, darbe genişliği modülasyonu (PWM) oluşturmak ve zamanlama hassasiyeti gerektiren diğer birçok görevi yerine getirmek için kullanılır. Bu mikrodenetleyicilerdeki donanım zamanlayıcıları, genellikle bir dizi kaydedici (register) kullanılarak yapılandırılır ve kontrol edilir. Temel olarak, bir donanım zamanlayıcısı şu bileşenlere sahip olur:

1. Sayıcılar (Counters): Genellikle 16-bit veya 32-bit olarak yapılandırılabilen sayıcılar belirli bir değeri sürekli olarak artırabilir veya azaltabilir. Bu sayıcılar genellikle zamanlayıcıların temelini oluşturur.
2. Ön ölçekleyiciler (Prescaler): Sayıcıların kaydedebileceği değerleri azaltmak veya artırmak için kullanılır. Bu, zaman dilimlerini ölçmek veya dönemleri ayarlamak için kullanılır.
3. Kesme (Interrupt) mekanizması: Zamanlayıcı belirli bir değeri aştığında, başka bir olay gerçekleştiğinde veya belirli bir koşulu sağladığında kesme (interrupt) oluşturulur. Bu, zamanlayıcı olayları tarafından tetiklenen belirli bir kod parçasını çalıştırmak için kullanılabilir.
4. Zamanlama modları: Zamanlayıcılar genellikle farklı zamanlama modlarını destekler. Örneğin, tek yönlü sayma, çift yönlü sayma, PWM üretimi gibi modlar bu zamanlayıcılarda kullanılabilir.

5. Çıkışlar ve PWM Üretimi: Zamanlayıcılar genellikle çıkışları kontrol etmek veya PWM sinyalleri oluşturmak için kullanılabilir. Böylelikle, motor kontrolü, LED parlaklığı kontrolü gibi birçok uygulama gerçekleştirilebilir.

Zamanlayıcıların çalışması, yapılandırma ve kontrol kaydedicilerinin doğru bir şekilde ayarlanmasına dayanır. Örneğin, belirli bir zaman dilimini ölçmek için bir zamanlayıcı yapılandırılır, sayıcı başlatılır ve belirli bir koşul gerçekleştiğinde bu sayıcı durdurulur veya belirli bir değere ulaştığında bir kesme oluşturulur.

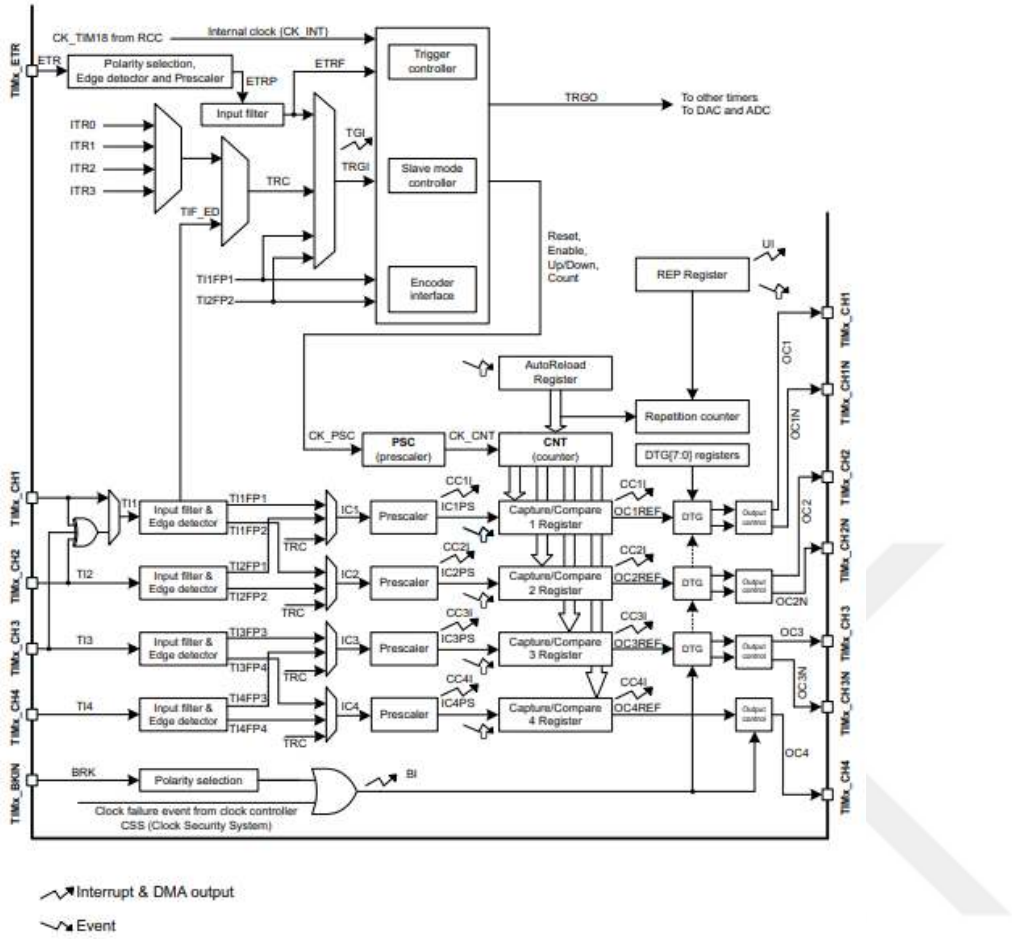
Tablo 2.3. STM32FXXX timer çeşitleri

Timer type		STM32 F04x /F070x6 /F03x (exclu-ding /F030x8 and /F030x)	STM32 F030xB /F030x8 /F05x /F09x /F07x (excluding F070x6)	STM32 F101 /F102 /F103 lines XL density (xF, xG)	STM32 F101 /F102 /F103 /F105 /F107 lines up to high-density (x4-xE)	STM32 F100 value line	STM32 F2 /F4 (exclu-ding /F401, /F411, /F410)	STM32 F401 /F411 /F410	STM32 F30X /F3x8 (exclu-ding /F378)	STM32 F37x	STM32 F334	STM32 F31x	STM32 F7 Series
Advanced		TIM1	TIM1	TIM1(1) TIM8(1)	TIM1(1) TIM8(1)	TIM1	TIM1 TIM8	TIM1	TIM1 TIM8(1) TIM20(1)	-	TIM1	TIM1 TIM8(1)	TIM1 TIM8
General purpose	32-bit	TIM2	TIM2	-	-	-	TIM2 TIM5	TIM2(1) TIM5	TIM2	TIM2 TIM5	TIM2	TIM2	TIM2 TIM5
	16-bit	TIM3	TIM3	TIM2 TIM3 TIM4 TIM5	TIM2 TIM3 TIM4(1) TIM5(1)	TIM2 TIM3 TIM4 TIM5(1)	TIM3 TIM4	TIM3(1) TIM4(1)	TIM3(1) TIM4(1) TIM19(1)	TIM3 TIM4 TIM19	TIM3	TIM3 TIM4	TIM3 TIM4
Basic		-	TIM6 TIM7(1)	TIM6 TIM7	TIM6(1) TIM7(1)	TIM6 TIM7	TIM6 TIM7	TIM6(1)	TIM6 TIM7(1)	TIM6 TIM7 TIM18	TIM6 TIM7	TIM6 TIM7(1)	TIM6 TIM7
1 channel		TIM14	TIM14	TIM10 TIM11 TIM13 TIM14	-	TIM13(1) TIM14(1)	TIM10 TIM11 TIM13 TIM14	TIM10(1) TIM11	-	TIM13 TIM14	-	-	TIM10 TIM11 TIM13 TIM14
2-channel		-	-	TIM9 TIM12	-	TIM12(1)	TIM9 TIM12	TIM9	-	TIM12	-	-	TIM9 TIM12
2-channel with complementary output		-	TIM15	-	-	TIM15	-	-	TIM15	TIM15	TIM15	TIM15	-
1-channel with complementary output		TIM16 TIM17	TIM16 TIM17	-	-	TIM16 TIM17	-	-	TIM16 TIM17	TIM16 TIM17	TIM16 TIM17	TIM16 TIM17	-
Low-power timer		-	-	-	-	-	-	LPTIM1(1)	-	-	-	-	LPTIM1
High-resolution timer		-	-	-	-	-	-	-	-	-	HRTIM	-	-

2.1.1. TIM1 ve TIM8 Zamanlayıcılarının Özellikleri

- 16 bitlik yukarı, aşağı, yukarı/aşağı otomatik yeniden yüklemeli sayaç içerir.
- Sayaç, saatinin bölünmesine (aynı zamanda "anında") izin veren 16 bitlik programlanabilir ön ölçekleyicisine sahiptir. Frekansı 1 ila 65535 arasındaki herhangi bir faktörle değiştirilebilir.
- 4'e kadar bağımsız kanal içerir: Giriş yakalama, çıkış karşılaştırması, PWM üretimi (Kenar ve Merkeze Hizalanmış Mod), tek darbeli mod çıkışı.
- Programlanabilir ölü zamanlı tamamlayıcı çıkışlar mevcuttur.
- Zamanlayıcıyı harici sinyallerle kontrol etmek ve ara bağlantı kurmak için senkronizasyon devresi ile birkaç zamanlayıcı bir arada bulunur.
- Zamanlayıcı kaydedicilerini yalnızca belirli sayıda döngüden sonra güncellemek için tekrarlama sayacıdır.
- Zamanlayıcının çıkış sinyallerini sıfırlama durumuna veya bilinen bir duruma getirmek için girişi kesmek gerekir.
- Olaylarda kesinti/DMA üretimi:
  - ✓ Güncelleme: sayaç taşması/eksikliği, sayaç başlatma (yazılım veya dahili/harici tetikleyici),
  - ✓ Tetikleme olayı (dahili/harici tetikleyiciyle sayaç başlatma, durdurma, başlatma veya sayma),
  - ✓ Giriş yakalama,
  - ✓ Çıkış karşılaştırması,
  - ✓ Ara girişi.
- Konumlandırma için artımlı (dörtlü) kodlayıcı ve sensör devresini destekler.
- Harici saat veya döngüden döngüye akım yönetimi için tetikleme girişi mevcuttur.

(RM0008, 2021)



Şekil 2.2. Gelişmiş kontrol zamanlayıcı blok şeması (RM0008, 2021)

Şekil 2.2'de gelişmiş kontrol zamanlayıcısının blok şeması mevcuttur. Burada; çekirdek Counter (CNT) yer almaktadır.

TIMX_ERT 72 MHz'e bağlanıyor. Bu işlemi osilatör ile de gerçekleştirebilir veya dışarıdan bir clock uygulanarak sayma işlemi gerçekleştirilebilir. PSC Prescaler (TIMxBKN, TIMXCH4, TIMxCH3, TIMxCH2, TIMxCH1) = giriş seçimi yapılır ve dahili (internal) clock'a bağlanır. Dahili clock PSC'ye girer ve ayarlanan frekansa bölünür. (Örneğin: yapılan çalışmada frekans 72 MHz'e ayarlanmıştır. PSC Prescalar 10 yazılırsa sonuç 7.2 MHz olarak sonuçlanır.)

AutoReload Register: Sayıcıların kaça kadar sayacağının belirlendiği kısımdır. (Örneğin:100'e kadar sayıp 100'e gelince çıkışta bir kesme (interrupt) oluşturmaktadır.) Sayma işlemini yaptıktan sonra çalışmayı işlemciye yani CC1'e gönderip işlemcinin uyarılması sağlanır.

2.1.2. Zaman Tabanı Oluşturucu

Zamanlayıcı, zaman tabanı oluşturucusu olarak kullanılabilir. Saat, ön ölçekleyici ve otomatik yeniden yükleme, tekrarlama sayacı (varsa) parametrelerine bağlı olarak 16 bitlik zamanlayıcı bir nanosaniyeden birkaç dakikaya kadar bir güncelleme olayı oluşturabilir. 32 bit zamanlayıcılar daha geniş bir aralık sağlar (AN4013, 2024) Güncelleme olayı süresi aşağıdaki gibi hesaplanır:

Güncelleme_olay = TIM_CLK/((PSC + 1)*(ARR + 1) *(RCR + 1))

TIM_CLK = zamanlayıcı saati girişi

PSC = 16 bitlik ön ölçekleyici kaydedicisi

ARR = 16/32-bit Otomatik yeniden yükleme kaydedicisi

RCR = 16 bit tekrarlama sayacı

ÖRNEK:

TIM_CLK = 72 MHz

Ön ölçekleyici = 1

Otomatik yeniden yükleme = 65535

Tekrarlama sayacı yok RCR = 0

Update_event = 72*(10^6)/((1 + 1)*(65535 + 1)*(1))

Güncelleme_olay = 549,3 Hz

ÖRNEK:

TIM_CLK = 72 MHz

Ön ölçekleyici = 1

Otomatik yeniden yükleme = 35999

Tekrarlama sayacı yok RCR = 0

Update_event = 72.000.000/ (35999 + 1)* (1 + 1)* (1))

Güncelleme_olay = 10.000 Hz=1ms

2.1.3. Kilitleme Mekanizması

Gelişmiş zamanlayıcı kaydedicileri ve bitleri, BDTR kaydedicisindeki LOCK bitlerini programlayarak kilitleme mekanizmasını kullanarak uygulamayı korumak amacıyla kullanılabilir veya kilitlenebilir. Tablo 2.4'te belirtildiği üzere üç kilitleme seviyesi vardır (AN4013, 2024).

Tablo 2.4. Kilitleme düzeyleri (AN4013, 2024).

Level 1		LOCK Level 2(1)		LOCK Level 3(2)	
Register	**Bits**	**Register**	**Bits**	**Register**	**Bits**
CR2	OISx	**CR2**	OISx	**CR2**	OISx
	OISxN		OISxN		OISxN
BDTR	DTG[7:0]	**BDTR**	DTG[7:0]	**BDTR**	DTG[7:0]
	BKE		BKE		BKE
	BKP		BKP		BKP
	AOE		AOE		AOE
	BK2E(3)		OSSR		OSSR
	BK2P(3)		OSSI		OSSI
	BKF[3:0](3)	**CCER**	CCxP	**CCER**	CCxP
	BK2F[3:0](3)		CCxNP		CCxNP
-	-	-	-	**CCMRx**	OCxM
-	-	-	-		OCxPE

Bu üç kilitleme seviyesi;

1. LOCK Seviye 2 = LOCK Seviye 1 + CC polarite bitleri (TIMx_CCER'deki CCxP/CCxNP bitleri).
2. LOCK Seviye 3 = LOCK Seviye 2 + CC kontrol bitleri (OCxM ve OCxPE.
3. STM32L4/F7/L5/G0/G4/WB/H7 Serisi ve STM32F30x/F3x8 hatlarında bulunan bitlerdir.

LOCK bitlerine sıfırlamadan sonra yalnızca bir kez yazılabilir. BDTR kaydedicisine yazıldıktan sonra içeriği bir sonraki sıfırlamaya kadar dondurulur (AN4013, 2024)

2.1.4. Saat Seçimi

Dahili Saat (CK_INT): Mikrodenetleyicinin dahili saat sinyalidir. Dahili olarak oluşturulur ve zamanlayıcının saati için kaynak olarak kullanılabilir.

Harici Saat Modu1-Harici Giriş Pini: Bu mod, zamanlayıcının saatini çalıştırmak için mikrodenetleyicinin belirli bir pinine bağlı harici bir sinyal kullanılmasını sağlar. Zamanlayıcı, harici sinyalin frekansına veya darbelerine göre sayacaktır.

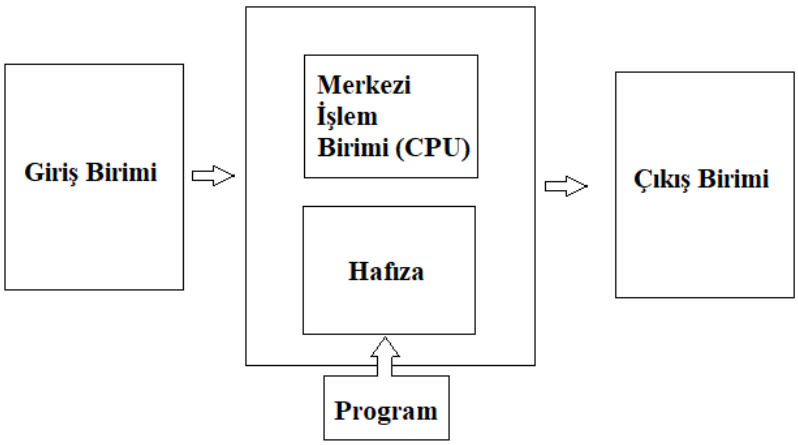
Harici Saat Modu2-Harici Tetikleme Girişi (ETR): Bu modda, zamanlayıcı, zamanlayıcının saatini kontrol etmek için harici bir tetikleme giriş pini (ETR) kullanabilir. Zamanlayıcı, harici tetikleyici olaylara göre başlar, durur veya sayar.

Bu saat kaynaklarının her biri, zamanlayıcıların farklı olaylara veya harici sinyallere göre nasıl çalıştırıldığı ve kontrol edildiği konusunda esneklik sunar. Saat kaynağının seçimi, özel uygulama gereksinimlerine ve zamanlayıcının istenen davranışına bağlıdır (RM0008, 2021).

STM32 mikrodenetleyicileri, donanım soyutlama katmanları (HAL) ve STM32CubeIDE gibi araçlar aracılığıyla zamanlayıcılar için bu saat kaynaklarını ayarlamak ve kontrol etmek için yapılandırma seçenekleri sağlar ve geliştiricilerin zamanlayıcının davranışını uygulama ihtiyaçlarına göre uyarlamasına olanak tanır.

3. TASARLANAN PLC'NİN DONANIMI

Bu bölümde bu yüksek lisans tez çalışması kapsamında STM32F103C8 mikrodenetleyicisi kullanılarak tasarlanan PLC donanımı açıklanmaktadır. PLC'ler besleme gerilimine ek olarak genelde üç temel donanım bileşeninden oluşur: Merkezi işlem birimi (CPU), (dijital/analog) giriş birimi ve (dijital/analog) çıkış birimi. Merkezi işlem biriminde giriş sinyallerine göre program yürütülür ve çıkış sinyalleri kontrol edilerek programın çalıştırılması ve işlenmesi sağlanır. Giriş birimi dış dünyadan gelen giriş sinyalleri CPU'ya ileterek tüm giriş sinyallerinin bağlandığı birimdir. Çıkış biriminde CPU'dan gelen çıkış sinyalleri çıkış elemanlarına yönlendirilir (Kontrol kalemi, t.y.). PLC'lerin genel yapısı Şekil 3.1'de görülmektedir. Giriş biriminden gelen giriş sinyallerini merkezi işlem birimindeki giriş arabiriminde işlenir ve PLC'nin RAM hafızasındaki giriş kaydedicileri güncellenir. Merkezi işlem birimi hafızaya yüklenmiş olan kullanıcı programını satır satır çalıştırır ve giriş sinyallerinin durumuna göre çıkış kaydedicilerini günceller. Bu işlem tamamlandığında çıkış kaydedicilerinin durumu çıkış birimine yüklenir. Bir PLC çevrimi olarak isimlendirilen bu işlemler sürekli olarak tekrarlanmaktadır (Kontrol kalemi, t.y.).



Şekil 3.1. PLC'lerin genel yapısı

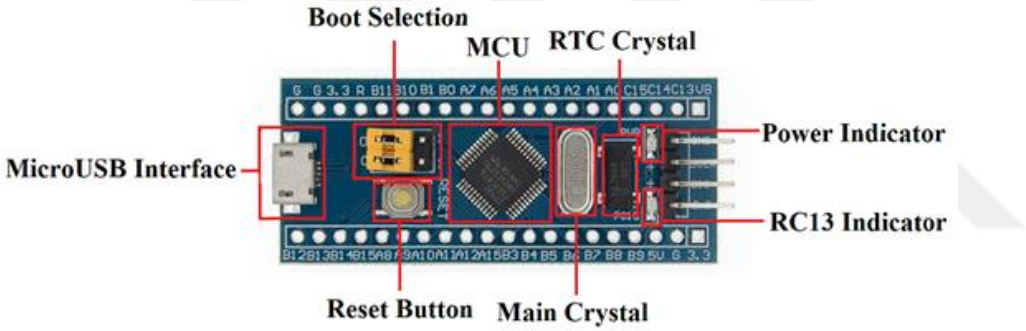
STM32F103C8 mikrodenetleyici kullanılarak tasarlanan PLC donanımı altı kısımdan oluşmaktadır:

3.1. Güç Kısmı

Güç kısmında iki tane DC gerilim girişi mevcuttur: bir tanesi 3.3V adaptör ve diğeri röle çıkışları için 12V 1 Amperlik başka bir adaptör kullanılarak gerçekleştirilmiştir.

3.2. Merkezi İşlem Birimi (CPU)

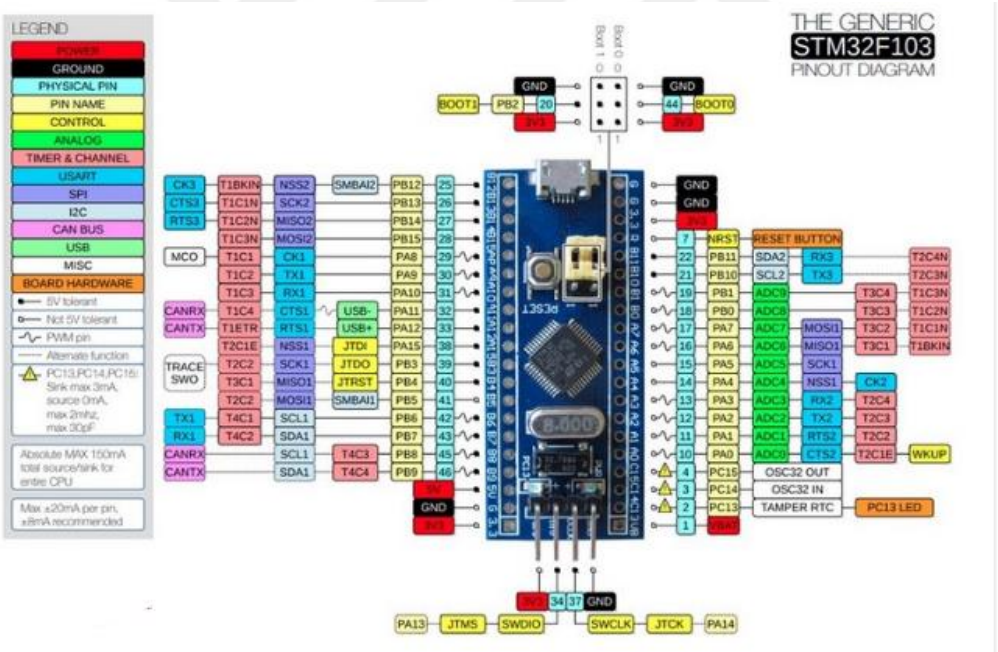
Bu tez çalışmasında merkezi işlem birimi olarak Şekil 3.2'de üstten görünümü verilmiş olan STM32F103C8T6 mikrodenetleyicisi temelli blue pill isimli kart kullanılmıştır. Bu kartın en belirgin özelliklerinden birisi çok güçlü bir Arduino Nano alternatifi olmasıdır. Bu mikrodenetleyici kartının üzerinde MicroUSB Interface, Boot Selection, MCU, RTC kristal, Power Indicator, RC13 Indicatör, Main kristal ve reset butonu yer almaktadır. (Diyot.net, t.y.)



Şekil 3.2. STM32F103C8T6 mikrodenetleyicisi temelli blue pill kartının üstten görünüşü

Burada MicroUSB Arabirimi: Bilgisayar veya diğer cihazlarla iletişim kurmak için kullanılan bir MicroUSB bağlantı noktasıdır. Boot Select: Kartın başlangıçta hangi yazılımı yükleyeceğini belirlemek için bir anahtar veya *jumper*ı ifade eder. Bu anahtar önyükleme seçeneklerini kontrol etmek için kullanılır. MCU (Microcontroller Unit): STM32F103C8T6 mikrodenetleyici entegresi kartın temel işlemcisidir ve programların yürütüldüğü ana bileşendir. RTC Crystal (Real-Time Clock Crystal): Gerçek zaman saati (RTC) devresi için kristal osilatör, kartın gerçek zamanlı saat ve takvim fonksiyonları sağlamasına yardımcı olur. Power Indicator: Kartın güç durumunu gösteren bir gösterge LED'i, kartın besleme alıp almadığını veya güç durumunu belirtmek için kullanılır. RC13 Indicator: Kartın belirli bir pininin durumunu gösteren bir gösterge LED'idir. RC13, bir GPIO pini olarak belirlenmiştir. Main crystal (ana kristal): Mikrodenetleyicinin ana

osilatör kristali, mikrodenetleyicinin ana saat kaynağını sağlar ve işlemcinin temel frekansını belirler. Reset Butonu: Mikrodenetleyiciyi sıfırlamak için kullanılan bu buton, kartın yazılım veya donanım sorunları durumunda yeniden başlatılması için kullanılır. Aynı zamanda bu kart üzerinde diğer harici bağlantı noktaları yer almaktadır. Bunlar genellikle GPIO (Genel Amaçlı Giriş/Çıkış) pinleri ve diğer iletişim arayüzleri için bağlantı noktalarıdır. Bu bağlantı noktaları, sensörler, ekranlar, motorlar ve diğer harici bileşenlerle iletişim kurmak için kullanılabilir. Besleme bağlantısı kartı beslemek için bir güç bağlantı noktasıdır. Genellikle 5V DC (veya 3.3V DC) ile çalışır. STM32F103C8 temelli blue pill geliştirme kartı 32 bit bir işlemci mimarisine sahip, maksimum 72 MHz CPU hızıyla Cortex-M3 çekirdeğini kullanır. Cortex – M serisi: Klasik 8/16 bit mikrodenetleyicilerin kullanıldığı ve düşük maliyet / düşük güç tüketimi gerektiren alanlarda kullanılmak üzere geliştirilen seridir. Son dönemde oldukça popüler olan STM32 mikrodenetleyicileri de bu seriye aittir. (Diyot.net, t.y.)

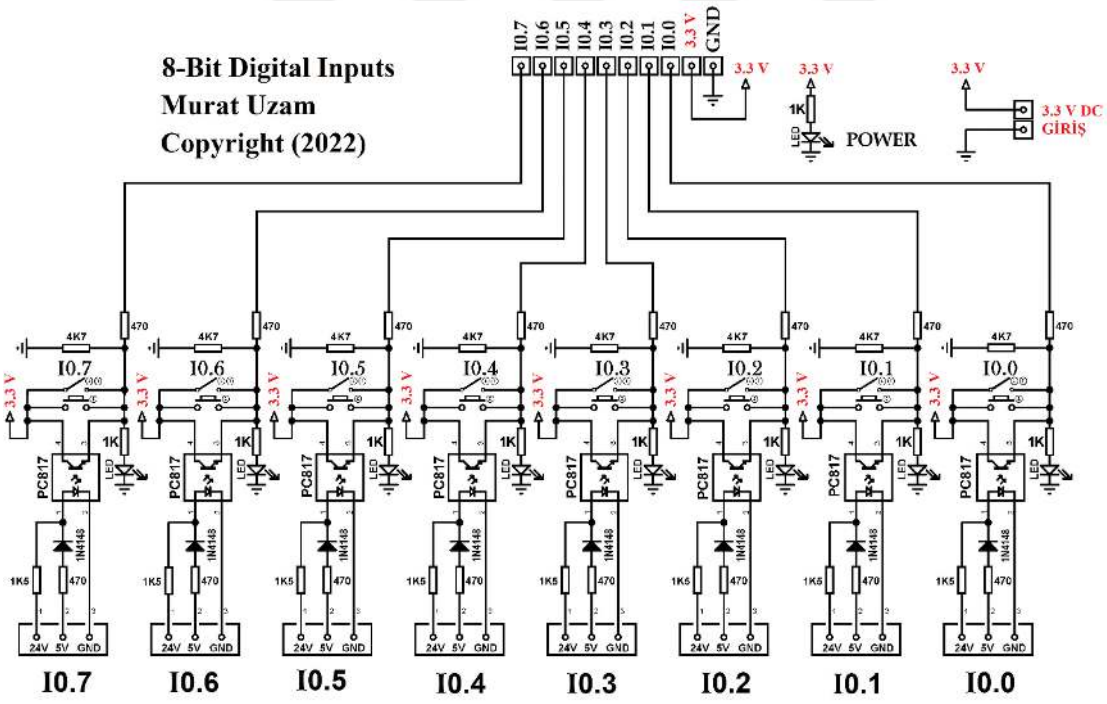


Şekil 3.3. STM32F103C8 mikrodenetleyicisi temelli blue pill kartının pin diyagramı

Şekil 3.3'te STM32F103C8 mikrodenetleyicisi temelli blue pill kartının pin diyagramı görülmektedir (STM32F103 pinout diagram, t.y). Proje kapsamındaki pinler kullanılırken bu kartın mevcut şeklinden yararlanılmıştır. Örneğin: PWM pin sayısı kart üzerinde mevcut olan dalgalı şekillerle ifade edilmiştir. Analog pinler Şekil 2.3'te belirtildiği üzere 10 tanedir. (AD0, AD1,…..AD9). PWM pin sayısı mevcut kartta 15 tanedir.

3.3. Dijital Girişler

Bu tez projesi kapsamında kullanılan 8 bit dijital PLC girişlerinin şeması Şekil 3.4'te görülmektedir. Bu şemaya uygun olarak gerçekleştirilmiş olan 8 bit dijital PLC giriş kartının üstten görünüşü de Şekil 3.5'te mevcuttur. PLC girişleri I0.7, I0.6, I0.5, I0.4, I0.3, I0.2, I0.1 ve I0.0 olarak isimlendirilmiştir. Bu dijital girişler için STM32F103C8 mikrodenetleyicisinin PB4, PB3, PA15, PA6, PA11, PA10, PA9, PA8 isimli pinleri kullanılmıştır. Her bir ayrık girişten 5V DC ya da 24V DC giriş sinyallerini kabul edilebilmektedir. Harici giriş sinyalleri NPN tipi PC817 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.

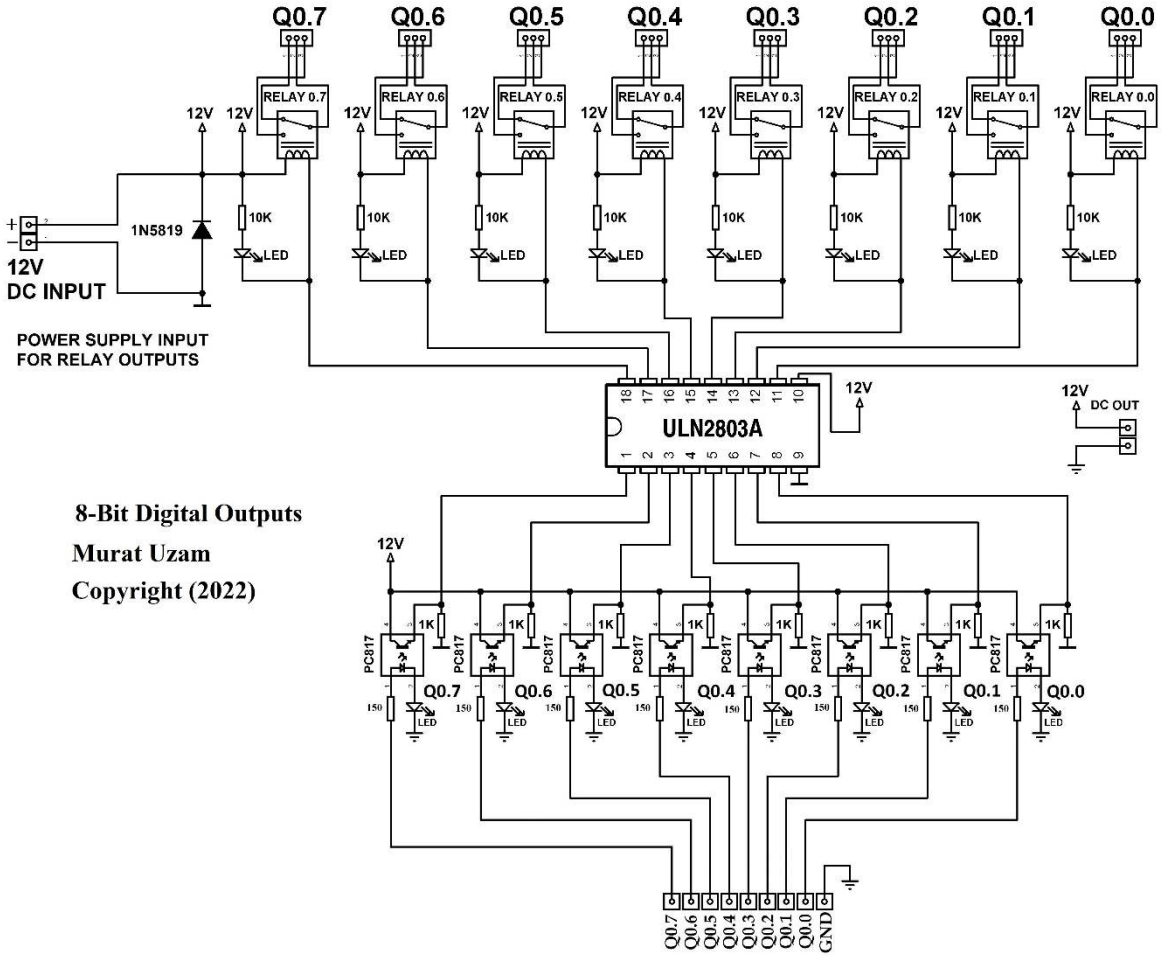


Şekil 3.4. 8 bit dijital PLC girişlerinin şeması

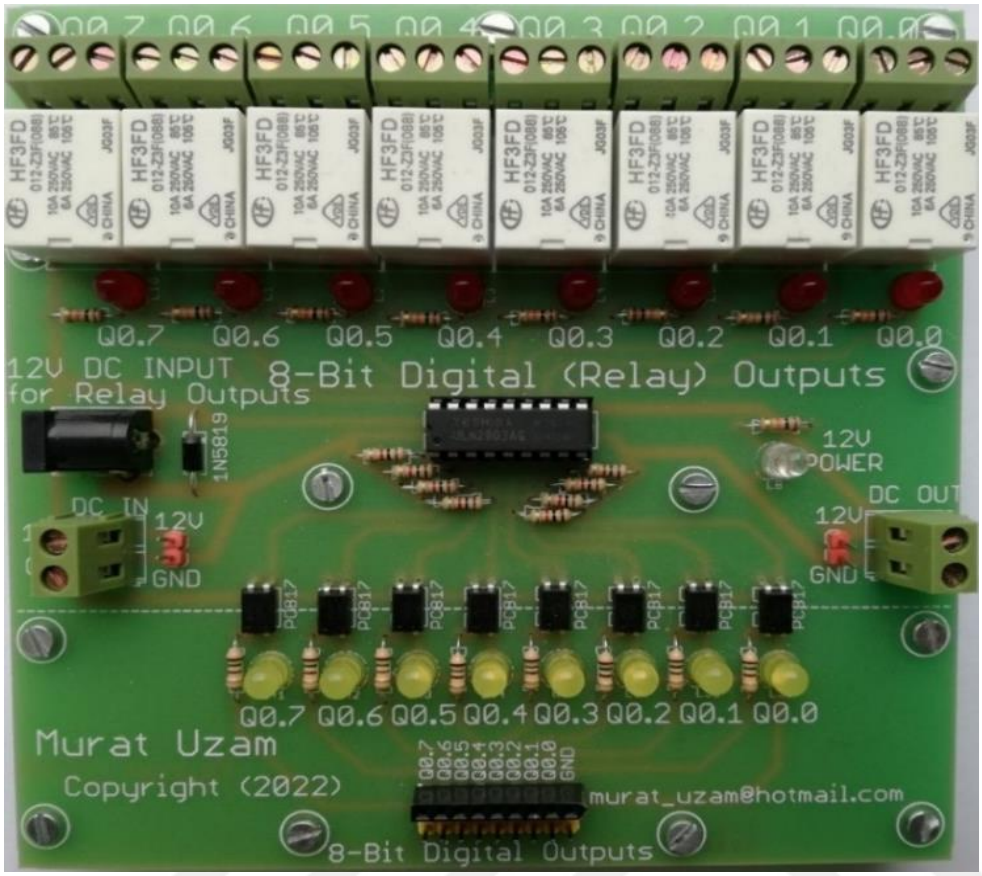
Şekil 3.5. 8 bit dijital PLC giriş kartının üstten görünüşü

3.4. Dijital Çıkışlar

Bu tez projesi kapsamında kullanılan 8 bit dijital PLC çıkışlarının şeması Şekil 3.6'da görülmektedir. Bu şemaya uygun olarak gerçekleştirilmiş olan 8 bit dijital PLC çıkış kartının üstten görünüşü de Şekil 3.7'de mevcuttur. PLC çıkışları Q0.7, Q0.6, Q0.5, Q0.4, Q0.3, Q0.2, Q0.1 ve Q0.0 olarak isimlendirilmiş ve 8 adet ayrık röle çıkışı düzenlenmiştir. Bu röleleri sürmek için STM32F103C8 mikrodenetleyicisinin PA4, PA5, PA6, PA7, PB0, PB1, PB10, PB11 isimli pinleri kullanılmıştır. Her röle 12V DC ile çalışmaktadır. Bu 8 röle bir tane ULN2803A darlington transistör sürücü entegresi yardımıyla sürülmektedir. STM32F103C8 mikrodenetleyicisi ile ULN2803A darlington transistör sürücü entegrelerini elektriksel olarak izole etmek için 8 tane NPN tipi PC817 optik yalıtıcı (opto-coupler) kullanılmıştır. Böylece röleleri sürmek için kullanılan 12V DC gerilim de STM32F103C8 mikrodenetleyicisinden elektriksel olarak izole edilmiştir. Her rölede, C (common – ortak uç), NC (normally closed – normalde kapalı) ve NO (normally open – normalde açık) olmak üzere üç bağlantı ucu olan SPDT (single pole double throw – tek grup iki konumlu) kontaklar mevcuttur. Devrede, her bir rölenin çalışmakta olduğunu gösteren bir LED mevcuttur. Ayrıca röleleri sürmek için kullanılan 12V DC gerilim kullanılmadan da çıkışların lojik durumlarını gözlemleyebilmek için her bir çıkış pinine bir LED bağlanmıştır.



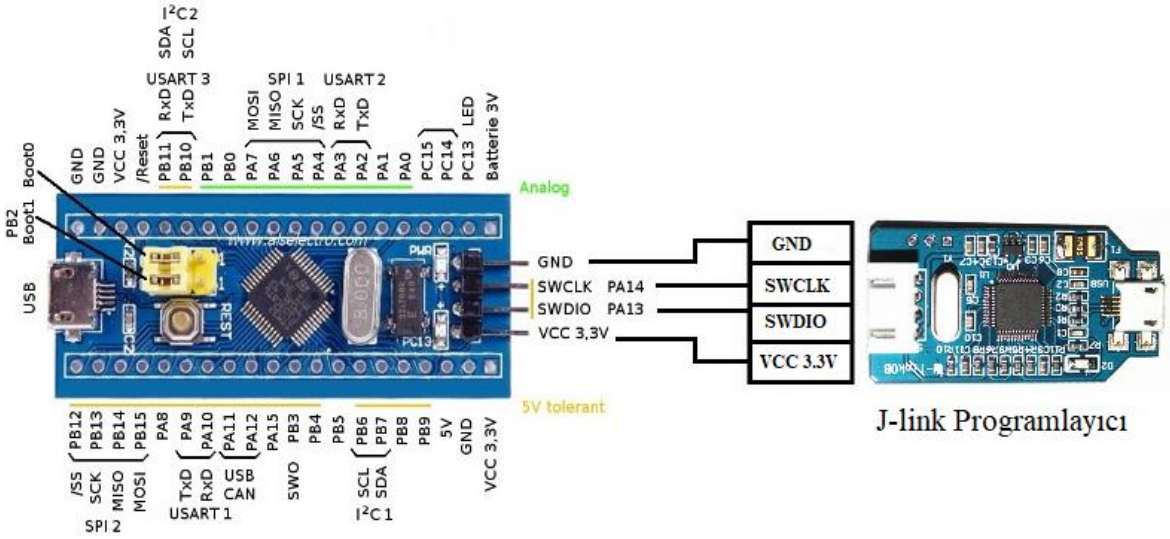
Şekil 3.6. 8 bit dijital PLC çıkışlarının şeması



Şekil 3.7. 8 bit dijital PLC çıkış kartının üstten görünüşü

3.5. Program Yükleme Kısmı

Bu projede kullanılan STM32F103C8 mikrodenetleyicisine kullanıcı programının yüklemesiyle ilgilidir. J-Link, bir STMicroelectronics STM32 mikrodenetleyicisine bir kullanıcı programını yüklemek ve hata ayıklamak için kullanılan bir donanımdır. Gerçekleştirilen PLC projesi kapsamında STM32F103C8 mikrodenetleyici ile birlikte kullanılmıştır. Şekil 3.8'de STM32F103C8 mikrodenetleyicisinin J-link programlayıcı donanımına bağlantısı görülmektedir. Bu sayede J-link, bir USB arayüzü aracılığıyla bilgisayara bağlanılıp ve bir yazılım geliştirme ortamı (IDE) veya hata ayıklayıcı yazılım aracılığıyla kullanabilmektedir.



Şekil 3.8. STM32F103C8 mikrodenetleyicisinin J-link programlayıcı donanımına bağlanması

STM32F103C8 mikrodenetleyicisinin J-link donanımını STM32CubeIDE programında çalıştırmak için şu adımlar izlenmiştir: Öncelikle J-link donanımı STM32F103C8 mikrodenetleyicisine Şekil 3.8'de görülen dört bağlantı kablosu ile bağlanmıştır. Sonrasında STM32 için tercih edilen STM32CubeIDE tümleşik geliştirme programının kurulumu yapılmış ve STM32CubeIDE, kod yazma, derleme, hata ayıklama ve J-link aracılığıyla programlama işlemlerinde kullanılmıştır. STM32CubeIDE geliştirme ortamının J-link ile iletişim kurabilmesi için gerekli olan J-Link sürücüleri ve yazılımı proje kapsamında kullanılan bilgisayara yüklenmiştir.

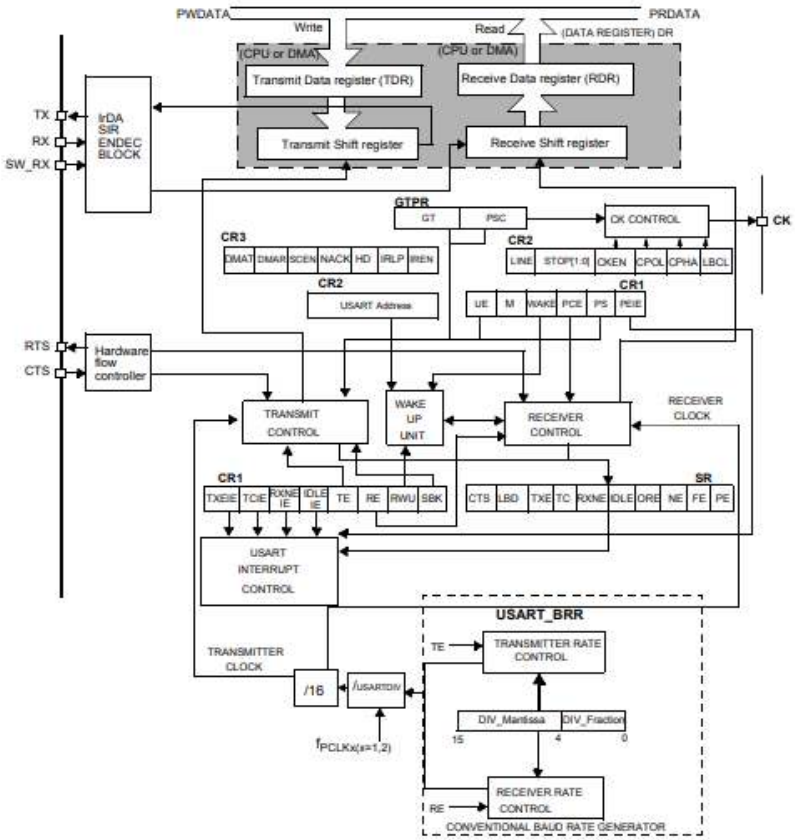
3.6. RS232 USB-TTL Seri Haberleşme Dönüştürücü

Şekil 3.9'daki RS232 USB-TTL seri haberleşme dönüştürücü modülü gerçekleştirilen tez projesi kapsamında bilgisayarda bir monitör ekranı oluşturup özellikle matematik, lojik ve kaydırma fonksiyonlarıyla ilgili işlem sonuçlarının gözlemlenebilmesi amacıyla kullanılmıştır. Bu seri haberleşme dönüştürücü modülü, bilgisayarlar veya diğer USB destekli cihazlar ile seri haberleşme protokollerini kullanarak iletişim kurmaya yarayan bir donanım modülüdür. Bu modüller genellikle UART Universal Asynchronous Receiver/Transmitter olarak da adlandırılır. Bu modüller genellikle seri haberleşme hızı, veri biti, dur biti ve parite gibi iletişim parametrelerini ayarlamak için yazılım veya

donanım düğmeleri sağlayan küçük cihazlardır. Bu tür modüller, genellikle mikrodenetleyiciler, Arduino ve diğer gömülü sistemlerle iletişim kurmak için ve hata ayıklama, veri alışverişi ve diğer seri haberleşme uygulamalarında yaygın olarak kullanılır. Aynı zamanda bu dönüştürücü modülü sensörler, ekranlar, motor sürücüleri gibi çeşitli elektronik bileşenlerle haberleşme işleminde de kullanılır.

Şekil 3.9. RS232 USB-TTL seri haberleşme dönüştürücü modülünün üstten görünüşü

RS232 USB-TTL seri haberleşme dönüştürücü modülü STM32F103C8 mikrodenetleyici kartına bağlanmıştır. Burada mikrodenetleyicinin alternatif fonksiyonlarını belirlemek için GPIO'ların Alternatif Fonksiyon kaydedicilerinden (AFR) yararlanılmıştır. AFR'ler özel kaydedicilerdir ve her pinin hangi alternatif fonksiyonu gerçekleştireceğini belirtirler. Mikrodenetleyici, bu kaydedicileri kullanarak pinlerin alternatif fonksiyonlarını yapılandırır. Alternatif fonksiyonlardan olarak USART arabirimi kullanılmıştır. İşlem pini USART'ın harici devreleri ile sağlanır. Şekil 3.10'da USART'ın blok diyagramı görülmektedir.



Şekil 3.10. USART blok diyagramı (RM0008, 2021)

Şekil 3.11'de görülen USART arabirimi için, transmit (Tx) ve receive (Rx) hatları gibi harici devreler gereklidir. Bir başka deyişle, USART (senkron) pinlerinin (TX, RX, GND) bağlamayı gerektirir. Burada göndermek istenilen verinin belirtilen yere gitmesi için TDR bloğu ve veri geldiğinde bu verinin okunabilmesi için bir RDR bloğu yer almaktadır. Ayrıca en aşağıda USART_BRR isimli haberleşme hızını kontrol eden bir birim bulunmaktadır. Gelen clock kaynağına göre hızın ayarlanması işlemleri yapılmaktadır. USART'ın hangi frekansta haberleştiğini bilmek önemlidir. Bu değer Baud Rate Register hesabı yapılırken kullanılmaktadır. Orta kısımda ayarlarla ilgili bitler bulunup burada kontrol USART ve kesme (interrupt) ayarlamalarının yapıldığı bölüm yer almaktadır.

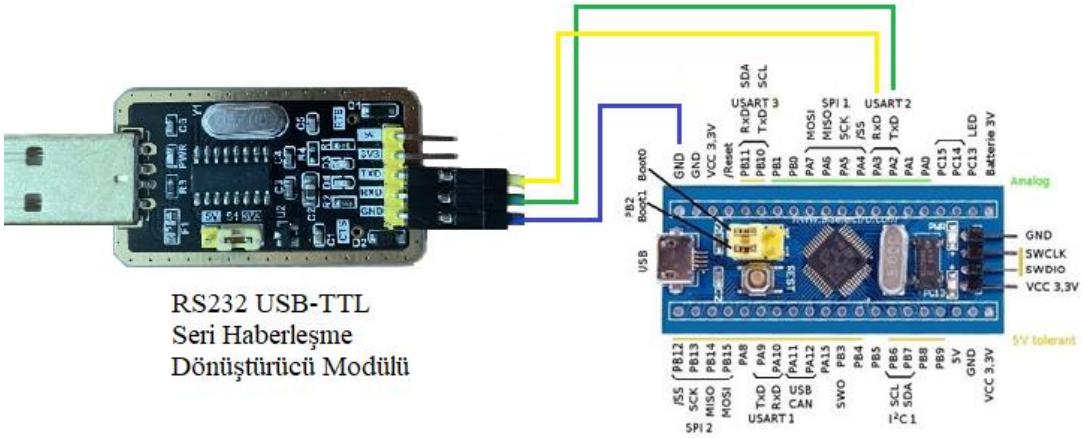
USART kullanılırken ilgili pinlerin ayarlaması yapılmalıdır. Bu pinler veri sayfasından takip edilerek gerçekleştirilir. Veri sayfasından yola çıkılarak çalışmanın gerçekleştirilmesinde UART2 (sekron) tercih edilmiştir. Tablo 3.1'de USART2'nin pin

eşlenmesi görülmektedir. GPIO giriş pinlerinin yapılandırılmasında bu tablo verilerinden faydalanılmıştır.

Tablo 3.1. USART2'nin pin eşlenmesi (RM0008, 2021)

Alternate functions	USART2_REMAP = 0	USART2_REMAP = 1[1]
USART2_CTS	PA0	PD3
USART2_RTS	PA1	PD4
USART2_TX	PA2	PD5
USART2_RX	PA3	PD6
USART2_CK	PA4	PD7

Tablo 3.1'de STM32F103C8 mikrodenetleyici kartının haberleşme pinleri PA2 ve PA3 pinleri olarak belirlenmiştir. Şekil 3.11'de belirtilen bu pinler kullanılarak RS232 USB-TTL seri haberleşme dönüştürücü modülünün mikrodenetleyiciye bağlantısı görülmektedir.

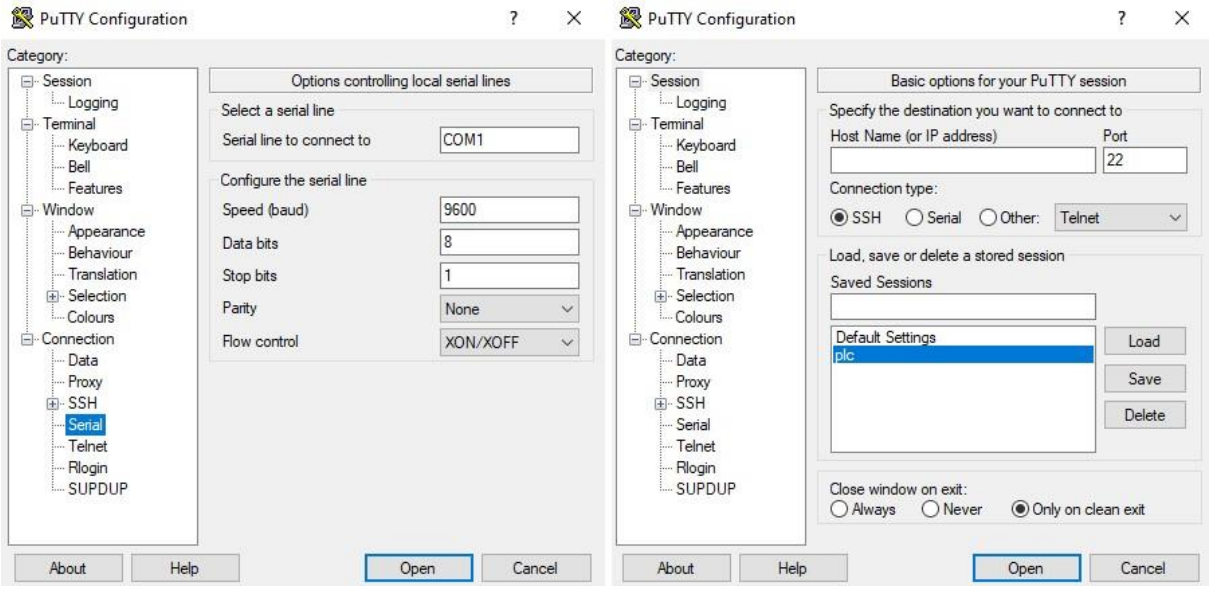


Şekil 3.11. RS232 USB-TTL seri haberleşme dönüştürücü modülünün mikrodenetleyiciye bağlanması

Bu modül, bilgisayar üzerinde sürücü yazılımı gerektirmektedir ve bu amaçla bu çalışmada kullanılan Putty sürücüsünün ekran görüntüsü Şekil 3.12'de görülmektedir. PuTTY, Windows işletim sistemi için geliştirilmiş olup kolay bir uygulama olmasıyla bilinir. PuTTY, bilgisayarlar arası uzak erişim ve dosya transferi sağlayan ücretsiz ve açık kaynaklı bir bilgisayar programıdır. Seri bağlantı protokollerini destekler. Bu nedenle, seri bağlantı üzerinden cihazlarla etkileşimde bulunmak için kullanılabilir.

PuTTY programı (Putty, t.y), bu tez çalışmasında kullanılan bilgisayarınıza indirilip kurulmuştur. Kurulum tamamlandıktan sonra PuTTY arayüzü açıldığında, bağlanmak

istenilen hedef bilgisayarın IP adresi girilir. Bağlantı türü (SSH, Telnet, Rlogin vb.) seçildikten sonra bağlantı portu ayarlanır. Çalışmada bağlantı türü olarak SSH seçilmiştir. Sonra "Bağlan" düğmesine basıldıktan sonra baud değeri girilir. Bu baud değeri bir frekans bandının genişliğini veya bant genişliğini ifade eder ve bu değer 9600 olarak kullanılmıştır. Bağlantı başarılı olduğunda, PuTTY penceresinde bir komut satırı açılır. Buradan hedef sistemde komutlar çalıştırılabilir, dosyalar yönetilebilir ve diğer işlemler gerçekleştirilebilir konuma gelir.



Şekil 3.12. Putty programına ait ekran görüntüsü

Program çalışabilir konuma geldikten sonra kullanılan modülün donanım bağlantısı ve konfigürasyon ayarlamaları yapılmalıdır. Bu bağlantıda Rx ve Tx birisi datayı gönderirken diğeri datayı almaktadır. Bilgisayarın Rx'i PLC'nin Tx'ine bağlanır. Tx ve Rx'ler birbirine çapraz bağlanır. Yani bilgisayarın bir bacağı datayı sürekli gönderirken diğer bacağından sürekli data alınır. Benzer işlem PLC'de gerçekleşir. Dolayısıyla TTL cihazının Tx pini STM32 kartının Rx pinine, Rx pini de STM32 kartının Tx pinine bağlanmıştır. Toprak pini (GND) STM32 kartının toprak pinine bağlanmıştır. Böylelikle bilgisayar ve PLC arasında haberleşme gerçekleşir.

Sonuçta PA2 ve PA3 pinlerinin sırasıyla Tx ve Rx olarak kullanılması amacıyla STM32CubeIDE geliştirme programında haberleşme modülü aktifleştirilmiş ve GPIO

ayarlamaları Hal kütüphanesinden yararlanılarak gerçekleştirilmiştir. Bu amaçla geliştirme programında yazılan C codu şu şekildedir:

```
// RS232 USB-TTL Seri Haberleşme Dönüştürücü Modülü GPIO
GPIO_InitStruct.Pin = GPIO_PIN_2;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;    // GPIO_PULLDOWN   GPIO_PULLUP
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

GPIO_InitStruct.Pin = GPIO_PIN_3;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;    // GPIO_PULLDOWN   GPIO_PULLUP
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

Bu sayede mikrodenetleyici üzerinde bulunan ve donanım bağlantısı için belirlenen pinler aktifleştirilmiştir.

RS232 USB-TTL seri haberleşme dönüştürücü modülü için STM32CubeIDE programında konfigürasyon: STM32CubeIDE programında yazılım tarafında veri gönderme ve alma işlemleri için bir fonksiyon kullanılmıştır. Bu sayede RS232 USB-TTL seri haberleşme dönüştürücü modülü ile iletişim kurması sağlanmıştır. Bu fonksiyon HAL kütüphanesinde yer alan USART fonksiyonu olup veri gönderme ve alma işlemlerini gerçekleştirir. USART fonksiyonu etkinleştirilip uygun pinlerin ve baud hızının (bit hızı) yapılandırılması sağlanır. Burada önemli olan BRR olarak ifade edilen Baud rate regülasyonudur. Proje kapsamında kullanılan USART2_setup() fonksiyonu aşağıda verilmiştir:

```
void USART2_setup(void)
{
	RCC->APB1ENR |= RCC_APB1ENR_USART2EN;

	USART2->CR1 &= (uint16_t)~((uint32_t)USART_CR1_UE);
	USART2->CR1 = 0x0000;
	USART2->CR2 = 0x0000;
	USART2->CR3 = 0x0000;

	USART2->BRR = 3750;		// 1875  3072 512   104.1666667 0x683

	USART2->CR1 |= USART_CR1_TE ;
	USART2->CR1 |= USART_CR1_UE ;
}
```

RCC->APB1ENR |= RCC_APB1ENR_USART2EN: USART2'nin clock sinyalini etkinleştirir. RCC (Reset and Clock Control) yapılandırma kaydedicisi, mikrodenetleyicinin çeşitli sistemlerini kontrol eder.

USART2->CR1 &= (uint16_t)~((uint32_t)USART_CR1_UE): USART_CR1_UE bitini temizler. UE (USART Enable) biti, USART modülünü açar veya kapatır.

USART2->CR1 = 0x0000;, USART2->CR2 = 0x0000;, USART2->CR3 = 0x0000: USART kontrol kaydedicilerini sıfırlar. Böylelikle, varolan herhangi bir yapılandırma temizlenmiş olur.

BRR (Baud Rate Register) değeri, baud hızını ayarlamak için kullanılır ve seri iletişim sinyalinin saniye başına düşen sayısını ifade etmektedir. Bu sinyalinin doğru şekilde hesaplanması gerekmektedir. Hesaplama işlemi genellikle mikrodenetleyicinin clock frekansına ve istenen baud hızına bağlıdır. Bölücü değerinin hesaplaması için ilk olarak

baud hızı, baud oranını belirlemek için BRR (Baud Rate Register) değeri kullanılarak ayarlanır. Bölücü değeri, aşağıdaki ifade edilen formülle hesaplanır:

STM32F103C8 mikrodenetleyicisinin kristal frekansı bu tez çalışmasında frekans değeri 72 MHz seçilmiştir ve baud hızı 9600 olarak belirlenmiştir. Bölücü değeri hesabı şu şekilde bulunur (RM0008, 2021):

$$BRR = \frac{\frac{fck + baudrate}{2}}{baudrate} = \frac{\frac{72.000.000 + 9600}{2}}{9600} = 3750$$

Burada; Fck, mikrodenetleyicinin çalışma frekansını, baudrate ise istenen baud hızını ifade etmektedir. Böylelikle, STM32CubeIDE programında USART2 ile belirtilen UART modülünün baud hızını ayarlar. Yani bu değer STM32CubeIDE programında fonksiyonda yer alan USART2->BRR yerine yazılır.

USART2->BRR = 3750: Baud oranı regülasyonunu yapılandırır. Burada, 3750 değeri, istenen baud hızını belirtir. Baud hızı, veri iletişimindeki sembol oranını belirtir.

Ayrıca mevcut C kodunda yer alan;

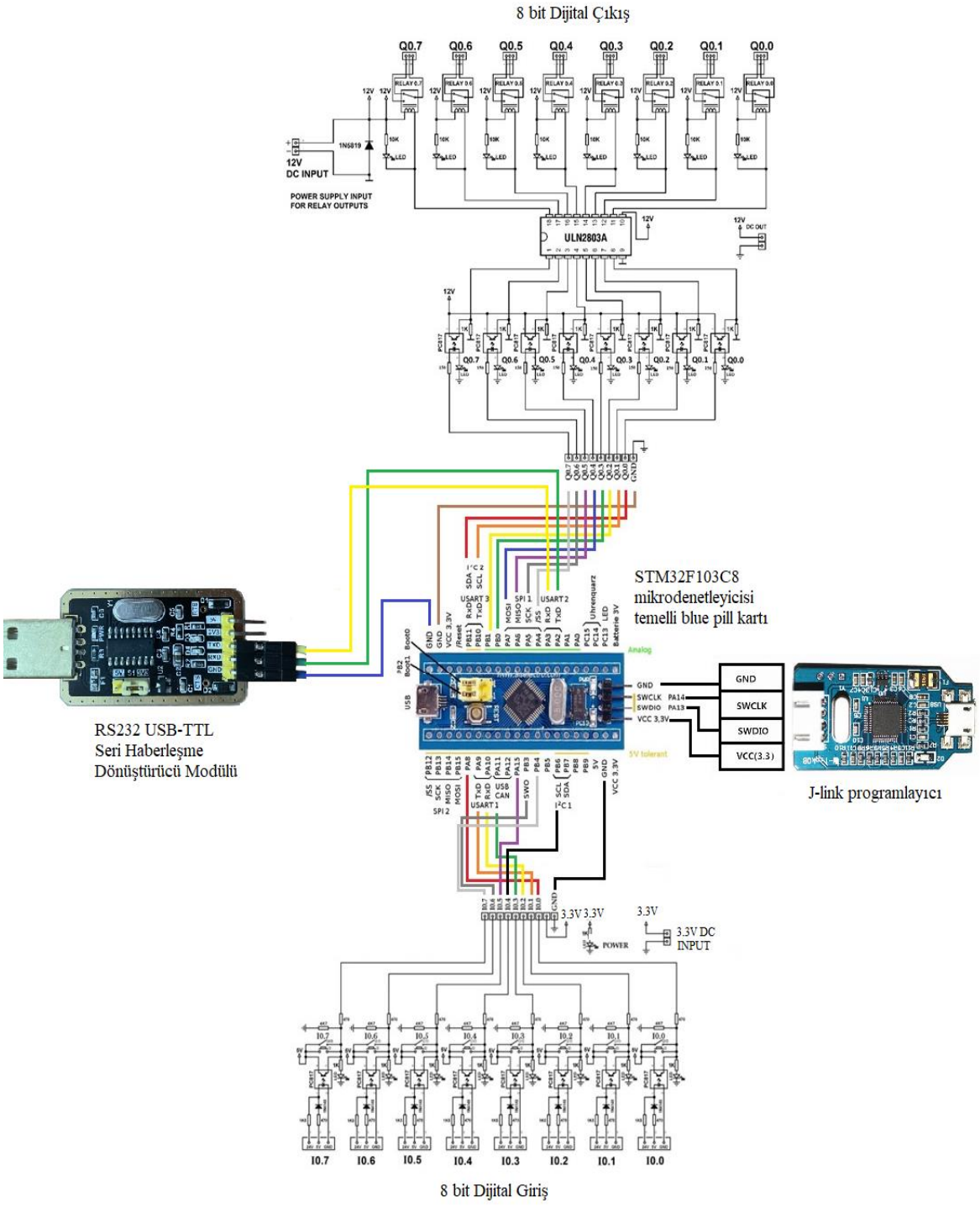
USART2->CR1 |= USART_CR1_TE: USART veri iletimini etkinleştirir. TE (Transmitter Enable) biti, veri gönderme işlevselliğini açar.

USART2->CR1 |= USART_CR1_UE: USART modülünü etkinleştirir. UE (USART Enable) biti, USART modülünü açar.

Kullanılan modülün donanım bağlantısı ve konfigürasyon ayarlamaları yapıldıktan sonra oluşturulan kod derlenir ve daha sonra STM32F103C8 mikronedetleyici kartına yüklenir. Putty programı yardımıyla RS 232 USB-TTL dönüştürücü ile iletişim kurulup işlevselliği gerçekleşir. Bu sayede proje kapsamında ihtiyaç duyulan ekranda bir monitör ihtiyacı karşılanmıştır.

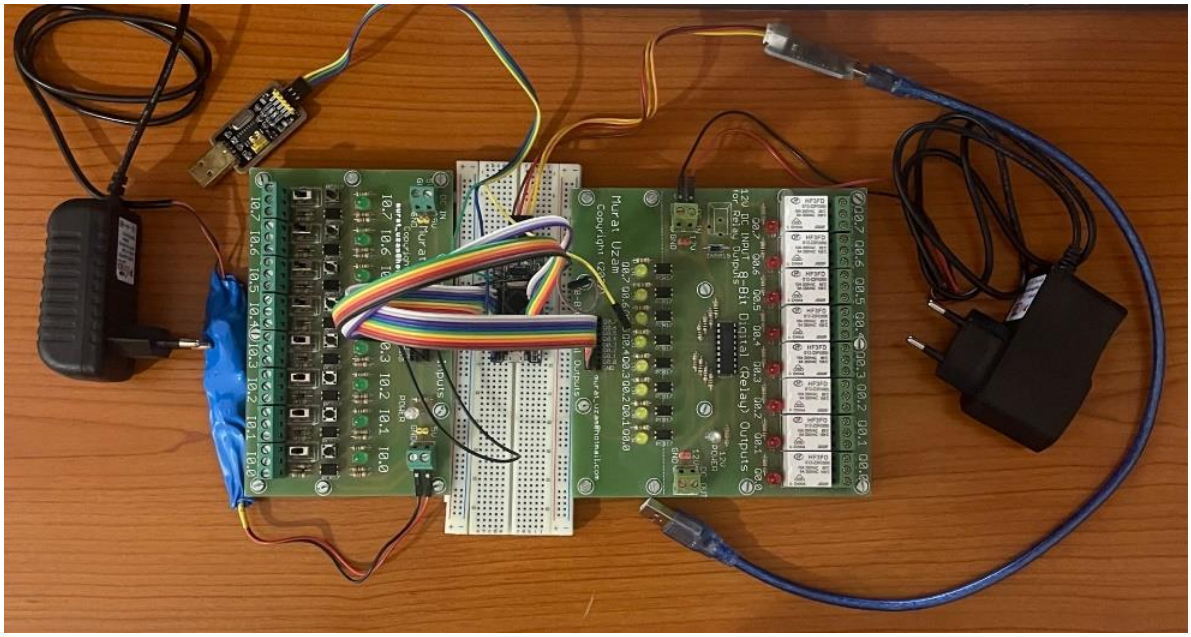
3.7. Tasarlanan PLC Donanımı

Sonuç olarak bu tez çalışması kapsamında STM32F103C8 mikrodenetleyici kullanılarak tasarlanan PLC'ye ait donanım şeması Şekil 3.13'te görülmektedir.



Şekil 3.13. Tasarlanan PLC'ye ait donanım şeması

Şekil 3.13'te görülen donanım şeması gereğince STM32F103C8 mikrodenetleyicisi temelli geliştirme kartının bir protoboard üzerine yerleştirilmesi ve gerekli bağlantıların kablolar kullanılarak gerçekleştirilmesi sonrasında elde edilen PLC donanımının üstten görünüşü Şekil 3.14'te verilmiştir.



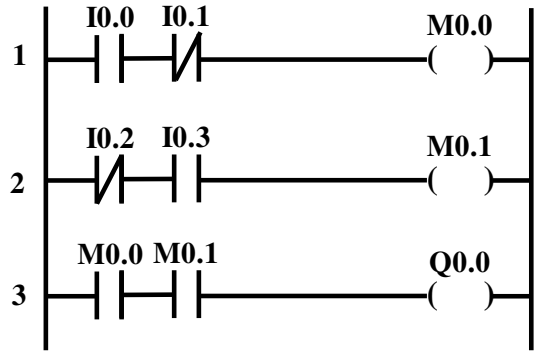
Şekil 3.14. Tasarlanan PLC'ye ait donanımın üstten görünüşü

4. TEMEL YAZILIM BİLEŞENLERİ

Bu yüksek lisans tez çalışmasında tasarlanan PLC'nin yazılımı C programlama Dili kullanılarak STM32CubeIDE (https://www.st.com/en/development-tools/stm32cubeide.html) bütünleşik geliştirme ortamı yardımıyla geliştirilmiştir. Bu kapsamda bu bölümde öncelikle PLC'lerin programlanması için kullanılan programlama dillerinden bahsedilmektedir. Sonrasında C programlama dili ile ilgili bazı önemli konular hakkında bir inceleme sunulmuştur. Üçüncü olarak STM32CubeIDE'nin kurulumu ve PLC projesini oluşturma adımları açıklanmaktadır. Sonrasında tasarlanan PLC'nin temel yazılım yapısı sunulmuştur. Zamanlayıcı fonksiyonlarında kullanılan referans zamanlama sinyallerinin üretilmesi amacıyla kullanılan kesme ile ilgili açıklamalar bu bölümdeki 5. kısımda yer alır. Dijital girişlerin kullanılması için çözülmesi zorunlu olan kontak atlaması problemi ve bu proje kapsamında bu problemin nasıl çözüldüğü ile ilgili açıklamalar bu bölümün son kısmında yer almaktadır.

4.1. PLC'lerin Programlanması

PLC'lerde kullanılan birçok programlama dili mevcuttur. En yaygın olarak kullanılan PLC programlama dili, merdiven diyagramı (ladder diagram) dilidir. Bu dil, şekil olarak bir merdivenin basamaklarına benzediği için bu adı almıştır. Yaygın olmasının sebebi de PLC'ler ortaya çıkmadan önce kullanılmakta olan elektromekanik kumanda sistemlerinde kullanılan şemalara benzemesindendir. Bu tez çalışmasında gerçekleştirilen fonksiyonlar genel olarak merdiven diyagramı sembolleri ile ifade edilecektir. Bir merdiven diyagramı basamaklardan (rungs) oluşur. Merdiven diyagramındaki basamaklar üstten alta doğru ve her basamak soldan sağa doğru olacak şekilde taranmaktadır. Her basamakta, açık veya kapalı kontaklarla temsil edilen girişlerin durumlarına göre çıkışın (dahili/harici röle bobininin veya ilgili fonksiyonun) durumu belirlenir (Otomasyonavm, t.y.). Şekil 4.1'de örnek bir merdiven diyagramı görülmektedir. Birinci basamakta I0.0 = 1 ve I0.1 = 0 ise dahili çıkış M0.0 = 1 olur. İkinci basamakta I0.2 = 0 ve I0.3 = 1 ise dahili çıkış M0.1 = 1 olur. Üçüncü basamakta M0.0 = 1 ve M0.1 = 1 ise harici çıkış Q0.0 = 1 olur.



Şekil 4.1. Örnek bir merdiven (ladder) diyagramı

4.2. C Dili Kullanılarak Programlama

C programlama dili 1972 yılında Dennis Ritchie tarafından tasarlanmıştır. C, ilk olarak Bell Laboratuvarında Unix işletim sistemi için geliştirilmiştir. Başlangıçta, C dili Unix işletim sistemi çekirdeğini ve temel araçlarını oluşturmak için kullanılmıştır. Ancak daha sonra C dilinin popülaritesi artmış ve genel amaçlı bir programlama dili haline gelmiştir. Bu yüzden C dili, çok çeşitli alanlarda kullanılan bir programlama dilidir. C dilinin hızlı ve düşük seviyeli yapısı, işletim sistemleri ve sistem yazılımları gibi yüksek performans gerektiren uygulamalarda sıkça tercih edilir. Aynı zamanda C dilinin taşınabilirliği ve düşük seviyeli programlama yetenekleri, gömülü sistemlerin geliştirilmesinde bu dilin seçilmesinin sebeplerindendir. Mikrodenetleyiciler, otomotiv sistemleri, tıbbi cihazlar ve endüstriyel kontrol sistemleri gibi birçok gömülü sistem, C dili kullanılarak geliştirilmektedir. Günümüzde nesneye yönelik programlama dilleri (C++, Java) ve script dilleri (JavaScript, JavaApplet, PHP) gibi programlama dilleri C programlama dilinden esinlenmiştir. Yani herhangi bir C programı hiçbir değişikliğe uğramadan veya çok az bir değişiklikle, başka bir derleyicide veya işletim sisteminde derlenebilir. Örneğin; ‘Windows’ işletim sistemlerinde yazılan bir C kodu, ‘Linux’, ‘UNIX’ veya ‘VAX’ gibi işletim sistemlerinde de derlenebilir.

4.2.1. C Dilinin Temel Özellikleri

Bir C programında aşağıda belirtilen özellikler yer almalıdır:

- Başlık dosyaları (#include "main.h", #include "definitions.h", #include <stdarg.h> gibi programın başlangıcında yer alır. Bu dosyalar, standart giriş/çıkış işlevlerini ve diğer önemli fonksiyonları içerir.
- C programlarının ana başlangıç noktası main() fonksiyonudur. Bu fonksiyon, programın çalışmasını başlatır ve diğer fonksiyonları çağırabilir. Gerçekleştirilen projede main() fonksiyonu şu şekilde başlar:

```
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_TIM1_Init();
  USART2_setup();
  // Programda kod burada yazılmıştır.
}
```

- Program içinde kullanılacak olan değişkenler tanımlanır.
- Her satırın sonuna mutlaka ‘ ; ’ işareti getirilmelidir.
- Her bloğun ve fonksiyonun başlangıcı ve bitişi sırasıyla ‘{’ ve ‘}’ sembolleri ile ifade edilir.
- Açıklama operatörü ‘/*, */’ sembolleri ile ifade edilir.

4.2.2. Veri Türleri

Veri türleri, bir programlama dilinde değişkenlerin içerdiği verinin türünü tanımlayan bir kavramdır. Bir değişkenin bellekte kaç bayt yer kaplayacağını, hangi tür verileri saklayabileceğini ve değişken üzerinde hangi işlemlerin yapılabileceğini belirler. Veri tipi,

değişkenin değerlerini belirleyip ve bu değerlerin hangi tür aritmetik işlemlerle işlenebileceğini belirler.

_Bool: Mantıksal (Boolean) değerleri temsil eder. Doğru (1) veya yanlış (0) değerlerini alabilir.

char: Bu veri tipi tek bir karakteri temsil eder. Örneğin, 'A', 'b', '?', '3' gibi tek bir karakter temsil edilir. “unsigned char” ve “signed char” olmak üzere iki çeşidi vardır.

1. ‘unsigned char’ ile 8 bitlik sayılar tanımlanır ve bu sayılar 0 ve 255 arasında değer alabilirler. Aşağıda görülen örnekte, veri tanımında değişken q, 0 ve 255 arasında herhangi bir değer alabilir ve burada q değişkeni 180 olarak değerlendirilmiştir. Aynı zamanda, a değişkeni karakter B’ye eşitlenmiştir.
   unsigned char q, a; q = 180; a=”B”;
2. ‘signed char’ veri tanımı, -128 ve +127 arasındaki sayıları tanımlamak için kullanılır. Aşağıdaki örnekte p değişkenine “-40” değeri ve z değişkenine ‘29’ değeri tanımlanmıştır. signed char p, z; p = -40; z = 29;

void: Genellikle fonksiyonların dönüş türü olarak kullanılır ve fonksiyonun geriye bir değer döndürmediğini belirtir.

int: Tamsayıları temsil eder. Örneğin, -10, 0, 42 gibi tam sayılar int veri tipi kullanılarak temsil edilir.

unsigned int: Sadece pozitif tamsayı değerleri alır. Örneğin, 0'dan büyük tamsayılar unsigned int veri tipi kullanılarak temsil edilir.

const: Bir değişkenin değerinin bir kez atanıp daha sonra değiştirilemeyeceğini belirtir.

const int: Bir tam sayı değişkeninin değeri sabitlenir ve daha sonra değiştirilemez. Örneğin: const int G = 29;

const char: Bir karakter değişkeninin değeri sabitlenir ve daha sonra değiştirilemez. Örneğin: const char INITIAL = 'G';

const float: Bir ondalıklı sayı değişkeninin değeri sabitlenir ve daha sonra değiştirilemez. Örneğin: const float Pİ = 3.14;

4.2.3. Aritmetik Operatörler

Aritmetik operatörler, matematiksel işlemleri gerçekleştirmek için kullanılan operatörlerdir. Bu operatörler, sayılar arasında toplama, çıkarma, çarpma, bölme gibi temel aritmetik işlemleri yapmak için kullanılır. C programlama dilinde kullanılan temel aritmetik operatörler Tablo 4.1'de listelenmiştir

Tablo 4.1. Aritmetik operatörler

Operatör	Örnek	Anlamı
Toplama (+)	a + b	İki sayının (a ve b) toplamını bulmak için kullanılır.
Çıkarma (-)	a - b	İki sayı (a'dan b'yi) arasındaki farkı bulmak için kullanılır.
Çarpma (*)	a * b	İki sayının (a ve b) çarpımını bulmak için kullanılır.
Bölme (/)	a / b	Bir sayının diğerine (a'nın b'ye) bölünmesi sonucu elde edilen bölümü bulmak için kullanılır.
Modüler Bölme (%)	a % b	Bir sayının diğerine bölünmesi (a'nın b'ye) sonucu elde edilen kalanı bulmak için kullanılır.

4.2.4. Atama Operatörleri

C programlama dilinde sıkça kullanılan atama operatörleri bir değişkenin değerini belirlemek veya değiştirmek için kullanılır ve Tablo 4.2.'de ifade edilmiştir.

Tablo 4.2. Atama operatörleri

Operatör	Örnek	Anlamı
=	x=5 , x=5	Atama
+=	x=+=2, x=x+2	Ekleyerek atama
-=	x=-=2, x=x-2	Eksilterek atama
*=	x=*=2, x=x*2	Çarparak atama
/=	x=/=2, x=x/2	Bölerek atama
%=	x=%=2, x=x%2	Bölüp, kalanını atama
++	x++ veya ++x, x=x+1	Bir arttırma
--	x-- veya --x, x=x-1	Bir azaltma

4.2.5. Mantıksal Operatörler

C programlama dilinde sıkça kullanılan mantıksal operatörler, mantıksal ifadelerin değerini değerlendirmek ve mantıksal ilişkileri kontrol etmek için kullanılan bu operatörler Tablo 4.3'te sunulmuştur.

Tablo 4.3. Mantıksal operatörler

Operatör	Örnek	Anlamı
Ve (&&)	x > 5 && x < y	x 5'den büyük ve y'den küçük mü?
Veya (‖)	x > 5 ‖ x < y	x 5'den büyük veya y'den küçük mü?

4.2.6. Karşılaştırma Operatörleri

İki değeri karşılaştırmak için kullanan bu operatörler Tabla 4.4'te sunulmuştur.

Tablo 4.4. Karşılaştırma operatörleri

Operatör	Örnek	Anlamı
>	sayı1 > sayı2	sayı1, sayı2'den büyük mü?
> =	sayı1 > = sayı2	büyük veya eşit mi?
=	sayı1 = sayı2	sayı1, sayı2'ye eşit mi?
<	sayı1 < sayı2	sayı1, sayı2'den küçük mü?
< =	sayı1 < = sayı2	sayı1, sayı2'den küçük veya eşit mi?
≠	sayı1 ≠ sayı2	sayı1, sayı2'ye eşit değil mi?

4.2.7. "while" Döngüsü

While döngüsü, belirli bir koşul doğru olduğu sürece bir kod bloğunu tekrar tekrar çalıştırmak için kullanılan bir kontrol yapısıdır. Koşul doğru olduğu sürece döngü devam eder, koşul yanlış olduğunda döngü sona erer.

```
while (koşul) {
   // kod bloğu
}
```

Koşul, her döngü tekrarından önce değerlendirilen bir mantıksal ifadedir. Bu ifade doğru (true) olduğu sürece döngü devam eder, yanlış (false) olduğunda döngü sona erer.

4.2.8. “for” Döngüsü

Bu döngü genellikle belirli bir sayıda tekrarlamak istenilen durumlar için kullanılır.

Yapısı:

```
for (başlangıç_değeri; koşul; adım) {
   // kod bloğu
}
```

For döngüsünde öncelikle döngünün başlangıç değeri belirlenir. Döngü tekrarından önce değerlendirilen bir mantıksal koşul ifadesi belirlenir. Bu ifade doğru (true) olduğu sürece döngü devam eder, yanlış (false) olduğunda döngü sona erer. Her döngü tekrarından sonra döngü değişkeninin nasıl güncelleneceğini belirten bir adım ifadesi yer alır. Bu adım, döngü değişkeninin arttırılması veya azaltılması gibi işlemleri içerebilir. Döngü tekrarı için çalıştırılacak olan kod parçaları kod bloğunda yer alır.

4.2.9. “if” Ve “else if” Yapısı

Bu yapı programların belirli koşullara bağlı olarak farklı kod bloklarını çalıştırmasını sağlayan kontrol yapılarıdır.

1. if Yapısı: if yapısı, bir koşul doğru olduğunda belirli bir kod bloğunu çalıştırmak için kullanılır.
   Yapısı: if (koşul) { kod bloğu }
   Örneğin;
   ```
   int x = 10;
   if (x > 5) {
   printf("x 5'ten büyüktür.\n");
   }
   ```
2. else if Yapısı: Bu yapı bir önceki if veya else if yapısının koşulu yanlış olduğunda, yeni bir koşulu kontrol etmek için kullanılır.
   Yapısı: else if (koşul) { kod bloğu }
   Örneğin:
   ```
   int y = 20;
   ```

```
if (y > 25) {
    printf("y 25'ten büyüktür.\n");
} else if (y == 20) {
    printf("y 20'ye eşittir.\n");
} else {
    printf("y 25'ten küçüktür.\n");
}
```

Bu örneklerde, if yapısı belirli bir koşulu kontrol eder ve koşul doğru ise ilgili kod bloğunu çalıştırır. else if yapısı ise bir önceki koşulun yanlış olduğunda yeni bir koşulu kontrol eder ve uygun kod bloğunu çalıştırır.

4.2.10. Sonsuz Döngü

Bir döngü işlemini sonsuz kere tekrarlarsa bu döngü sonsuz döngü olarak ifade edilir. Örneğin while döngüsü için:

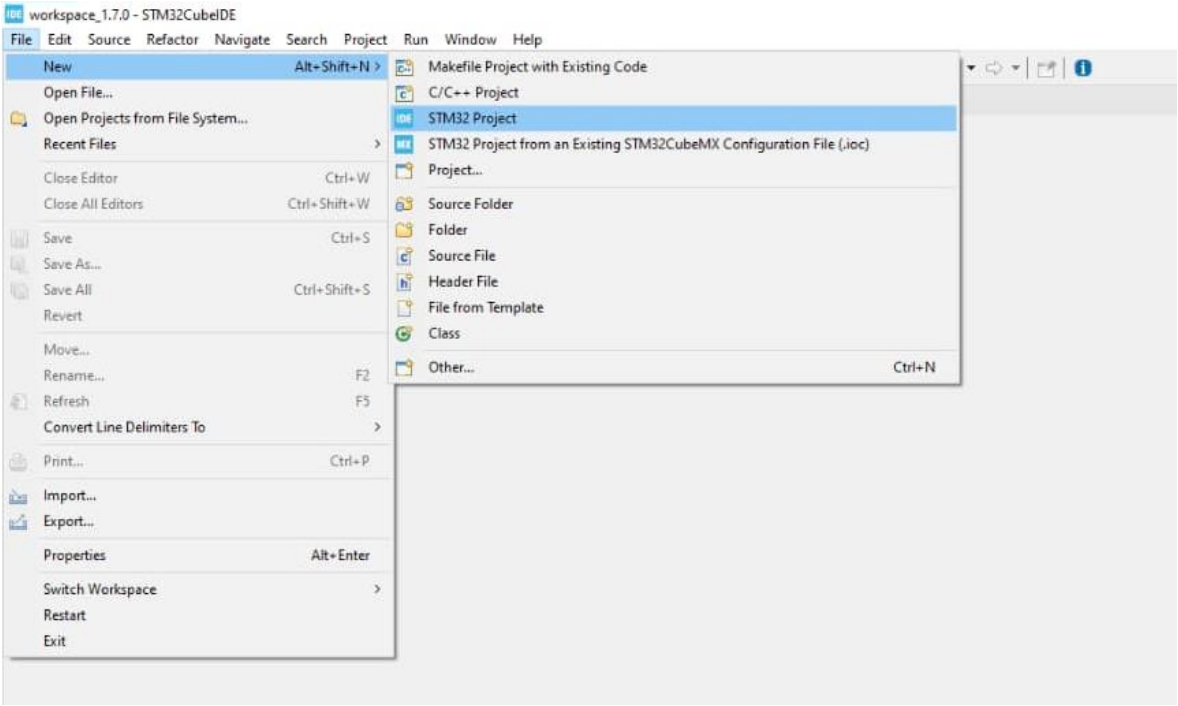
```
while(1)
{
  i = i++
}
```

4.3. STM32CubeIDE'nin Kurulumu ve PLC Projesini Oluşturma

Bu kısımda tasarımı gerçekleştirilen PLC'nin STM32CubeIDE bütünleşik geliştirme ortamı kullanılarak gerçekleştirilmesi için gerekli işlemler sırasıyla anlatılmaktadır.

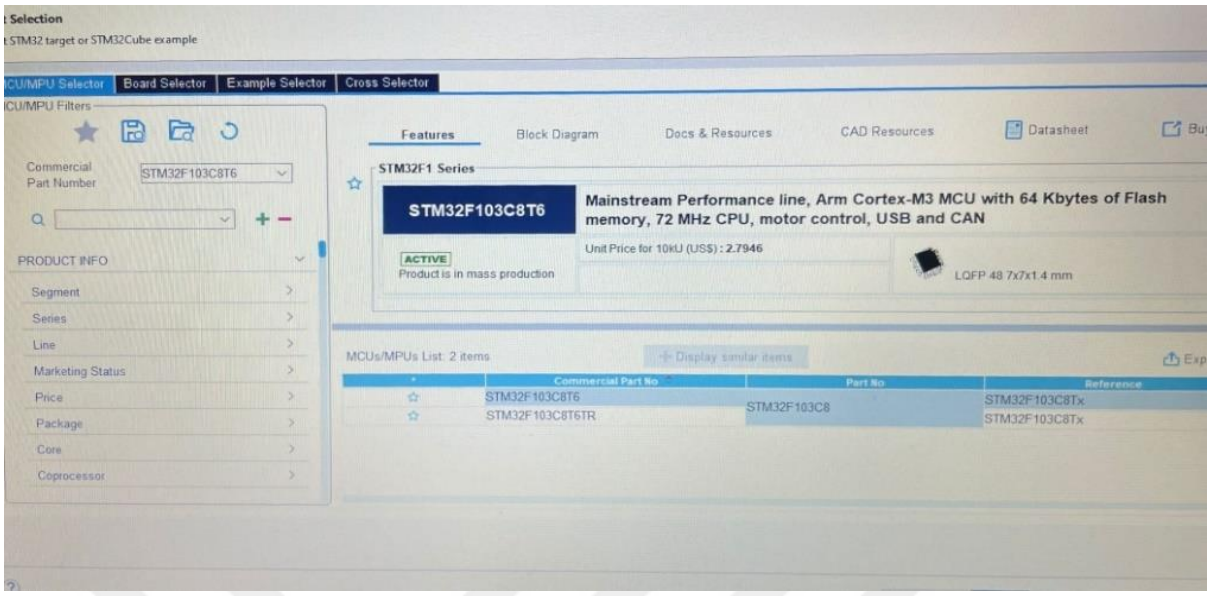
1. STM32CubeIDE yazılımı https://www.st.com/en/development-tools/stm32cubeide.html internet sitesinden indirilerek kurulumu gerçekleştirilmiştir. Kurulum tamamlandıktan sonra programda proje oluşturabilmek için gerekli adımlar tamamlanmalıdır. Bu adımlar mikrodenetleyici kartının seçimi, programa isim verme, kart ayarları, kod oluşturma, kod derleme alanı, debug alanı basamaklarını içermektedir.

2. STM32CubeIDE geliştirme ortamının indirme işlemi gerçekleştirildikten sonra yapılan PLC tasarımının projesi için yeni bir çalışma alanı oluşturulmuştur. Bu işlem için Şekil 4.2'de görülen proje oluşturma basamakları takip edilmelidir. Burada *File* > *New* > *STM32Project*'e basarak proje oluşturma işlemi gerçekleştirilir.



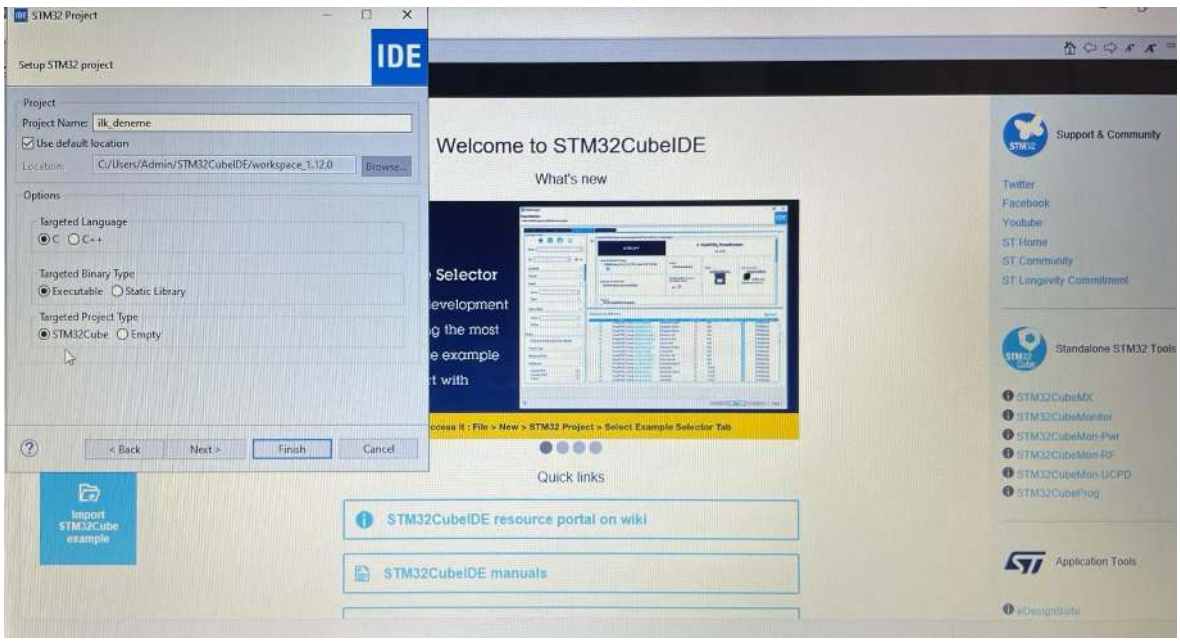
Şekil 4.2. STM32CubeIDE programında proje oluşturma

3. *STM32 Project* kısmına tıklandıktan sonra ekranda Şekil 4.3'te görüldüğü gibi *Target Selection* bölümü açılır. Sayfanın sol üst kösesinde *MCU/MPU Selector – Board Selector – Example Selector – Cross Selector* gibi sekmeler yer almaktadır. Tez konusunda belirtilen STM32F103C8 mikrodenetleyici kartının seçilebilmesi için *MCU/MPU Selector* sekmesinde yer alan *Commercial Part Number* alanına belirlenmiş olunan STM32F103C8T6 mikrodenetleyici kartının adı yazılır ve bu sayede eşleşme gerçekleştirilir.



Şekil 4.3. STM32CubeIDE programında mikrodenetleyici kartının seçilmesi

4. STM32F103C8 mikrodenetleyici kartı seçildikten sonra proje adının belirlenmesi için Şekil 4.4'te görüldüğü gibi bir sekme açılır. Burada *Project Name* bölümünde proje ismi belirlendikten sonra STM32F103C8 mikrodenetleyici kartının (blue pill kartı) bilgisayara bağlanabilmesi için Şekil 4.5'te görüldüğü gibi bir dönüştürücü seçilir ve *Finish* butonuna basılır.



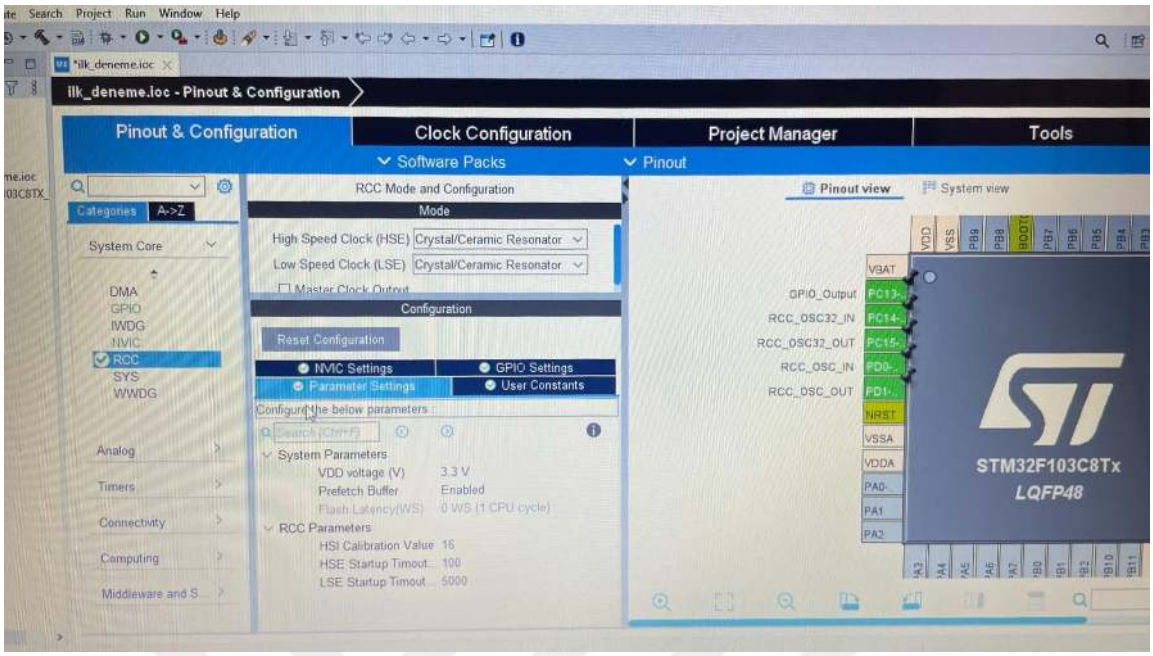
Şekil 4.4. Projeye isim verme işlemi

Şekil 4.5. Bilgisayar bağlantısı için dönüştürücü seçimi

5. *Finish* butonuna bastıktan sonra mikrodenetleyici kartına ait ayarlamaların (Pin ayarlamalarının, Clock ayarlamalarının ve proje için gerekli olabilecek genel ayarlamaların) yapılması gerekir:

- Input/Output (Giriş-Çıkış),
- System Core (DMA-GPIO-IWDG-RCC-SYS-WWDG),
- Analog (ADC1-ADC2), Timers (RTC-TIM1-TIM2-TIM3-TIM4),
- Connectivity (CAN-I2C1- I2C2-SPI1-SPI2-USART1- USART2- USART3-USB),
- Computing (CRC) ve Middleware (FATFS-FREERTOS-USB_DEVICE)

olarak ifade edilen ayarlarının hepsi ilgili olan bu alanda seçilip gerçekleştirilir. STM32F103C8T6 mikrodenetleyici kartına ayarlamalarının yapılabilmesi için Şekil 4.6'da görüldüğü gibi bir sayfa açılır.



Şekil 4.6. Pin konfigürasyon ayarlamaları

Açılan bu sayfada:

1. Pinout & Configuration

2. Clock Configuration

3. Project Manager

4. Tools

gibi kutucuklar yer almaktadır. Bu bölümler şu durumları ifade etmektedir:

1. Pinout & Configuration Bölümü: Bu bölüm pin ayarlaması için ve kristal osilatörün yerinin belirlenmesi için gerekmektedir. Bunun için Şekil 4.6'da sağ tarafta yer alan pinoutview ve system core sekmelerinde işlemler yapılır. Bu işlem basamakları şöyledir:

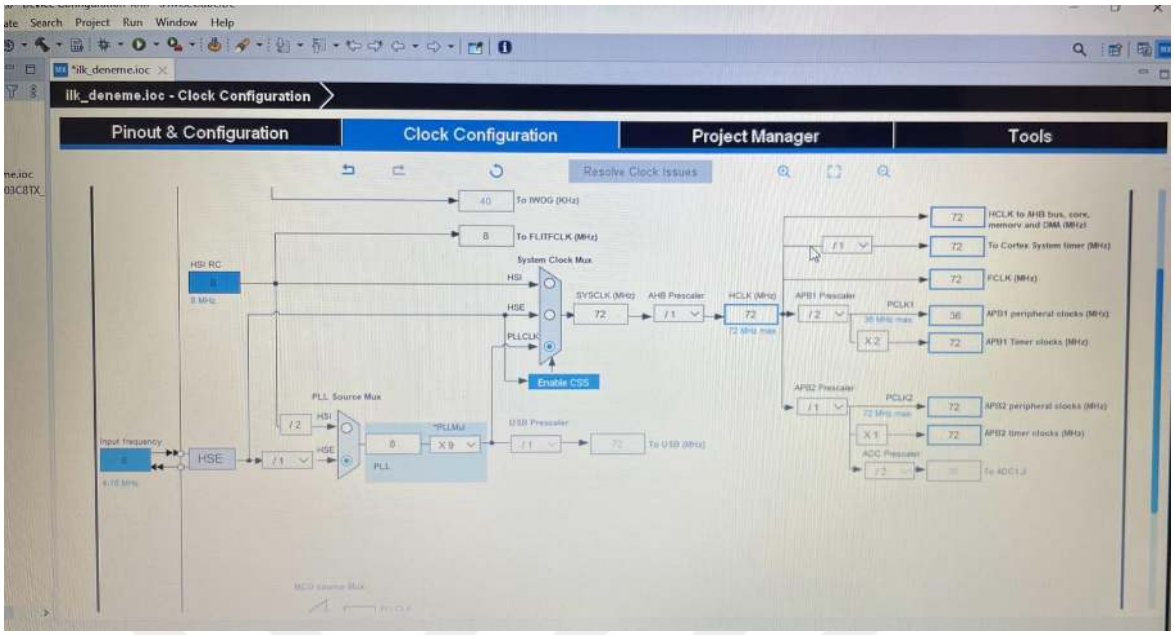
1. Pinout view kısmından pin ayarlamaları

   1. Adım: PC13 pini GPIO_Output ledin bağlandığı kısım ayarlanır, PC13, genellikle birçok STM32 mikrodenetleyicisinde bulunan bir GPIO pinidir. Bu pin, genellikle LED'leri kontrol etmek için kullanılır.

2. Adım: Harici osilatör ve dahili osilatör seçimi yapılır. Harici bir osilatör kullanmak, hassas zamanlama veya yüksek hızlı işlemler gerektiren uygulamalarda, mikrodenetleyicinin dahili osilatörüne kıyasla daha doğru ve istikrarlı saat sinyalleri sağlayabilir. Bu kapsamda harici osilatör seçilerek mikrodenetleyicinin harici osilatör için ayarlama yapılmıştır. Bu işlem için mikrodenetleyicisinin PD0 ve PD1 olarak ifade edilen GPIO pinleri arayüz olarak kullanılır. Bu işlem RCC (Reset ve Clock Control - Sıfırla ve Saat Kontrolü) modülüyle ilişkilendirilirler ve STM32CubeIDE programında PD0 (RCC_OSC_IN) harici osilatör için giriş pini iken, ve PD1(RCC_OSC_OUT) çıkış pini olarak kullanılır. Benzer şekilde PC14 ve PC15 pinleri de harici 32.768 kHz kristal osilatörü için kullanılır. PC14, RCC_OSC32_IN (harici 32.768 kHz osilatör giriş pini) olarak işlev görürken, PC15, RCC_OSC32_OUT (harici 32.768 kHz osilatör çıkış pini) olarak işlev görür. STM32CubeIDE programında PC14(RCC_OSC32_IN) ve PC15(RCC_OSC32_OUT) olarak seçilir.

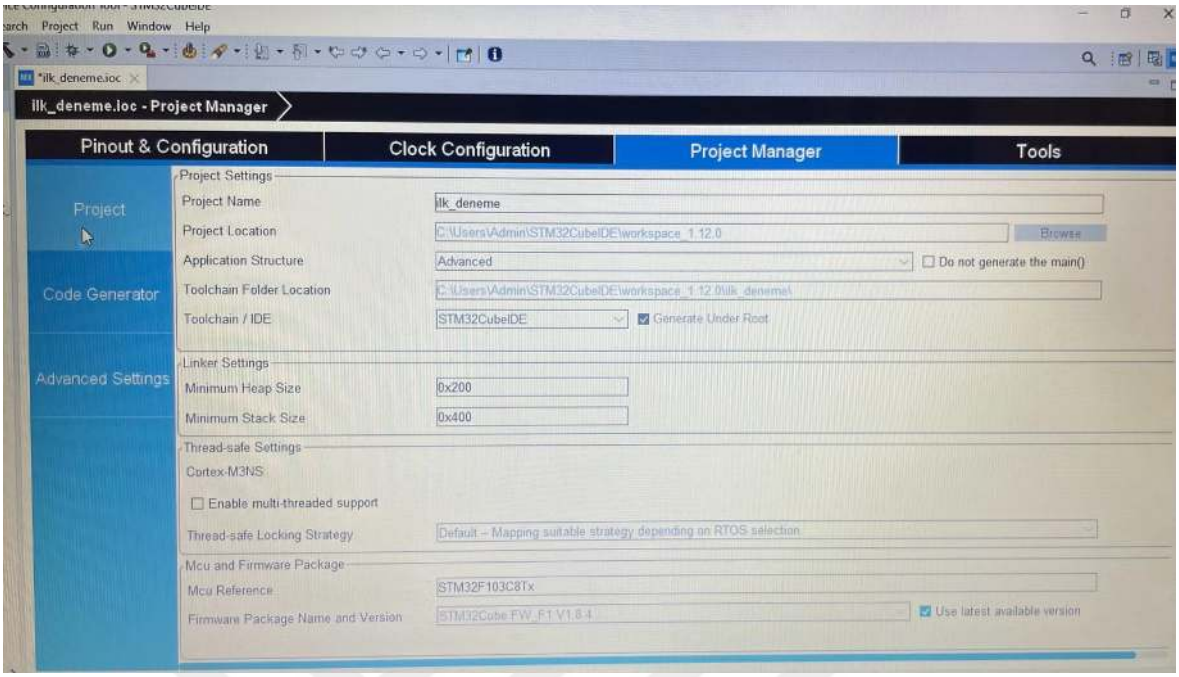
2. *System Core* sekmesinden sistemin ayarlarını kontrol etmek için RCC ayarları yapılır. Burada *Mode* kısmında yer alan *High Spead Clock* (HSE) *ve Low Speed Clock* (LSE) crystal/ceramic resonator seçilir ve bu sayede kristal osilatörün yeri otomatik olarak belirlenir. *SYS* bölümünden *debug serial wire* seçilir. Bu sayede kodları test edip hata ayıklama işlemi gerçekleştirilir.

2. Clock Configuration Bölümü: Bu bölümde işlemcinin çalışma hızını belirlemek için clock ayarları yapılır. Şekil 4.7'de STM32F103C8T6 mikrodenetleyici kartının Clock ayarları görülmektedir. Kullanılan STM32F103C8T6 mikrodenetleyicisi 72 MHz'de çalışabildiği için çalışma hızı 72 MHz olarak belirlenmiştir.



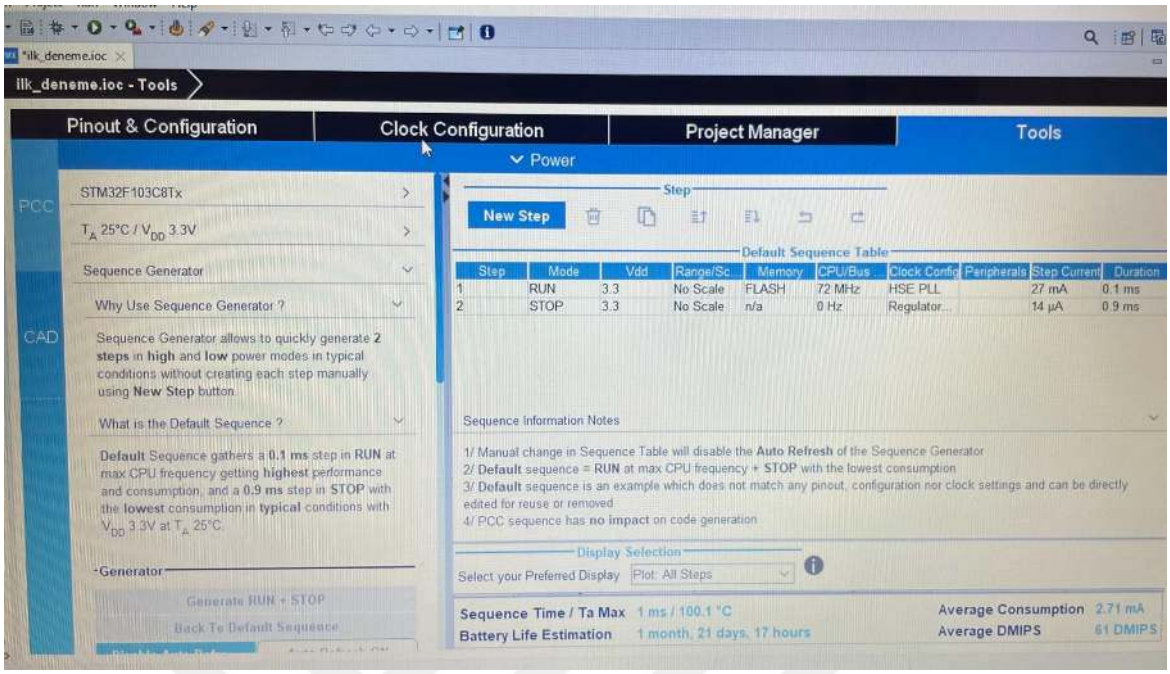
Şekil 4.7. Clock konfigürasyon ayarlamaları

3. Project Manager Bölümü: Bu bölüm, kullanıcıların projelerini oluşturmasına, yönetmesine ve yapılandırmasına yardımcı olan bir araçtır. Şekil 4.8'de Project Manager ayarlamaları görülmektedir. Bu kısımda yeni bir proje oluşturma, mevcut projeleri açma, projelerin yapılandırma parametrelerini ayarlama, kütüphaneleri ekleme veya yönetme ve projeyi derleme işlemleri gerçekleştirilir.



Şekil 4.8. Project Manager ayarlamaları

4. Tools Bölümü: Bu bölüm, kullanıcının geliştirme sürecini kolaylaştıran çeşitli ek fonksiyonları ifade eder. Bu araçlar, kod yazma yardımcıları, otomatik tamamlama özellikleri, kod analizi araçları, derleme ayarları, hata ayıklama araçları, performans analizi araçları ve diğer geliştirme yardımcıları gibi çeşitli özellikleri içerir. Şekil 4.9'da görüldüğü üzere proje kapsamında nelerin belirlendiği, projenin ismi, genel ayarları ve önerilen ayarların ne olduğu özet bilgi olarak kullanıcılara sunulmaktadır.



Şekil 4.9. Tools ayarlamaları

7. Kod oluşturma: Önceki kısımlarda belirtilen temel ayarlamalar yapıldıktan sonra son işlem basamağı kod oluşturmadır. Burada kod oluşturma, kod derleme alanı ve debug alanı işlemleri yapılıp böylelikle hedef projenin çalışması sağlanır. Bahsedilen tüm ayarlamalar yapıldıktan sonra STM32CubeIDE programında sol üst bölümünde yer alan *Project* kısmına gelip *Generate Code* butonuna tıklanır. Buraya tıklandıktan sonra Şekil 4.10'da görülen kaynak dosyaları ile birlikte proje sayfaları açılır.

Şekil 4.10. Proje oluşturulduktan sonra açılan kaynak sayfalarının görüntüsü

Burada:

main.c olarak adlandırılan kaynak dosyasında bu tez çalışması için gerekli temel yazılımı ifade eden temel fonksiyonlar dahil olmak üzere tüm fonksiyonları tanımları yer almaktadır.

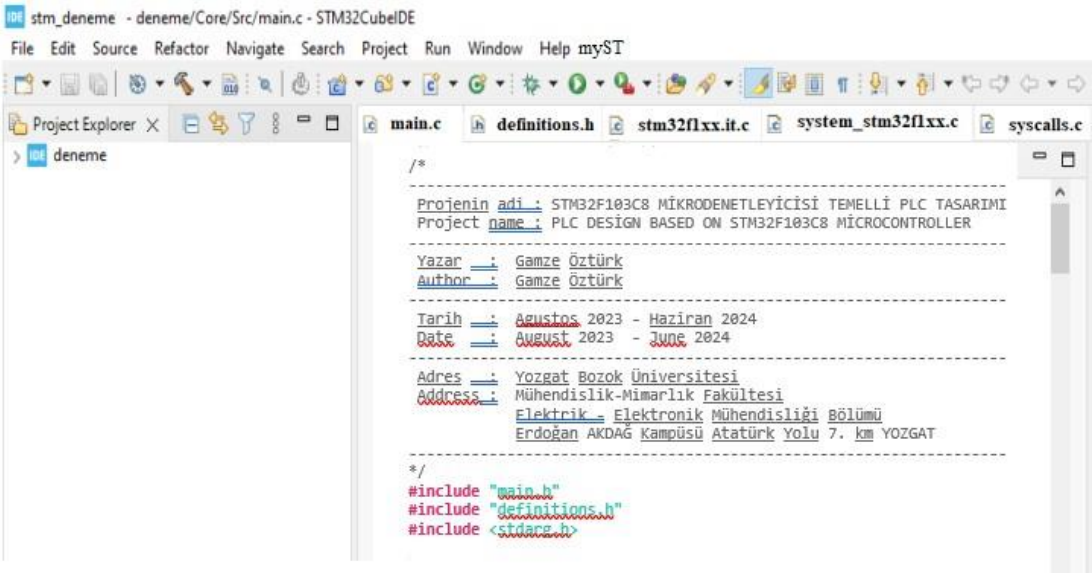
definitions.h olarak ifade edilen kaynak dosyasında proje kapsamında kullanılan fonksiyonların karakterlerinin tanımlaması işlemi yapılmıştır.

stm32f1xx_hal_gpio.c olarak ifade edilen kaynak dosyasında program içine gömülü olan ve gerçekleştirilen tez çalışması için büyük destek sağlayan otomatik kod oluşturulmasına yardımcı olan bir Hal kütüphanesi yer almaktadır.

stm32fxx_it.c olarak ifade edilen kaynak dosyasında ise zaman gecikmesi elde etmek için kesme (interrupt) çalışması yapılmıştır.

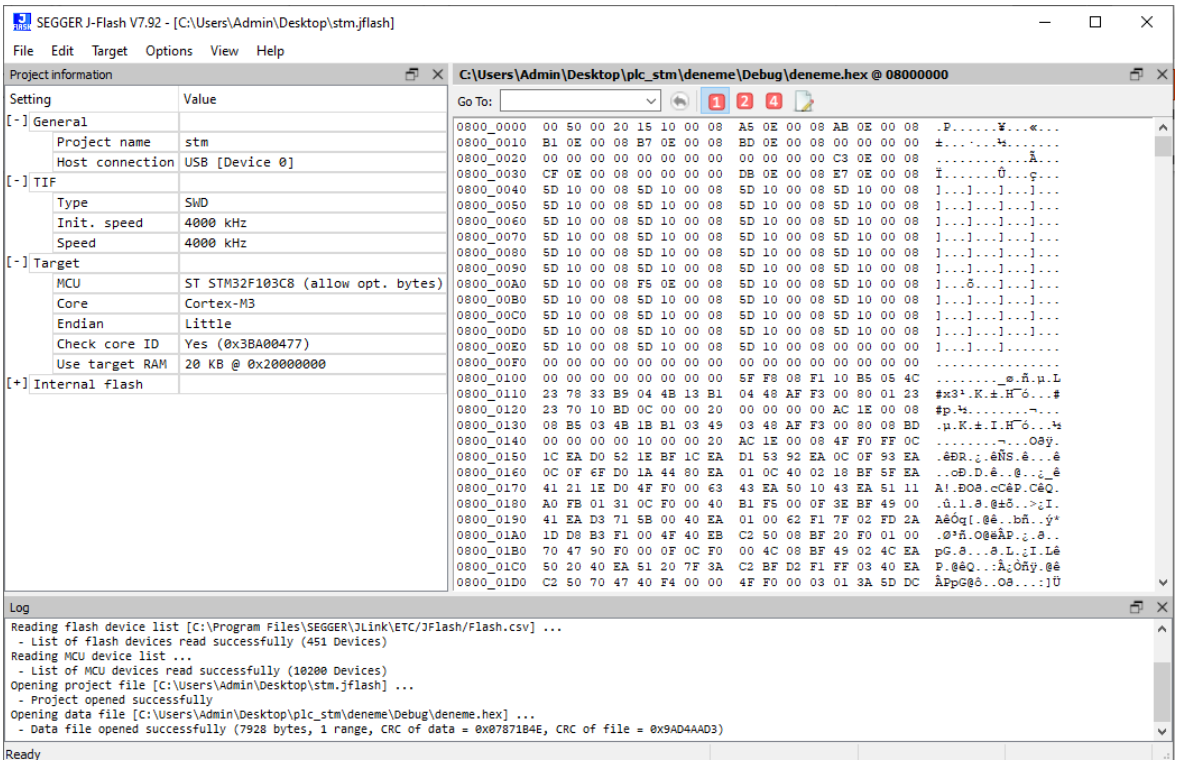
Aynı zamanda sistemin çalışmasına destek sağlayan diğer tüm kaynak dosyaları otomatik olarak proje sayfasında yer almıştır.

8. Şekil 4.10'da görülen tüm kaynak dosyaları seçilip Enter tuşuna basıldıktan sonra Şekil 4.11'de sağ tarafta görüldüğü gibi tüm proje dosyaları açılır. **main.c** kaynak dosyasına tıklanarak STM32F103C8 Mikrodenetleyicisi Temelli PLC Tasarımlı tez çalışması gerçekleştirilmiştir.



Şekil 4.11. Projeye ait tüm proje kaynak dosyalarının ekran görüntüsü

9. Son işlem olarak geliştirilen programı kullanılan bilgisayardan STM32F103C8T6 mikrodenetleyicisine yüklemek için SEGGER Microcontroller Systems tarafından geliştirilen bir yazılım aracı (https://www.segger.com/products/debug-probes/j-link/technology/flash-download/) kullanılır. Bu yazılım, mikrodenetleyicilere yerleştirilen programları ve veri dosyalarını hızlı ve güvenilir bir şekilde yazmak veya silmek için kullanılır. Şekil 4.12'de bu programın ekran görüntüsü mecvuttur.



Şekil 4.12. SEGGER J-Flash programının ekran görüntüsü

4.4. Tasarlanan PLC'nin Temel Yazılımı

Bir PLC'de 1- girişleri güncelleme, 2- kullanıcı programını çalıştırma, 3- çıkışları güncelleme işlemleri sonsuz bir döngü içinde sürekli olarak gerçekleştirilmektedir. Bu sebeple bu işlemler bir PLC döngüsü (PLC cycle) olarak isimlendirilir. Program sonsuz döngüye girmeden önce başlangıç şartlarının ilk PLC taramasında düzenlenmesi gerekir. Ana programda yer alan HAL_Init(), SystemClock_Config(), MX_GPIO_Init(), MX_TIM1_Init(), ve USART2_setup() isimli fonksiyonlar bu amaçla kullanılmaktadır.

Temel yazılımda ana programda yer alan get_inputs() ve send_outputs() isimli iki fonksiyon vardır. get_inputs() fonksiyonu girişleri ve send_outputs() fonksiyonu da

çıkışları güncellemek amacıyla kullanılır. Kullanıcı programı, get_inputs() ve send_outputs() fonksiyonları arasına yerleştirilen bir ‘while’ döngüsü içerisinde yer alır. “get_inputs()” fonksiyonu ‘get_inputs’ ismi ile “send_outputs()” fonksiyonu da ‘send_outputs’ ismi ile projeye eklenmiştir. Bu fonksiyonların içerikleri Ek-1’de mevcuttur. get_inputs() fonksiyonu çalıştığında STM32F103C8 mikrodenetleyicisinin dijital giriş olarak tanımlanmış olan PB4, PB3, PA15, PA6, PA11, PA10, PA9, PA8 pinlerinin 0 veya 1 olan konumları okunarak sonlu darbe tepkisi (FIR) filtreleri ile filtrelendikten sonra bu sinyallerin konumları sırasıyla I0.7, I0.6, I0.5, I0.4, I0.3, I0.2, I0.1 ve I0.0 olarak isimlendirilmiş olan giriş değişkenlerine kaydedilmektedir. send_outputs() fonksiyonu çalıştığında ise Q0.7, Q0.6, Q0.5, Q0.4, Q0.3, Q0.2, Q0.1 ve Q0.0 olarak isimlendirilmiş olan çıkış değişkenlerinin durumları dijital çıkış olarak kullanılan röleleri sürmek için STM32F103C8 mikrodenetleyicisinin dijital çıkış olarak tanımlanmış olan PA4, PA5, PA6, PA7, PB0, PB1, PB10, PB11 isimli pinlerine kaydedilmektedir. Temel yazılımda bu fonksiyonlara ek olarak, diğer tüm fonksiyonlarda sıklıkla kullanılan değişkenlerin tanımlarının yapıldığı tanımlama fonksiyonları da mevcuttur. Bu tanımlamalar ‘definitions.h’ isimli bir kaynak dosyasına kaydedilmiş ve projeye eklenmiştir.

4.4.1 Temel Yazılım ve Tanımlama Fonksiyonları

Yazılımın temelini oluşturan; giriş/çıkış birimleri, zamanlayıcılar (timers), işaretçiler (markers), durum bitleri, oluşturulan fonksiyonlar birer değişken olarak program hafızasında yerleşmiştir. Bunlar 16 bitlik hafıza yapıları olarak tüm bitleri ayrı ayrı tanımlanacak şekilde program hafızasında yerleşmiştir. Şekil 4.13’te 16 bitlik hafıza yapısı ve bu yapıdaki kaydedici tanımlamaları görülmektedir. Benzer yapıdaki tüm hafıza yapıları Ek-1’de mevcuttur.

```
union R16bit
{
        unsigned int bits;
        struct {
                unsigned n0:1;
                unsigned n1:1;      union R16bit INPUTS;
                unsigned n2:1;      union R16bit TIMERS;
                unsigned n3:1;      union R16bit TEMPS;
                unsigned n4:1;      union R16bit OUTPUTS;
                unsigned n5:1;      union R16bit MARKER0;
                unsigned n6:1;      union R16bit MARKER1;
                unsigned n7:1;      union R16bit MARKER2;
                unsigned n8:1;      union R16bit MARKER3;
                unsigned n9:1;      union R16bit REDS;
                unsigned n10:1;     union R16bit FEDS;
                unsigned n11:1;     union R16bit LATCHS;
                unsigned n12:1;     union R16bit SRFFS;
                unsigned n13:1;
                unsigned n14:1;
                unsigned n15:1;
        };
};
```

Şekil 4.13. 16 bitlik hafıza yapısı ve tanımlamaları

Bu 16 bitlik hafıza ‘definitions.h’ isimli kaynak dosyasında ‘union’ olarak tanımlanmış ve program hafızasına kaydedilmiştir. Yine bu dosyada 16 bitlik değişkenlerin tüm bitleri ayrı ayrı tanımlanmıştır. ‘definitions.h’ dosyası Ek-1’de verilmiştir. Şekil 4.14’te örnek bazı bit tanımlamaları görülmektedir.

#define Q00 OUTPUTS.n0	#define I00 INPUTS.n0
#define Q01 OUTPUTS.n1	#define I01 INPUTS.n1
#define Q02 OUTPUTS.n2	#define I02 INPUTS.n2
#define Q03 OUTPUTS.n3	#define I03 INPUTS.n3
#define Q04 OUTPUTS.n4	#define I04 INPUTS.n4
#define Q05 OUTPUTS.n5	#define I05 INPUTS.n5
#define Q06 OUTPUTS.n6	#define I06 INPUTS.n6
#define Q07 OUTPUTS.n7	#define I07 INPUTS.n7
#define Q10 OUTPUTS.n8	#define I10 INPUTS.n8
#define Q11 OUTPUTS.n9	#define I11 INPUTS.n9
#define Q12 OUTPUTS.n10	#define I12 INPUTS.n10
#define Q13 OUTPUTS.n11	#define I13 INPUTS.n11
#define Q14 OUTPUTS.n12	#define I14 INPUTS.n12
#define Q15 OUTPUTS.n13	#define I15 INPUTS.n13
#define Q16 OUTPUTS.n14	#define I16 INPUTS.n14
#define Q17 OUTPUTS.n15	#define I17 INPUTS.n15
#define OUTPUT OUTPUTS.bits	#define INPUT INPUTS.bits
#define M00 MARKER0.n0	#define M20 MARKER1.n0
#define M01 MARKER0.n1	#define M21 MARKER1.n1
#define M02 MARKER0.n2	#define M22 MARKER1.n2
#define M03 MARKER0.n3	#define M23 MARKER1.n3
#define M04 MARKER0.n4	#define M24 MARKER1.n4
#define M05 MARKER0.n5	#define M25 MARKER1.n5
#define M06 MARKER0.n6	#define M26 MARKER1.n6
#define M07 MARKER0.n7	#define M27 MARKER1.n7
#define M10 MARKER0.n8	#define M30 MARKER1.n8
#define M11 MARKER0.n9	#define M31 MARKER1.n9
#define M12 MARKER0.n10	#define M32 MARKER1.n10
#define M13 MARKER0.n11	#define M33 MARKER1.n11
#define M14 MARKER0.n12	#define M34 MARKER1.n12
#define M15 MARKER0.n13	#define M35 MARKER1.n13
#define M16 MARKER0.n14	#define M36 MARKER1.n14
#define M17 MARKER0.n15	#define M37 MARKER1.n15
#define M0 MARKER0.bits	#define M1 MARKER1.bits

Şekil 4.14. 16 bitlik hafıza yapılarına ait örnek bit tanımlamaları

Bu bit tanımlamalarına ek olarak, fonksiyonların çalışmasında anlık bilgilerin tutulduğu 16 bitlik veya 16 bitlik 64'lü gruplar şeklinde değişkenler de mevcuttur. Program hafızasında yer alan bu değişkenlerin tanımlanması Şekil 4.15'de görülmektedir ve benzer kaydedici ve değişkenlerin tanımlandığı kaynak dosyası Ek-1'de verilmiştir.

```
#define TEMP TEMPS.n0
#define LOGIC0 0x0000
#define LOGIC1 0x0001
unsigned int TEMP_REG;
unsigned int CV[64];
_Bool CQ[64];
_Bool QU[64];
_Bool QD[64];
_Bool CURED[64];
_Bool CDRED[64];
unsigned int TON[64];
_Bool TONQ[64];
_Bool TONRED[64];
unsigned int TOF[64];
_Bool TOFQ[64];
_Bool TOFRED[64];
```

Şekil 4.15. Durum bitlerinin tanımlanması

Burada TEMP_REG geçici bilgi tutan 16 bitlik bir değişkendir. CV, CQ, QU, QD, CURED ve CDRED sayıcılara ait; TON, TONQ ve TONRED düz zaman rölesine ait; TOF, TOFQ ve TOFRED ise ters zaman rölesine ait Bool türü değişkenleri ifade etmektedir.

Temel yazılımda yer alan “get_inputs(void)” ve “send_outputs(void)” fonksiyonları Şekil 4.16’da görüldüğü gibi tanımlanmıştır.

```
unsigned int get_inputs (void);
unsigned int send_outputs (void);
```

Şekil 4.16. get_inputs() ve send_outputs() fonksiyonlarının tanımlanması

4.4.2. Ana Program

Bu projede yazılımın ana programı ‘int main (void)’dir. Portların giriş/çıkış tanımlamaları ile zamanlayıcı, zamanlayıcı kesme, alternatif fonksiyonlar burada tanımlanmaktadır. ‘get_inputs ()’ ve ‘send_outputs ( )’ fonksiyonları burada kullanılır. Kullanıcı esas PLC kullanıcı programını ‘get_inputs ()’ ve ‘send_outputs ()’ fonksiyonları arasındaki kısma yazar. Şekil 4.17’de ana fonksiyon ‘int main (void)’ görülmektedir. Ayrıca zamanlayıcı kesmesi **stm32f1xx_it.c** kaynak dosyasında yer almakta olup içeriği Ek-1’de mevcuttur.

```
#include "main.h"
#include "definitions.h"
int main(void)
{
        HAL_Init();
        SystemClock_Config();
        MX_GPIO_Init();
        MX_TIM1_Init();
        USART2_setup();
while (1)
        {
        TIMER = TIMER_Val;
//__________Timerlar için kullanılan referans zamanlama sinyalleri:
        re_T10ms = re_RTS (T_10ms, 61);
        re_T100ms = re_RTS (T_100ms, 62);
        re_T1s = re_RTS (T_1s, 63);
        get_inputs();
//-------------- Kullanıcı Programı Burada Başlar------------------
//-------------- Kullanıcı Programı Burada Biter--------------------
        send_outputs();
        }
 }
```

Şekil 4.17. Ana fonksiyon ‘int main (void)’

4.4.2.1. PLC ve GPIO Bağlantısı ve Yazılımı

Bu kısımda STM32F103C8 mikrodenetleyicisinin PLC dijital giriş ve dijital çıkış kartlarına bağlanırken yapılan işlemlerden ve yazılımdan bahsedilmektedir. Bu bağlantılar bir mikrodenetleyicide GPIO (genel amaçlı giriş/çıkış – general purpose input/output) adı verilen dijital sinyal giriş ve çıkışlarını sağlayan pinleriyle gerçekleştirilir. Mikrodenetleyiciler, çeşitli elektronik sistemlerde bilgi alışverişi yapmak veya dış cihazlarla iletişim kurmak için GPIO pinlerini kullanır. GPIO bağlantı noktalarının bağlantı noktası biti, yazılım tarafından ayrı ayrı yapılandırılabilir. STM32'de GPIO adlandırılması genellikle harflerle ifade edilir ve bunlara Port adı verilir.

Bu pinler çeşitli modlarda çalışabilir:

1. **Giriş** (Digital Input): Dış dünyadan dijital bilgi almak için kullanılırlar.
2. **Çıkış** (Digital Output): Mikrodenetleyiciden dış dünyaya dijital bilgi göndermek için kullanılırlar.
3. **Analog:** Bir çevre birimine giriş/çıkış olarak kullanılabilir.
4. **Alternatif Fonksiyon:** Dijital Giriş/Çıkış yerine başka birimin pini olarak kullanılabilir.

Pinlerin durumlarını kontrol etmek için gerekli olan 4 tane bileşen mevcuttur:

1. **Push-Pull:** Push-pull çıkış modunda, bir pin hem yüksek (1) hem de düşük (0) voltaj seviyelerini sağlayabilir. Bu mod, pinin hem yüksek hem de düşük durumda aktif sürücü olduğu anlamına gelir. Yani, pin sinyal gönderirken yüksek bir voltaj seviyesine geçer ve sinyal göndermediğinde düşük bir voltaj seviyesine geçer.

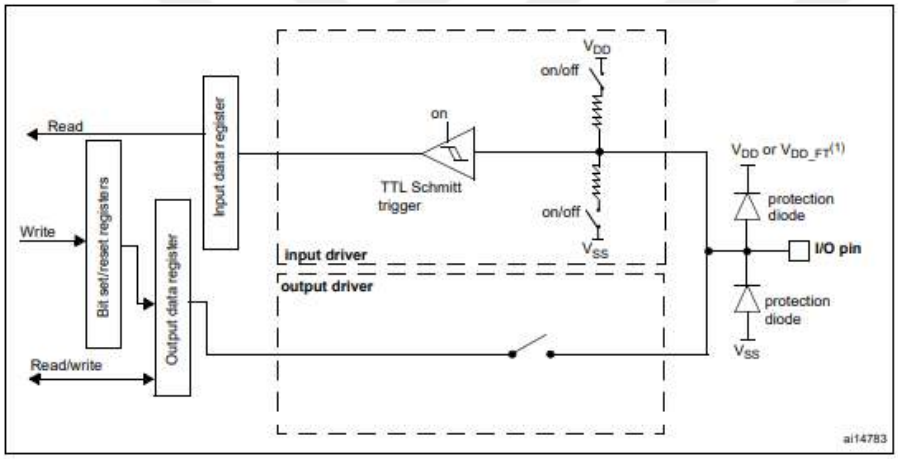
2. **Pull-Up:** Pull-up çıkış modunda, bir pin yüksek (1) voltaj seviyesini sağlar. Ancak, doğrudan bağlı bir cihaz veya devre olmadığında, bir direnç yardımıyla sabit bir yüksek voltaj seviyesinde kalır. Böylece pin, bağlı olan bir cihaz tarafından "düşük" (0) olarak çekilene kadar yüksek seviyede kalır.

3. **Pull-Down:** Pull-down çıkış modunda, bir pin düşük (0) voltaj seviyesini sağlar. Ancak, doğrudan bağlı bir cihaz veya devre olmadığında, bir direnç yardımıyla sabit bir düşük voltaj seviyesinde kalır. Böylece pin, bağlı olan bir cihaz tarafından "yüksek" (1) seviyeye çekilene kadar düşük seviyede kalır.

4. **Open-Drain (Açık Kolektör):** Open-drain çıkış modunda, bir pin yalnızca düşük (0) voltaj seviyesini sağlayabilir. Yüksek (1) seviyeyi sağlamak için dış devre tarafından çekilmelidir. Bu mod, tipik olarak bir pull-up direnciyle kullanılır. Bu, birden fazla cihazın aynı hat üzerinde paylaşılarak çakışmasını önler. I^2C ve SMBus gibi haberleşme protokolleri bu çıkış modunu sıkça kullanır.

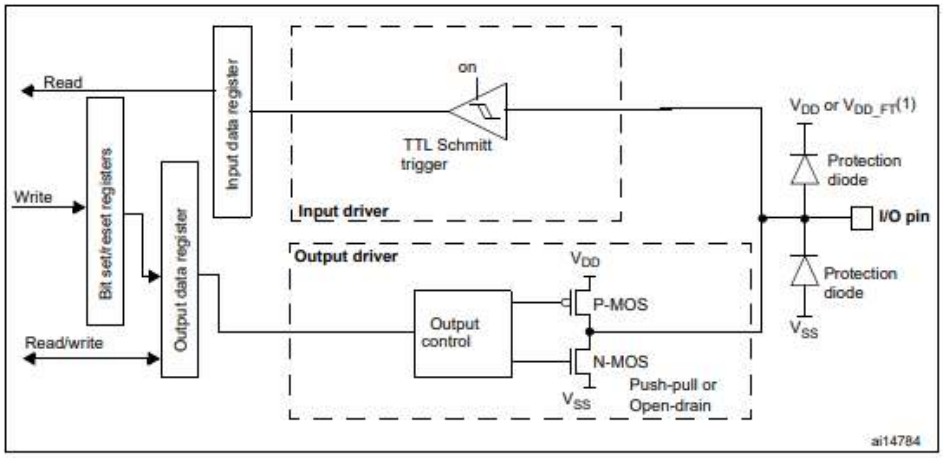
4.4.2.1.1. Kullanılan Pinlerin Modları

Giriş konfigürasyonu: Sistem giriş olarak yani içeriye bilgi almak için çalışır**.** Bu yapı her bir pin için geçerlidir. Şekil 4.18'de giriş konfigürasyonu görülmektedir. Giriş konfigürasyonunda pull up/pull down dirençleri ayarlanabilir. Sisteme bir buton bağlanıldığında içeriden pull up/ pull down dirençleri aktif edilebilir.



Şekil 4.18. Giriş konfigürasyonu (RM0008, 2021)

Çıkış konfigürasyonu: her bir pin için geçerlidir. Şekil 4.19'da çıkış konfigürasyonu görülmektedir.



Şekil 4.19. Çıkış konfigürasyonu (RM0008, 2021)

Şekil 4.19'da görülen çıkış konfigürasyonunda içeri giren sinyal için schmitt trigger (tetikleme girişi) devresi vardır. Aynı zamanda Pull up ve Pull down dirençleri vardır. İçeriden yazılımsal olarak aktive edilen push-pull/open drain ayarlaması yapılabilir. Push-pull, pull-up, pull-down ve open-drain, mikrodenetleyicilerin GPIO (Genel Amaçlı Giriş/Çıkış) pinlerinde kullanılan çıkış modlarıdır. Bunlar, mikrodenetleyici pinlerinin çıkışlarının nasıl davranacağını ve hangi voltaj seviyelerini sağlayacaklarını belirler.

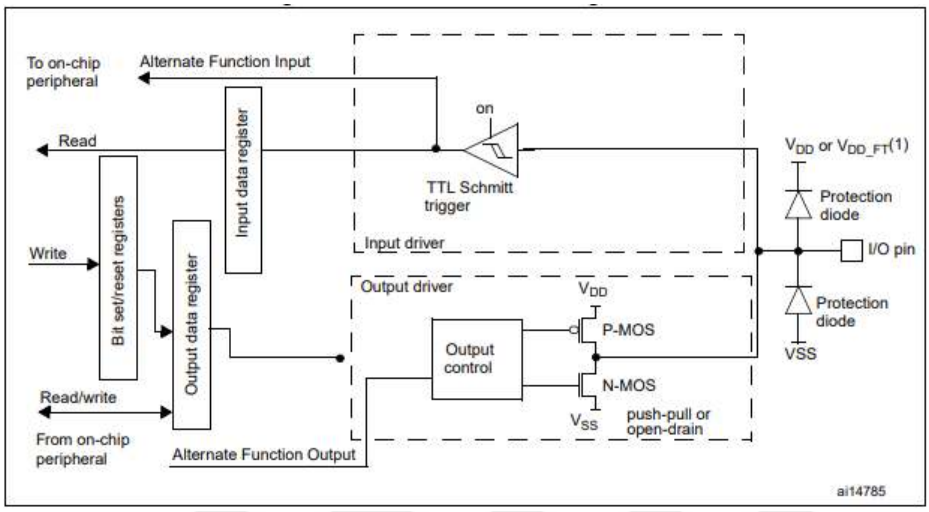
Analog Modu: Bir çevre birimine giriş/çıkış olarak kullanılabilir.

Alternatif Fonksiyon Modu: "Alternatif fonksiyon modu" mikrodenetleyicilerin (STM32 gibi) pinlerinin farklı işlevleri yerine getirebilme yeteneğini ifade eder. Bu, mikrodenetleyicinin donanımının birden fazla amaç için kullanılabilmesine olanak tanır. STM32 mikrodenetleyicilerinde, pinlerin alternatif fonksiyonlarını belirlemek için özel bir kaydedici olan AFR (Alternate Function Registers) bulunur. Bu kaydediciler, bir pinin hangi alternatif işlevi gerçekleştirebileceğini belirler. Örneğin, bir pinin bir GPIO pini olarak mı yoksa bir UART veya SPI arabirimi için mi kullanılacağını belirler.

Alternatif modları kullanmak için genellikle iki adım vardır:

1. Pinin alternatif fonksiyonunu seçmek için ilgili AFR'yi yapılandırmak.
2. Pinin giriş, çıkış veya alternatif mod olmasını sağlayan GPIO yapılandırmasını ayarlamak.

Bu alternatif modlar, mikrodenetleyicinin bir dizi farklı işlevi yerine getirebilmesini sağlar ve sınırlı pin sayısını en verimli şekilde kullanmayı mümkün kılar. Bu, özellikle karmaşık sistemlerde veya çoklu donanım arabirimlerinin kullanıldığı uygulamalarda önemlidir.



Şekil 4.20. Alternatif fonksiyon konfigürasyonu şeması (RM0008, 2021)

Şekil 4.20'deki alternatif fonksiyon konfigürasyonu şemasının yapısı Şekil 4.19'da görülen çıkış konfigürasyonu yapısına benzemektedir. Buna ek olarak alternatif fonksiyon giriş ve çıkış GPIO'larını içermektedir. Bunlar mikrodenetleyicinin alternatif fonksiyonlarını kontrol etmek için kullanılır. Mikrodenetleyicinin pinlerinin hangi alternatif fonksiyonları gerçekleştireceğini belirlemek için özel kaydediciler mevcuttur. Bu bilgiler her mikrodenetleyicide olduğu gibi STM32F103C8 mikrodenetleyicisinin veri sayfasında ifade edilmiştir.

Bu tez çalışmasında RS232 USB-TTL Seri Haberleşme Dönüştürücü Modülü bölümünde ifade edilen haberleşme (seri) biriminin pini olarak alternatif fonksiyon (USART) kullanılmıştır ve bu fonksiyondan ilgili kısımda bahsedilmiştir**.**

4.4.2.1.2. GPIO Ayar Kaydedicileri ve Kontrol Kaydedicileri

GPIO (genel amaçlı giriş/çıkış) pinlerini yapılandırmak ve kontrol etmek için kullanılan kaydediciler, genellikle mikrodenetleyicilerin donanım kaynaklarında bulunur. Bu kaydediciler, GPIO pinlerinin çalışma modunu (giriş veya çıkış), çıkış durumunu, dahili pull-up/pull-down dirençlerini ve diğer özelliklerini yapılandırmak için kullanılan küçük

bir bellek birimi bulunur. STM32 gibi mikrodenetleyicilerde, kaydediciler çeşitli bit alanlarına (bitfields) ayrılarak kontrol edilir. Bu kaydediciler GPIO ayar kaydedicileri ve kontrol kaydedicileri olarak belirtilir (AN4899, 2022). Tablo 4.5'te GPIO ayar kaydedicileri ve kontrol kaydedicileri yer almaktadır. Burada ifade edilen GPIO ayar kaydedicileri mikrodenetleyicilerin GPIO pinlerini yapılandırmak için kullanılır. Bu kaydediciler, belirli bir GPIO pininin başlangıçta nasıl yapılandırılacağını ve çalışma modunu belirlemek için kullanılır. Bu GPIO pin ayarlamaları genellikle bir kez yapılır ve ardından pinin işlevi programın çalışması boyunca sabit kalır. Yani giriş olarak tanımlanmışsa giriş, çıkış olarak tanımlanmışsa çıkış olarak kullanılmaya devam eder.

Tablo 4.5. GPIO ayar kaydedicileri ve kontrol kaydedicileri

GPIO ayar kaydedicileri	GPIO kontrol kaydedicileri
GPIO_MOSDER: Mod ayarları	GPIOx_IDR: Input Data
GPIO_OSPEED: Hız ayarları	GPIOx_ODR: Output Data
GPIO_OTYPE: Çıkış tipi	GPIOx_BSRR: Bit Set/Reset
GPIO_PULLD: Pull Up/Pull Down	GPIOx_BRR: Bit Reset
GPIO_AFRH: Alternatif Fonksiyon	
GPIO_AFRL: Alternatif Fonksiyon	

GPIO kontrol kaydedicileri ilgili pin için kontrol işlemi gerçekleştirir. Örneğin ilgili pin çıkış olarak ayarlanırsa dışarıya 1 veya 0 verir, ilgili pin giriş olarak ayarlanırsa dışarıdan 1 veya 0 bilgisi alır.

Bu projede kullanılan ayar kaydedicilerinin ayrıntıları aşağıda sunulmuştur (AN4899, 2022).

RCC_AHBENR (Reset Clock Control Register)

Mikrodenetleyici üreticileri, özellikle STM32 gibi ARM tabanlı mikrodenetleyicilerde bu tür bir kaydedici sağlarlar. Bu kaydedici, belirli donanım birimini etkinleştirmek veya devre dışı bırakmak için kullanılan bit alanlarını içerir. Güç verildiğinde PLC'nin kontrol edilmesi için kullanılır. Dolayısıyla Clock kaynağını aktif eden kaydedici bu projede kullanılmıştır. Her bir çevre birimin clock sinyalini açma/kapama pini vardır. Kullanılan

çevre birimi A, B, C, D, F portlarından hangisiyse onunla ilgili biti en başta aktif etmek gerekir. Bu aktivasyonun sağlanabilmesi için ayar kaydedicileri kullanılır. Aktif bit 0 yapıldığında o biriminin clock'u yetkisiz (disable - aktif olmaz) durumunda olur, 1 yapıldığında o birimin clock'u yetkilenmiş (enable - aktif) durumunda olur. Örneğin, RCC_AHBENR'deki bir bit, GPIO portlarını etkinleştirmek veya devre dışı bırakmak için kullanılabilir. Bir GPIO portunun kullanılabilmesi için bu bitin ilgili biti RCC_AHBENR kaydedicisinde ayarlanmalıdır. Benzer şekilde, diğer donanım birimleri de (DMA kontrolcüsü, USB kontrolcüsü vb.) bu kaydedici aracılığıyla etkinleştirilebilir veya devre dışı bırakılabilir.

GPIOX_MODER

GPIO portları genellikle GPIO_MODER (GPIO modu kaydedicisi) gibi kaydediciler aracılığıyla kontrol edilir. Bu kaydedici, GPIO pinlerinin modunu belirler (giriş, çıkış, alternatif fonksiyon veya analog). Kaydedicideki her bit, bir pin çiftine karşılık gelir ve bu bitleri set ederek veya temizleyerek pinleri buna göre yapılandırılır. Burada "GPIOX" belirli bir GPIO portunu ifade eder, "X" ise bir port numarası için bir yer tutucu olur (örneğin, GPIOA, GPIOB, GPIOC vb.). "GPIOX_MODER", belirli bir GPIO portunun mod kaydedicisidir.

GPIO_OTYPER

GPIO_OTYPER, GPIO portunun pin çıkış tipini belirler. Her GPIO pinine iki temel çıkış tipi atanabilir: push-pull ve open-drain. Push-pull çıkış tipinde, pin hem yüksek (1) hem de düşük (0) seviyelerde sürücü yeteneğine sahiptir. Open-drain çıkış tipinde ise, pin sadece toprağa bağlı (0) olduğunda sürücü yeteneğine sahiptir. Yüksek seviyede (1), pinde herhangi bir sürücü bulunmaz. Bu durumda, pinin yükseltilmesi genellikle bir pull-up direnciyle gerçekleştirilir. GPIO_OTYPER kaydedicisindeki her bit, bir GPIO pini için belirli bir çıkış tipini temsil eder. Set edilen bit, open-drain çıkış tipini belirtirken resetlenen bit, push-pull çıkış tipini belirtir.

SPEED REGISTER

SPEED REGISTER, bir mikrodenetleyici GPIO pinlerinin hızını ayarlamak için kullanılan bir kaydediciyi ifade eder. Birçok mikrodenetleyici mimarisinde, GPIO pinlerinin çıkış hızı veya sürme hızı, bir "speed register" kullanılarak ayarlanabilir. Bu kaydedici, her bir GPIO pini için ayrı bir bit veya bir grup bit içerir. Bu bitlerin durumu, belirli bir pini düşük (LOW), orta (MEDİUM) veya yüksek (HIGH) hızda sürmek için kullanılır. Dolayısıyla, "speed register" GPIO pinlerinin sürme hızını kontrol eden bir araçtır. Bu kaydedici, elektrik sinyallerinin hızlı veya yavaş hareket etmesini gerektiren örneğin yüksek hızlı iletişim arayüzlerinde veya zaman hassasiyeti olan sistemlerde uygulamalarda kullanışlıdır.

GPIO_PULLD

GPIO_PULLD, GPIO pinlerinin üzerinde bulunan entegre pull-down direncini kontrol etmek için kullanılan bir kaydediciyi ifade eder. Her GPIO pini için bir bit kullanılır. Böylece her pin ayrı ayrı yapılandırılabilir. Bu kaydedici, bir mikrodenetleyicinin donanım belgelerinde belirtilen kaydedicilerden biridir ve GPIO pinlerinin davranışını programlama sürecinde kontrol etmek için kullanılır. Pull-down direnci, bir GPIO pini etkisiz olduğunda (yani, ne yüksek ne de düşük bir mantıksal seviyede olduğunda) pini sabit bir düşük seviyeye çeker. Bu, pini açık veya yüksek empedanslı bir durumdayken düşük seviyede tutar. Böylece pinden gelen girişlerin rastgele veya yanlış okunması önlenir.

Bu projede kullanılan kontrol kaydedicilerinin ayrıntıları aşağıda sunulmuştur (AN4899, 2022).

GPIOx_IDR

GPIOx_IDR, GPIO pinlerinden gelen verileri okumak için kullanılan bir kaydedicinin adıdır. IDR ifadesi, Input Data Register'ı temsil eder. Bu kaydedici, genellikle dış dünyadan gelen verileri okumak veya pinlerin durumunu izlemek için kullanılır. Burada "x", bir GPIO portunun belirli bir harfini veya numarasını temsil eder. Her GPIO pini, bağlı olduğu fiziksel devrelerden gelen sinyalleri okumak için bir IDR kullanır. Bu kaydedici, ilgili pinlerin mevcut durumunu içerir. Örneğin, 8 nolu pinin değerini okumak

için IDR8 bitiyle ilgilenir ve IDR8 bitinde 8 nolu pinin durumu okunur. GPIO_IDR kaydedicisi, her GPIO pini için bir bit içerir ve her bit, ilgili pinin durumunu gösterir. Örneğin, bir bitin 1 olması, ilgili pinin yüksek seviyede olduğunu, 0 olması ise düşük seviyede olduğunu gösterir.

GPIOx_ODR

GPIOx_ODR, GPIO pinlerine çıkış vermek için kullanılan bir kaydedicinin adıdır. ODR ifadesi, Output Data Register'ı temsil eder. GPIO pinlerine dijital sinyal göndermek için kullanılır. Burada "x", bir GPIO portunun belirli bir harfini veya numarasını temsil eder. Her GPIO pini, bağlı olduğu fiziksel devrelere dijital sinyal göndermek için bir ODR kullanır. Örneğin, bir pinin yüksek seviyede mi yoksa düşük seviyede mi olması gerektiğini belirlemek için bu kaydedici kullanılır. ODR11 biti 1 yapıldığında 11 pininde çıkışta 1 görünür. Yani bir bitin 1 olması, ilgili pinin yüksek seviyede olduğunu, 0 olması ise düşük seviyede olduğunu belirtir.

GPIOx_BSRR

GPIOx_BSRR, GPIO portunun "Bit Set/Reset Register"ını ifade eder. Burada "x", bir GPIO portunun belirli bir harfini veya numarasını temsil eder. Örneğin, GPIOA_BSRR veya GPIOB_BSRR gibi. Bu kaydedici, belirli bir GPIO portunun pinlerinin durumunu değiştirmek (set veya reset yapmak) için kullanılır. "Bit Set" işlevi ile belirli bir pinin durumu "1" yapılırken, "Bit Reset" işlevi ile belirli bir pinin durumu "0" yapılır. Her bir GPIO portu için bir BSRR kaydedicisi vardır ve her bit, ilgili pinin durumunu temsil eder. Bu kaydedici, genellikle birden fazla pinin durumunu aynı anda değiştirmek için kullanılır.

GPIOx_BRR

GPIOx_BRR, GPIO portunun "Bit Reset Register"ını ifade eder. Burada "x", bir GPIO portunun belirli bir harfini veya numarasını temsil eder. Örneğin, GPIOA_BRR veya GPIOB_BRR gibi. Bu kaydedici, belirli bir GPIO portunun pinlerinin durumunu sıfırlamak (reset etmek) için kullanılır. Bit Reset işlevi ile belirli bir pinin durumu "0" olarak ayarlanır. Her bir GPIO portu için bir BRR kaydedicisi vardır ve her bit, ilgili pinin

durumunu temsil eder. Bu kaydedici, genellikle belirli bir GPIO portunun tüm pinlerini sıfırlamak için kullanılır. Özellikle, bir GPIO portunun tüm çıkışlarını sıfırlamak için hızlı bir çözüm sağlar.

GPIOx_LCKR

GPIOx_LCKR, GPIO portunun "Lock Register"ını ifade eder. Burada "x", bir GPIO portunun belirli bir harfini veya numarasını temsil eder. Örneğin, GPIOA_LCKR veya GPIOB_LCKR gibi. Bu kaydedici, belirli bir GPIO portunun pinlerinin konfigürasyonunu kilitlemek için kullanılır. GPIO pinlerinin konfigürasyonu, genellikle yazılım tarafından yapılandırılır. Ancak, bazen yanlışlıkla veya istenmeyen durumlarda yapılandırma değiştirilmesini engellemek için bu işlev kullanılır. Her bir GPIO portu için bir LCKR kaydedicisi vardır ve her bit, ilgili pinin kilidinin durumunu temsil eder. Bu kaydedici, genellikle belirli bir GPIO portunun tüm pinlerinin konfigürasyonunu aynı anda kilitlemek veya kilidini açmak için kullanılır.

Programda GPIO sürücüsünün yazılma adımları:

1. Kullanılan portarın ilgili GPIO portlarına ait clock kaynakları aktif edilir.
2. İlgili pinlerin çalışma ayarları gerçekleştirilir.
3. Yön/Fonksiyon giriş, çıkış, ya da alternatif fonksiyonlarından hangisi kullanılacaksa yön ayarlanır.
4. Hız: Hızları belirlenir. (Low, Middle, High)

   Bu adımlar STM32CubeIDE geliştirme programında proje dosyası açma adımları gerçekleştirdikten sonra otomatik kod oluşturularak elde edilir ve Hal kütüphanesi olarak ifade edilen bu dosya içerisinden GPIO için gerekli olan bağlantıların gerçekleştirilmesi için kolaylık sağlanır.

5. Bu kod sayfası içerisinde dirençler pull up/pull down direnci gerekli durumlarda belirlenir. GPIO bağlantıları ayarlamaları yapılırken kullanılır.

Böylelikle bu proje kapsamında kullanılan dijital giriş ve dijital çıkış amacıyla kullanılan GPIO ayarlamaları aşağıda görüldüğü şekilde gerçekleştirilmiştir.

Input (Giriş) GPIO ayarlamaları;

//////////////////////////////// B4 input I0.7

GPIO_InitStruct.Pin = GPIO_PIN_4;

GPIO_InitStruct.Mode = GPIO_MODE_INPUT;

GPIO_InitStruct.Pull = GPIO_NOPULL; // GPIO_PULLDOWN GPIO_PULLUP

HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

//////////////////////////////// B3 input I0.6

GPIO_InitStruct.Pin = GPIO_PIN_3;

GPIO_InitStruct.Mode = GPIO_MODE_INPUT;

GPIO_InitStruct.Pull = GPIO_NOPULL; // GPIO_PULLDOWN GPIO_PULLUP

HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

//////////////////////////////// A15 input I0.5

GPIO_InitStruct.Pin = GPIO_PIN_15;

GPIO_InitStruct.Mode = GPIO_MODE_INPUT;

GPIO_InitStruct.Pull = GPIO_NOPULL; // GPIO_PULLDOWN GPIO_PULLUP

HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

//////////////////////////////// B6 input I0.4

GPIO_InitStruct.Pin = GPIO_PIN_6;

GPIO_InitStruct.Mode = GPIO_MODE_INPUT;

GPIO_InitStruct.Pull = GPIO_NOPULL; // GPIO_PULLDOWN GPIO_PULLUP

HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

//////////////////////////////// A11 input I0.3

GPIO_InitStruct.Pin = GPIO_PIN_11;

GPIO_InitStruct.Mode = GPIO_MODE_INPUT;

GPIO_InitStruct.Pull = GPIO_NOPULL; // GPIO_PULLDOWN GPIO_PULLUP

```
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
//////////////////////////////  A10 input I0.2
GPIO_InitStruct.Pin = GPIO_PIN_10;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;    // GPIO_PULLDOWN   GPIO_PULLUP
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
//////////////////////////////  A9 input I0.1
GPIO_InitStruct.Pin = GPIO_PIN_9;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;   // GPIO_PULLDOWN   GPIO_PULLUP
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
//////////////////////////////  A8 input I0.0
GPIO_InitStruct.Pin = GPIO_PIN_8;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;   // GPIO_PULLDOWN   GPIO_PULLUP
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

Sonuç olarak GPIO giriş pinlerinin yapılandırılması şu şekildedir:

Yukarıda ifade edilen düzenlemelerle B portunun 4. (sırasıyla 3., 15., 6., 11., 10., 9., 8.) pini giriş olarak tanımlanmış, GPIO_MODE_INPUT modunda yapılandırılmış ve bu pin dijital giriş I0.7 (I0.6, I0.5, I0.4, I0.3, I0.2, I0.1, I0.0)'ye bağlanmıştır.

Ayrıca her bir pin için pull-up veya pull-down direnci kullanılacak şekilde ayarlanmıştır.

Output (Çıkış) GPIO ayarlamaları;

```
//////////////////////////////  A4 output Q0.7
GPIO_InitStruct.Pin = GPIO_PIN_4;
```

```
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP; //GPIO_MODE_OUTPUT_OD
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
////////////////////////////////  A5 output Q0.6
GPIO_InitStruct.Pin = GPIO_PIN_5;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP; //GPIO_MODE_OUTPUT_OD
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
////////////////////////////////  A6 output Q0.5
GPIO_InitStruct.Pin = GPIO_PIN_6;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP; //GPIO_MODE_OUTPUT_OD
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
 ////////////////////////////////  A7 output Q0.4
GPIO_InitStruct.Pin = GPIO_PIN_7;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP; //GPIO_MODE_OUTPUT_OD
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
////////////////////////////////  B0 output Q0.3
GPIO_InitStruct.Pin = GPIO_PIN_0;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP; //GPIO_MODE_OUTPUT_OD
```

```
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
//////////////////////////////  B1 output Q0.2
GPIO_InitStruct.Pin = GPIO_PIN_1;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP; //GPIO_MODE_OUTPUT_OD
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
//////////////////////////////  B10 output Q0.1
GPIO_InitStruct.Pin = GPIO_PIN_10;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP; //GPIO_MODE_OUTPUT_OD
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
//////////////////////////////  B11 output Q0.0
GPIO_InitStruct.Pin = GPIO_PIN_11;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP; //GPIO_MODE_OUTPUT_OD
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
```

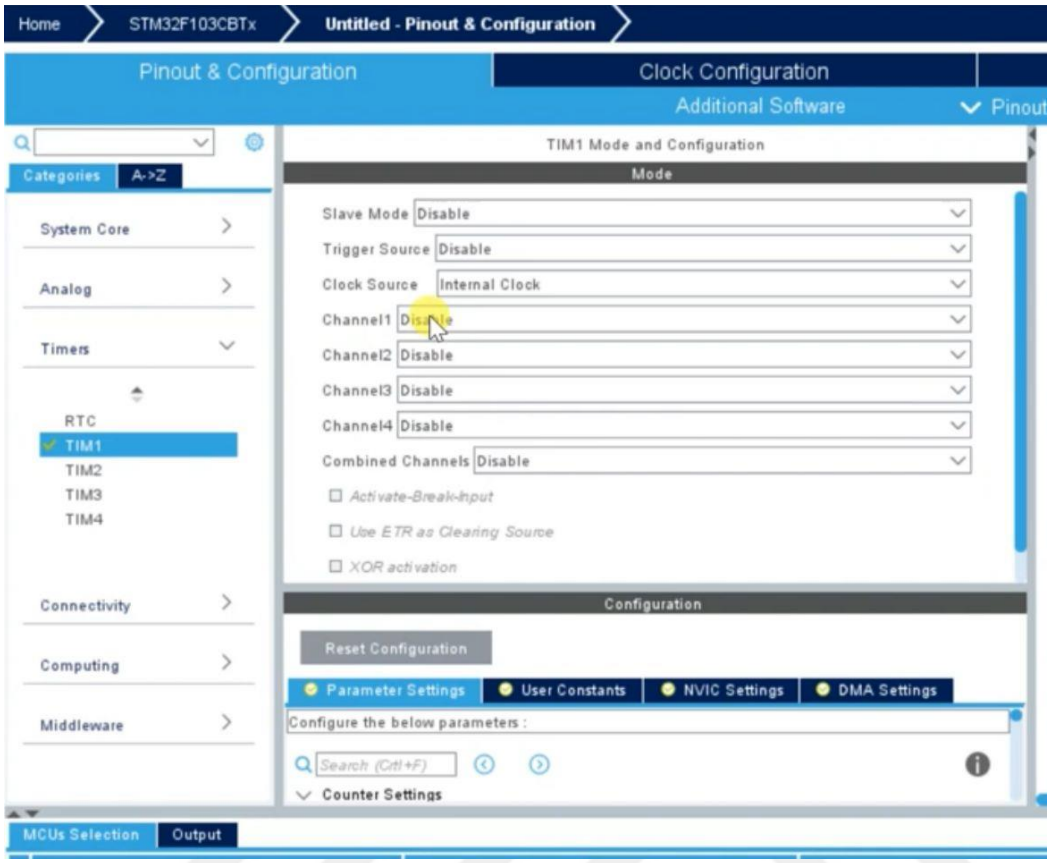
Sonuç olarak GPIO çıkış pinlerinin yapılandırılması şu şekildedir:

Yukarıda ifade edilen düzenlemelerle A portundaki 4. (sırasıyla, 5., 6., 7., 0., 1., 10., 11.) pinin çıkış olarak yapılandırılması ve "Push-Pull" çıkış tipiyle kullanılması sağlanmakta

olup bu pin dijital çıkış Q0.7 (sırasıyla, Q0.6, Q0.5, Q0.4, Q0.3, Q0.2, Q0.1, Q0.0)'ye bağlanmıştır.

4.5. Kesme (Interrupt)

Bu kısımda bu tez çalışması kapsamında gerçekleştirilen PLC projesi kapsamında kullanılan kesme (interrupt) hakkında açıklamalar yer almaktadır. Kesme (interrupt), bir bilgisayar veya mikrodenetleyici tarafından anlık tepki verilmesi gereken olayların meydana gelmesi durumunda normal program akışının geçici olarak durdurularak öncelikli bir işlemin gerçekleştirilmesi için kullanılan bir mekanizmadır. Bu olaylar, harici bir cihazın tetiklenmesi, bir zamanlayıcının/sayacının belirli bir değere ulaşması veya bir harici sinyalin algılanması gibi çeşitli nedenlerden kaynaklanabilir. Kesmeler, sistemlerin gerçek zamanlı veya zaman uyumlu tepkiler vermesini sağlar. Örneğin, bir düğmeye basıldığında bir işlemi tetiklemek veya bir sensörden gelen verileri hızlı bir şekilde işlemek için kesmeler kullanılabilir. Gerçekleştirilen bu PLC projesinde, zamanlayıcı fonksiyonlarında kullanılmak üzere referans zamanlama sinyallerinin üretilmesi amacıyla Timer 1 kesmesi kullanımıştır. Bu amaçla gerekli ayarlamalar STM32CubeIDE programında Şekil 4.21'de görüldüğü gibi gerçekleştirilmiştir. Timer 1 (TIM1) bir donanım sayıcısı olduğundan çalıştırılmak üzere ayarlandığında normal program taraması gerçekleşirken arka planda çalışmaya devam eder.



Şekil 4.21. STM32CubeIDE programında Timer 1 ile ilgili ayarlamaların yapılması

Timers bölümünde gerçek zaman saati olan RTC, TIM1, TIM2, TIM3 VE TIM4 sayıcıları mevcuttur. Bu tez çalışmasında Timer 1 (TIM1) kullanılmıştır. Şekil 4.21'de görülen ekranda TIM1'in üzerine tıklandığı zaman programda TIM1 modu ile ilgili bilgiler yer almaktadır. Bunlar:

- **Slave mode:** harici bir kaynaktan clock sinyali alarak yani bir tetikleme sinyali alınarak işlemler yaptırılabilir.
- **Trigger source:** harici bir kaynaktan bir sinyal (kesme sinyali, kare dalga sinyali vs.) alınıp ölçülebilir. Böylelikle bu sinyallerin her birinde palse oluşur.
- **Clock source**: dahili (internal) clock ya da harici (external) clock (etr2) seçiminin yapıldığı bölümdür.

  Dahili clock yaygın olarak timer kesmesiyle oluşabilecek sinyali kullanarak yapılacak işlemlerde tercih edilir. Harici clock (ETR2) harici bir kaynak bağlayarak

(sinüs dalgası vs) etr2 nin pinindeki sinyale göre ölçme işlemleri yapılır. Seçildiği durumlarda kart üzerinde otomatik atama yapılır ve ayarlanması gerekmektedir. Dolayısıyla ETR2'de oluşan sinyale (pals) göre işlemler yapılmış olur.

- **Channel1:**

  output compare bölümü iki farklı kanaldan iki farklı sinyal bağlayarak bu sinyallerin karşılaştırılabilmesine imkan verir.

  capture direct mode bölümü bir sinyalin periyoduna göre o sinyalin yakalanabilmesini sağlar. Örneğin 50 Hz'lik bir sinyalde 50 Hz'lik bir palse'i 1, 2, 3… gibi saymaya başlar. Kaç tane 50 Hz'lik periyod yakalanırsa o periyoda göre sayma işlemi yapar.

Projede Clock kaynağı olarak dahili clock tercih edilmiştir. Böylelikle timer arka planda çalışır ve sayma işlemi yapar. Sayma işlemi yapıldıkça bir alt ve üst limit ile belirlenen sayıya göre kesme işlemi gerçekleşir.

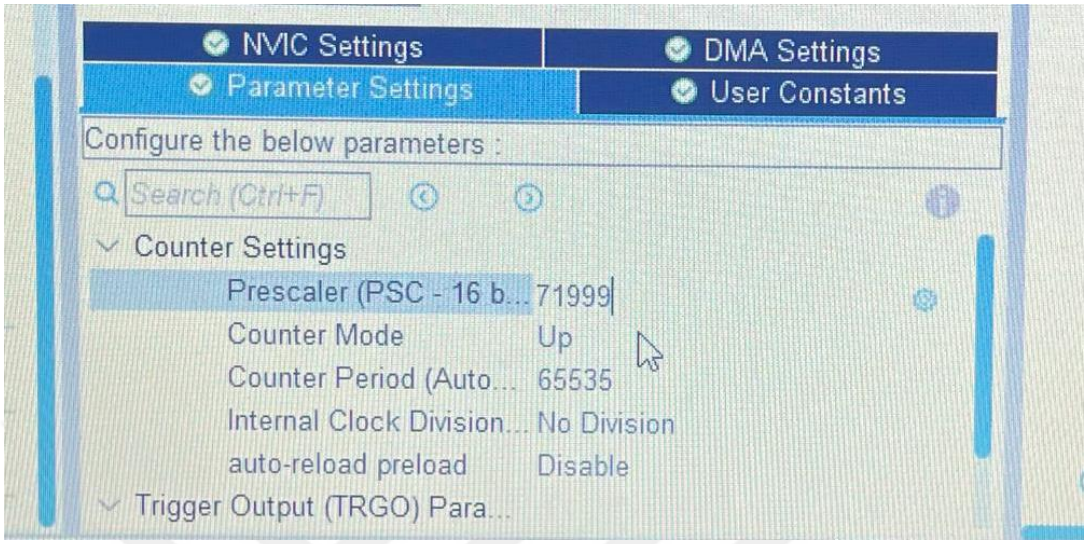
Timer modu belirlendikten sonra programda üç adımda ayarlamalar yapılır. Bunlar:

1. Pin&configurasyon bölümünde Şekil 4.22'de görüldüğü gibi bazı ayarlar yapılır:

Parametre ayarları

- **Prescaler PSC:** Sayıcının üst limitinin belirtildiği bölümdür. Yani sayıcının nereye kadar sayacağını gösterir. Bu sayma işlemi 0'dan başlayarak saymaz 1'den başlayarak saydığı için PSC alanına belirlenen sayının 1 eksiği yazılır. 72MHz için 71999 yazılmalıdır.
- **Counter Mode:** Bu kısımda Up-down-center aligned mode1-2-3 gibi seçenekler yer alır ve burası sayma işleminin aşağı-yukarı doğru sayma şeklinin belirlendiği bölümdür.
- **Counter Period**: Kesme işlemi bittikten sonra sayma periyodu burada ifade edildiği gibi her seferinde tekrar yüklenir. Yine periyot için belirlenen sayıdan 1 çıkartılarak yazılmalıdır.
- **Auto-reload preload modu:** Disable ve Enable sekmeleri açılır. Burada sayma işleminin her seferinde tekrar edilip/edilmemesinin seçimi yapılır. Sayma işlemi

her seferinde tekrar edilip bitirildikten sonra sayma işlemini bitirmeyip başa dönüp belirtilen değerin tekrar yüklenmesi işlemi gerçekleştirilir.



Şekil 4.22. Timer parametre ayarları

- **NVIC ayarları:** Kesme kontrol ayarlarının yapıldığı kısımdır. TIM1'in update interrupt ayarlaması yapılır. Bu her güncelleme yani her saymayı bitirmede bir kesme oluşması istenildiğinde aktifleştirilir.
- **User Constansts ayarları** ayarlama yapılmamıştır.
- **DMA ayarları** bölümünde ayarlama yapılmamıştır.

2. Clock konfigürasyon ayarlamaları;
   HCLK 72 MHz seçilmiştir ve bu seçim yapıldıktan sonra gerekli olan frekans (Hz) değerleri otomatik olarak hesaplanır.
3. Project Manager bölümde isim ve klasör ayarlamaları yapıldıktan sonra Toolchain seçilir ve Code Genarator bölümünde gerekli dosyaların otomatik olarak gelmesini işaretledikten sonra kod oluştur butonuna basılır.

Bütün ayarlamalar Şekil 4.21 ve 4.22'de belirlenen bölümlere yazıldıktan sonra STM32CubeIDE programı girilen verilere göre Timer bilgilerini içeren ve aşağıda görülen kod oluşturulur:

```
static void MX_TIM1_Init(void) {
  RCC->APB2ENR |= RCC_APB2ENR_TIM1EN;
  //32320
  // configure TIM1
  TIM1->CR1 = 0;              // dir=0 - upcounting
  TIM1->ARR = 1;              // upper bound
  TIM1->PSC = 35999;          // f=72MHz   F= f/((psc+1)*(ARR+1))  1000hz -> 1 ms
  TIM1->EGR = TIM_EGR_UG;     // Generate an update event to reload the Prescaler and the Repetition counter values immediately
  // start communication
  TIM1->SR = 0;      // clear status
  TIM1->DIER = TIM_DIER_UIE;    // Update interrupt enable
  NVIC_EnableIRQ(TIM1_UP_IRQn);
  TIM1->CR1 |= TIM_CR1_CEN;
}
```

Burada zamanlayıcıyı (TIM1) yapılandırılıp, mevcut kod incelendiğinde:

- RCC->APB2ENR |= RCC_APB2ENR_TIM1EN; Bu satır, RCC_APB2ENR kaydındaki TIM1EN bitini ayarlayarak Zamanlayıcı 1 için saati etkinleştirir.

Zamanlayıcı Yapılandırması:

- TIM1->CR1 = 0; Zamanlayıcı 1'in kontrol kaydı 1'i yapılandırır. Yönü yukarı doğru saymaya ayarlar (DIR=0).
- TIM1->ARR = 1; Zamanlayıcının otomatik yeniden yükleme değerini (üst sınır) 1 olarak ayarlar.
- TIM1->PSC = 35999; Belirli bir frekansa ulaşmak için ön ölçekleyici değerini yapılandırır. Bu durumda, 72 MHz saat frekansıyla 1000 Hz (1 ms periyot) frekans üretecek şekilde ayarlanmıştır.

- TIM1->EGR = TIM_EGR_UG: Ön Ölçekleyici ve Tekrarlama sayacı değerlerini yeniden yüklemek için bir güncelleme olayı oluşturur.

Kesme Yapılandırması:

- TIM1->SR = 0; Durum kaydını temizler.
- TIM1->DIER = TIM_DIER_UIE; Güncelleme kesmesini etkinleştirir.
- NVIC_EnableIRQ(TIM1_UP_IRQn); Zamanlayıcı 1 Güncelleme IRQ'su için Yuvalanmış Vektör Kesinti Denetleyicisindeki (NVIC) kesmeyi etkinleştirir.

Zamanlayıcı Çalışmasını Başlatma:

- TIM1->CR1 |= TIM_CR1_CEN; Zamanlayıcı işlemini başlatarak Sayaç Etkinleştirme bitini ayarlar.

Bu düzenlemeler, 1'lik bir ARR değeri, 35999'luk bir ön ölçekleyici (72 MHz'lik bir saat ile 1 ms'lik bir periyoda ulaşmak için) kullanarak 1 ms'lik bir periyotluk kesinti oluşturmak ve güncelleme kesmesini etkinleştirmek için Timer 1'i belirli yapılandırmalarla ayarlar. TIM1 ve kesme kullanılarak aşağıda görülen C kodu yardımıyla PLC projesindeki zamanlayıcılarda referans zamanlama sinyalleri olarak kullanılmak üzere 10 ms, 100 ms ve 1 saniyelik periyotlara sahip üç tane sinyal üretilmektedir.

```
void TIM1_UP_IRQHandler(void)
{
    if (TIM1->SR&TIM_SR_UIF)
    {
        TIM1->SR &= ~TIM_SR_UIF;
        T_CNT1 = T_CNT1+1;
        T_CNT2 = T_CNT2+1;
        T_CNT3 = T_CNT3+1;
        if(T_CNT1>=5)
```

```
            {
                T_10ms=!T_10ms;
                T_CNT1 =0;
            }
            if(T_CNT2>=50)
            {
                T_100ms=!T_100ms;
                T_CNT2 =0;
            }
            if(T_CNT3>=500)
            {
                HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_13);
                T_1s=!T_1s;
                T_CNT3 =0;
            }
            TIMER_Val++;
            if(TIMER_Val>65534)TIMER_Val=0;
        }
        TIM1->SR = 0;
}
```

Oluşturulan bu C Kodu sayesinde Timer 1'in (TIM1) güncelleme kesintisini işleyen bir kesme işlemi yapılmaktadır. Şöyle ki:

- Kesme Kontrolü: Kod, Timer 1'in (TIM1) durum kaydedicisinde (SR) güncelleme kesme bayrağının (UIF) ayarlanıp ayarlanmadığını kontrol ederek başlar. Bu koşul, güncelleme olayının meydana gelip gelmediğini doğrular.

- Bayrak Temizleme: Güncelleme olayı onaylandıktan sonra kod, TIM_SR_UIF tamamlayıcısı ile bit düzeyinde AND işlemini kullanarak ve bunu durum kaydedicisine tekrar atayarak UIF'yi temizler.
- Zamanlayıcı Sayaçlarının Artışı: Üç zamanlayıcı sayma değişkeni (T_CNT1, T_CNT2, T_CNT3) her güncelleme olayında gerektiğinde 1 değer arttırılır.
- Aralık Tabanlı Bayrak Güncellemesi: Belirli sayaçların (T_CNT1, T_CNT2, T_CNT3) belirli değerlere (sırasıyla 5, 50, 500) ulaşıp ulaşmadığını kontrol eden üç koşullu blok vardır. Bu sayaçlar belirlenen değerlere ulaştığında bayraklar (T_10ms, T_100ms, T_1s) değiştirilir (toggle yapılır).
- HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_13); Bu satır, bir LED'i veya başka bir çıkışı kontrol etmek için kullanılan GPIOC'un 13 numaralı pininin durumunu değiştirir.
- Genel Zamanlayıcı Değeri Güncellemesi: TIMER_Val, her güncelleme olayında artırılır ve 65534 ile sınırlandırılır.
- Bayrak Sıfırlama ve Temizleme: Son olarak kod, durum kaydedicisinde tüm bekleyen kesmeleri ona 0 yazarak siler (TIM1->SR = 0).

Böylelikle 10 ms, 100 ms ve 1 saniye periyota sahip T_10ms, T_100ms ve T_1s isimli üç tane sinyal üretilmiş olur.

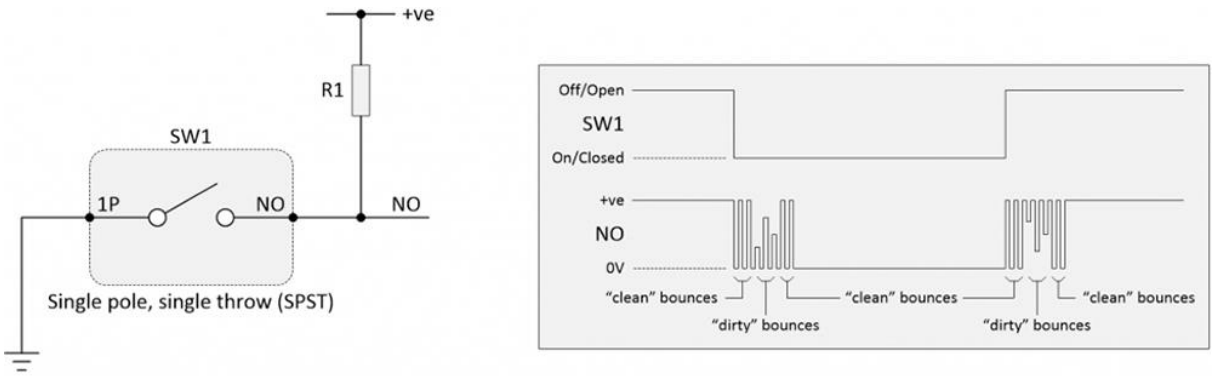
4.6. Kontak Atlaması Problemi ve Proje Kapsamında Çözümü

Bu kısımda kontak atlaması problemin nedenleri ve bu problemin giderilebilmesi için gerekli işlemler incelenmektedir. Kontak atlaması, genellikle elektriksel veya elektromekanik sistemlerde karşılaşılan bir sorundur ve elektriksel sistemlerin güvenilirliğini ve performansını etkilediği için oluşan sorunların tam olarak çözülmesi önemlidir. Bu terim, bir devredeki kontakların beklenmedik bir şekilde açılıp kapanması veya bağlantılarının kopması anlamına gelir ve sistemlerin düzgün çalışmasını engelleyebilir veya bozabilir. Bu nedenle, kontak atlaması sorunları genellikle dikkatlice tespit edilmeli ve giderilmelidir. Kontak atlaması, özellikle röleler, anahtarlar, düğmeler veya diğer mekanik veya elektronik kontaklar gibi elektromekanik cihazlarla ilişkili olabilir. Kontak atlamasının nedenleri şunlardır:

- Fiziksel Hasar veya Kirlenme: Kontakların fiziksel olarak hasar görmesi, aşınması veya kirlenmesi kontak atlaması problemini tetikleyebilir. Bu, kontakların istenmeyen şekilde temasını etkileyebilir.
- Kötü Bağlantılar: Elektrik devrelerindeki kötü bağlantılar, kontakların istenmeyen şekilde kesilmesine veya kapanmasına neden olabilir. Bu, tellerin gevşek olması, lehim noktalarının zayıf olması veya bağlantı noktalarının korozyona uğraması gibi durumlarla ilişkilendirilebilir.
- Elektriksel Gürültü: Elektrik devrelerindeki gürültü veya parazitler, kontakların istenmeyen şekilde hareket etmesine neden olabilir. Bu, devredeki diğer cihazlardan kaynaklanan elektriksel parazitler veya ortamdan gelen radyo frekansı (RF) gürültüsü olabilir.
- Kontak Basıncı: Elektromekanik kontakların yeterli basınca sahip olmaması kontak atlaması problemini tetikleyebilir. Kontakların yeterince sıkı olmaması veya düğmelerin düzgün çalışmaması bu tür sorunlara yol açabilir.
- Termal Etkiler: Elektromekanik kontaklar, sıcaklık değişikliklerine hassas olabilir. Yüksek veya düşük sıcaklıklar kontakların istenmeyen şekilde hareket etmesine veya temasını kaybetmesine neden olabilir.

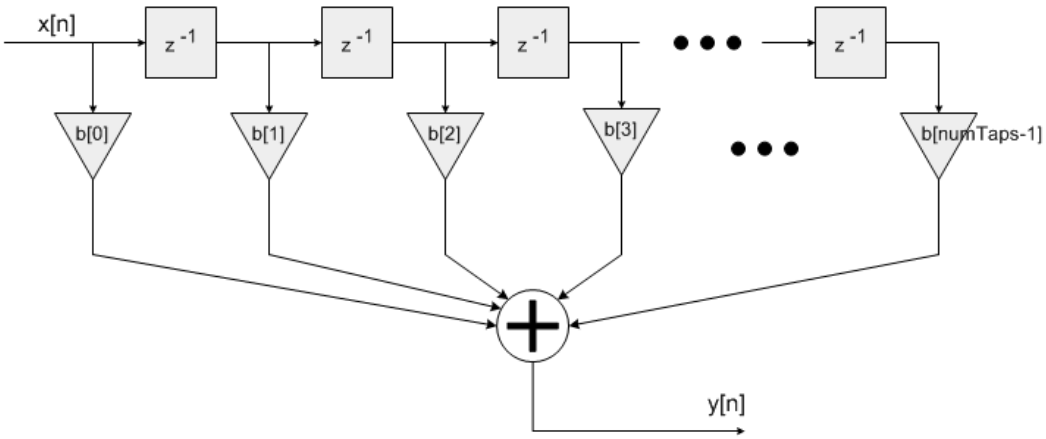
Kontak atlaması (contact bouncing) problemi Şekil 4.23'te ifade edilmiştir. Bir buton, bir anahtarı, bir röle ya da kontaktör kontağı gibi mekanik bir kontak açıldığında çok nadiren bir konumdan diğerine düzgün bir geçiş gerçekleşir. Kontak, açıldığında ya da kapandığında sabit bir konuma gelinceye kadar defalarca açma-kapanma gerçekleşir. Bu durum mekanik titreşim oluşturup sıçramalar meydana getirir. Aynı zamanda bir metal başka bir metale çok yavaş bastırıldığı veya bırakıldığı durumlar düşünülürse farklı bir nedenden dolayı (tozlaşma, paslanma vs.) değme-değmeme anı oluşabilir. Şekil 4.23'te görülen SW1, normalde açık kontağı olan tek kutuplu tek atımlı (SPST) bir anahtardır. SW1 anahtarı açıkken R1 direncinin alt ucu ile normalde açık kontağın birleşim yeri olan NO çıkış noktasındaki gerilim +ve'ye eşittir. SW1 anahtarı kapalıyken NO çıkış noktasındaki gerilim ise 0V'a eşittir. Şeklin sağ tarafında görülen dalga şekillerinden gözlemleneceği üzere SW1 anahtarı açıkken kapatılmak üzere anahtara baskı uygulandığında NO çıkış noktasındaki gerilim kontak atlaması problemi sebebiyle bir süre

+ve ile 0V arasında salındıktan sonra 0V olur. Benzer şekilde SW1 anahtarı kapalıyken açılmak üzere anahtara baskı uygulandığında NO çıkış noktasındaki gerilim kontak atlaması problemi sebebiyle bir süre 0V ile +ve arasında salındıktan sonra +ve'ye eşit olur. Bu örnek dikkate alındığında eğer bu şekildeki NO çıkış noktası bir dijital sisteme bilgi vermek için kullanılırsa bu defalarca açılma-kapanma sebebiyle pek çok hatalı 0'dan 1'e veya 1'den 0'a değişim algılanır (Digikey, t.y.).



Şekil 4.23. Kontak atlaması problemi (Digikey, t.y.)

Kontak atlaması probleminin çözümü için RC alçak geçiren filtresi, yazılımsal tasarımı veya yazılımsal filtre kullanılarak sorun çözümlenir. Gerçekleştirilen tez çalışmasında yazımsal filtre tasarımı tercih edilmiştir. Şekil 4.24'te görülen Sonlu Darbe Tepkisi (FIR) filtresi kullanılmıştır. Sonlu Darbe Tepkisi (FIR) filtreleri dijital sinyal işleme (DSP) alanında kullanılan bir filtreleme tekniğidir ve belirli parametreleri belirledikten sonra tasarım gerçekleşir.



Şekil 4.24. Sonlu darbe tepkisi (FIR) filtresi (Blog – DTA Mühendislik, 2020)

FIR filtresinde örnekler arasında bir örnek gecikmesi sağlanmaktadır. Sinyal sonlu bir önceki örneğiyle çarpılır ve bir dizi sabit katsayı kullanılarak filtrelenir. Bu katsayılar, filtrelenen sinyalin frekans tepkisini belirler ve istenilen frekans tepkisi doğrudan katsayılarla elde edilebilir. FIR filtrelerinin önemli bir özelliği, sabit bir grup gecikmesine sahip olmalarıdır. Yani sinyalin girişinden çıkışına kadar geçen zaman sabit kalır ve matematiksel olarak şu şekilde ifade edilir:

eskiInputDeger = (filtreKatsayısı x inputDeger) + (eskiInputDeger x (1-filtreKatsayısı))

filtreKatsayısı = 0.1 olsun;

1. döngü: butona basılmadı

inputDeger=0 , eskiInputDeger=0

eskiInputDeger = 0x0.1 + 0x0.9 = 0

.

.

.

199. döngü : butona basıldı

inputDeger=1 , eskiInputDeger=0

eskiInputDeger = 1x0.1 + 0x0.9 = 0.1

200. döngü : buton halen basılı

inputDeger=1 , eskiInputDeger=0.1

eskiInputDeger = 1x0.1 + 0.1x0.9 = 0.19

201. döngü : buton halen basılı

inputDeger=1 , eskiInputDeger=0.19

eskiInputDeger = 1x0.1 + 0.19x0.9 = 0.271

202. döngü : buton halen basılı

inputDeger=1 , eskiInputDeger=0.271

eskiInputDeger = 1x0.1 + 0.271x0.9 = 0.3439

.

.

n. döngü buton halen basılı eskiInputDeger=1

n+1 döngü de butonu bıraktık:

inputDeger=0 , eskiInputDeger=1

eskiInputDeger = 0x0.1 + 1x0.9 = 0.9

n+2 döngü de butonu bıraktık:

inputDeger=0 , eskiInputDeger=0.9

eskiInputDeger = 0x0.1 + 0.9x0.9 = 0.81

n+3 döngü de butonu bıraktık:

inputDeger=0 , eskiInputDeger=0.81

eskiInputDeger = 0x0.1 + 0.81x0.9 = 0.729

.

.

.

Devam edecektir…

Bahsedilen işlemlerin formülü STM32CubeIDE programında şu şekilde ifade edilir:

filteredState[index] = ((1-alpha) * (float)current_state) + (alpha * filteredState[index]);

Burada alpha değerinin belirli bir aralıkta (0.001 ile 0.999 arasında) olması gerektiğine dikkat edilmelidir. Bu değer, FIR filtresinin davranışını ve tepki süresini etkiler. Değer ne kadar büyükse, FIR filtresinin girişe daha fazla tepki gösterme eğilimi artar. Bu yüzden alpha değeri 0.999 olarak belirlenmiştir.

Tercih edilen STM32CubeIDE programında FIR (Sonlu Darbe Tepkisi) filtresinin uygulanması için C dilinde bir fonksiyon tanımlanmıştır. Kodun genel yapısı ve kullanılan değişkenler şu şekildedir:

```
const float alpha = 0.999;  // 0.001 ~ 0.999
void read_and_filter_input(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, int index)
{
   _Bool current_state = HAL_GPIO_ReadPin(GPIOx, GPIO_Pin);
   filteredState[index] = ((1-alpha) * (float)current_state) + (alpha * filteredState[index]);
}
unsigned int get_inputs(void)
{
        read_and_filter_input(GPIOA,GPIO_IDR_IDR8,0);
        read_and_filter_input(GPIOA,GPIO_IDR_IDR9,1);
        read_and_filter_input(GPIOA,GPIO_IDR_IDR10,2);
        read_and_filter_input(GPIOA,GPIO_IDR_IDR11,3);
        read_and_filter_input(GPIOB,GPIO_IDR_IDR6,4);
        read_and_filter_input(GPIOA,GPIO_IDR_IDR15,5);
        read_and_filter_input(GPIOB,GPIO_IDR_IDR3,6);
        read_and_filter_input(GPIOB,GPIO_IDR_IDR4,7);

        for(int indx=0;indx<8;indx++)
        {
                if(filteredState[indx]>=0.5f)bit_set(INPUT,BIT(indx));
                else bit_clear(INPUT,BIT(indx));
         }
```

```
    return 0;
}
```

Yukarıda görülen C kodunda;

alpha: FIR filtresinde kullanılacak olan ağırlık katsayısı. Bu katsayı, FIR filtresinin güncellenen çıkış değerini hesaplamak için kullanılır.

filteredState[]: FIR filtresinin çıkış değerlerini tutan bir dizidir.

read_and_filter_input(): Belirli bir GPIO pininin okunması ve FIR filtresinin uygulanması için kullanılan fonksiyondur. Bu fonksiyon, belirli bir GPIO pininin mevcut durumunu okur, FIR filtresine sokar ve sonucu filteredState dizisine kaydeder.

get_inputs(): Tüm girişlerin okunması ve FIR filtresinin uygulanması için kullanılan fonksiyondur. Bu fonksiyon, belirli GPIO pinlerinin okunması ve FIR filtresinin uygulanması için read_and_filter_input() fonksiyonunu çağırır. Ardından, FIR filtresinden geçirilmiş çıkışlar, belirli bir eşik değerine göre bit düzeylerine dönüştürülerek bir çıkış değeri oluşturulur. Her bir durum için belirli bir eşik değeri (0.5) üzerinden karar verir ve buna göre bir bitin set edilip edilmeyeceğini belirler. Bu işlem, bir bit işlemi olarak bilinen bir işlemle gerçekleştirilir. İlk olarak, bir döngü kullanılarak her bir filtrelenmiş giriş durumu (filteredState[] dizisinde) incelenir. Ardından, her bir durum için şu adımlar gerçekleştirilir:

filteredState[indx] >= 0.5f: Filtrelenmiş giriş durumu belirli bir eşik değerinden büyük veya eşitse, if koşulu doğrudur. Bu durumda, ilgili bit set edilir.

bit_set(INPUT, BIT(indx)): İlgili bit, INPUT değişkeninin belirli bir konumunda (BIT(indx)) set edilir. bit_set() fonksiyonu, belirli bir biti set etmek için kullanılır. Aksi takdirde, yani filtrelenmiş giriş durumu belirli bir eşik değerinden küçükse, else bloğu çalışır.

bit_clear(INPUT, BIT(indx)): İlgili bit, INPUT değişkeninin belirli bir konumunda (BIT(indx)) reset edilir. bit_clear() işlevi, belirli bir biti temizlemek (reset etmek) için kullanılır. Filtrelenmiş giriş durumuna göre belirli bir eşik değeri üzerinden karar verir ve buna göre ilgili biti temizler.

Böylelikle, belirli bir giriş sinyalinin FIR filtresinden geçirilerek düzeltilmiş bir çıkış sinyali üretilir. FIR filtresi, bir önceki adımda elde edilen çıkışı ve mevcut örneği dikkate alarak bir ağırlıklandırma işlemi uygular. Bu, giriş sinyalindeki dalgalanmaları düzeltmeye yardımcı olur ve istikrarlı bir çıkış sağlar. Bu sistem dijital ve analog sistemlerde değerlerin dalgalanmasından dolayı çok kullanılır. Analog okumalarda filtreye konulduğunda biraz gecikmeli olur ama ani sıçramalar olmaz. Dijital sistemlerde de yüksek frekanslı buton girişleri engellenmiş olur.

5. KONTAK VE RÖLE TEMELLİ FONKSİYONLAR

Bu tez projesinde Boole (ikili-binary) değişkenleri üzerinde işlem yapmak amacıyla oluşturulan kontak ve röle temelli fonksiyonlar bu bölümde açıklanmaktadır. Toplam 29 tane kontak ve röle temelli fonksiyon gerçekleştirilmiştir. Kontak ve röle temelli fonksiyonlarla ilgili geliştirilen C kodları, doğruluk tabloları ve semboller Tablo 5.1'de sunulmuştur. Kontak ve röle temelli fonksiyonlarla ilgili açıklamalar aşağıda verilmiştir:

1. **ld** fonksiyonu bir merdiven diyagramındaki en soldaki normalde açık kontak olarak işlem yapar. Bilgiyi 'in' girişinden alır ve TEMP üzerinden dışarı aktarır. Eğer gelen bilgi 0 ise, çıkış (TEMP) 0 olur. Gelen bilgi 1 ise, çıkış (TEMP) 1 olur.
2. **ld_not** fonksiyonu bir merdiven diyagramındaki en soldaki normalde kapalı kontak olarak işlem yapar. Bilgiyi 'in' girişinden alır ve 'in' değerinin tersini TEMP üzerinden dışarı aktarır. Eğer gelen bilgi 0 ise, çıkış (TEMP) 1 olur. Gelen bilgi 1 ise, çıkış (TEMP) 0 olur.
3. **not** fonksiyonu lojik DEĞİL (NOT) kapısı olarak işlem yapar. Giriş bilgisini TEMP üzerinden alır ve girişin değilini çıkışa TEMP üzerinden aktarır. Giriş değeri 0 ise çıkış (TEMP) 1'e, giriş değeri 1 ise çıkış (TEMP) 0'a dönüşür.
4. **or** fonksiyonu iki girişli bir lojik VEYA (OR) kapısı olarak görev yapar. Bir girişi 'IN' değişkeninden alırken, diğer girişi bir önceki fonksiyondaki bilgiyi saklayan TEMP'ten alır. İşlem sonucunu tekrar TEMP üzerinden dışarı aktarır.
5. **or_not** fonksiyonu girişlerden birisinin değili alınmış iki girişli bir lojik VEYA (OR) kapısı olarak görev yapar. Girişlerden birini 'in' değişkeninden, diğerini ise TEMP değişkeninden alır. Ancak TEMP'ten alınan bilginin değilini alarak işlem yapar ve oluşan işlem sonucunu TEMP ile dışarı aktarır.
6. **nor** fonksiyonu iki girişli bir lojik VEYA DEĞİL (NOR) kapısı olarak görev yapar. Girişlerden birini 'in' değişkeninden, diğer girişi ise TEMP'ten alır. İşlem sonucunu TEMP ile dışarı aktarır.
7. **and** fonksiyonu iki girişli bir lojik VE (AND) kapısı olarak görev yapar. Girişlerden birini 'in' değişkeninden, diğer girişi ise TEMP'ten alır. İşlem sonucunu TEMP ile dışarı aktarır.

8. **and_not** fonksiyonu girişlerden birisinin değili alınmış iki girişli bir lojik VE (AND) kapısı olarak görev yapar. Girişlerden birini ‘in’ değişkeninden, diğer girişi ise TEMP’ten alır. TEMP’ten alınan değerin değilini alıp işleme devam eder ve işlem sonucunu TEMP ile dışarı aktarır.
9. **nand** fonksiyonu iki girişli bir VE DEĞİL (NAND) kapısı olarak görev yapar. Girişlerden birini ‘in’ değişkeninden, diğer girişi ise TEMP’ten alır ve işlem sonucunu TEMP ile dışarı aktarır.
10. **xor** fonksiyonu iki girişli bir ÖZEL VEYA (EX-OR) kapısı olarak görev yapar. Girişlerden birini ‘in’ değişkeninden, diğer girişi ise TEMP’ten alır. İşlem sonucunu TEMP ile dışarı aktarır.
11. **xnor** fonksiyonu iki girişli bir ÖZEL VEYA DEĞİL (EX-NOR) kapısı olarak görev yapar. Girişlerden birini ‘in’ değişkeninden, diğer girişi ise TEMP’ten alır. İşlem sonucunu TEMP ile dışarı aktarır.
12. **xor_not** fonksiyonu iki girişten birinin değili alınmış ÖZEL VEYA (EX-OR) kapısı olarak görev yapar. Bir girişi 'in' değişkeninden alırken, diğer girişi TEMP’ten alır. TEMP’ten alınan değerin değili alınarak işlem gerçekleştirilir ve sonuç TEMP ile dışarıya aktarılır.
13. **out** fonksiyonu merdiven diyagramında röle (dâhili ya da harici) çıkışı olarak görev yapar. İkili giriş bilgisini TEMP’ten alır ve çıkış yine TEMP ile dışarı aktarılmış olur. Giriş değeri 0 ise, çıkış (TEMP) 0 olurken; eğer giriş değeri 1 ise, çıkış (TEMP) 1 olur.
14. **out_not** fonksiyonu merdiven diyagramında out_not fonksiyonu değillenmiş röle (dâhili ya da harici) çıkışı olarak görev yapar. İkili giriş bilgisini TEMP’ten alır ve çıkış yine TEMP ile dışarı aktarılmış olur. Giriş değeri 0 ise, çıkış (TEMP) 1 olurken; eğer giriş değeri 1 ise, çıkış (TEMP) 0 olur.
15. **in_out** fonksiyonu bir giriş ve girişe bağlı bir çıkıştan oluşur. Eğer giriş bilgisi IN 1 ise çıkış (TEMP) 1, eğer giriş bilgisi 0 ise çıkış (TEMP) 0 olur.
16. **inv_out** fonksiyonu bir değillenmiş giriş ve bu girişe bağlı bir çıkıştan meydana gelmektedir. IN giriş değeri 0 olduğunda, çıkış OUT 1 olarak belirlenirken; IN giriş değeri 1 olduğunda ise çıkış (OUT) 0 olur.
17. **r_edge** fonksiyonu yükselen kenar algılayıcı olarak görev yapar. İkili değişken “IN”, TEMP ile alınır ve fonksiyonun çıkışı yine TEMP üzerinden dışarıya iletilir.

“r_edge” fonksiyonu çalıştırılmadan önce, “IN” giriş sinyali TEMP kaydedicisine yüklenmiş olmalıdır. İşlemin sonunda, TEMP “OUT” çıkış sinyali olarak dışarı gönderilir. Merdiven diyagramında pozitif geçiş algılama kontağı olarak tanımlanan bu fonksiyon aynı zamanda --| P |-- sembolüyle gösterilir. “IN” giriş değişkeni 0’dan 1’e değiştiğinde, “OUT” çıkış değişkeni sadece bir PLC tarama süresi boyunca 1 yapılır. Diğer tüm durumlarda çıkış “OUT” 0 olarak kalır.

18. **f_edge** fonksiyonu düşen kenar algılayıcı olarak görev yapar. İkili değişken “IN”, TEMP kaydedicisiyle alınır ve fonksiyonun çıkışı yine TEMP üzerinden dışarıya iletilir. ‘f_edge’ fonksiyonu çalıştırılmadan evvel, “IN” giriş sinyali TEMP'e yüklenir. İşlevin sonunda, TEMP “OUT” çıkış sinyali olarak dışarı gönderilir. Merdiven diyagramında negatif geçiş algılama kontağı olarak tanımlanan bu fonksiyon --| N |-- sembolüyle ifade edilir. “IN” giriş değişkeni 1’den 0’a değiştiğinde, “OUT” çıkış değişkeni sadece bir PLC tarama süresi boyunca 1 yapılır. Diğer tüm durumlarda çıkış “OUT”, 0 olarak kalır.
19. **latch** fonksiyonunda iki giriş ve bir çıkış değişkeni bulunmaktadır. Aktif 1 yüklemeli giriş (EN), fonksiyon çalıştırılmadan önce TEMP'e yüklenir. Aktif 1 yüklemeli giriş (EN) = 0 olduğunda, Q çıkışında bir değişiklik olmaz ve Q mevcut değerini korur. Ancak aktif 1 yüklemeli giriş (EN) = 1 olduğunda ise D girişindeki mantıksal değer Q çıkışına yüklenir.
20. **mid_out** hatortası çıkış fonksiyonudur. Boole giriş değişkeni olan IN’in lojik durumunu Boole hatortası çıkışı değişkeni TEMP’e yükler.
21. **mid_out_not** değillenmiş hatortası çıkış fonksiyonudur. Boole giriş değişkeni IN’in değilini, değillenmiş hatortası çıkışı değişkeni TEMP’e yükler.
22. **_set** fonksiyonunda bir Boole giriş değişkeni (TEMP) ve bir Boole çıkış değişkeni (OUT) mevcuttur. Giriş değişkeni (TEMP) 0 olduğunda herhangi bir işlem yapılmaz ve çıkışın (OUT) durumu değişmez. Giriş değişkeni (TEMP) 1 olduğunda ise çıkış (OUT) set edilir (1 yapılır).
23. **_reset** fonksiyonunda bir Boole giriş değişkeni (TEMP) ve bir Boole çıkış değişkeni (OUT) mevcuttur. Giriş değişkeni (TEMP) 0 olduğunda herhangi bir işlem yapılmaz ve çıkışın (OUT) durumu değişmez. Giriş değişkeni (TEMP) 1 olduğunda ise çıkış (OUT) reset edilir (0 yapılır).

24. **SR** fonksiyonunda iki tane Boole giriş değişkeni (S, R1) ve bir tane Boole çıkış değişkeni (OUT) mevcuttur. Bu fonksiyon reset öncelikli bir SR mandalı olarak görev yapar.

    S = 0, R1 = 0 ise çıkış değişkeni OUT bir önceki değerini korur.

    S = 1, R1 = 0 ise çıkış değişkeni OUT set (1) yapılır.

    S = 0, R1 = 1 ise çıkış değişkeni OUT reset (0) yapılır.

    S = 1, R1 = 1 ise çıkış değişkeni OUT reset (0) yapılır.

    Eğer set ve reset aynı anda aktif olursa (S=1, R1=1) çıkış (OUT) 0 olur.

25. **RS** fonksiyonunda iki tane Boole giriş değişkeni (R, S1) ve bir tane Boole çıkış değişkeni (OUT) mevcuttur. Bu fonksiyon set öncelikli bir RS mandalı olarak görev yapar.

    R = 0, S1 = 0 ise çıkış değişkeni OUT bir önceki değerini korur.

    R = 1, S1 = 0 ise çıkış değişkeni OUT reset (0) yapılır.

    R = 0, S1 = 1 ise çıkış değişkeni OUT set (1) yapılır.

    R = 1, S1 = 1 ise çıkış değişkeni OUT set (1) yapılır.

    Eğer set ve reset aynı anda aktif olursa (R=1, S1=1) çıkış (OUT) 1 olur.

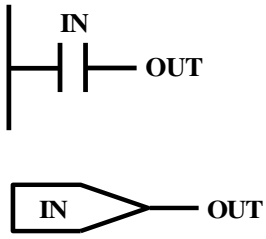
26. **r_toggle** fonksiyonunda bir Boole-giriş-değişkeni IN (TEMP üzerinden alınmaktadır), bir Boole-çıkış-değişkeni (OUT) ve bir indeks değeri i (0, 1, …, 63) bulunmaktadır. Bu fonksiyon IN giriş sinyalinin yükselen kenarını algılamak amacıyla RED_TOGGLE[i] özgün hafıza bitini kullanır. Çalıştırılmadan önce IN giriş sinyali TEMP'e yüklenmelidir. Böylelikle OUT çıkış sinyali, fonksiyonun sonunda OUT'a yüklenmiş olacaktır. Zamanlama diyagramından görüleceği gibi IN giriş sinyalinin 0'dan 1'e değiştiği anlarda OUT çıkış sinyali, toggle (1 ise 0, 0 ise 1) yapılmaktadır. Diğer tüm durumlarda OUT çıkış sinyali değişmeden kalır.

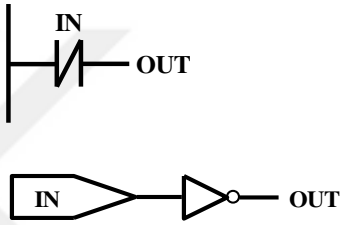
27. **f_toggle** fonksiyonunda bir Boole-giriş-değişkeni IN (TEMP üzerinden alınmaktadır), bir Boole-çıkış-değişkeni (OUT) ve bir indeks değeri i (0, 1, …, 63) bulunmaktadır. Bu fonksiyon IN giriş sinyalinin düşen kenarını algılamak amacıyla FED_TOGGLE[i] özgün hafıza bitini kullanır. Çalıştırılmadan önce IN giriş sinyali TEMP'e yüklenmelidir. Böylelikle OUT çıkış sinyali, fonksiyonun sonunda OUT'a yüklenmiş olacaktır. Zamanlama diyagramından görüleceği gibi IN giriş sinyalinin 1'den 0'a değiştiği anlarda çıkış sinyali (Output), toggle (1 ise 0, 0 ise 1) yapılmaktadır. Diğer tüm durumlarda OUT çıkış sinyali değişmeden kalır.

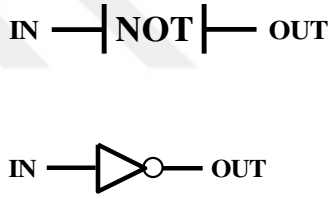
28. **adrs_re** fonksiyonunda bir Boole-giriş-değişkeni IN (TEMP üzerinden alınmaktadır), bir Boole-giriş-değişkeni adrs, bir Boole-çıkış-değişkeni OUT (TEMP üzerinden dışarı gönderilmektedir) ve bir indeks değeri i (0, 1, …, 63) bulunmaktadır. Bu fonksiyon Boole-giriş-değişkeni adrs'nin yükselen kenarını algılamak amacıyla adrs_RED[i] özgün hafıza bitini kullanır. Fonksiyon çalıştırılmadan önce IN giriş sinyalinin TEMP'e yüklenmiş olması gerekmektedir. Zamanla diyagramında belirtildiği üzere IN=1 olduğunda, Address giriş sinyali 0'dan 1'e değiştiği anda (1 PLC tarama süresi kadar) TEMP çıkış sinyali 1 olur. Diğer tüm durumlarda TEMP çıkış sinyali 0 olarak kalacaktır.
29. **adrs_fe** fonksiyonunda bir Boole-giriş-değişkeni IN (TEMP üzerinden alınmaktadır), bir Boole-giriş-değişkeni adrs, bir Boole-çıkış-değişkeni OUT (TEMP üzerinden dışarı gönderilmektedir) ve bir indeks değeri i (0, 1, …, 63) bulunmaktadır. Bu fonksiyon Boole-giriş-değişkeni adrs'nin düşen kenarını algılamak amacıyla adrs_FED[i] özgün hafıza bitini kullanılır. Fonksiyon çalıştırılmadan önce IN giriş sinyalinin TEMP'e yüklenmiş olması gerekmektedir. Zamanla diyagramında belirtildiği üzere IN=1 olduğunda, Address giriş sinyali 1'dan 0'a değiştiği anda (1 PLC tarama süresi kadar) TEMP çıkış sinyali 1 olur. Diğer tüm durumlarda TEMP çıkış sinyali 0 olarak kalacaktır.

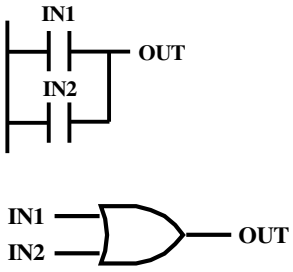
Tablo 5.1. Kontak ve röle temelli fonksiyonlar

<table>
<tr><th>No</th><th>STMCubeIDE
Programındaki
C Kodu</th><th>Doğruluk Tablosu</th><th>Merdiven Diyagramı
Sembolü /
Şematik sembol</th></tr>
<tr><td>1</td><td>ld

void ld(_Bool in){
if (in==0) TEMP=0;
else TEMP=1;
}</td><td><table><tr><td>IN</td><td>OUT</td></tr><tr><td>in</td><td>TEMP</td></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table></td><td> 
</td></tr>
<tr><td>2</td><td>ld_not

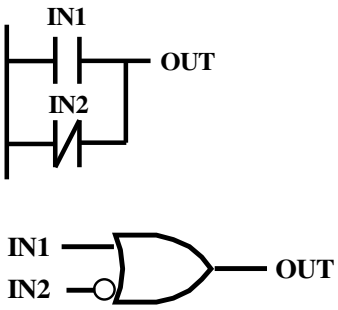
void ld_not(_Bool in){
if (in==0) TEMP=1;
else TEMP=0;
}</td><td><table><tr><td>IN</td><td>OUT</td></tr><tr><td>in</td><td>TEMP</td></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table></td><td> 
</td></tr>
<tr><td>3</td><td>not

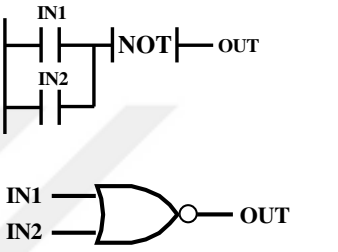
void not (void){
if (TEMP) TEMP=0;
else TEMP=1;
}</td><td><table><tr><td>IN</td><td>OUT</td></tr><tr><td>TEMP</td><td>TEMP</td></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table></td><td> 
</td></tr>
<tr><td>4</td><td>or

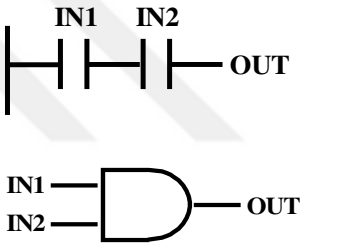
void or (_Bool in)
{
if (TEMP || in) TEMP=1;
else TEMP=0;
}</td><td><table><tr><td>IN1</td><td>IN2</td><td>OUT</td></tr><tr><td>TEMP</td><td>in</td><td>TEMP</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table></td><td> 
</td></tr>
</table>

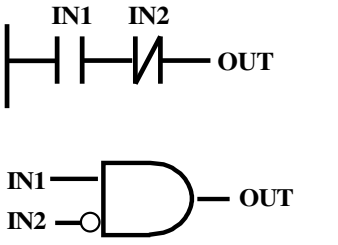
Tablo 5.1. Kontak ve röle temelli fonksiyonlar (devam)

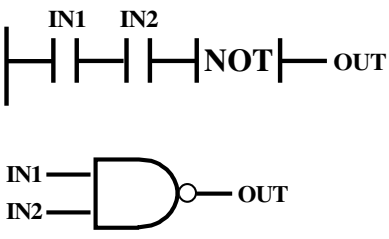
<table>
<tr>
<td>5</td>
<td>or_not

void or_not (_Bool in)
{
if (TEMP==1 || in==0)
TEMP=1;
else TEMP=0;
}</td>
<td>
<table>
<tr><th>IN1</th><th>IN2</th><th>OUT</th></tr>
<tr><td>TEMP</td><td>in</td><td>TEMP</td></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>6</td>
<td>nor

void nor (_Bool in)
{
if (TEMP==0 && in==0)
TEMP=1;
else TEMP=0;
}</td>
<td>
<table>
<tr><th>IN1</th><th>IN2</th><th>OUT</th></tr>
<tr><td>TEMP</td><td>in</td><td>TEMP</td></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>7</td>
<td>and

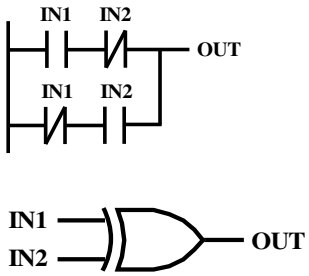
void and (_Bool in)
{
if (TEMP && in)
TEMP=1;
else TEMP=0;
}</td>
<td>
<table>
<tr><th>IN1</th><th>IN2</th><th>OUT</th></tr>
<tr><td>TEMP</td><td>in</td><td>TEMP</td></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td>
<td> 
</td>
</tr>
<tr>
<td>8</td>
<td>and_not

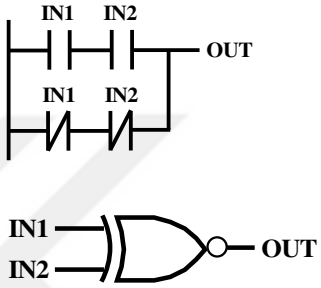
void and_not (_Bool in)
{
if (TEMP==1 && in==0)
TEMP=1;
else TEMP=0;
}</td>
<td>
<table>
<tr><th>IN1</th><th>IN2</th><th>OUT</th></tr>
<tr><td>TEMP</td><td>in</td><td>TEMP</td></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>0</td></tr>
</table>
</td>
<td> 
</td>
</tr>
<tr>
<td>9</td>
<td>nand

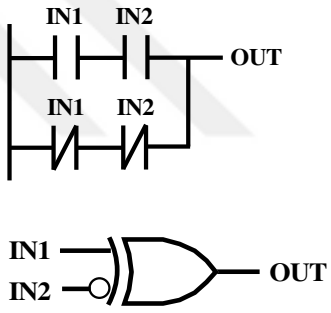
void nand (_Bool in)
{
if (TEMP && in)
TEMP=0;
else TEMP=1;
}</td>
<td>
<table>
<tr><th>IN1</th><th>IN2</th><th>OUT</th></tr>
<tr><td>TEMP</td><td>in</td><td>TEMP</td></tr>
<tr><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>0</td></tr>
</table>
</td>
<td> 
</td>
</tr>
</table>

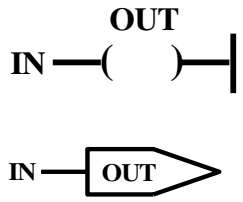
Tablo 5.1. Kontak ve röle temelli fonksiyonlar (devam)

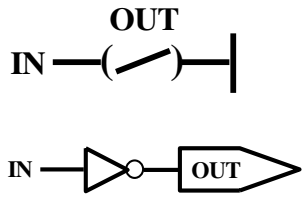
<table>
<tr>
<td>10</td>
<td>xor

void xor (_Bool in)
{
if (TEMP != in)
TEMP=1;
else TEMP=0;
}</td>
<td>
<table>
<tr><td>IN1</td><td>IN2</td><td>OUT</td></tr>
<tr><td>TEMP</td><td>in</td><td>TEMP</td></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>0</td></tr>
</table>
</td>
<td> 
</td>
</tr>
<tr>
<td>11</td>
<td>xnor

void xnor (_Bool in)
{
if (TEMP == in)
TEMP=1;
else TEMP=0;
}</td>
<td>
<table>
<tr><td>IN1</td><td>IN2</td><td>OUT</td></tr>
<tr><td>TEMP</td><td>in</td><td>TEMP</td></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>1</td></tr>
</table>
</td>
<td> 
</td>
</tr>
<tr>
<td>12</td>
<td>xor_not

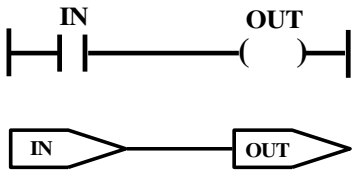
void xor_not (_Bool in)
{
in=!in;
if (TEMP != in)
TEMP=1;
else TEMP=0;
}</td>
<td>
<table>
<tr><td>IN1</td><td>IN2</td><td>OUT</td></tr>
<tr><td>TEMP</td><td>in</td><td>TEMP</td></tr>
<tr><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td>
<td> 
</td>
</tr>
<tr>
<td>13</td>
<td>out

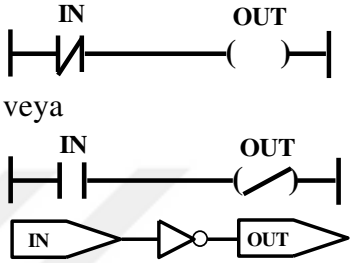
_Bool out (void)
{
if (TEMP==0) TEMP=0;
else TEMP=1;
return TEMP;
}</td>
<td>
<table>
<tr><td>IN</td><td>OUT</td></tr>
<tr><td>TEMP</td><td>TEMP</td></tr>
<tr><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td></tr>
</table>
</td>
<td> 
</td>
</tr>
<tr>
<td>14</td>
<td>out_not

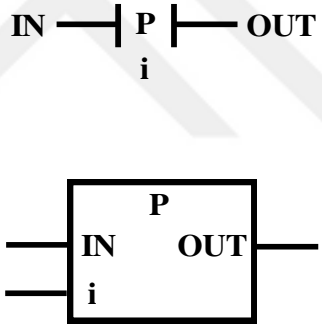
_Bool out_not (void)
{
if (TEMP==0) TEMP=1;
else TEMP=0;
}</td>
<td>
<table>
<tr><td>IN</td><td>OUT</td></tr>
<tr><td>TEMP</td><td>TEMP</td></tr>
<tr><td>0</td><td>1</td></tr>
<tr><td>1</td><td>0</td></tr>
</table>
</td>
<td> 
</td>
</tr>
</table>

Tablo 5.1. Kontak ve röle temelli fonksiyonlar (devam)

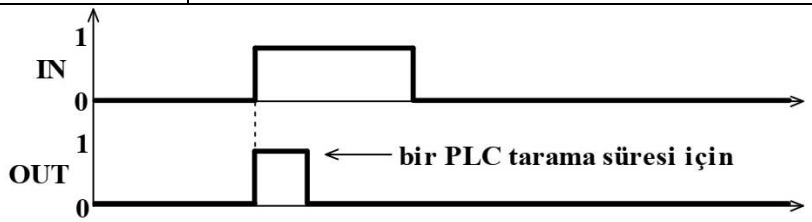
<table>
<tr>
<td>15</td>
<td>in_out

void in_out (_Bool in)
{
if (in) TEMP=1;
else TEMP=0;
}</td>
<td>
<table>
<tr><td>IN</td><td>OUT</td></tr>
<tr><td>in</td><td>TEMP</td></tr>
<tr><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td></tr>
</table>
</td>
<td> 
</td>
</tr>
<tr>
<td>16</td>
<td>inv_out

_Bool inv_out (_Bool in)
{
return (!in);
}</td>
<td>
<table>
<tr><td>IN</td><td>OUT</td></tr>
<tr><td>in</td><td>(!in)</td></tr>
<tr><td>0</td><td>1</td></tr>
<tr><td>1</td><td>0</td></tr>
</table>
</td>
<td> 
</td>
</tr>
<tr>
<td rowspan="2">17</td>
<td>r_edge

void r_edge (const int i)
{
if(TEMP==0)
{
RED[i]=1;
}
else if (RED[i]==1)
{
TEMP=1
RED[i] =0;
}
else
{
TEMP=0;
}
}</td>
<td colspan="2"> 

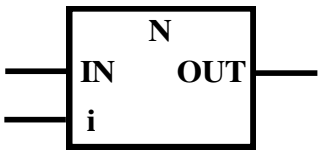
IN = TEMP
OUT = TEMP

RED[i], i = 0, 1, …, 63</td>
</tr>
<tr>
<td colspan="2"> 

r_edge fonsiyonunun zamanlama diyagramı</td>
</tr>
</table>

Tablo 5.1. Kontak ve röle temelli fonksiyonlar (devam)

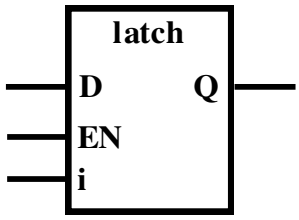
<table>
<tr>
<td rowspan="2">18</td>
<td>f_edge
void f_edge (const int i)
{
if(TEMP)
{
FED[i]=1;
TEMP=0;
}
else if (FED[i]==1)
{
TEMP=1;
FED[i]=0;
}
else
{
TEMP=0;
}
}</td>
<td> 



IN = TEMP
OUT = TEMP

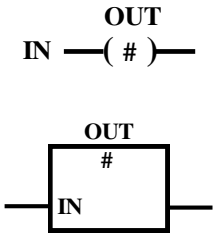
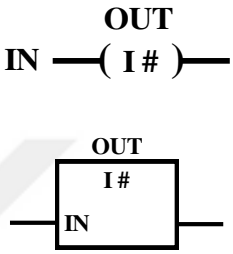
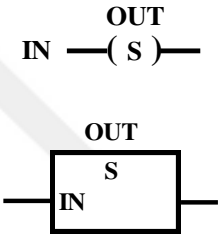
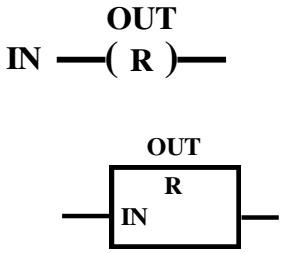
FED[i], i = 0, 1, …, 63</td>
</tr>
<tr>
<td colspan="2"> 

f_edge fonsiyonunun zamanlama diyagramı</td>
</tr>
<tr>
<td>19</td>
<td>latch

_Bool latch(_Bool D, const int i)
{
if (TEMP)
LB[i]=D;
return LB[i];
}</td>
<td><table>
<tr><th>EN</th><th>D</th><th>Q_t</th><th>Q_{t+1}</th><th>Yorum</th></tr>
<tr><td>TEMP</td><td>D</td><td>LB [i]</td><td>LB [i]</td><td></td></tr>
<tr><td>0</td><td>×</td><td>Q_t</td><td>Q_t</td><td>Değişim yok</td></tr>
<tr><td>1</td><td>0</td><td>×</td><td>0</td><td>Reset</td></tr>
<tr><td>1</td><td>1</td><td>×</td><td>1</td><td>Set</td></tr>
</table>
× : (don’t care) ne olursa olsun.</td>
<td> 

LB[i], i = 0, 1, …, 63</td>
</tr>
</table>

Tablo 5.1. Kontak ve röle temelli fonksiyonlar (devam)

<table>
<tr>
<td>20</td>
<td>mid_out
_Bool mid_out (_Bool m_out)
{
m_out=TEMP;
return m_out;
}</td>
<td>
<table>
<tr><td>IN</td><td>OUT</td></tr>
<tr><td>TEMP</td><td>m_out</td></tr>
<tr><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td></tr>
</table>
</td>
<td> 
</td>
</tr>
<tr>
<td>21</td>
<td>mid_out_not
_Bool mid_out_not (_Bool m_out)
{
m_out=!TEMP;
return m_out;
}</td>
<td>
<table>
<tr><td>IN</td><td>OUT</td></tr>
<tr><td>TEMP</td><td>m_out</td></tr>
<tr><td>0</td><td>1</td></tr>
<tr><td>1</td><td>0</td></tr>
</table>
</td>
<td> 
</td>
</tr>
<tr>
<td>22</td>
<td>_set
_Bool _set (_Bool out)
{
if (TEMP) out=1;
return (out);
}</td>
<td>
<table>
<tr><td>IN</td><td>OUT</td></tr>
<tr><td>TEMP</td><td>out</td></tr>
<tr><td>0</td><td>Değişim yok</td></tr>
<tr><td>1</td><td>Set</td></tr>
</table>
</td>
<td> 
</td>
</tr>
<tr>
<td>23</td>
<td>_reset
_Bool _reset (_Bool out)
{
if (TEMP) out=0;
return (out);
}</td>
<td>
<table>
<tr><td>IN</td><td>OUT</td></tr>
<tr><td>TEMP</td><td>out</td></tr>
<tr><td>0</td><td>Değişim yok</td></tr>
<tr><td>1</td><td>Reset</td></tr>
</table>
</td>
<td> 
</td>
</tr>
</table>

Tablo 5.1. Kontak ve röle temelli fonksiyonlar (devam)

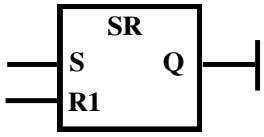
24

SR

```
_Bool SR (_Bool S, _Bool
R1, _Bool out) {
if (R1) out=0;
else if (S) out=1;
return (out);
}
```

S	R1	Q_{t+1}	Yorum
S	R1	out	
0	0	Q_t	Değişim yok
1	0	1	Set
×	1	0	Reset

× : ne olursa olsun.



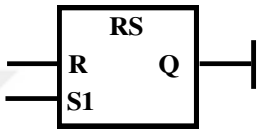
25

RS

```
_Bool RS (_Bool R, _Bool
S1, _Bool out)
{
if (S1) out=1;
else if (R) out=0;
return (out);
}
```

R	S1	Q_{t+1}	Yorum
R	S1	out	
0	0	Q_t	Değişim yok
1	0	0	Reset
×	1	1	Set

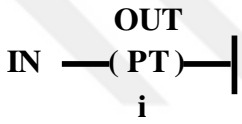
× : ne olursa olsun.

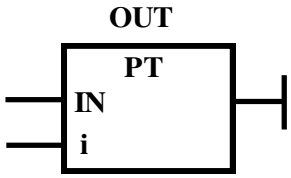


26

r_toggle

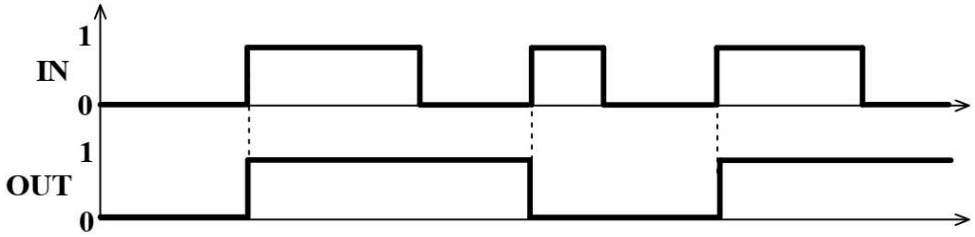
```
_Bool r_toggle(const int i,
_Bool out)
{
        if(TEMP==0)
        {

RED_TOGGLE[i]=1;
        }
        else                if
(RED_TOGGLE[i]==1)
        {
        RED_TOGGLE[i]
=0;              out=!out;
        }
        return out;
}
```





IN = TEMP
OUT = out

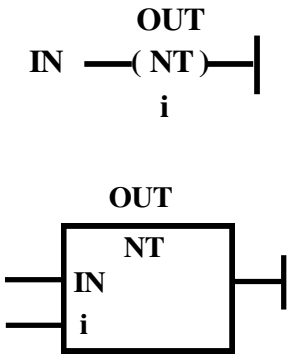
RED_TOGGLE [i], i = 0, 1, …, 63



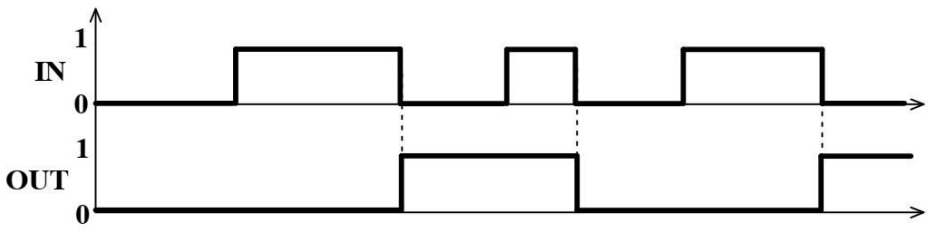
r_toggle fonsiyonunun zamanlama diyagramı

Tablo 5.1. Kontak ve röle temelli fonksiyonlar (devam)

<table>
<tr>
<td rowspan="2">27</td>
<td>f_toggle

_Bool f_toggle(const int i, _Bool out)
{
if(TEMP==1)
{
FED_TOGGLE[i]=1;
}
else if (FED_TOGGLE[i]==1)
{
FED_TOGGLE[i] =0;
out = !out;
}
return out;
}</td>
<td> 

IN = TEMP
OUT = out

FED_TOGGLE [i], i = 0, 1, …, 63</td>
</tr>
<tr>
<td colspan="2"> 

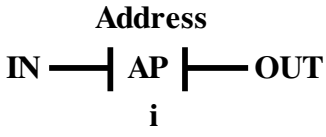
f_toggle fonsiyonunun zamanlama diyagramı</td>
</tr>
</table>

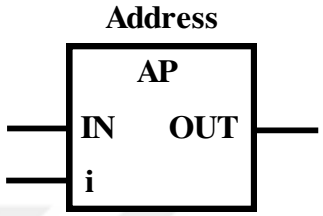
Tablo 5.1. Kontak ve röle temelli fonksiyonlar (devam)

28

adrs_re

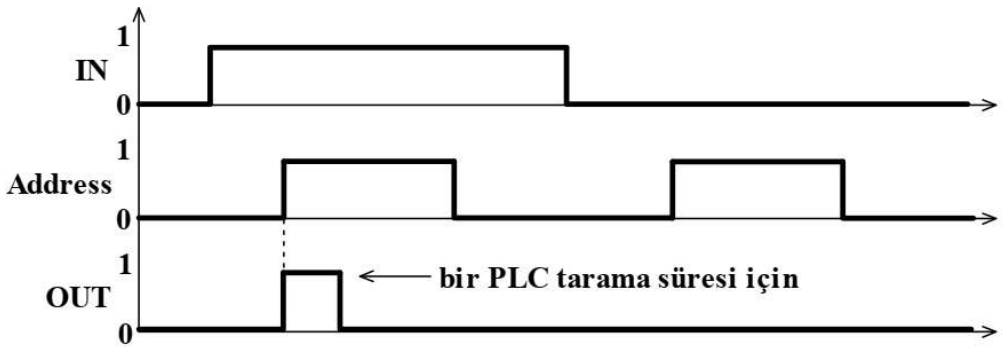
```
void adrs_re(const int i, _Bool adrs)
{
if(TEMP && (adrs==0) )
        {
        adrs_RED[i]=1;
        TEMP=0;
        }
if(TEMP && (adrs==0) && (adrs_RED[i]==0))
        {
        TEMP=0;
        }
if(TEMP && (adrs==1) && (adrs_RED[i]==1))
        {
         adrs_RED[i]=0;
         TEMP=1;
        }
else    {
        TEMP=0;
        }
}
```





IN = TEMP
Address = adrs
OUT = TEMP

adrs_RED [i], i = 0, 1, …, 63



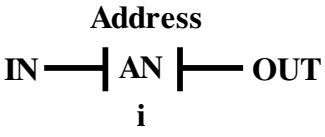
adrs_re fonsiyonunun zamanlama diyagramı

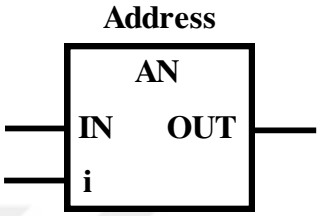
Tablo 5.1. Kontak ve röle temelli fonksiyonlar (devam)

29

adrs_fe

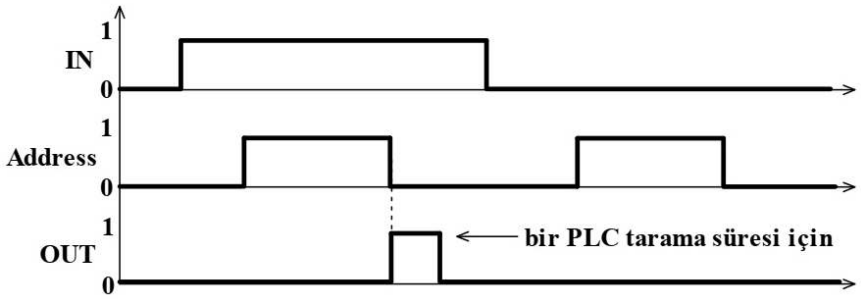
```
void adrs_fe(const int i,_Bool adrs)
{
if(TEMP && (adrs==1) )
        {
        adrs_FED[i]=1;
        TEMP=0;
        }
if(TEMP && (adrs==1) && (adrs_FED[i]==0))
        {
        TEMP=0;
        }
if(TEMP && (adrs==0) && (adrs_FED[i]==1))
        {
        adrs_FED[i]=0;
         TEMP=1;
        }
else    {
        TEMP=0;
        }
}
```





IN = TEMP
Address = adrs
OUT = TEMP

adrs_FED [i], i = 0, 1, …, 63



adrs_fe fonsiyonunun zamanlama diyagramı

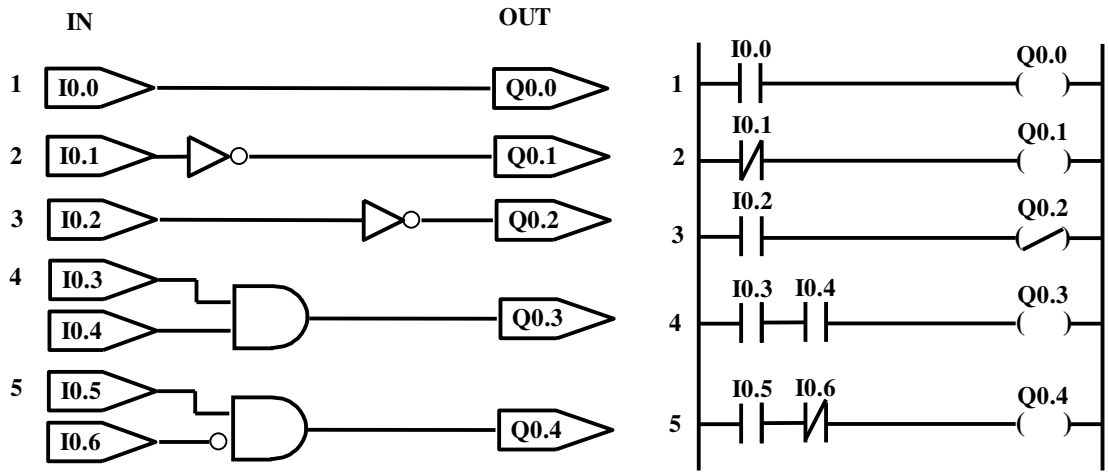
5.1. Kontak ve Röle Temelli Fonksiyon Örnekleri

Bu kısımda kontak ve röle temelli fonksiyonlara ait örnekler yer almaktadır. Her bir örnek için şematik diyagram, merdiven diyagramı ve kullanıcı programına ait C kodu sunulmuştur.

5.1.1. Örnek 1

Şekil 5.1'de şematik ve merdiven diyagramı görülen Örnek 1'e ait kullanıcı programı aşağıda verilmiştir:

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

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

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

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

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



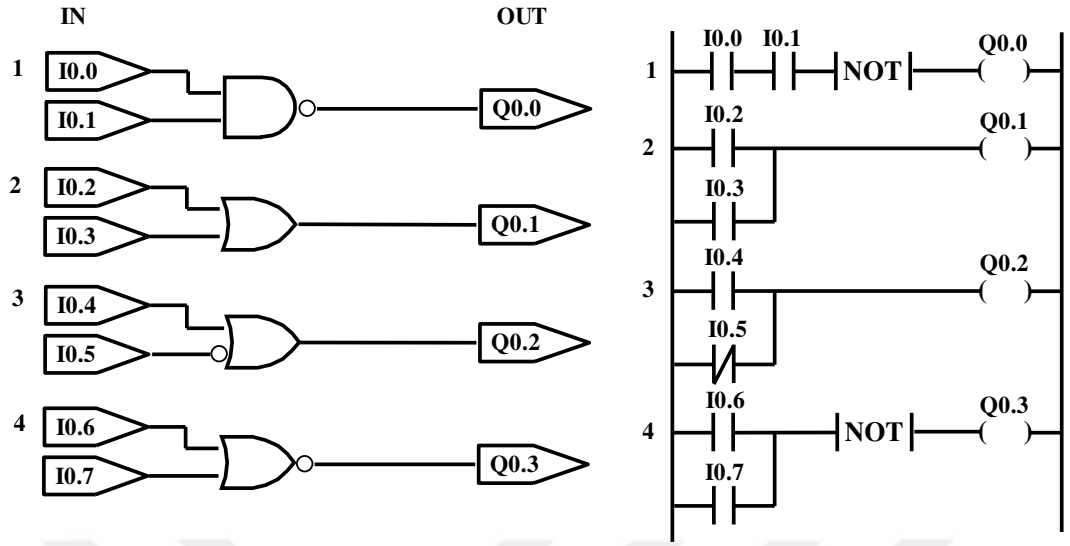
Şekil 5.1. Örnek 1'e ait şematik diyagram ve merdiven diyagramı

Şekil 5.1'de görülen Örnek 1'e ait oluşturulan C kodunda: Basamak 1'de, giriş I0.0'dan bir değeri yükler ve bu değeri Q0.0 çıkışına gönderir. Basamak 2'de, I0.1 girişinden bir değeri yükler ve bu değerin mantıksal değilini alır, ardından bu değeri Q0.1 çıkışına gönderir. Basamak 3'te, I0.2 girişinden bir değeri yükler, bu değerin mantıksal değilini out_not () fonksiyonu yardımıyla Q0.2 çıkışına gönderir. Basamak 4'te, I0.3 girişinden bir değeri yükler, ardından bu değeri I0.4 girişinden gelen bir değerle mantıksal VE işlemine tabi tutar ve sonucu Q0.3 çıkışına gönderir. Basamak 5'te, I0.5 girişinden bir değeri yükler, ardından bu değeri I0.6 girişinden gelen bir değerin değiliyle mantıksal VE işlemine tabi tutar ve sonucu Q0.4 çıkışına gönderir.

5.1.2. Örnek 2

Şekil 5.2'de şematik ve merdiven diyagramı görülen Örnek 2'ye ait kullanıcı programı aşağıda verilmiştir:

```
ld (I00);              //basamak1
nand (I01);
Q00 = out();

ld (I02);              //basamak2
or (I03);
Q01 = out();

ld (I04);              //basamak3
or_not (I05);
Q02 = out();

ld (I06);              //basamak4
nor (I07);
Q03 = out();
```



Şekil 5.2. Örnek 2'e ait şematik diyagram ve merdiven diyagramı

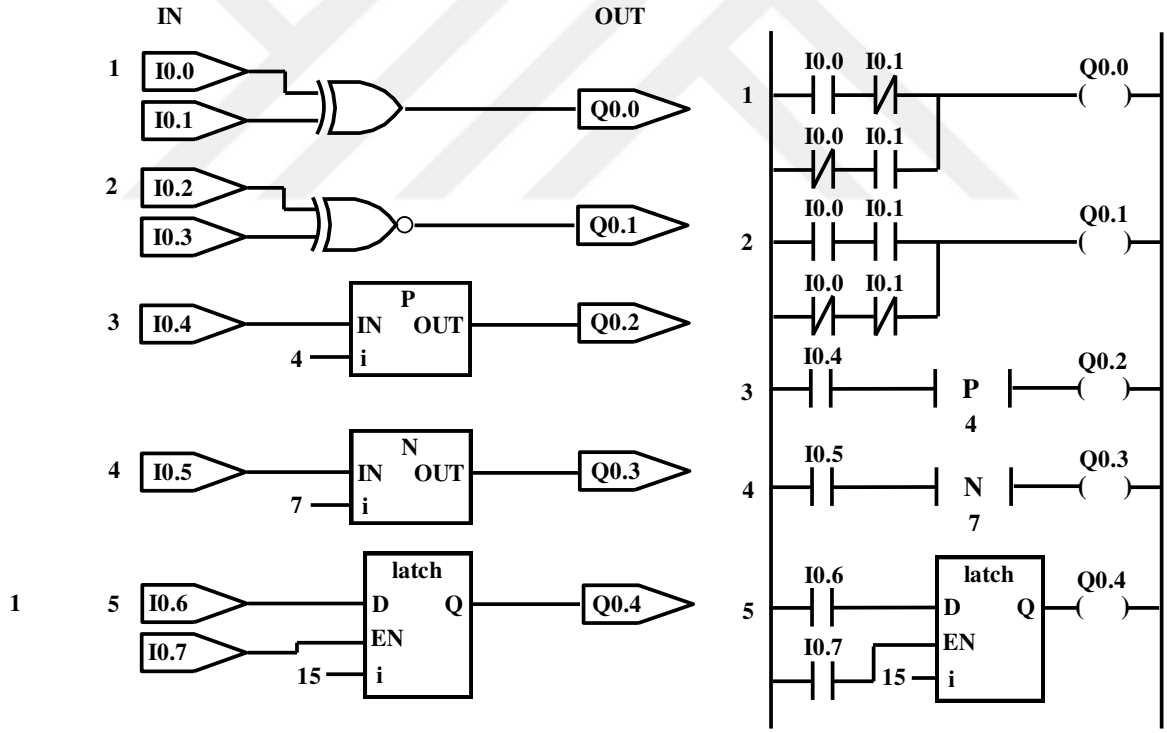
Şekil 5.2'te görülen Örnek 2'ye ait oluşturulan C kodunda: Basamak 1'de, I0.0'dan yüklenen değer ile I0.1'den yüklenen değer NAND işlemine tabi tutulur ve bulunan sonuç Q0.0 çıkışına gönderilir. Basamak 2'de, I0.2'den yüklenen değer ile I0.3'ten yüklenen değer OR işlemine tabi tutulur ve bulunan sonuç Q0.1 çıkışına gönderilir. Basamak 3'te, I0.4'ten yüklenen değer ile I0.5'ten yüklenen değerin değili OR işlemine tabi tutulur ve bulunan sonuç Q0.2 çıkışına gönderilir. Basamak 4'te, I0.6'dan yüklenen değer ile I0.7'den yüklenen değer NOR işlemine tabi tutulur ve bulunan sonuç Q0.3 çıkışına gönderilir.

5.1.3. Örnek 3

Şekil 5.3'te şematik ve merdiven diyagramı görülen Örnek 3'e ait kullanıcı programı aşağıda verilmiştir:

```
ld (I00);                //basamak1
xor (I01);
Q00 = out();

ld (I02);                //basamak2
xnor (I03);
Q01 = out();
```

```
ld (I04);                    //basamak3
r_edge (4);
Q02 = out();

ld (I05);                    //basamak4
f_edge (7);
Q03 = out();

ld (I07);                    //basamak5
Q04 = latch(I06,15);
```



Şekil 5.3. Örnek 3'e ait şematik diyagram ve merdiven diyagramı

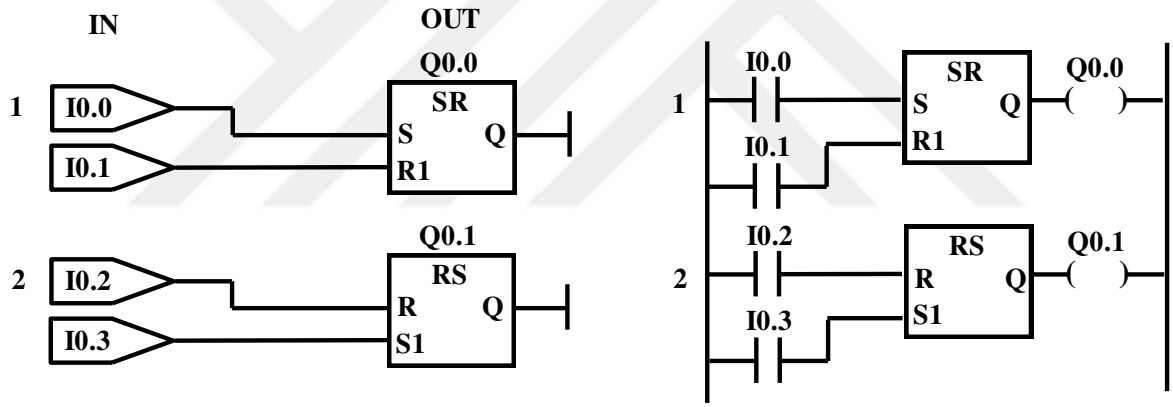
Şekil 5.3'te görülen Örnek 3'e ait oluşturulan C kodunda: Basamak 1'de, I0.0'dan yüklenen değer ile I0.1'den yüklenen değer EXOR işlemine tabi tutulur ve bulunan sonuç Q0.0 çıkışına gönderilir. Basamak 2'de, I0.2'den yüklenen değer ile I0.3'ten yüklenen

değer EXNOR işlemine tabi tutulur ve bulunan sonuç Q0.1 çıkışına gönderilir. Basamak 3'te, I0.4'ten alınan giriş sinyalinin yükselen kenarı algılanarak sonuç Q0.2 çıkışına gönderilir. Basamak 4'te, I0.5'ten alınan giriş sinyalinin düşen kenarı algılanarak sonuç Q0.3 çıkışına gönderilir. Basamak 5'te, indeks değeri i = 15, yetkileme girişi EN = I0.7, D = I0.6 ve çıkışı Q = Q0.4 olan bir latch gerçekleştirilmiştir.

5.1.4. Örnek 4

Şekil 5.4'te şematik ve merdiven diyagramı görülen Örnek 4'e ait kullanıcı programı aşağıda verilmiştir:

```
Q00 = SR (I00, I01, Q00);     //basamak1

Q01 = RS (I02, I03, Q01);     //basamak2
```



Şekil 5.4. Örnek 4'e ait şematik diyagram ve merdiven diyagramı

Şekil 5.4'te görülen Örnek 4'e ait oluşturulan C kodunda: Basamak 1'de, giriş sinyalleri S = I0.0, R1 = I0.1 ve çıkışı Q = Q0.0 olan reset öncelikli bir SR fonksiyonu gerçekleştirilmiştir. Basamak 2'de, giriş sinyalleri R = I0.2, S1 = I0.3 ve çıkışı Q = Q0.1 olan set öncelikli bir RS fonksiyonu gerçekleştirilmiştir.

6. SAYICI FONKSİYONLARI

PLC'de sayıcılar, dahili veya harici olayların veya sinyallerin, yukarı ya da aşağıya doğru sayılmasını sağlayan fonksiyonlardır. PLC'lerde genel olarak üç temel sayma fonksiyonu kullanılmaktadır: İleri Sayıcı (Up Counter-CTU), Geri Sayıcı (Down Counter-CTD), İleri/Geri Sayıcı (Up/Down Counter-CTUD). Bu fonksiyonlar sayıcı içeriğini arttıran veya azaltan ve içerik belli bir değere ulaştığında sayıcı durum kaydedicisine veriyi ileterek çalışmaktadır. Dolayısıyla bu bölümde üç temel sayma fonksiyonu açıklanarak bu tez projesi kapsamında bu fonksiyonların nasıl gerçekleştirildiği anlatılmaktadır.

6.1. İleri Sayıcı (Up Counter-CTU)

İleri Sayıcı (Up Counter-CTU) bir sayma değerinin kurma değerine ulaştığını belirlemek amacıyla kullanılır. Tablo 6.1'de İleri Sayıcı (Up Counter-CTU)'nın sembolü ve doğruluk tablosu ve Şekil 6.1'de ileri sayıcı fonksiyonu için oluşturulan C kodu yer almaktadır. Tablo 6.1'de belirtildiği üzere ileri sayıcı, **CU** girişinde tespit edilen yükselen kenar, **CV** değerinin bir değer arttırılmasına neden olur. Sayıcı **PV** değerine ulaştığında sayıcı çıkışı Q set (ON-1) yapılır. Sayıcı çıkışı Q set olduktan sonra CU girişindeki her bir yükselen kenar CV değerinin bir değer arttırılmasına sebep olur. CU girişindeki her bir yükselen kenar sayıcının sayma değeri maksimum sayma değerine (PVmax) ulaşana kadar bir değer arttırılmasını sağlar. PVmax sayıcının çözünürlüğüne bağlıdır. Örneğin: 8 bitlik bir sayıcının PVmax değeri = 2^8 = 255'tir. 32 bitlik bir sayıcının PVmax değeri = 2^{32} = 4.294.967.296'dır. R reset girişi sayıcının çıkışını 0 (OFF) yapmak ve CV sayma değerini sıfırlamak için kullanılır (Uzam, 2022).

Şekil 6.1'de görülen C kodunda belirtiği üzere fonksiyona ait iki tane Boole-giriş-değişkeni (ileri sayma girişi CU ve reset girişi Reset (bir sıfırlama sinyali), bir tane Boole-çıkış değişkeni Q ve giriş sinyali CU'nun yükselen kenarını algılamak amacıyla kullanılan özgün bir Bool değişkeni (CURED[i]) vardır. Kurma değeri PV, 1 - 4.294.967.296 arası bir sayıyı ifade eder. İndeks değeri i, 0 ile 63 arası bir sayıdır ve CU giriş sinyalinin yükselen kenarını algılamak için kullanılacak CURED[i] değişkenini, sayıcı çıkışını ifade

eden CQ[i] değişkenini ve sayma değerlerini tutmak için kullanılacak olan CV[i] değişkenini belirlemek için kullanılmaktadır.

Şekil 6.1'de görülen C kodunda şu işlemler yapılmaktadır:

İndeks değeri i 64'ten küçükse:

Reset girişi R'deki sinyal 1'e eşitse sayma değeri CV[i] sıfırlanır.

Reset girişi R'deki sinyal 0'a eşitse, aşağıdaki işlemler gerçekleştirilir:

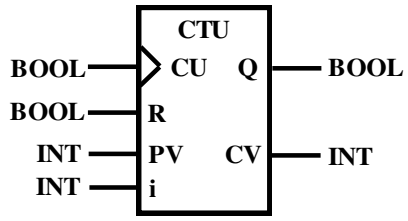
CU 0'a eşitse, CURED[i] 1 yapılır.

CU 1'e eşitse, CURED[i] 1 ise ve CV[i] 0xFFFFFFFF'ten (4.294.967.296) küçükse

CURED[i] sıfırlanır ve CV[i] 1 arttırılır.

Son olarak CV[i]'deki değer (sayma değeri) PV (işaretsiz bir tam sayı)'den büyük veya eşitse çıkış CQ[i] set edilir. CV[i]'deki değer PV'den küçükse CQ[i] reset edilir.

Tablo 6.1. İleri sayıcı (Up Counter-CTU)nın sembolü ve doğruluk tablosu

<table>
<tr><th>Sembol</th><th colspan="3">Doğruluk Tablosu</th></tr>
<tr><td rowspan="8"> 

CU: İleri sayma girişi
R: Reset girişi
PV: Preset count value (Kurma değeri)
Q: Sayıcı Çıkışı
CV: Current count value (Sayma değeri)
i: İndeks değeri, i = 0, 1, …, 63</td><td>CU</td><td>R</td><td>İşlem</td></tr>
<tr><td>×</td><td>1</td><td>Sayma değeri CV'yi sıfır yap (CV:= 0).</td></tr>
<tr><td>0</td><td>0</td><td>NOP (No Operation – hiçbir işlem yapma)</td></tr>
<tr><td>1</td><td>0</td><td>NOP</td></tr>
<tr><td>↓</td><td>0</td><td>NOP</td></tr>
<tr><td>↑</td><td>0</td><td>Eğer CV < PVmax ise CV'yi bir arttır (CV:= CV + 1).</td></tr>
<tr><td>×</td><td>×</td><td>Eğer CV ≥ PV ise sayıcı çıkışı Q'yu set et (ON – 1),
değilse sayıcı çıkışı Q'yu reset et (OFF – 0).</td></tr>
</table>

×: ne olursa olsun

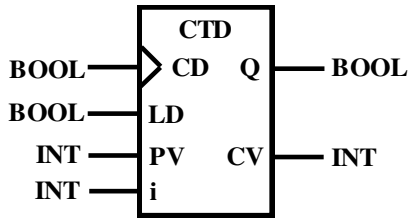
```
void CTU (_Bool CU,_Bool R,unsigned int PV,unsigned int i)
{
        if (i<64)
        {
                if(R==1)
                {
            CV[i]=0;}
                else
                  {
                     if(CU==0) {CURED[i]=1;}
                     if( (CU==1) && (CURED[i]==1) && (CV[i]< 0xFFFFFFFF))
                      { CURED[i]=0;
                        CV[i]++;
                      }
                  }

           if (CV[i]>=PV) CQ[i]=1;
           else CQ[i]=0;
        }
}
```

Şekil 6.1. İleri sayıcı (Up Counter-CTU) için geliştirilen C kodu

6.2. Geri Sayıcı (Down Counter-CTD)

Geri sayıcı (down counter-CTD) bir sayma değerinin belirli bir set değerinden geri doğru sayma esnasında sıfır değerine ulaştığını belirlemek için kullanılır. Tablo 6.2'de geri sayıcının sembolü ve doğruluk tablosu ve Şekil 6.2'de fonksiyon için geliştirilen C kodu yer almaktadır. Şekil 6.2'den görüldüğü üzere kurma değeri **PV**, sayıcının saymaya başlayacağı sayı değerini belirler. **CD** girişindeki her bir yükselen kenar, eğer CV>0 ise CV değerinin bir değer azalmasına neden olur. Sayıcının sayma değeri 0 olduğunda sayıcı çıkışı Q set (ON-1) yapılır ve geri sayma işlemi durur. **LD** yükleme girişi, çıkış **Q**'nun 0 (OFF) yapılmasına ve CV'ye PV kurma değerinin yüklenmesine neden olur (Uzam, 2022).

Tablo 6.2. Geri sayıcı (Down Counter-CTD)'nın sembolü ve doğruluk tablosu

<table>
<tr><th>Sembol</th><th colspan="3">Doğruluk Tablosu</th></tr>
<tr><td rowspan="6"> 

CD: Count down input
(Geri sayma girişi)
LD: Load input
(Yükleme girişi)
PV: Preset count value (Kurma değeri)
Q: Counter output
(Sayıcı çıkışı)
CV: Current count value (Sayma değeri)
i: İndeks değeri, i = 0, 1, …, 63</td><td>CU</td><td>R</td><td>İşlem</td></tr>
<tr><td>×</td><td>1</td><td>Sayma değeri CV'ye kurma değeri PV'yi yükle (CV := PV).</td></tr>
<tr><td>0</td><td>0</td><td>NOP (No Operation – hiç bir işlem yapma)</td></tr>
<tr><td>1</td><td>0</td><td>NOP</td></tr>
<tr><td>↓</td><td>0</td><td>NOP</td></tr>
<tr><td>↑</td><td>0</td><td>Eğer CV > 0 ise CV'yi bir azalt
(CV = CV - 1).</td></tr>
<tr><td>×</td><td>×</td><td>Eğer CV = 0 ise CV'yi değiştirmeden tut ve sayıcı çıkışı Q'yu set et (ON – 1) değilse sayıcı çıkışı Q'yu reset et (OFF – 0).</td></tr>
</table>

×: ne olursa olsun

```
void CTD (_Bool CD, _Bool Load, unsigned int PV, unsigned int i)
{
        if (i<64)
        {
                if(LD==1)
                {
                CV[i]=PV;
                }
                else
                  {
                    if(CD==0) {CDRED[i]=1;}
                    if((CD==1) &&  (CDRED[i]==1) && (CV[i]>0))
                      {
                        CDRED[i]=0;
                         CV[i]--;
                      }
                    }
   if (CV[i]==0) CQ[i]=1;
   else CQ[i]=0;
        }
}
```

Şekil 6.2. Geri sayıcı (Down Counter-CTD) için geliştirilen C kodu

Şekil 6.2'de görülen C kodunda fonksiyona ait iki tane Boole-giriş-değişkeni (geri sayma girişi CD ve yükleme girişi LD, bir tane Boole-çıkış değişkeni Q ve giriş sinyali CD'nin yükselen kenarını algılamak amacıyla kullanılan özgün bir Bool değişkeni (CDRED[i]) vardır. İndeks değeri i, 0 ile 63 arası bir sayıdır ve CD giriş sinyalinin yükselen kenarını algılamak için kullanılacak CDRED[i] değişkenini, sayıcı çıkışını ifade eden CQ[i] değişkenini ve sayma değerlerini tutmak için kullanılacak olan CV[i] değişkenini belirlemek için kullanılmaktadır. Kurma değeri PV, 1-4.294.967.296 arası bir sayıyı ifade eder.

Şekil 6.2'de görülen C kodunda şu işlemler yapılmaktadır:

İndeks değeri i 64'ten küçükse:

LD = 1 olduğunda, CV[i], PV değerine set edilir.

LD = 0 olduğunda, aşağıdaki işlemler gerçekleştirilir:

CD 0'a eşitse, CDRED[i] set edilir (1 yapılır).

CD 1'e eşitse, CDRED[i] 1 ise ve CV[i] 0'dan büyükse:

CDRED[i] sıfırlanır ve CV[i] 1 azaltılır.

CV[i] 0'a eşitse CQ[i] set edilir. CV[i] 0'a eşit değilse CQ[i] reset edilir.

6.3. İleri/Geri Sayıcı (Up/Down Counter-CTUD)

İleri/geri sayıcı için "CU (ileri sayma)" ve "CD (geri sayma)" olmak üzere iki giriş bulunmaktadır. "CU" girişinde tespit edilen yükselen kenarda ileri sayma, "CD" girişinde tespit edilen yükselen kenarda ise geri sayma işlemi yapılır. "PVmax" sayıcının çözünürlüğüne bağlı olan maksimum sayma değerini ifade eder ve 8 bitlik bir sayıcının; PVmax değeri = 2^8 = 255 iken, 32 bitlik bir sayıcının PVmax değeri = 2^{32} = 4.294.967.296'dır. Tablo 6.3'te ileri/geri sayıcının sembolü ve doğruluk tablosu ve Şekil 6.3'te ileri/geri sayıcı fonksiyonu için oluşturulan C kodu yer almaktadır. Tablo 6.3'ten görüleceği üzere ileri/geri sayıcı, CV < PVmax olduğunda CU girişindeki bir yükselen kenar, sayıcı sayma değeri CV'nin bir değer arttırılmasına neden olur. CV>0 olduğunda CD girişindeki her bir yükselen kenar, sayıcı sayma değeri CV'nin bir değer azalmasına neden olur.

CV ≥ PV olduğunda sayıcı çıkışı QU set (ON-1) yapılır.

CV < PV olduğunda ise sayıcı çıkışı QU reset (OFF-0)yapılır.

CV = 0 olduğunda sayıcı çıkışı QD set (ON-1) yapılır.

CV > 0 olduğunda ise sayıcı çıkışı QD reset (OFF-0) yapılır.

R reset girişi aktif olduğunda sayıcı çıkışı 0 (OFF) konumuna getirilir ve CV sayma değerini sıfırlanır. R reset girişi aktif değilken LD yükleme girişi aktif olduğunda ise, CV'ye PV kurma değeri yüklenir. Sayıcı sıfır değerine ulaştığında geri sayma durur. Aynı şekilde sayıcı PVmax değerine ulaşınca ileri sayma durur (Uzam, 2022)

Şekil 6.3'te görülen C kodunda belirtiği üzere fonksiyona ait dört tane Boole-giriş-değişkeni (ileri sayma değişkeni CU, geri sayma girişi CD ve yükleme girişi LD, reset girişi R ve iki tane Boole-çıkış değişkeni QU ve QD bulunur. CU'nun yükselen kenarını algılamak amacıyla kullanılan özgün bir Bool değişkeni (CURED[i]) ve geri sayma girişi CD'nin yükselen kenarını algılamak için kullanılan özgün bir Bool (CDRED[i]) değişkeni mevcuttur. İndeks değeri i, 0 ile 63 arası bir sayıdır ve CU giriş sinyalinin yükselen kenarını algılamak için kullanılacak CURED[i] değişkenini, CD giriş sinyalinin yükselen kenarını algılamak için kullanılacak CDRED[i] değişkenini, ileri sayıcı çıkışını ifade eden QU[i] değişkenini, geri sayıcı çıkışını ifade eden QD[i] değişkenini ve sayma değerlerini tutmak için kullanılacak olan CV[i] değişkenini belirlemek için kullanılmaktadır. Kurma değeri PV, 1 - 4.294.967.296 arası sabit bir sayıyı ifade eder.

Şekil 6.3'te görülen C kodunda şu işlemler yapılmaktadır:

İndeks değeri i 64'ten küçükse:

R = 1 ise sayma değeri CV[i] sıfırlanır.

R = 0 ise aşağıdaki işlemler gerçekleştirilir:

LD = 1 olduğunda, CV[i], PV değerine set edilir.

LD = 0 olduğunda, aşağıdaki işlemler gerçekleştirilir:

CU 0'a eşitse, CURED[i] set edilir (1 yapılır).

CD 0'a eşitse, CDRED[i] set edilir (1 yapılır).

CU 1'e eşitse, CDRED[i] 1 ise ve CV[i] 0xFFFFFFFF'ten (4.294.967.296) küçükse:

CURED[i] sıfırlanır ve CV[i] 1 arttırılır.

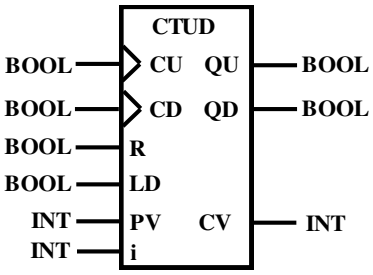
CD 1'e eşitse, CDRED[i] 1 ise ve CV[i] 0'dan büyükse:

CDRED[i] sıfırlanır ve CV[i] 1 azaltılır.

CV[i] PV'den büyük veya eşitse QU[i] set edilir. CV[i] PV'den küçükse QU[i] reset edilir.

CV[i] 0’a eşitse QD[i] set edilir. CV[i] 0’a eşit değilse QD[i] reset edilir.

Tablo 6.3. İleri/Geri sayıcı (Up/Down Counter-CTUD)'nın sembolü ve doğruluk tablosu

Sembol	Doğruluk Tablosu				
 CU: Count up input (İleri sayma girişi) CD: Count down input (Geri sayma girişi) R: Reset input (Reset girişi) LD: Load input (Yükleme girişi) PV: Preset count value (Kurma değeri) QU: Output up (İleri çıkışı) QD: Output down (Geri çıkışı) CV: Current count value (Sayma değeri) i: İndeks değeri, i = 0, 1, …, 63	R	LD	CU	CD	İşlem
	1	×	×	×	Sayma değeri CV'yi sıfır yap (CV:= 0).
	0	1	×	×	Sayma değeri CV'ye kurma değeri PV'yi yükle (CV:= PV).
	0	0	0	0	NOP (No Operation – hiç bir işlem yapma)
	0	0	0	1	NOP
	0	0	1	0	NOP
	0	0	1	1	NOP
	0	0	0	↓	NOP
	0	0	1	↓	NOP
	0	0	↓	0	NOP
	0	0	↓	1	NOP
	0	0	↓	↓	NOP
	0	0	↑	↑	NOP
	0	0	↑	0	Eğer CV < PVmax ise CV'yi bir arttır (CV := CV + 1).
	0	0	0	↑	Eğer CV > 0 ise CV'yi bir azalt (CV := CV – 1).
	×	×	×	×	Eğer CV ≥ PV ise QU çıkışını set et (ON - 1), değilse QU çıkışını reset et (OFF - 0).
	×	×	×	×	Eğer CV = 0 ise QD çıkışını set et (ON - 1), değilse QD çıkışını reset et (OFF - 0).

×: ne olursa olsun

```
void CTUD (_Bool CU,_Bool CD,_Bool R,_Bool LD,unsigned int PV, unsigned int i)
{
   if (i<64)
     {
          if(R==1)
          { CV[i]=0;}
          else
          {
              if(LD==1) {CV[i]=PV;}
              else
                {
                    if(CU==0) {CURED[i]=1;}
                    if(CD==0) {CDRED[i]=1;} //      if(!( ((CU==1) && (CURED[i]=1)) &&
((CD==1) && (CDRED[i]=1))
                   if( (CU==1) && (CURED[i]==1) && (CV[i]< 0xFFFFFFFF))
                        { CURED[i]=0;
                          CV[i]++;
                        }
                       if((CD==1) &&  (CDRED[i]==1) && (CV[i]>0))
                          {
                              CDRED[i]=0;
                              CV[i]--;
                          }
                       }
                  }
          if (CV[i]>=PV) QU[i]=1;
          else QU[i]=0;
          if (CV[i]==0) QD[i]=1;
          else QD[i]=0;
  }
}
```

Şekil 6.3. İleri/Geri sayıcı (Up/Down Counter-CTUD) için geliştirilen C kodu

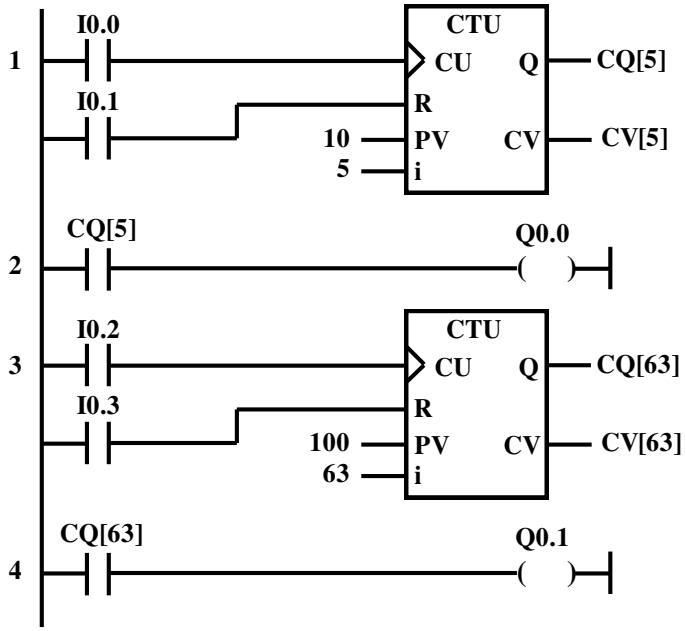
6.4. Sayıcı Fonksiyon Örnekleri

Bu kısımda sayıcı fonksiyonlarına ait örnekler yer almaktadır. Her bir örnek için merdiven diyagramı ve kullanıcı programına ait C kodu sunulmuştur.

6.4.1. Örnek 1

Şekil 6.4'te merdiven diyagramı görülen Örnek 1'e ait kullanıcı programı aşağıda verilmiştir:

```
CTU (I00,I01,10,5); //Basamak 1

in_out(CQ[5]);        //Basamak 2

Q00=out();


CTU (I02,I03,100,63); //Basamak 3

in_out(CQ[63]);       //Basamak 4

Q01=out();
```



Şekil 6.4. Örnek 1'e ait merdiven diyagramı

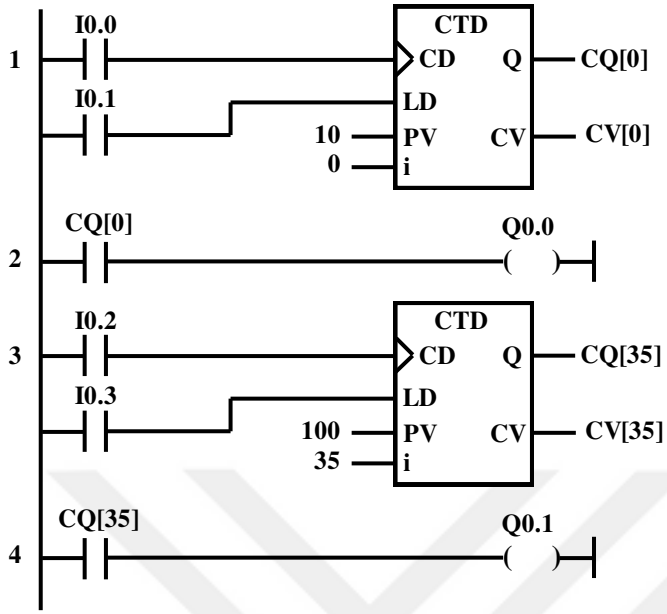
Şekil 6.4'te görülen Örnek 1'e ait oluşturulan C kodunda: Basamak 1'de, CTU fonksiyonu şu şekilde gerçekleştirilmiştir: İleri sayma girişi CU = I0.0, ve reset girişi R = I0.1 olarak belirlenmiştir. İndeks değeri i = 5 olduğu için, CU'nun yükselen kenarını algılamak için kullanılacak olan Bool değişkeni CURED[5] ve ileri sayıcı durum biti (ileri sayıcı çıkışı) CQ[5] olan 5 numaralı ileri sayıcı seçilmiştir. CU girişinden uygulanacak olan yükselen kenar sinyallerini saymak için kullanılan değişken CV[5] olur. Kurma değeri PV = 10 olduğu için CV[5] değişkendeki sayma değeri 10 veya daha büyük bir sayı olursa (CV ≥ 10) sayıcı durum biti (ileri sayıcı çıkışı) CQ[5] = 1 olur. Basamak 2'de, sayıcı durum biti CQ[5]'in değeri Q0.0 çıkışına gönderilir. Böylelikle CQ[5] sayıcı durum bitinin lojik seviyesi Q0.0'dan gözlemlenebilir. Basamak 3'te, CTU fonksiyonu şu şekilde gerçekleştirilmiştir: İleri sayma girişi CU = I0.2, ve reset girişi R = I0.3 olarak belirlenmiştir. İndeks değeri i = 63 olduğu için, CU'nun yükselen kenarını algılamak için kullanılacak olan Bool değişkeni CURED[63] ve ileri sayıcı durum biti (ileri sayıcı çıkışı) CQ[63] olan 63 numaralı ileri sayıcı seçilmiştir. CU girişinden uygulanacak olan yükselen kenar sinyallerini saymak için kullanılan değişken CV[63] olur. Kurma değeri PV = 100 olduğu için CV[63] değişkendeki sayma değeri 100 veya daha büyük bir sayı olursa (CV ≥ 100) sayıcı durum biti (ileri sayıcı çıkışı) CQ[63] = 1 olur. Basamak 4'te, sayıcı durum biti CQ[63]'in değeri Q0.1 çıkışına gönderilir. Böylelikle CQ[63] sayıcı durum bitinin lojik seviyesi Q0.1'den gözlemlenebilir.

6.4.2. Örnek 2

Şekil 6.5'te merdiven diyagramı görülen Örnek 2'ye ait kullanıcı programı aşağıda verilmiştir:

```
CTD (I00,I01,10,0); //Basamak 1

in_out(CQ[0]);        //Basamak 2

Q00 = out();


CTD (I02,I03,100,35); //Basamak 3

in_out(CQ[35]);         //Basamak 4
```

Q01 = out();



Şekil 6.5. Örnek 2'ye ait merdiven diyagramı

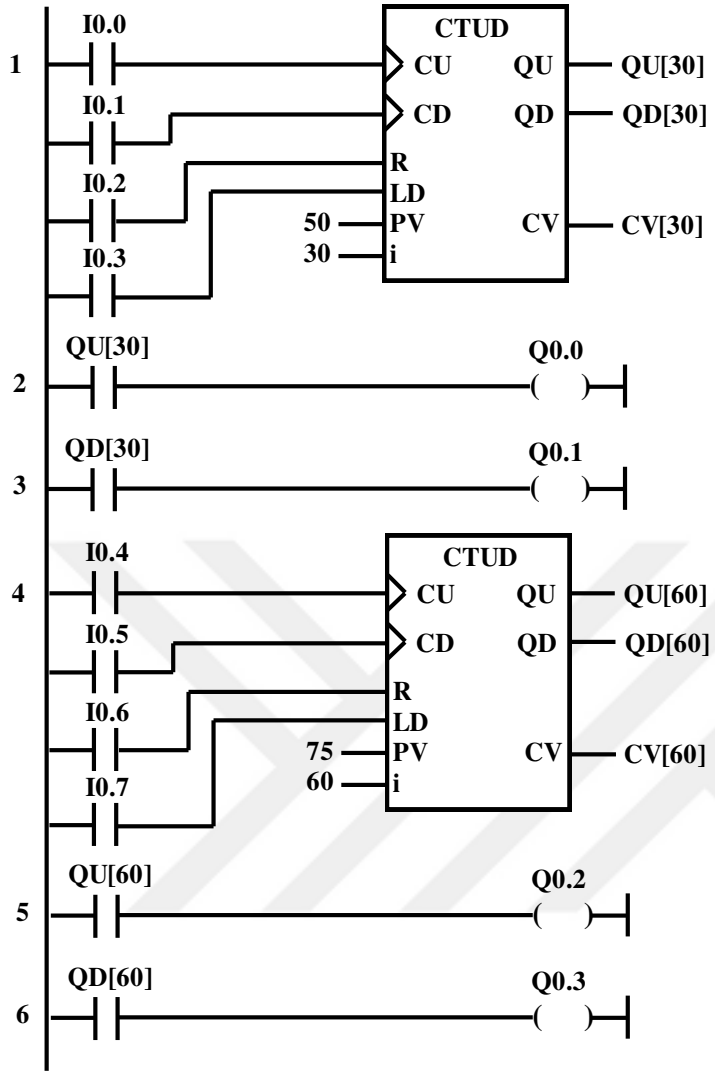
Şekil 6.5'te görülen Örnek 2'ye ait oluşturulan C kodunda: Basamak 1'de, CTD fonksiyonu şu şekilde gerçekleştirilmiştir: Geri sayma girişi CD = I0.0, ve load girişi LD = I0.1 olarak belirlenmiştir. İndeks değeri i = 0 olduğu için, CD'nin yükselen kenarını algılamak için kullanılacak olan Bool değişkeni CDRED[0] ve geri sayıcı durum biti (geri sayıcı çıkışı) CQ[0] olan 0 numaralı geri sayıcı seçilmiştir. CD girişinden uygulanacak olan yükselen kenar sinyallerini saymak için kullanılan değişken CV[0] olur. Kurma değeri PV = 10 olduğu için LD = I0.1 = 1 olduğunda CV[0] = 10 yapılır. CV[0] > 0 olduğunda geri sayma girişi CD = I0.0'a uygulanan her bir yükselen kenar sinyalle birlikte CV[0]'daki sayma değeri bir azaltılır. CV[0] = 0 olduğunda sayıcı durum biti (geri sayıcı çıkışı) CQ[0] = 1 olur. Basamak 2'de, sayıcı durum biti CQ[0]'ın değeri Q0.0 çıkışına gönderilir. Böylelikle CQ[0] sayıcı durum bitinin lojik seviyesi Q0.0'dan gözlemlenebilir. Basamak 3'te, CTD fonksiyonu şu şekilde gerçekleştirilmiştir: Geri sayma girişi CD = I0.2, ve load girişi LD = I0.3 olarak belirlenmiştir. İndeks değeri i = 35 olduğu için, CD'nin yükselen kenarını algılamak için kullanılacak olan Bool değişkeni CDRED[35] ve geri sayıcı durum biti (geri sayıcı çıkışı) CQ[35] olan 35 numaralı geri sayıcı seçilmiştir. CD girişinden uygulanacak olan yükselen kenar sinyallerini saymak için kullanılan değişken CV[35] olur. Kurma değeri PV = 100 olduğu için LD = I0.3 = 1 olduğunda

CV[35] = 100 yapılır. CV[35] > 0 olduğunda geri sayma girişi CD = I0.2'ye uygulanan her bir yükselen kenar sinyalle birlikte CV[35]'deki sayma değeri bir azaltılır. CV[35] = 0 olduğunda sayıcı durum biti (geri sayıcı çıkışı) CQ[35] = 1 olur. Basamak 4'te, sayıcı durum biti CQ[35]'in değeri Q0.1 çıkışına gönderilir. Böylelikle CQ[35] sayıcı durum bitinin lojik seviyesi Q0.1'den gözlemlenebilir.

6.4.3. Örnek 3

Şekil 6.6'da merdiven diyagramı görülen Örnek 3'e ait kullanıcı programı aşağıda verilmiştir:

```
CTUD (I00,I01,I02,I03,50,30); //Basamak 1
in_out(QU[30]);               //Basamak 2
Q00 = out ();
in_out(QD[30]);               //Basamak 3
Q01 = out ();

CTUD (I04,I05,I06,I07,75,60); //Basamak 4
in_out(QU[60]);               //Basamak 5
Q02 = out ();
in_out(QD[60]);               //Basamak 6
Q03 = out ();
```



Şekil 6.6. Örnek 3'e ait merdiven diyagramı

Şekil 6.6'da görülen Örnek 3'e ait oluşturulan C kodunda: Basamak 1'de, CTUD fonksiyonu şu şekilde gerçekleştirilmiştir: İleri sayma girişi CU = I0.0, geri sayma girişi CD = I0.1, reset girişi R = I0.2 ve load girişi LD = I0.3, kurma değeri PV = 50, indeks değeri i = 30 olarak belirlenmiştir. i = 30 olduğu için CU'nun yükselen kenarını algılamak için kullanılacak olan Bool değişkeni CURED[30], CD'nin yükselen kenarını algılamak için kullanılacak olan Bool değişkeni CDRED[30], sayma değerlerini saklamak için kullanılan değişken CV[30], ileri sayıcı durum biti QU[30] ve geri sayıcı durum biti QD[30] olarak seçilir. CV[30] < PVmax olmak kaydıyla ileri sayma girişi CU = I0.0'a uygulanan her yükselen kenar sinyali ile CV[30]'daki sayma değeri bir arttırılır ve CV[30] > 0 olmak kaydıyla geri sayma girişi CD = I0.1'e uygulanan her yükselen kenar sinyali ile

CV[30]'daki sayma değeri bir azaltılır. Reset girişi R = I0.2 = 1 olduğunda ise CV[30] = 0 yapılır. Kurma değeri PV = 50 olduğu için yükleme girişi LD = I0.3 = 1 olduğunda CV[30] = 50 yapılır. CV[30] ≥ PV olduğunda ileri sayıcı durum biti QU[30] = 1 olur. CV[0] = 0 olduğunda geri sayıcı durum biti QD[30] = 1 olur. Basamak 2'de, ileri sayıcı durum biti QU[30]'ın değeri Q0.0 çıkışına gönderilir. Böylelikle QU[30] sayıcı durum bitinin lojik seviyesi Q0.0'dan gözlemlenebilir. Basamak 3'te, geri sayıcı durum biti QD[30]'ın değeri Q0.1 çıkışına gönderilir. Böylelikle QD[30] sayıcı durum bitinin lojik seviyesi Q0.1'den gözlemlenebilir.

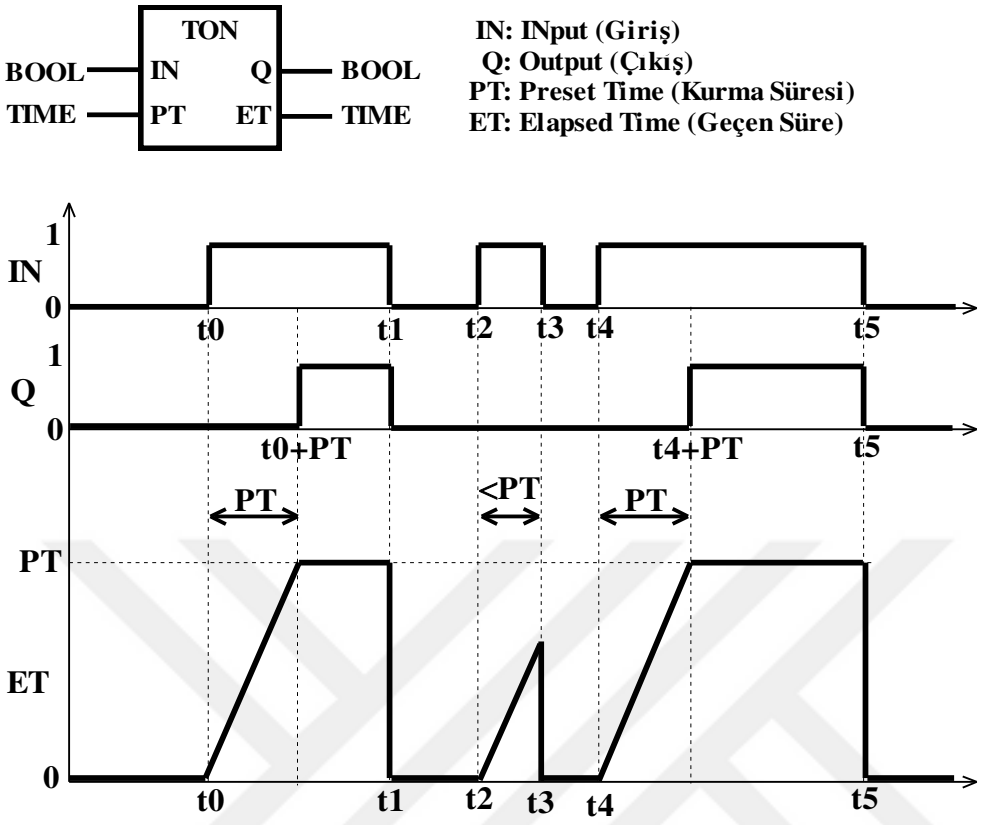
Basamak 4'te, CTUD fonksiyonu şu şekilde gerçekleştirilmiştir: İleri sayma girişi CU = I0.4, geri sayma girişi CD = I0.5, reset girişi R = I0.6 ve load girişi LD = I0.7, kurma değeri PV = 75, indeks değeri i = 60 olarak belirlenmiştir. i = 60 olduğu için CU'nun yükselen kenarını algılamak için kullanılacak olan Bool değişkeni CURED[60], CD'nin yükselen kenarını algılamak için kullanılacak olan Bool değişkeni CDRED[60], sayma değerlerini saklamak için kullanılan değişken CV[60], ileri sayıcı durum biti QU[60] ve geri sayıcı durum biti QD[60] olarak seçilir. CV[60] < PVmax olmak kaydıyla ileri sayma girişi CU = I0.4'e uygulanan her yükselen kenar sinyali ile CV[60]'daki sayma değeri bir arttırılır ve CV[60] > 0 olmak kaydıyla geri sayma girişi CD = I0.5'e uygulanan her yükselen kenar sinyali ile CV[60]'daki sayma değeri bir azaltılır. Reset girişi R = I0.6 = 1 olduğunda ise CV[30] = 0 yapılır. Kurma değeri PV = 75 olduğu için yükleme girişi LD = I0.7 = 1 olduğunda CV[60] = 75 yapılır. CV[60] ≥ PV olduğunda ileri sayıcı durum biti QU[60] = 1 olur. CV[60] = 0 olduğunda geri sayıcı durum biti QD[60] = 1 olur. Basamak 5'te, ileri sayıcı durum biti QU[60]'ın değeri Q0.2 çıkışına gönderilir. Böylelikle QU[60] sayıcı durum bitinin lojik seviyesi Q0.2'den gözlemlenebilir. Basamak 6'da, geri sayıcı durum biti QD[60]'ın değeri Q0.3 çıkışına gönderilir. Böylelikle QD[60] sayıcı durum bitinin lojik seviyesi Q0.3'ten gözlemlenebilir.

7. ZAMANLAYICI FONKSİYONLARI

Bu bölümde zamanlayıcı fonksiyonlarının proje kapsamında nasıl gerçekleştirildiği anlatılmaktadır. Zamanlayıcı fonksiyonları, programlama dillerinde belirli bir işlemin belirli bir süre sonra veya belirli aralıklarla otomatik olarak çalışmasını sağlayan özel fonksiyonlardır. Bu tür fonksiyonlar genellikle zamanlanmış görevlerin gerçekleştirilmesinde, periyodik işlemlerde veya belirli bir süre içinde bir işlemin tamamlanması gerektiği durumlarda kullanılır. Bu fonksiyonlar, programların otomatik olarak belirli bir zaman dilimi sonunda bir işlem gerçekleştirmesini veya belirli bir süre boyunca belirli bir işlemi tekrarlamasını sağlar. Gerçekleştirilen bu tez çalışmasında gecikmenin yapısına bağlı olarak düz zaman rölesi (On Delay Timer - TON) ve ters zaman rölesi (OFF Delay Timer - TOF) olmak üzere iki zamanlayıcı fonksiyonu gerçekleştirilmiştir. Bu iki fonksiyonda gerekli olan referans zaman sinyallerinin elde edilmesi için, mikrodenetleyicinin 32 bitlik Timer1 dâhili zamanlayıcı modülü kullanılmıştır.

7.1. Düz Zaman Rölesi (On Delay Timer - TON)

Giriş sinyali (IN), kurma süresi kadar (PT) bir süre uygulandıktan sonra çıkış sinyali gecikmeli olarak lojik 0 seviyesinden lojik 1 seviyesine konum değiştiren zaman röleleri olarak bilinir. Şekil 7.1'de düz zaman rölesinin (TON) sembolü ve zamanlama diyagramı görülmektedir. Düz zaman rölesinin çıkışının (Q) gecikmeli olarak lojik 0 seviyesinden lojik 1 seviyesine konum değiştirebilmesi için giriş sinyali (IN) en az kurma süresi (PT) kadar uygulanmalıdır (t0+PT; t4+PT). Eğer giriş sinyali (IN) kurma süresi (PT) dolmadan lojik 1 seviyesinden lojik 0 seviyesine konum değiştirirse (t3) zaman gecikmesi iptal edilmiş olur. Hem giriş sinyalinin (IN) hem de çıkış sinyalinin (Q) lojik 1 seviyesinde olduğu durumda; giriş sinyalinin (IN) lojik 1 seviyesinden lojik 0 seviyesine konum değiştirmesiyle birlikte çıkış sinyali (Q) de lojik 1 seviyesinden lojik 0 seviyesine değişir (t1; t5) (Uzam, 2022).

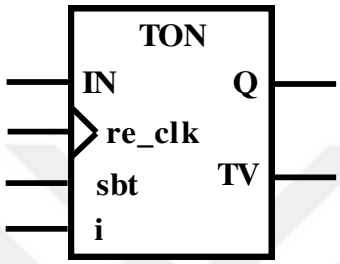


Şekil 7.1. Düz zaman rölesinin (TON) sembolü ve zamanlama diyagramı (Uzam, 2022)

Tablo 7.1'de düz zaman rölesinin bu proje kapsamında kullanılan (TON) sembolü ve C kodu mevcuttur. Burada oluşturulan C kodunda TON düz zaman rölesi fonksiyonundaki parametreler: re_clk, sbt ve i. **re_clk:** bir referans zamanlama sinyalinin yükselen kenarını belirtir. Her ne kadar re_clk girişinde bir yükselen kenar sembolü varsa da TON fonksiyonunun içerisinde re_clk girişinden uygulanacak olan sinyalin yükselen kenarını algılamak için bir işlem yapılmamaktadır. TON fonksiyonunun problemsiz bir şekilde çalışabilmesi için re_clk girişine uygulanacak olan sinyalin bu proje kapsamında geliştirilmiş olan referans zamanlama sinyallerinin yükselen kenarı olan re_T10ms (10 ms), re_T100ms (100 ms) veya re_T1s (1 s) sinyallerinden biri olması gerekir. **sbt:** 32-bitlik çözünürlükte 1 - 4.294.967.296 arası bir zaman sabitidir. Kurma süresini (PT) elde etmek için kullanılır. **i,** indeks numarasıdır ve 0'dan 63'e kadar olan 64 tane düz zaman rölesini ifade eder. **Kurma süresi PT** = sbt* re_clk formülüyle bulunur. Buradaki re_clk, bu girişe bağlanacak olan referans zamanlama sinyalinin periyodu olarak hesaba katılır.

TON fonksiyonunun çalışması şöyledir: İndeks değeri i, 64'ten küçük olduğunda: Giriş sinyali IN = 0 ise, sayma kaydedicisi TON[i] ve zamanlayıcı çıkışı TONQ[i] sıfırlanır. Giriş sinyali IN = 1 ve zamanlayıcı çıkışı TONQ[i] = 0 ise; re_clk sinyalinin 1 olması durumunda (kullanılan referans zamanlama sinyalinin yükselen kenarında) sayma kaydedicisi TON[i] bir değer arttırılır ve TON[i] = sbt ise, TONQ[i] set edilir.

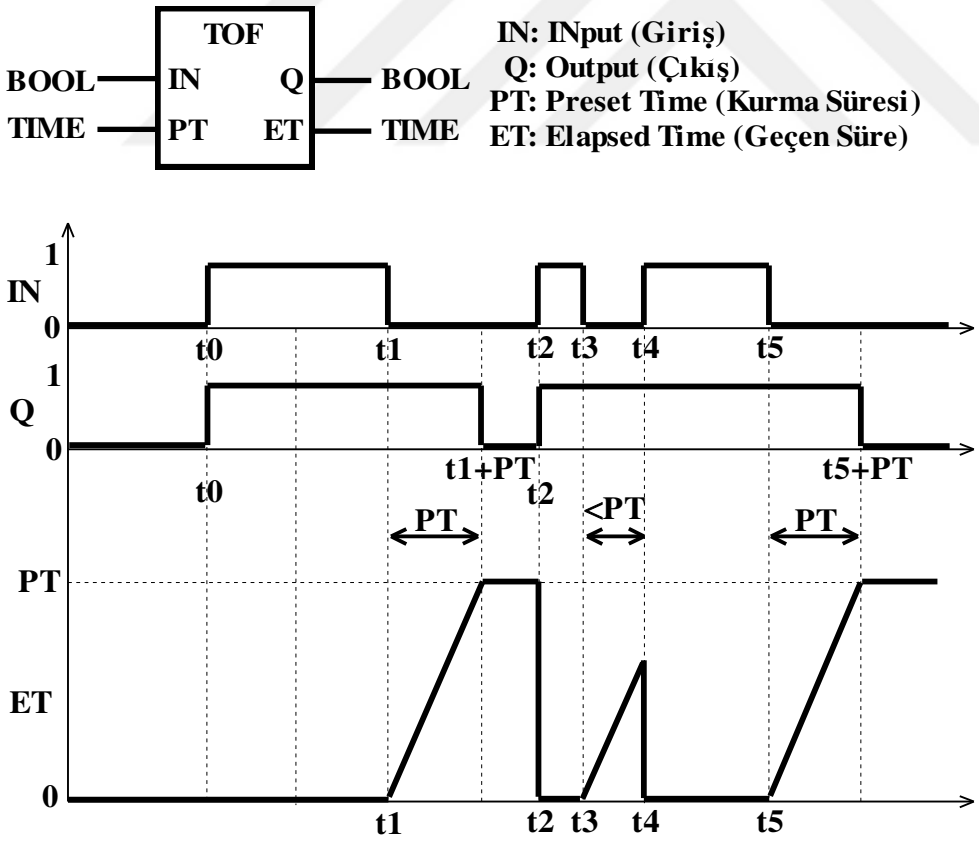
Tablo 7.1. Düz zaman rölesinin (TON) sembolü ve C kodu

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

PT = sbt x re_clk
ET = TV x re_clk</td>
<td>IN (giriş sinyali, TEMP üzerinden) = 0 veya 1
re_clk (üç referans zamanlama sinyalinden birinin yükselen kenarı) = re_T10ms (10 ms), re_T100ms (100 ms), veya re_T1s (1 s)
sbt (zaman sabiti) = 1, 2, ..., 4.294.967.296
i (indeks girişi - Özgün zamanlayıcı numarası) = 0, 1, …, 63
Q (zamanlayıcı çıkışı - zamanlayıcı durum biti) = TONQ[i], i = 0, 1, …, 63
TV (zamanlayı kaydedicisindeki sayma değeri) = TON[i], i = 0, 1, …, 63)</td>
</tr>
<tr>
<td>Geliştirilen C Kodu</td>
<td colspan="2">
<pre>
void ton (_Bool re_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(re_clk)
                        {
                                TON[i]++;
                                 if (TON[i]==sbt)
                                        TONQ[i]=1;
                        }
                }
        }
}
</pre>
</td>
</tr>
</table>

7.2. Ters Zaman Rölesi (Off Delay Timer- TOF)

Çıkış sinyalini (Q) lojik 0 yapma işlemini giriş sinyalinin (IN) lojik 1 seviyesinden lojik 0 seviyesine düşmesinden sonra belirli bir süre (PT) geciktirerek gerçekleştiren zaman rölelerine ters zaman rölesi denir. Şekil 7.2'de ters zaman rölesinin (TOF) sembolü ve zamanlama diyagramı görülmektedir. Giriş sinyalinin (IN) lojik 1 olmasıyla birlikte (t0; t2) çıkış sinyali (Q) de lojik 1 olur ve giriş sinyali lojik 0 seviyesine düştükten sonra (t1; t5) kurma süresi (PT) geçinceye kadar bu şekilde kalmaya devam eder. Giriş sinyalinin (IN) lojik 0 seviyesine düşmesiyle birlikte geçen süre (ET) artmaya başlar (t1 - t1+PT arası; t5 - t5+PT arası). Geçen sürenin (ET) artması, geçen süre değeri kurma (PT) süresine eşit oluncaya kadar devam eder. Bu iki süre eşit olunca (ET=PT: t1+PT; t5+PT) çıkış sinyali lojik 0 yapılır ve geçen süre (ET) durdurulur. Eğer giriş sinyali (IN) lojik 0 seviyesinde kurma süresinden (PT) daha az bir süre kalıp (t3-t4 arası) tekrar lojik 1 seviyesine dönerse (t4) bu durumda çıkış sinyali (Q) lojik 1 seviyesinde kalmaya devam eder (Uzam, 2022).

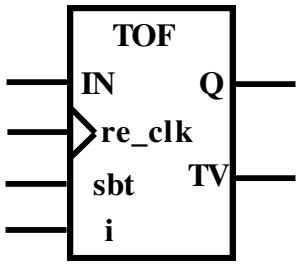


Şekil 7.2. Ters zaman rölesinin (TOF) sembolü ve zamanlama diyagramı

Tablo 7.2'de ters zaman rölesinin bu proje kapsamında kullanılan (TOF) sembolü ve C kodu mevcuttur. Burada oluşturulan C kodunda TOF ters zaman rölesi fonksiyonundaki parametreler: re_clk, sbt ve i. **re_clk:** bir referans zamanlama sinyalinin yükselen kenarını belirtir. Her ne kadar re_clk girişinde bir yükselen kenar sembolü varsa da TOF fonksiyonunun içerisinde re_clk girişinden uygulanacak olan sinyalin yükselen kenarını algılamak için bir işlem yapılmamaktadır. TOF fonksiyonunun problemsiz bir şekilde çalışabilmesi için re_clk girişine uygulanacak olan sinyalin bu proje kapsamında geliştirilmiş olan referans zamanlama sinyallerinin yükselen kenarı olan re_T10ms (10 ms), re_T100ms (100 ms) veya re_T1s (1 s) sinyallerinden biri olması gerekir. **sbt:** 32-bitlik çözünürlükte 1 - 4.294.967.296 arası bir zaman sabitidir. Kurma süresini (PT) elde etmek için kullanılır. **i,** indeks numarasıdır ve 0'dan 63'e kadar olan 64 tane ters zaman rölesini ifade eder. **Kurma süresi PT** = sbt* re_clk formülüyle bulunur. Buradaki re_clk, bu girişe bağlanacak olan referans zamanlama sinyalinin periyodu olarak hesaba katılır.

TOF fonksiyonunun çalışması şöyledir: İndeks değeri i, 64'ten küçük olduğunda: Giriş sinyali IN (TEMP) = 1 ve zamanlayıcı çıkışı TOFQ[i] = 0 ise, TOFQ[i] = 1 yapılır (set edilir). Giriş sinyali IN (TEMP) = 1 ve zamanlayıcı çıkışı TOFQ[i] = 1 ise, sayma kaydedicisi TOF[i] sıfırlanır. Giriş sinyali IN (TEMP) = 0 ve TOFQ[i] = 1 ise; re_clk sinyalinin 1 olması durumunda (kullanılan referans zamanlama sinyalinin yükselen kenarında) sayma kaydedicisi TOF[i] bir değer arttırılır ve TOF[i] = sbt ise, TOFQ[i] = 0 yapılır (resetlenir).

Tablo 7.2. Ters zaman rölesinin (TOF) sembolü ve C kodu

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

PT = sbt x re_clk
ET = TV x re_clk</td>
<td>IN (giriş sinyali, TEMP üzerinden) = 0 veya 1
re_clk (üç referans zamanlama sinyalinden birinin yükselen kenarı) = re_T10ms (10 ms), re_T100ms (100 ms), veya re_T1s (1 s)
tcnst (zaman sabiti) = 1, 2, ..., 4.294.967.296
i (indeks girişi - Özgün zamanlayıcı numarası) = 0, 1, …, 63
Q (zamanlayıcı çıkışı - zamanlayıcı durum biti) = TOFQ[i], i = 0, 1, …, 63
TV (zamanlayı kaydedicisindeki sayma değeri) = TOF[i], i = 0, 1, …, 63)</td>
</tr>
<tr>
<td>Geliştirilen C Kodu</td>
<td colspan="2">
<pre>
void tof (_Bool re_clk, unsigned int sbt,unsigned int i)
{
        if (i<64)
        {
                if(TEMP){
                        if (TOFQ[i]==1)   TOF[i]=0;
                        else   TOFQ[i]=1;
                }
                else if (TOFQ[i]==1)
                {
                        if(re_clk)
                        {
                                TOF[i]++;
                                if (TOF[i]==sbt)
                                        TOFQ[i]=0;
                        }
                }
        }
}
</pre>
</td>
</tr>
</table>

7.3. Zamanlayıcı Fonksiyon Örnekleri

Bu kısımda zamalayıcı fonksiyonlarına ait örnekler yer almaktadır. Her bir örnek için merdiven diyagramı ve kullanıcı programına ait C kodu sunulmuştur.

7.3.1. Örnek 1

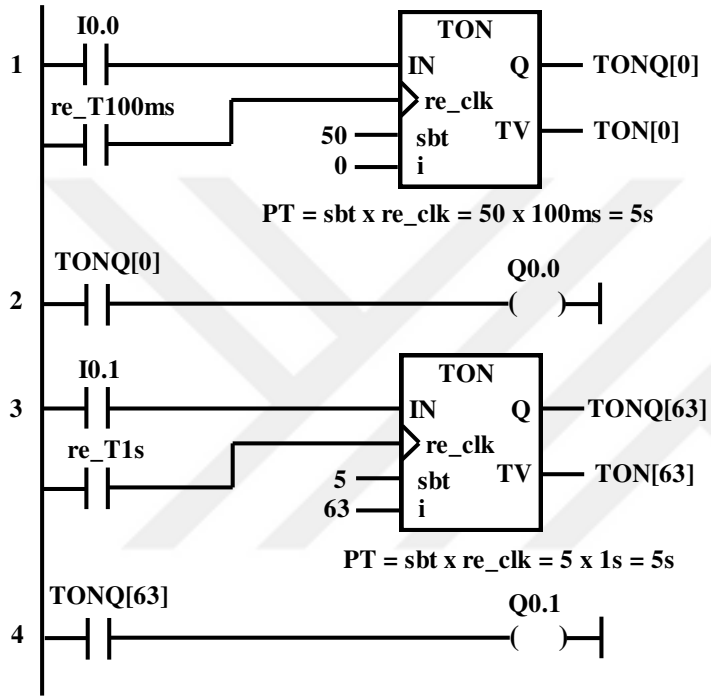
Şekil 7.3'te merdiven diyagramı görülen Örnek 1'e ait kullanıcı programı aşağıda verilmiştir:

```
ld (I00);                    //Basamak 1
ton (re_T100ms, 50,0);

Q00 = TONQ[0];       //Basamak 2

ld (I01);                     //Basamak 3

ton (re_T1s, 5,63);
Q01 = TONQ[63];      //Basamak 4
```

Şekil 7.3'te görülen Örnek 1'e ait oluşturulan C kodunda: Basamak 1'de, TON fonksiyonu şu şekilde gerçekleştirilmiştir: Giriş sinyali IN = I0.0 olarak belirlenmiştir. i = 0 olduğu için zaman gecikme değerlerini saymak için TON[0] kaydedicisi ve zamanlayıcı durum biti (veya zamanlayıcı çıkışı) TONQ[0] olarak seçilmiştir. re_clk girişinden uygulanan sinyal re_T100ms ve sbt = 5 olduğu için kurma süresi PT = 100ms x 5 = 5 s'dir. Basamak 2'de, zamanlayıcı çıkışı TONQ[0]'ın değeri Q0.0 çıkışına gönderilir. Böylelikle TONQ[0] zamanlayıcı çıkışının lojik seviyesi Q0.0'dan gözlemlenebilir.

Basamak 3'te, TON fonksiyonu şu şekilde gerçekleştirilmiştir: Giriş sinyali IN = I0.1 olarak belirlenmiştir. i = 63 olduğu için zaman gecikme değerlerini saymak için TON[63]

kaydedicisi ve zamanlayıcı durum biti (veya zamanlayıcı çıkışı) TONQ[63] olarak seçilmiştir. re_clk girişinden uygulanan sinyal re_T1s ve sbt = 5 olduğu için kurma süresi PT = 1s x 5 = 5 s'dir. Basamak 4'te, zamanlayıcı çıkışı TONQ[63]'ın değeri Q0.1 çıkışına gönderilir. Böylelikle TONQ[63] zamanlayıcı çıkışının lojik seviyesi Q0.1'den gözlemlenebilir.



Şekil 7.3. Örnek 1'e ait merdiven diyagramı

7.3.2. Örnek 2

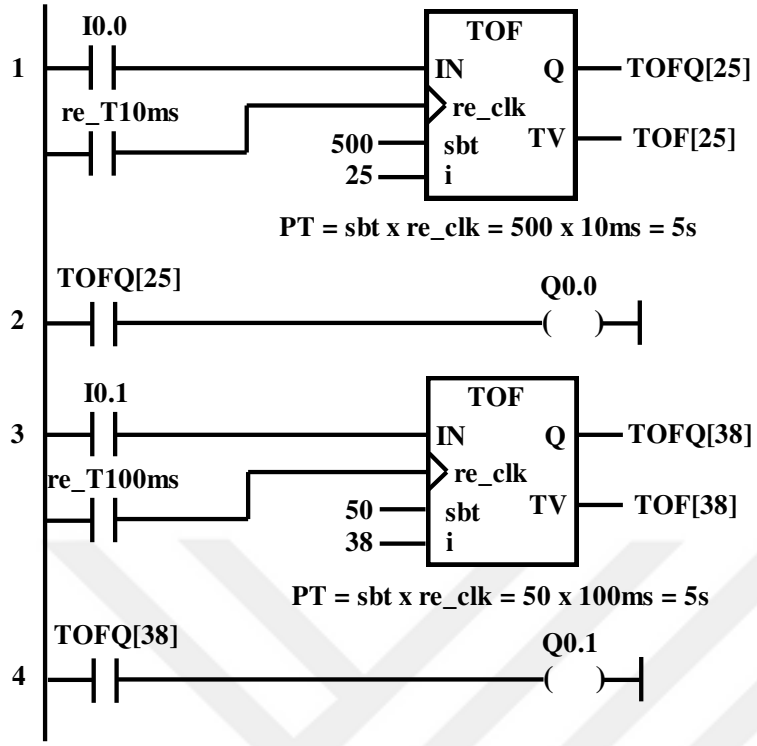
Şekil 7.4'te merdiven diyagramı görülen Örnek 2'ye ait kullanıcı programı aşağıda verilmiştir:

```
ld (I00);                    //Basamak 1
tof (re_T10ms, 500,25);

Q00 = TOFQ[25];    //Basamak 2
```

```
ld (I01);                    //Basamak 3


tof (re_T100ms, 50,38);
Q01 = TOFQ[38];    //Basamak 4
```

Şekil 7.4'te görülen Örnek 2'ye ait oluşturulan C kodunda: Basamak 1'de, TOF fonksiyonu şu şekilde gerçekleştirilmiştir: Giriş sinyali IN = I0.0 olarak belirlenmiştir. i = 25 olduğu için zaman gecikme değerlerini saymak için TOF[25] kaydedicisi ve zamanlayıcı durum biti (veya zamanlayıcı çıkışı) TOFQ[25] olarak seçilmiştir. re_clk girişinden uygulanan sinyal re_T10ms ve sbt = 500 olduğu için kurma süresi PT = 10ms x 500 = 5 s'dir. Basamak 2'de, zamanlayıcı çıkışı TOFQ[25]'in değeri Q0.0 çıkışına gönderilir. Böylelikle TOFQ[25] zamanlayıcı çıkışının lojik seviyesi Q0.0'dan gözlemlenebilir.

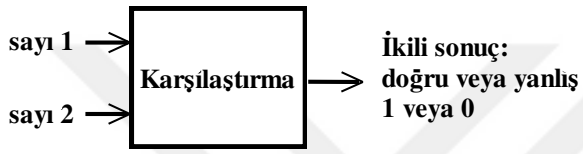
Basamak 3'te, TOF fonksiyonu şu şekilde gerçekleştirilmiştir: Giriş sinyali IN = I0.1 olarak belirlenmiştir. i = 38 olduğu için zaman gecikme değerlerini saymak için TOF[38] kaydedicisi ve zamanlayıcı durum biti (veya zamanlayıcı çıkışı) TOFQ[38] olarak seçilmiştir. re_clk girişinden uygulanan sinyal re_T100ms ve sbt = 50 olduğu için kurma süresi PT = 100ms x 50 = 5 s'dir. Basamak 4'te, zamanlayıcı çıkışı TOFQ[38]'ın değeri Q0.1 çıkışına gönderilir. Böylelikle TOFQ[38] zamanlayıcı çıkışının lojik seviyesi Q0.1'den gözlemlenebilir.



Şekil 7.4. Örnek 2'ye ait merdiven diyagramı

8. KARŞILAŞTIRMA FONKSİYONLARI

PLC programlarında sıklıkla sayısal değerlerin karşılaştırılması gerekir. Bu kısımda bu değerlerin karşılaştırılabilmesi için bu proje kapsamında gerçekleştirilen karşılaştırma fonksiyonları açıklanmaktadır. Şekil 8.1'de veri karşılaştırma işleminin genel şekli görülmektedir. Burada sayı1 ve sayı2 karşılaştırılacak iki sayısal değeri ifade etmektedir. Karşılaştırma işleminin sonucuna göre Boole-çıkış-değeri işlem sonucu doğruysa 1, yanlışsa 0 olur.



Şekil 8.1. Veri karşılaştırma işleminin genel şekli

Karşılaştırma fonksiyonlarında altı farklı işlem yapılır. Bunlar;

GT (Greater Than) = sayı1, sayı2'den büyüktür (sayı1 > sayı2)

GE (Greater than or Equal to) = sayı1, sayı2'den büyük veya eşittir (sayı1 ≥ sayı2)

EQ (EQual to) = sayı1, sayı2'ye eşittir (sayı1 = sayı2)

LT (Less Than) = sayı1, sayı2'den küçüktür (sayı1 < sayı2)

LE (Less than or Equal to) = sayı1, sayı2'den küçük veya eşittir (sayı1 ≤ sayı2)

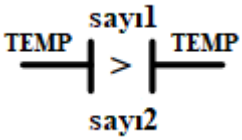
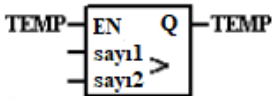
NE (Not Equal to) = sayı1, sayı2'ye eşit değildir (sayı1 ≠ sayı2)

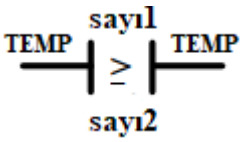
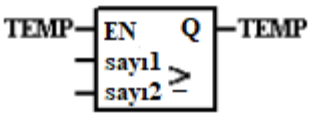
Tablo 8.1'de altı karşılaştırma fonksiyonunun merdiven diyagramı sembolü, şematik sembolü ve geliştirilen C Kodu görülmektedir. Bu fonksiyonlarda aktif 1 yetkileme girişi EN (TEMP), iki giriş değişkeni "in1" ve "in2" ile bir çıkış değişkeni OUT (TEMP) mevcuttur. Bu fonksiyonlar yetkileme girişi EN (TEMP) aktif olduğunda giriş değişkenlerini karşılaştırılarak işlem yapar ve sonucu TEMP çıkış değişkeni üzerinden dışarıya gönderir (Harmanda, 2011). Bu tabloda görülen karşılaştırma işlemleri sayı1,

sayı2'nin unsigned int veri tipi olması durumu için tanımlanmıştır. Benzer karşılaştırma işlemleri diğer veri tipleri için de tanımlanabilir.

Tablo 8.1. Karşılaştırma fonksiyonları

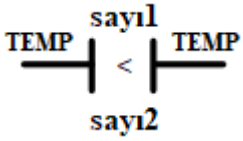
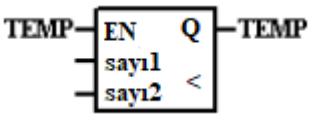
Tanım	Merdiven diyagramı sembolü	Şematik sembol	Geliştirilen C Kodu

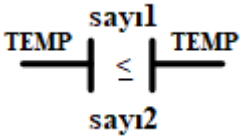
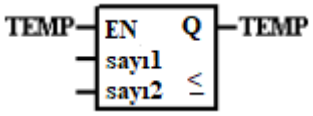
GT
sayı1, sayı2'den büyük mü?
		sayı1 > sayı2
		 
		EN : Aktif 1 yetkileme girişi (TEMP) = 0 veya 1
		sayı1 : giriş değişkeni 1
		sayı2 : giriş değişkeni 2
		Q : Boole çıkış değişkeni, (TEMP) = 0 veya 1
			void GT (unsigned int in1, unsigned int in2)
			{
			if (TEMP && (in1 > in2))
			TEMP=1;
			else
			TEMP=0;
			}

GE
sayı1, sayı2'den büyük veya eşit mi?
		sayı1 ≥ sayı2
		 
		EN : Aktif 1 yetkileme girişi (TEMP) = 0 veya 1
		sayı1 : giriş değişkeni 1
		sayı2 : giriş değişkeni 2
		Q : Boole çıkış değişkeni, (TEMP) = 0 veya 1
			void GE (unsigned int in1, unsigned int in2)
			{
			if (TEMP && (in1 >= in2))
			TEMP=1;
			else
			TEMP=0;
			}

Tablo 8.1. Karşılaştırma fonksiyonları (devam)

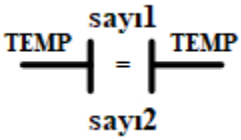
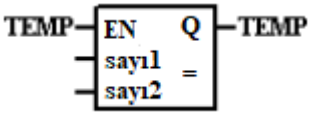
Tanım	Merdiven diyagramı sembolü	Şematik sembol	Geliştirilen C Kodu

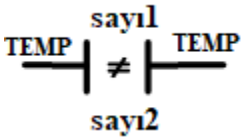
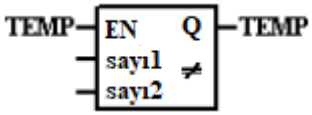
LT
sayı1, sayı2'den küçük mü?
		sayı1 < sayı2
		 
		EN : Aktif 1 yetkileme girişi (TEMP) = 0 veya 1
		sayı1 : giriş değişkeni 1
		sayı2 : giriş değişkeni 2
		Q : Boole çıkış değişkeni, (TEMP) = 0 veya 1
			void LT (unsigned int in1, unsigned int in2)
			{
			if (TEMP && (in1 < in2)) TEMP=1;
			else TEMP=0;
			}

LE
sayı1, sayı2'den küçük veya eşit mi?
		sayı1 ≤ sayı2
		 
		EN : Aktif 1 yetkileme girişi (TEMP) = 0 veya 1
		sayı1 : giriş değişkeni 1
		sayı2 : giriş değişkeni 2
		Q : Boole çıkış değişkeni, (TEMP) = 0 veya 1
			void LE (unsigned int in1, unsigned int in2)
			{
			if (TEMP && (in1 <= in2)) TEMP=1;
			else TEMP=0;
			}

Tablo 8.1. Karşılaştırma fonksiyonları (devam)

Tanım	Merdiven diyagramı sembolü	Şematik sembol	Geliştirilen C Kodu

EQ
sayı1,
sayı2'ye
eşit
mi?
		sayı1 = sayı2
		 
		EN : Aktif 1 yetkileme girişi (TEMP) = 0 veya 1
		sayı1 : giriş değişkeni 1
		sayı2 : giriş değişkeni 2
		Q : Boole çıkış değişkeni, (TEMP) = 0 veya 1
			void EQ (unsigned int in1, unsigned int in2)
			{
			if (TEMP && (in1 == in2)) TEMP=1;
			else TEMP=0;
			}

NE
sayı1,
sayı2'ye
eşit değil
mi?
		sayı1 ≠ sayı2
		 
		EN (Boole yetkileme girişi, TEMP) = 0 veya 1
		sayı1 : giriş değişkeni 1
		sayı2 : giriş değişkeni 2
		Q : Boole çıkış değişkeni, (TEMP) = 0 veya 1
			void NE (unsigned int in1, unsigned int in2)
			{
			if (TEMP && (in1 != in2)) TEMP=1;
			else TEMP=0;
			}

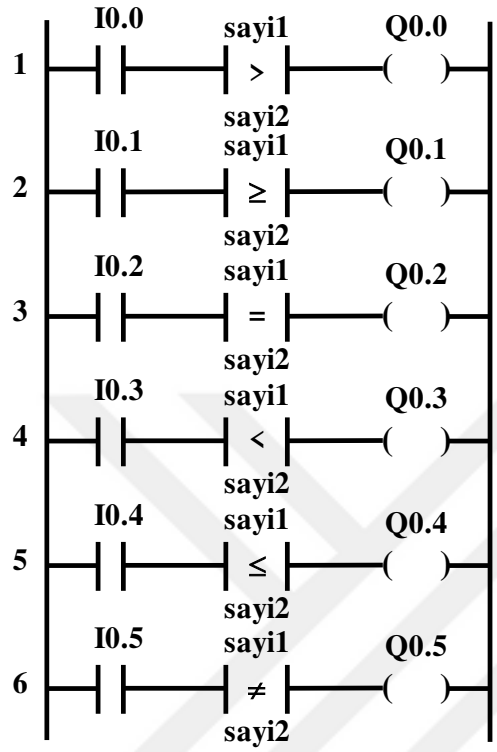
8.1. Karşılaştırma Fonksiyon Örneği

Bu kısımda karşılaştırma fonksiyonlarına ait bir örnek yer almaktadır. Şekil 8.1'de merdiven diyagramı görülen Örnek 1'e ait kullanıcı programı aşağıda verilmiştir:

```
unsigned int sayi1 = 2200;

unsigned int sayi2 = 10;

ld (I00);                    //Basamak 1
GT (sayi1, sayi2);
Q00 = out();

ld (I01);                    //Basamak 2
GE (sayi1, sayi2);
Q01 = out();

ld (I02);                    //Basamak 3
EQ (sayi1, sayi2);
Q02 = out();

ld (I03);                    //Basamak 4
LT (sayi1, sayi2);
Q03 = out();

ld (I04);                    //Basamak 5
LE (sayi1, sayi2);
```

```
Q04 = out();


ld (I05);                    //Basamak 6
NE (sayi1, sayi2);
Q05 = out();
```

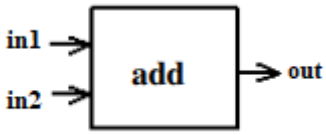
Şekil 8.2'de görülen Örnek 1'e ait oluşturulan C kodunda: Öncelikle sayi1 ve sayi2'ye değer ataması yapılmaktadır (sayi1 = 2200, sayi2 = 10). Farklı sonuçlar elde etmek için bu değerlerin değiştirilerek programın tekrar çalıştırılması gerekir. Basamak 1'de, I0.0 = 1 olduğunda "sayi1'in değeri sayi2'nin değerinden büyük mü?" şeklinde bir karşılaştırma işlemi yapılıp sonuç Q0.0 no'lu çıkışa gönderilmektedir. Basamak 2'de, I0.1 = 1 olduğunda "sayi1'in değeri sayi2'nin değerinden büyük veya eşit mi?" şeklinde bir karşılaştırma işlemi yapılıp sonuç Q0.1 no'lu çıkışa gönderilmektedir. Basamak 3'te, I0.2 = 1 olduğunda "sayi1'in değeri sayi2'nin değerine eşit mi?" şeklinde bir karşılaştırma işlemi yapılıp sonuç Q0.2 no'lu çıkışa gönderilmektedir. Basamak 4'te, I0.3 = 1 olduğunda "sayi1'in değeri sayi2'nin değerinden küçük mü?" şeklinde bir karşılaştırma işlemi yapılıp sonuç Q0.3 no'lu çıkışa gönderilmektedir. Basamak 5'te, I0.4 = 1 olduğunda "sayi1'in değeri sayi2'nin değerinden küçük veya eşit mi?" şeklinde bir karşılaştırma işlemi yapılıp sonuç Q0.4 no'lu çıkışa gönderilmektedir. Basamak 6'da, I0.5 = 1 olduğunda "sayi1'in değeri sayi2'nin değerine eşit değil mi?" şeklinde bir karşılaştırma işlemi yapılıp sonuç Q0.5 no'lu çıkışa gönderilmektedir.



Şekil 8.2. Örnek 1'e ait merdiven diyagramı

9. ARİTMETİK FONKSİYONLAR

Aritmetik fonksiyonlar PLC'lerde sayısal veriler üzerinde matematiksel işlemler yapmak için kullanılan fonksiyonlardır. Bunlar genellikle PLC programlama yazılımı tarafından sağlanan özel fonksiyon blokları veya komutlar olarak temsil edilir. Aritmetik fonksiyonlar genellikle toplama, çıkarma, çarpma, bölme gibi temel aritmetik işlemleri gerçekleştirirler. Bu işlemler hem değişik veri tiplerinde tanımlanan iki değişkenin içeriğine veya bir değişkenle bir sabit sayıya uygulanabilir. Bu tür işlemler, bir veya birden fazla değeri alarak bir işlem gerçekleştirir ve sonucu hesaplar. Hesaplanan sonucu daha sonra hafızaya kaydeder. Şekil 9.1'de örnek olarak bir toplama fonksiyonuna ait işlem şeması görülmektedir. Bu bölümde bu proje kapsamında aritmetik fonksiyonları gerçekleştirmek için altı tane fonksiyon tanımlanmıştır. Bunların dört tanesi Tablo 9.1'de yer alan ADD (addition - toplama), SUB (subtraction - çıkartma), MUL (multiplication - çarpma) ve DIV (division - bölme) fonksiyonlarıdır. Bu fonksiyonların yetkileme girişi EN (TEMP) aktif olduğunda giriş değişkenlerinden gelen verilerle işlem yapar ve sonucu çıkış değişkenine kaydeder (Harmanda, 2011). Bu fonksiyonlarda iki giriş değişkeni olan "in1" ve "in2" ile bir çıkış değişkeni OUT (TEMP_REG) mevcuttur. Diğer iki tanesi ise Tablo 9.2'de yer alan INC (increment - bir arttırma) ve DEC (decrement - bir azaltma) fonksiyonlarıdır. INC ve DEC fonksiyonlarında bir giriş değişkeni olan "in" ve bir çıkış değişkeni OUT (TEMP_REG) mevcuttur. Bu bölümde incelenen aritmetik işlemler giriş ve çıkış verilerinin unsigned int veri tipinde olması durumu için tanımlanmıştır. Benzer aritmetik işlemler diğer veri tipleri için de tanımlanabilir.



Şekil 9.1. ADD (toplama) fonksiyonuna ait şema

Bu bölümde incelenen aritmetik fonksiyonlar şunlardır:

ADD (in1+ in2), in1, in2, out (çıkış), iki değeri topla, sonucu çıkışa kaydet

SUB (in1 - in2), in1, in2, out (çıkış), in 1'den in2'yi çıkart, sonucu çıkışa kaydet

MUL (in1 x in2), in1, in2, out (çıkış), iki değeri çarp, sonucu çıkışa kaydet

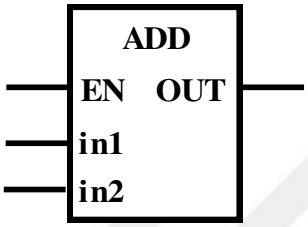
DIV (in1/ in2), in1, in2, out (çıkış), in1'i in2'ye böl, sonucu çıkışa kaydet

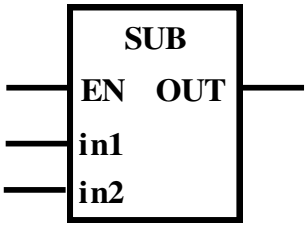
INC (IN+1), in, out (çıkış), değeri bir artır, sonucu çıkışa kaydet

DEC (IN - 1), in, out (çıkış), değeri bir azalt, sonucu çıkışa kaydet

Tablo 9.1. Aritmetik fonksiyonlar (ADD, SUB, MUL, DIV)

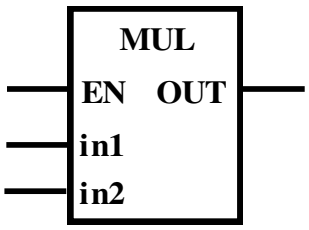
Tanım	Şematik sembol	Geliştirilen C Kodu

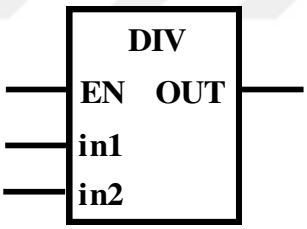
Aktif 1 yetkileme girişi (active high enable input) EN = 0 olduğunda hiçbir işlem yapılmaz. EN = 1 olduğunda in1 ve in2 giriş değişkenlerinin içeriklerini toplayıp sonucu çıkış değişkeni OUT'a depolar. OUT:=in1+in2	
	EN : Aktif 1 yetkileme girişi (TEMP üzerinden alınır) (0 veya 1)
	in1 : Giriş değişkeni 1
	in2 : Giriş değişkeni 2
	OUT : Çıkış değişkeni (işlem sonucu TEMP_REG'e depolanır)
		unsigned int ADD (unsigned int in1, unsigned int in2)
		{
		if (TEMP) TEMP_REG=in1+in2;
		return TEMP_REG;
		}

Aktif 1 yetkileme girişi EN = 0 olduğunda hiçbir işlem yapılmaz. EN = 1 olduğunda in1 değişkeninin içeriğinden in2 değişkeninin içeriğini çıkarıp sonucu çıkış değişkeni OUT'a depolar. OUT:=in1−in2	
	EN : Aktif 1 yetkileme girişi (TEMP üzerinden alınır) (0 veya 1)
	in1 : Giriş değişkeni 1
	in2 : Giriş değişkeni 2
	OUT : Çıkış değişkeni (işlem sonucu TEMP_REG'e depolanır)
		unsigned int SUB (unsigned int in1, unsigned int in2)
		{
		if (TEMP) TEMP_REG=in1−in2;
		return TEMP_REG;
		}

Tablo 9.1. Aritmetik fonksiyonlar (ADD, SUB, MUL, DIV) (devam)

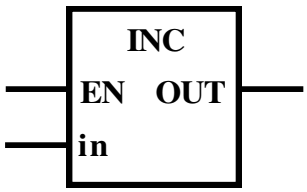
Tanım	Şematik sembol	Geliştirilen C Kodu

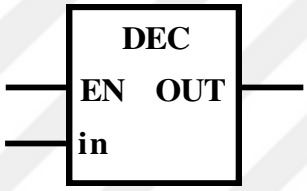
Aktif 1 yetkileme girişi EN = 0 olduğunda hiçbir işlem yapılmaz. EN = 1 olduğunda in1 giriş değişkenin içeriği ile in2 giriş değişkeninin içeriğini çarpıp elde edilen sonucu çıkış değişkeni OUT'a depolar. (OUT:=in1xin2)	
	EN : Aktif 1 yetkileme girişi (TEMP üzerinden alınır) (0 veya 1)
	in1 : Giriş değişkeni 1
	in2 : Giriş değişkeni 2
	OUT : Çıkış değişkeni (işlem sonucu TEMP_REG'e depolanır)
		unsigned int MUL (unsigned int in1, unsigned int in2)
		{
		if (TEMP) TEMP_REG=in1*in2;
		return TEMP_REG;
		}

Aktif 1 yetkileme girişi EN = 0 olduğunda hiçbir işlem yapılmaz. EN = 1 olduğunda in1 giriş değişkenin içeriğini in2 giriş değişkeninin içeriğine böler ve elde edilen tam sayı bölme işlemi sonucunu çıkış değişkeni OUT'a depolar. (OUT:=in1/in2)	
	EN : Aktif 1 yetkileme girişi (TEMP üzerinden alınır) (0 veya 1)
	in1 : Giriş değişkeni 1
	in2 : Giriş değişkeni 2
	OUT : Çıkış değişkeni (işlem sonucu TEMP_REG' e depolanır)
		unsigned int DIV (unsigned int in1, unsigned int in2)
		{
		if (TEMP) TEMP_REG=in1/in2;
		return TEMP_REG;
		}

Tablo 9.2. INC ve DEC fonksiyonları

Tanım	Şematik sembol	Geliştirilen C Kodu

Aktif 1 yetkileme girişi EN = 0 olduğunda hiçbir işlem yapılmaz. EN = 1 olduğunda INC fonksiyonu in giriş değişkeninin içeriğini 1 değer arttırıp sonucu çıkış değişkeni OUT'a depolar. (OUT := in + 1)	
	EN : Aktif 1 yetkileme girişi (TEMP üzerinden alınır) (0 veya 1)
	in : Giriş değişkeni
	OUT : Çıkış değişkeni (işlem sonucu TEMP_REG'e depolanır)
		unsigned int INC (unsigned int in)
		{
		if (TEMP) TEMP_REG= in + 1;
		else TEMP_REG=in;
		return TEMP_REG;
		}

Aktif 1 yetkileme girişi EN = 0 olduğunda hiçbir işlem yapılmaz. EN = 1 olduğunda DEC fonksiyonu in giriş değişkeninin içeriğini 1 değer azaltıp sonucu sonuç değişkeni OUT'a depolar. (OUT := in - 1)	
	EN : Aktif 1 yetkileme girişi (TEMP üzerinden alınır) (0 veya 1)
	in : Giriş değişkeni
	OUT : Çıkış değişkeni (işlem sonucu TEMP_REG'e depolanır)
		unsigned int DEC (unsigned int in)
		{
		if (TEMP) TEMP_REG= in - 1;
		else TEMP_REG=in;
		return TEMP_REG;
		}

9.1. Aritmetik Fonksiyon Örnekleri

Bu kısımda aritmetik fonksiyonlarına ait örnekler yer almaktadır. Her bir örnek için merdiven diyagramı ve kullanıcı programına ait C kodu sunulmuştur.

9.1.1. Örnek 1

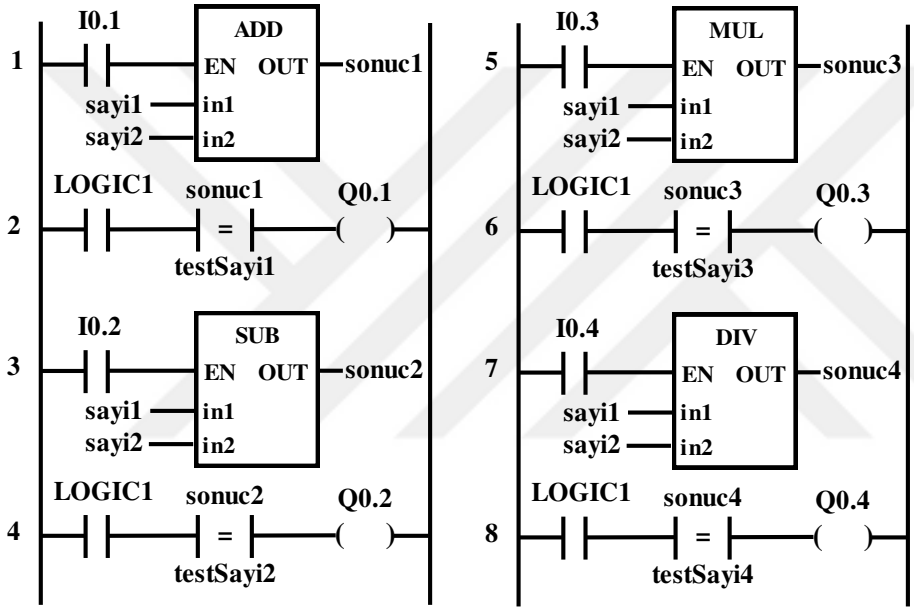
Şekil 9.2'de merdiven diyagramı görülen Örnek 1'e ait kullanıcı programı aşağıda verilmiştir:

```
unsigned int sonuc1, sonuc2, sonuc3, sonuc4 = 0;
unsigned int testSayi1 = 2210;
unsigned int testSayi2 = 2190;
unsigned int testSayi3 = 22000;
unsigned int testSayi4 = 220;
unsigned int sayi1 = 2200;
unsigned int sayi2 = 10;

ld (I01);                       //Basamak 1
sonuc1 = ADD (sayi1,sayi2);// sayi ile sayi2'yi topla

ld (LOGIC1);                    //Basamak 2
EQ (sonuc1,testSayi1);
Q01 = out();      // sonucu testSayi1 ile karşılaştır eşitse Q01 = 1, değilse Q01 = 0.

ld (I02);                       //Basamak 3
sonuc2 = SUB (sayi1,sayi2); // sayi1'den sayi2'yi çıkar
```

```
ld (LOGIC1);                    //Basamak 4
EQ (sonuc2,testSayi2);
Q02 = out();      // sonuc2'yi testSayi2 ile karşılaştır eşitse Q02 = 1, değilse Q02 = 0.

ld(I03);                        //Basamak 5
sonuc3 = MUL (sayi1,sayi2);// sayi1 ile sayi2'yi çarp

ld (LOGIC1);                    //Basamak 6
EQ (sonuc3,testSayi3);
Q03 = out();      // sonuc3'ü testSayi3 ile karşılaştır eşitse Q03 = 1, değilse Q03 = 0.

ld (I04);                       //Basamak 7
sonuc4 = DIV (sayi1,sayi2);  // sayi1'i sayi2'ye böl

ld (LOGIC1);                    //Basamak 8
EQ (sonuc4,testSayi4);
Q04 = out();      // sonuc4'ü testSayi4 ile karşılaştır eşitse Q04 = 1, değilse Q04 = 0.
```

Şekil 9.2'de görülen Örnek 1'e ait oluşturulan C kodunda: İlk PLC taramasında bir defaya mahsus olmak üzere şu atama işlemleri gerçekleştirilir: sonuc1 = sonuc2 = sonuc3 = sonuc4 = 0, testSayi1 = 2210, testSayi2 = 2190, testSayi3 = 22000, testSayi4 = 220, sayi1 = 2200, sayi2 = 10. Farklı sonuçlar elde etmek için bu değerlerin değiştirilerek programın tekrar çalıştırılması gerekir. Basamak 1'de, I0.1 = 1 yapıldığında sonuc1 := sayi1 + sayi2 toplama işlemi gerçekleştirilir. Basamak 2'de, "sonuc1'in değeri testSayi1'nin değerine eşit mi?" şeklinde bir karşılaştırma işlemi yapılıp sonuç Q0.1 no'lu çıkışa gönderilmektedir. Basamak 3'te, I0.2 = 1 yapıldığında sonuc2 := sayi1 – sayi2 çıkarma işlemi gerçekleştirilir. Basamak 4'te, "sonuc2'nin değeri testSayi2'nin değerine eşit mi?"

şeklinde bir karşılaştırma işlemi yapılıp sonuç Q0.2 no'lu çıkışa gönderilmektedir. Basamak 5'te, I0.3 = 1 yapıldığında sonuc3 := sayi1 * sayi2 çarpma işlemi gerçekleştirilir. Basamak 6'da, "sonuc3'ün değeri testSayi3'ün değerine eşit mi?" şeklinde bir karşılaştırma işlemi yapılıp sonuç Q0.3 no'lu çıkışa gönderilmektedir. Basamak 7'de, I0.4 = 1 yapıldığında sonuc4 := sayi1 / sayi2 bölme işlemi gerçekleştirilir. Basamak 8'de, "sonuc4'ün değeri testSayi4'ün değerine eşit mi?" şeklinde bir karşılaştırma işlemi yapılıp sonuç Q0.4 no'lu çıkışa gönderilmektedir.



Şekil 9.2. Örnek 1'e ait merdiven diyagramı

9.1.2. Örnek 2

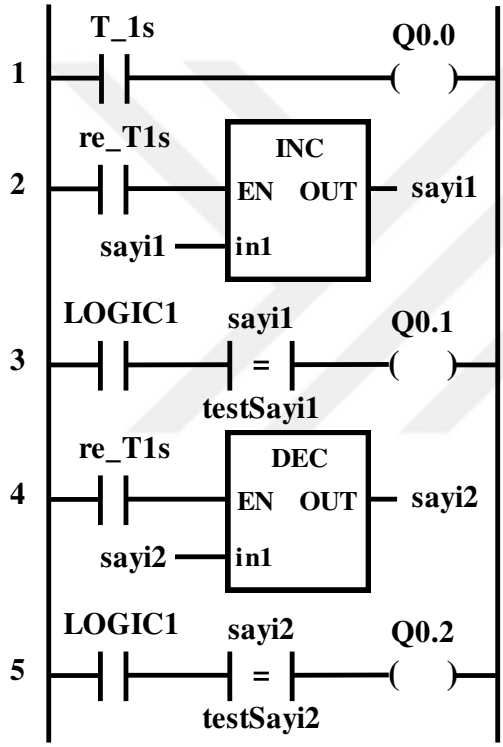
Şekil 9.3'te merdiven diyagramı görülen Örnek 2'ye ait kullanıcı programı aşağıda verilmiştir:

```
unsigned int testSayi1 = 10;

unsigned int testSayi2 = 15;

unsigned int sayi1 = 0;

unsigned int sayi2 = 30;
```

```
Q00 = in_out(T_1s);          //Basamak 1  - T_1s: 1 saniyelik periyoda sahip sinyal

 ld (re_T1s);                //Basamak 2
sayi1 = INC (sayi1);         // re_T1s: 1 saniyelik periyoda sahip sinyalin yükselen kenarı

ld (LOGIC1);                 //Basamak 3
EQ (sayi1,testSayi1);
Q01 = out();      // sayi1'i testSayi1 ile karşılaştır eşitse Q01 = 1, değilse Q01 = 0.

 ld (re_T1s);                //Basamak 4
sayi2 = DEC (sayi2);         // re_T1s: 1 saniyelik periyoda sahip sinyalin yükselen kenarı

ld (LOGIC1);                 //Basamak 5
EQ (sayi2,testSayi2);
Q02 = out();      // sayi2'yi testSayi2 ile karşılaştır eşitse Q02 = 1, değilse Q02 = 0.
```

Şekil 9.3'de görülen Örnek 2'ye ait oluşturulan C kodunda: İlk PLC taramasında bir defaya mahsus olmak üzere şu atama işlemleri gerçekleştirilir: testSayi1 = 10, testSayi2 = 15, sayi1 = 0, sayi2 = 30. Farklı sonuçlar elde etmek için bu değerlerin değiştirilerek programın tekrar çalıştırılması gerekir. Basamak 1'de, Q0.0 = T_1s yapılır. T_1s, 1 saniyelik periyoda sahip olan bir referans zamanlama sinyalidir. Bu sinyal 0.5 saniye lojik 1 ve 0.5 saniye lojik 0 olacak şekilde sürekli olarak değişeceğinden bu değişim Q0.0 numaralı PLC çıkışından gözlemlenecektir. Basamak 2'de, re_T1s = 1 olduğunda yani 1 saniyelik periyoda sahip sinyalin (T_1s) yükselen kenarında (re_T1s) sayi1 = sayi1 + 1 işlemi gerçekleştirilir. Bunun anlamı, PLC çalışmaya başladıktan sonra her bir saniyede sayi1 değişkeninin değeri 1 arttırılacak demektir. Böylelikle ilk anda sayi1 = 0 olduğu için PLC çalışmaya başladıktan 10 saniye sonra sayi1 = 10 olacaktır. Basamak 3'de, "sayi1'in değeri testSayi1'nin değerine eşit mi?" şeklinde bir karşılaştırma işlemi yapılıp sonuç Q0.1

no'lu çıkışa gönderilmektedir. Basamak 4'te, re_T1s = 1 olduğunda yani 1 saniyelik periyoda sahip sinyalin (T_1s) yükselen kenarında (re_T1s) sayi2 = sayi2 – 1 işlemi gerçekleştirilir. Bunun anlamı, PLC çalışmaya başladıktan sonra her bir saniyede sayi2 değişkeninin değeri 1 azaltılacak demektir. Böylelikle ilk anda sayi2 = 30 olduğu için PLC çalışmaya başladıktan 15 saniye sonra sayi2 = 15 olacaktır. Basamak 5'te, "sayi2'in değeri testSayi2'nin değerine eşit mi?" şeklinde bir karşılaştırma işlemi yapılıp sonuç Q0.2 no'lu çıkışa gönderilmektedir.



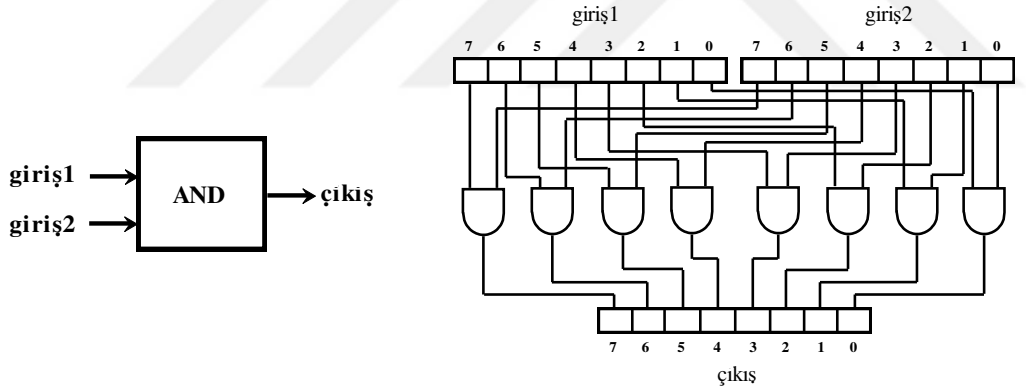
Şekil 9.3. Örnek 2'ye ait merdiven diyagramı

10. LOJİK VE KAYDIRMA FONKSİYONLARI

Bu bölümde bu tez çalışması kapsamında gerçekleştirilen AND, OR, NAND, NOR, XOR, XNOR ve NOT lojik fonksiyonları ile SHIFT_L ve SHIFT_R kaydırma fonksiyonları açıklanmaktadır.

10.1. Lojik Fonksiyonları

Bir lojik fonksiyon AND, OR, NAND, NOR, exclusive-OR (XOR) ve exclusive-NOR (XNOR) lojik işlemlerini iki giriş değişkenine (kaydediciye) ve NOT (değil) lojik işlemini sadece bir giriş değişkenine (kaydediciye) uygular ve bulunan sonucu çıkış değişkenine (kaydediciye) depolar. Örneğin Şekil 10.1.(a)'da görülen AND lojik fonksiyonu giriş1 ve giriş2 olarak isimlendirilmiş kaynaklardan aldığı iki veri girişini lojik AND işlemine tabi tutup sonucu çıkış değişkenine kaydeder. Örnek olarak AND fonksiyonunun 8 bitlik değişkenlere uygulanması Şekil 10.1.(b)'de görüldüğü gibi gerçekleştirilir.

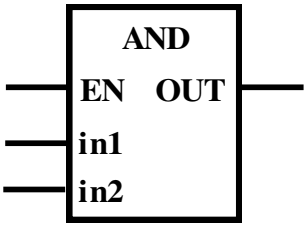


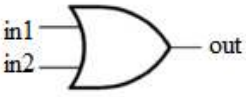
Şekil 10.1. a) AND fonksiyonu b) AND fonksiyonunun 8 bitlik değişkenlere uygulanması

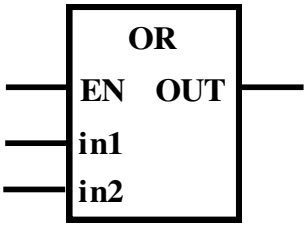
Tablo 10.1'de lojik fonksiyonlara ait işlemlerin sembolleri ve geliştirilen C kodları birlikte sunulmuştur. Bu fonksiyonlar yetkileme girişi EN (TEMP) aktif olduğunda "in1" ve "in2" giriş değişkenlerinden gelen verilerle işlem yapar ve sonucu çıkış değişkeni OUT (TEMP_REG) kaydeder. Diğer fonksiyonlardan farklı olarak NOT lojik fonksiyonunda giriş değişkeni (in) bir tanedir. Bu bölümde incelenen lojik işlemler giriş ve çıkış verilerinin unsigned int veri tipinde olması durumu için tanımlanmıştır. Benzer lojik işlemler, lojik işlem yapmanın anlamlı olduğu diğer veri tipleri için de tanımlanabilir.

Tablo 10.1. Lojik fonksiyonlar

<table>
<tr><th>Fonksiyon</th><th>Lojik işlemin sembolü ve doğruluk tablosu</th><th>Lojik Fonksiyonunun Sembolü</th><th>Geliştirilen C Kodu</th></tr>
<tr><td>AND</td><td> 

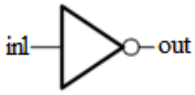
<table>
<tr><th>in1</th><th>in2</th><th>out</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table></td><td> 

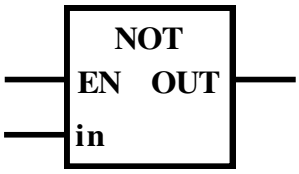
EN : Aktif 1 yetkileme girişi (TEMP üzerinden alınır) (0 veya 1)
in1 : Giriş değişkeni 1
in2 : Giriş değişkeni 2
OUT : Çıkış değişkeni (işlem sonucu TEMP_REG'e depolanır)</td><td>unsigned int AND (unsigned int in1, unsigned int in2)
{
if (TEMP)
TEMP_REG = in1&in2;
return TEMP_REG;
}</td></tr>
<tr><td>OR</td><td> 

<table>
<tr><th>in1</th><th>in2</th><th>out</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> 

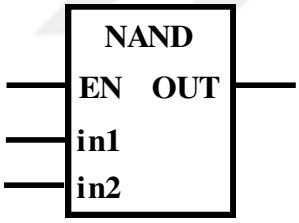
EN : Aktif 1 yetkileme girişi (TEMP üzerinden alınır) (0 veya 1)
in1 : Giriş değişkeni 1
in2 : Giriş değişkeni 2
OUT : Çıkış değişkeni (işlem sonucu TEMP_REG'e depolanır)</td><td>unsigned int OR (unsigned int in1, unsigned int in2)
{
if (TEMP)
TEMP_REG = in1|in2;
return TEMP_REG;
}</td></tr>
</table>

Tablo 10.1. Lojik fonksiyonlar (devam)

<table>
<tr><th>Fonksiyon</th><th>Lojik işlemin sembolü ve doğruluk tablosu</th><th>Lojik Fonksiyonunun Sembolü</th><th>Geliştirilen C Kodu</th></tr>
<tr><td>NOT</td><td> 

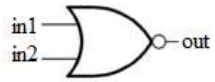
<table>
<tr><th>in</th><th>out</th></tr>
<tr><td>0</td><td>1</td></tr>
<tr><td>1</td><td>0</td></tr>
</table></td><td> 

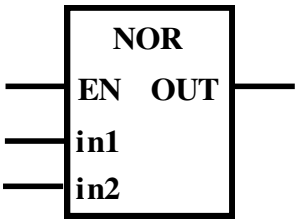
EN : Aktif 1 yetkileme girişi (TEMP üzerinden alınır) (0 veya 1)
in : Giriş değişkeni
OUT : Çıkış değişkeni (işlem sonucu TEMP_REG'e depolanır)</td><td>unsigned int NOT (unsigned int in)
{
if (TEMP)
TEMP_REG = ~ in;
return TEMP_REG;
}</td></tr>
<tr><td>NAND</td><td> 

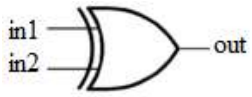
<table>
<tr><th>in1</th><th>in2</th><th>out</th></tr>
<tr><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>0</td></tr>
</table></td><td> 

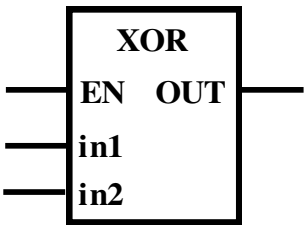
EN : Aktif 1 yetkileme girişi (TEMP üzerinden alınır) (0 veya 1)
in1 : Giriş değişkeni 1
in2 : Giriş değişkeni 2
OUT : Çıkış değişkeni (işlem sonucu TEMP_REG'e depolanır)</td><td>unsigned int NAND (unsigned int in1, unsigned int in2)
{
if (TEMP)
TEMP_REG = ~ (in1&in2);
return TEMP_REG;
}</td></tr>
</table>

Tablo 10.1. Lojik fonksiyonlar (devam)

<table>
<tr><th>Fonksiyon</th><th>Lojik işlemin sembolü ve doğruluk tablosu</th><th>Lojik Fonksiyonunun Sembolü</th><th>Geliştirilen C Kodu</th></tr>
<tr><td>NOR</td><td> 

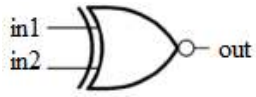
<table>
<tr><th>in1</th><th>in2</th><th>out</th></tr>
<tr><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>0</td></tr>
</table></td><td> 

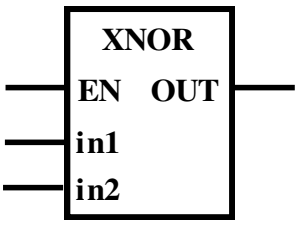
EN : Aktif 1 yetkileme girişi (TEMP üzerinden alınır) (0 veya 1)
in1 : Giriş değişkeni 1
in2 : Giriş değişkeni 2
OUT : Çıkış değişkeni (işlem sonucu TEMP_REG'e depolanır)</td><td><pre>unsigned int NOR (unsigned int in1, unsigned int in2)
{
        if (TEMP)
            TEMP_REG = ~ (in1|in2);
        return TEMP_REG;
}</pre></td></tr>
<tr><td>XOR</td><td> 

<table>
<tr><th>in1</th><th>in2</th><th>out</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>0</td></tr>
</table></td><td> 

EN : Aktif 1 yetkileme girişi (TEMP üzerinden alınır) (0 veya 1)
in1 : Giriş değişkeni 1
in2 : Giriş değişkeni 2
OUT : Çıkış değişkeni (işlem sonucu TEMP_REG'e depolanır)</td><td><pre>unsigned int XOR (unsigned int in1, unsigned int in2)
{
    if (TEMP)
        TEMP_REG = in1^in2;
    return TEMP_REG;
}</pre></td></tr>
</table>

Tablo 10.1. Lojik fonksiyonlar (devam)

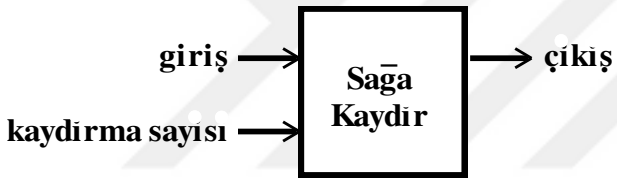
<table>
<tr><th>Fonksiyon</th><th>Lojik işlemin sembolü ve doğruluk tablosu</th><th>Lojik Fonksiyonunun Sembolü</th><th>Geliştirilen C Kodu</th></tr>
<tr><td>XNOR</td><td> 

<table>
<tr><th>in1</th><th>in2</th><th>out</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>1</td></tr>
</table></td><td> 

EN : Aktif 1 yetkileme girişi (TEMP üzerinden alınır) (0 veya 1)
in1 : Giriş değişkeni 1
in2 : Giriş değişkeni 2
OUT : Çıkış değişkeni (işlem sonucu TEMP_REG'e depolanır)</td><td>unsigned int XNOR (unsigned int in1, unsigned int in2)
{
if (TEMP)
TEMP_REG = ~ (in1^in2);
return TEMP_REG;
}</td></tr>
</table>

10.2. Kaydırma Fonksiyonları

Bir kaydırma (Shift) fonksiyonu bir değişken (kaydedici) içerisindeki bitleri sağa ya da sola hareket ettirir. Örneğin Şekil 10.2'de giriş değişkenindeki (kaydedici) veriyi kaydırma sayısı ile belirlenen sayıda sağa kaydırıp elde edilen sonucu çıkış değişkenine (kaydedici) depolayan bir sağa kaydırma fonksiyonu gösterilmektedir. Bu durumda giriş değişkenindeki bitler kaydırma sayısı ile belirlenen sayıda sağa kaydırılırken soldan (MSB) kaydırma sayısı kadar 0 değeri sonuca eklenmektedir. Sola kaydırma fonksiyonu da benzer şekilde çalışır. Fakat kaydırma işlemi kaydırma sayısı ile belirlenen sayıda kaydırma yönü sola olacak şekilde gerçekleştirilir. Bu durumda giriş değişkenindeki bitler kaydırma sayısı ile belirlenen sayıda sola kaydırılırken sağdan (LSB) kaydırma sayısı kadar 0 değeri sonuca eklenmektedir.

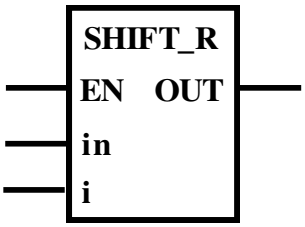


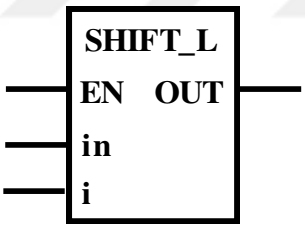
Şekil 10.2. Sağa kaydırma fonksiyonu

Bu proje kapsamında Tablo 10.2'de görülen SHIFT_R (sağa kaydırma) ve SHIFT_L (sola kaydırma) fonksiyonları gerçekleştirilmiştir Bu fonksiyonlar yetkileme girişi EN (TEMP) aktif olduğunda "in" giriş değişkeninden gelen verileri alarak sağa veya sola kaydırma işlemi yaparak sonucu çıkış değişkeni OUT'a (TEMP_REG) kaydeder. Tablo 10.2'de kaydırma fonksiyonlarına ait işlemlerin sembolleri ve geliştirilen C kodları birlikte sunulmuştur. Bu kısımda incelenen kaydırma işlemleri giriş ve çıkış verilerinin unsigned int veri tipinde olması durumu için tanımlanmıştır. Benzer kaydırma işlemleri diğer veri tipleri için de tanımlanabilir.

Tablo 10.2. Kaydırma fonksiyonları

Fonksiyon	Kaydırma Fonksiyonunun Sembolü	Geliştirilen C Kodu

SHIFT_R (Sağa Kaydırma Fonksiyonu)	
	EN : Aktif 1 yetkileme girişi (TEMP üzerinden alınır) (0 veya 1)
	in : Giriş değişkeni
	i : kaydırma sayısı (1, 2, …, 32)
	OUT : Çıkış değişkeni (işlem sonucu TEMP_REG'e depolanır)
		unsigned int SHIFT_R (unsigned int in, unsigned int i)
		{
		if ((i<33)&& (TEMP))
		TEMP_REG = (in << i);
		return TEMP_REG;
		}

SHIFT_L (Sola Kaydırma Fonksiyonu)	
	EN : Aktif 1 yetkileme girişi (TEMP üzerinden alınır) (0 veya 1)
	in : Giriş değişkeni
	i : kaydırma sayısı (1, 2, …, 32)
	OUT : Çıkış değişkeni (işlem sonucu TEMP_REG'e depolanır)
		unsigned int SHIFT_L (unsigned int in, unsigned int i)
		{
		if ((i<33)&& (TEMP))
		TEMP_REG = (in >> i);
		return TEMP_REG;
		}

10.3. Lojik Fonksiyon Örneği

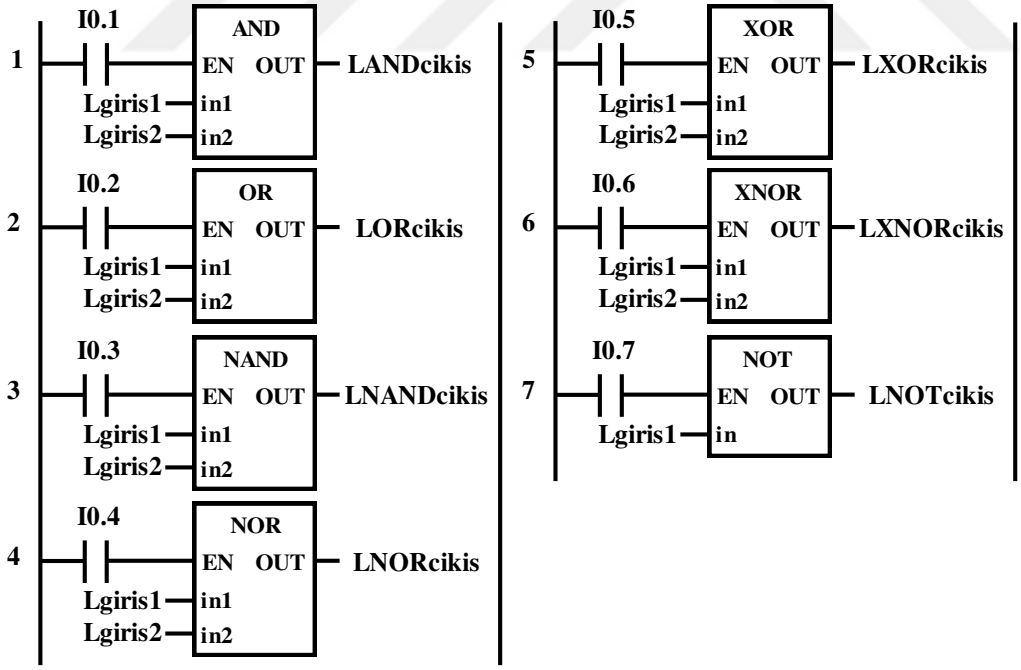
Bu kısımda lojik fonksiyonlara ait bir örnek yer almaktadır. Bu örnek için merdiven diyagramı ve kullanıcı programına ait C kodu sunulmuştur. Şekil 10.1'de merdiven diyagramı görülen Örnek 1'e ait kullanıcı programı aşağıda verilmiştir:

```
unsigned int Lgiris1 = 0b0000000000001100;

unsigned int Lgiris2 = 0b0000000000001010;

unsigned int LANDcikis = 0;

unsigned int LORcikis = 0;

unsigned int LNANDcikis = 0;

unsigned int LNORcikis = 0;

unsigned int LXORcikis = 0;

unsigned int LXNORcikis = 0;

unsigned int LNOTcikis = 0;


ld (I01);                    //Basamak 1

LANDcikis = AND (Lgiris1, Lgiris2);


ld (I02);                    //Basamak 2

LORcikis = OR (Lgiris1, Lgiris2);


ld (I03);                    //Basamak 3

LNANDcikis = NAND (Lgiris1, Lgiris2);


ld (I04);                    //Basamak 4

LNORcikis = NOR (Lgiris1, Lgiris2);
```

```
ld (I05);                    //Basamak 5
LXORcikis = XOR (Lgiris1, Lgiris2);

ld (I06);                    //Basamak 6
LXNORcikis = XNOR (Lgiris1, Lgiris2);

ld (I07);                    //Basamak 7
LNOTcikis = NOT (Lgiris1);

if(I01)_printf("AND işleminin sonucu = 0x%08X\n\r ", LANDcikis);
if(I02)_printf("OR işleminin sonucu = 0x%08X\n\r ", LORcikis);
if(I03)_printf("NAND işleminin sonucu = 0x%08X\n\r ", LNANDcikis);
if(I04)_printf("NOR işleminin sonucu = 0x%08X\n\r ", LNORcikis);
if(I05)_printf("XOR işleminin sonucu = 0x%08X\n\r ", LXORcikis);
if(I06)_printf("XNOR işleminin sonucu = 0x%08X\n\r ", LXNORcikis);
if(I07)_printf("NOT işleminin sonucu = 0x%08X\n\r ", LNOTcikis);
```

Şekil 10.3'te görülen Örnek 1'e ait oluşturulan C kodunda: İlk PLC taramasında bir defaya mahsus olmak üzere şu atama işlemleri gerçekleştirilir: Lgiris1 = 0b0000000000001100, Lgiris2 = 0b0000000000001010, LANDcikis = LORcikis = LNANDcikis = LNORcikis = LXORcikis = LXNORcikis = LNOTcikis = 0. Farklı sonuçlar elde etmek için bu değerlerin değiştirilerek programın tekrar çalıştırılması gerekir. Basamak 1'de, I0.1 = 1 yapıldığında LANDcikis := Lgiris1 AND Lgiris2 lojik işlemi gerçekleştirilir. Mevcut Lgiris1 ve Lgiris2 giriş değerlerine göre LANDcikis = 0b0000000000001000 olur. Basamak 2'de, I0.2 = 1 yapıldığında LORcikis := Lgiris1 OR Lgiris2 lojik işlemi gerçekleştirilir. Mevcut Lgiris1 ve Lgiris2 giriş değerlerine göre LORcikis = 0b0000000000001110 olur. Basamak 3'te, I0.3 = 1 yapıldığında LNANDcikis := Lgiris1

NAND Lgiris2 lojik işlemi gerçekleştirilir. Mevcut Lgiris1 ve Lgiris2 giriş değerlerine göre LNANDcikis = 0b0000000000000111 olur. Basamak 4'te, I0.4 = 1 yapıldığında LNORcikis := Lgiris1 NOR Lgiris2 lojik işlemi gerçekleştirilir. Mevcut Lgiris1 ve Lgiris2 giriş değerlerine göre LNORcikis = 0b0000000000000001 olur. Basamak 5'te, I0.5 = 1 yapıldığında LXORcikis := Lgiris1 XOR Lgiris2 lojik işlemi gerçekleştirilir. Mevcut Lgiris1 ve Lgiris2 giriş değerlerine göre LXORcikis = 0b0000000000000110 olur. Basamak 6'da, I0.6 = 1 yapıldığında LXNORcikis := Lgiris1 XNOR Lgiris2 lojik işlemi gerçekleştirilir. Mevcut Lgiris1 ve Lgiris2 giriş değerlerine göre LXNORcikis = 0b0000000000001001 olur. Basamak 7'de, I0.7 = 1 yapıldığında LNOTcikis := Lgiris1 NOT Lgiris1 lojik işlemi gerçekleştirilir. Mevcut Lgiris1 giriş değerine göre LNOTcikis = 0b1111111111110011 olur. C kodunda olup merdiven diyagramında yer almayan kısımda ise bulunan lojik işlem sonuçlarının Bölüm 3'te anlatılan RS232 USB-TTL seri haberleşme dönüştürücü modülü ve PuTTY yazılımı ile bilgisayar ekranında gösterilmesi işlemi gerçekleştirilir.



Şekil 10.3. Örnek 1'e ait merdiven diyagramı

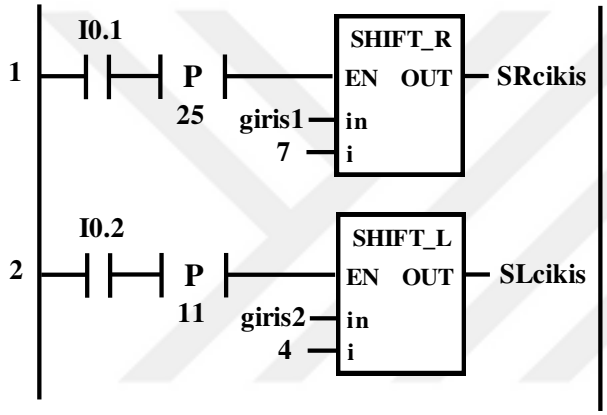
10.4. Kaydırma Fonksiyonu Örneği

Bu kısımda kaydırma fonksiyonlara ait bir örnek yer almaktadır. Bu örnek için merdiven diyagramı ve kullanıcı programına ait C kodu sunulmuştur. Şekil 10.2'de merdiven diyagramı görülen Örnek 2'ye ait kullanıcı programı aşağıda verilmiştir:

```
unsigned int giris1 = 0b1011000000000000;

unsigned int giris2 = 0b0000000000111100;

unsigned int SRcikis = 0;

unsigned int SLcikis = 0;


ld (I01);                    //Basamak 1

r_edge (25);

SRcikis = SHIFT_R (giris1, 7);


ld (I02);                    //Basamak 2

r_edge (11);

SRcikis = SHIFT_L (giris2, 4);


if(I01)_printf("Sağa kaydırma işleminin sonucu = 0x%08X\n\r ", SRcikis);

if(I02)_printf("Sola kaydırma işleminin sonucu = 0x%08X\n\r ", SLcikis);
```

Şekil 10.4'te görülen Örnek 2'ye ait oluşturulan C kodunda: İlk PLC taramasında bir defaya mahsus olmak üzere şu atama işlemleri gerçekleştirilir: giris1 = 0b1011000000000000, giris2 = 0b0000000000111100, SRcikis = SLcikis = 0. Farklı sonuçlar elde etmek için bu değerlerin değiştirilerek programın tekrar çalıştırılması gerekir. Basamak 1'de, I0.1'in yükselen kenarı algılanırsa SHIFT_R sağa kaydırma fonksiyonu yardımıyla, giris1 giriş değişkenindeki veri (0b1011000000000000) 7 kez sağa kaydırılır

ve bulunan sonuç SRcikis çıkış değişkenine yüklenir. Mevcut giris1 giriş değerine göre SRcikis = 0b0000000101100000 olur. Basamak 2'de, I0.2'in yükselen kenarı algılanırsa SHIFT_L sola kaydırma fonksiyonu yardımıyla, giris2 giriş değişkenindeki veri (0b0000000000111100) 4 kez sola kaydırılır ve bulunan sonuç SLcikis çıkış değişkenine yüklenir. Mevcut giris2 giriş değerine göre SLcikis = 0b0000001111000000 olur. C kodunda olup merdiven diyagramında yer almayan kısımda ise bulunan kaydırma işlemi sonuçlarının Bölüm 3'te anlatılan RS232 USB-TTL seri haberleşme dönüştürücü modülü ve PuTTY yazılımı ile bilgisayar ekranında gösterilmesi işlemi gerçekleştirilir.



Şekil 10.4. Örnek 2'ye ait merdiven diyagramı

11. SONUÇ

PLC'ler günümüzde endüstriyel otomasyon sistemlerinde çok yaygın olarak kullanılan kontrol cihazlarıdır. Pek çok yerli ve yabancı PLC üreticisi değişik kapasiteye ve hıza sahip PLC'leri farklı ihtiyaçları karşılamak üzere üretmektedir. PLC'lerin tasarımı konusunda daha önce yapılmış olan tez çalışmaları ve benzer şekilde konuyla ilgili Türkçe ve İngilizce yazılmış kitaplar da mevcuttur. Daha önce PLC'lerle ilgili yapılan tez çalışmalarında 8 dijital giriş/8 dijital çıkış ve 5 MHz'de çalışan bir PLC, 7 dijital giriş/6 dijital çıkışlı bir PLC ve PIC16F877 mikrodenetleyicisi kullanılarak 6 dijital girişli/5 dijital çıkışlı PLC'ler gerçekleştirilmiştir. Çalışma hızları nispeten yavaş olan bu çalışmalarda kullanılan mikrodenetleyiciler düşük maliyetli olması sebebiyle tercih edilmiştir. Bu yüksek lisans tez çalışması kapsamında özellikle küçük çaplı uygulamalar için hem hızlı hem de çok düşük maliyetli bir PLC tasarlanmış ve gerçekleştirilmiştir. Bu projede STMicroelectronics tarafından üretilen 32-bitlik STM32F103C8 (ARM Cortex-M3) mikrodenetleyicisi kullanılarak 72 MHz gibi yüksek bir hızda çalışan 8 dijital girişli/8 dijital çıkışlı bir PLC tasarımı gerçekleştirilmiştir. Böylelikle zaman-kritik sistemler için kullanılması söz konusu olabilecek bir PLC altyapısı ortaya konulmuştur. Bu tasarımın geliştirilmesiyle hem çok hızlı hem de geniş işlem gücü ve çevresel uyumluluğu sayesinde düşük maliyetli ve aynı zamanda yerli ve milli bir PLC tasarımı oluşturulmuştur. Uygulamada, PLC'nin temel işlevlerini gerçekleştirmek için gerekli olan I/O bağlantıları, STM32F103C8 mikrodenetleyici üzerindeki GPIO pinleri kullanılarak sağlanmıştır. Bu, harici bileşenlere olan ihtiyacı azaltıp tasarımı daha ekonomik hale getirmiştir. Yapılan çalışma STM32CubeIDE arayüz programı gibi popüler bütünleşik geliştirme ortamı üzerinde C dili kullanılarak gerçekleştirilmiştir. Bu da projenin daha da erişilebilir olmasını sağlamaktadır. Bu tez çalışmasıyla STM32F103C8 Mikrodenetleyicisi Temelli PLC Tasarımı yerli ve milli PLC olarak geliştirilmiştir. Ayrıca bu tez çalışmasının ileride yapılması söz konusu olabilecek daha kapsamlı çalışmalara örnek ve temel teşkil edeceği öngörülerek çalışmanın ileri aşaması olarak endüstriyel PLC'lerdekine benzer bir kullanıcı arayüzü geliştirilmesi önerilir.

12. KAYNAKÇA

AN4013 Application note - Introduction to timers for STM32 MCUs. (March 2024). https://www.st.com/resource/en/application_note/an4013-stm32-crossseries-timer-overview-stmicroelectronics.pdf

AN4899 Application note - STM32 microcontroller GPIO hardware settings and low-power consumption (March 2022). https://www.st.com/resource/en/application_note/an4899-stm32-microcontroller-gpio-hardware-settings-and-lowpower-consumption-stmicroelectronics.pdf

Blog – DTA Mühendislik. (2020, Nisan 03). Haziran 01, 2024 tarihinde DTA web sitesi: https://blog.dta.com.tr/filtrelere-giris-fir-ve-iir-filtreleri/ adresinden alındı

Deltaww. (t.y.). Mayıs 31, 2024 tarihinde Deltaww web sitesi: https://www.deltaww.com/en-us/products/PLC-Programmable-Logic-Controllers/ALL/ adresinden alındı

Digikey. (t.y.). Haziran 01, 2024 tarihinde Digikey web sitesi: https://www.digikey.com/-/media/Images/Article%20Library/TechZone%20Articles/2021/February/How%20to%20Implement%20Hardware%20Debounce%20for%20Switches%20and%20Relays/article-2021february-how-to-implement-hardware-fig1_fullsize.jpg?la=en&ts=fe043efa-d706-4 adresinden alındı

Diyot.net. (t.y.). Temmuz 01, 2024 tarihinde Diyot.net web sitesi: https://diyot.net/stm32f103/ adresinden alındı

Erkol, H. O. (2008). Mikrodenetleyici tabanlı PLC donanımı ve yazılımının gerçekleştirilmesi. [Yüksek lisans tezi, Erciyes Üniversitesi, Fen Bilimleri Enstitüsü].

Gmtcontrol. (t.y.). Mayıs 31, 2024 tarihinde Gmtcontrol web sitesi: https://gmtcontrol.com/ adresinden alındı

Harmanda, A. (2011). 16 bitlik bir PIC mikrodenetleyicisi ile bir programlanabilir lojik denetleyici tasarımı ve uygulaması. [Yüksek lisans tezi, Niğde Üniversitesi, Fen Bilimleri Enstitüsü].

Kılıçarslan, S. (2014). Mikrodenetleyici temelli PLC'er için programlama yazılımının gerçekleştirilmesi. [Yüksek lisans tezi, Gaziosmanpaşa Üniversitesi, Fen Bilimleri Enstitüsü].

Kitiş, Ş. (2007). PIC16F84 mikrodenetleyicisi ile bir programlanabilir lojik denetleyici tasarımı ve uygulaması. [Yüksek lisans tezi, Niğde Üniversitesi, Fen Bilimleri Enstitüsü].

Kontrol kalemi. (t.y.). Mayıs 31, 2024 tarihinde Kontrol kalemi web sitesi: https://www.kontrolkalemi.com/plc-nedir-plc-ne-ise-yarar/ adresinden alındı

Mikrodev. (t.y.). Mayıs 31, 2024 tarihinde Mikrodev web sitesi: https://www.mikrodev.com/tr/ adresinden alındı

Omron. (t.y). Mayıs 31, 2024 tarihinde Omron web sitesi: https://industrial.omron.com.tr/tr/products/programmable-logic-controllers adresinden alındı

Otomasyonavm. (t.y.). Haziran 01, 2024 tarihinde Otomasyonavm web sitesi: https://www.otomasyonavm.com/tr/ladder-diagram adresinden alındı

Putty. (t.y). Haziran 01, 2024 tarihinde Putty web sitesi: https://www.putty.org/ adresinden alındı

Rafat, Ö. F. (2010). PIC16F877 mikrodenetleyicisi ile bir PLC tasarımı. [Yüksek lisans tezi, Selçuk Üniversitesi, Fen Bilimleri Enstitüsü].

RM0008 Reference manual - STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced Arm®-based 32-bit MCUs (February 2021). https://www.st.com/resource/en/reference_manual/rm0008-stm32f101xx-stm32f102xx-STM32F103XX-stm32f105xx-and-stm32f107xx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

Siemens. (t.y.). Mayıs 31, 2024 tarihinde Siemens web sitesi: https://www.siemens.com/global/en/products/automation/systems/industrial/plc.html adresinden alındı

STM32F103 pinout diagram. (t.y). Mayıs 31, 2024 tarihinde wikipedia web sitesi: https://upload.wikimedia.org/wikipedia/commons/9/90/Stm32f103_pinout_diagram.png adresinden alındı

Tongur, V. (2008). Atmega128 tabanlı PLC tasarımı. [Yüksek lisans tezi, Selçuk Üniversitesi, Fen Bilimleri Enstitüsü].

Uzam, M. (2013). *PIC16F877A temelli PLC*. Birsen Yayınevi. http://www.birsenyayinevi.com/pic16f877-a-temelli-plc-murat-uzam adresinden alındı

Uzam, M. (2022). *PIC16F877A temelli PLC sürüm 2.0* . Kodlab Yayınevi. https://www.kodlab.com/ana-sayfa/626-pici6f877a-temelli-plc-sueruem-20-9786257440448.html adresinden alındı

EKLER

Ek-1 STM32CubeIDE geliştirme programında PLC tasarımı için gerçekleştirilen C kodu

MAIN BÖLÜMÜ

```
#include "main.h"
#include "definitions.h"
#include <stdarg.h>

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM1_Init(void);
float filteredState[8] = {0};


int main(void)
{

	HAL_Init();
	SystemClock_Config();
	MX_GPIO_Init();
	MX_TIM1_Init();
	USART2_setup();

	//_Bool tmp=0;
	while (1)
	{

		TIMER = TIMER_Val;
//_____________________________________________Timerlar için kullanılacak
		re_T10ms = re_RTS (T_10ms, 61);
		re_T100ms = re_RTS (T_100ms, 62);
		re_T1s = re_RTS (T_1s, 63);

		get_inputs();
//_____________________________________________
		//Kullanıcı Programı burada yer alır

//_____________________________________________

		send_outputs();
	}

}

void SystemClock_Config_HSI(void)
```

```
{
	RCC_OscInitTypeDef RCC_OscInitStruct = {0};
	RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

	RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
	RCC_OscInitStruct.HSIState = RCC_HSI_ON;
	RCC_OscInitStruct.HSICalibrationValue =
RCC_HSICALIBRATION_DEFAULT;
	RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
	if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
	{
		Error_Handler();
	}


	RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
	RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
	RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
	RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
	RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
	RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

	if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) !=
HAL_OK)
	{
		Error_Handler();
	}
}


void SystemClock_Config(void)
{
	RCC_OscInitTypeDef RCC_OscInitStruct = {0};
	RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

	RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;// high
speed external 8Mhz
	RCC_OscInitStruct.HSEState = RCC_HSE_ON;
	RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
	RCC_OscInitStruct.HSIState = RCC_HSI_ON;
	RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON; // PLL
	RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;//8MHz
	RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;  // 8x9=72MHz
	if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
	{
		Error_Handler();
	}
```

```
	RCC_ClkInitStruct.ClockType                                                    =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                        |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
	RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
	RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
	RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
	RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

	if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) !=
HAL_OK)
	{
		Error_Handler();
	}
}

static void _printchar(char **str, int c)
{
	if (str) {
		**str = c;
		++(*str);
	}
	else
	{
		//USART1->TDR = (ch & (uint16_t)0x01FF);
		//	while ((USART1->ISR & USART_ISR_TC) == (uint16_t) RESET);

		while ( (USART2->SR & USART_SR_TC) != USART_SR_TC);
		USART2->DR = c;
	}
}
#define PAD_RIGHT 1
#define PAD_ZERO 2
static int _prints(char **out, const char *string, int width, int pad)
{
	register int pc = 0, padchar = ' ';

	if (width > 0) {
		register int len = 0;
		register const char *ptr;
		for (ptr = string; *ptr; ++ptr) ++len;
		if (len >= width) width = 0;
		else width -= len;
		if (pad & PAD_ZERO) padchar = '0';
	}
	if (!(pad & PAD_RIGHT)) {
		for ( ; width > 0; --width) {
			_printchar (out, padchar);
```

```
                        ++pc;
                }
        }
        for ( ; *string ; ++string) {
                _printchar (out, *string);
                ++pc;
        }
        for ( ; width > 0; --width) {
                _printchar (out, padchar);
                ++pc;
        }

        return pc;
}
/* the following should be enough for 32 bit int */
#define PRINT_BUF_LEN 12

static int _printi(char **out, int i, int b, int sg, int width, int pad, int letbase)
{
        char print_buf[PRINT_BUF_LEN];
        register char *s;
        register int t, neg = 0, pc = 0;
        register unsigned int u = i;

        if (i == 0) {
                print_buf[0] = '0';
                print_buf[1] = '\0';
                return _prints (out, print_buf, width, pad);
        }

        if (sg && b == 10 && i < 0) {
                neg = 1;
                u = -i;
        }

        s = print_buf + PRINT_BUF_LEN-1;
        *s = '\0';

        while (u) {
                t = u % b;
                if( t >= 10 )
                        t += letbase - '0' - 10;
                *--s = t + '0';
                u /= b;
        }

        if (neg) {
                if( width && (pad & PAD_ZERO) ) {
```

```
                    _printchar (out, '-');
                    ++pc;
                    --width;
                }
                else {
                    *--s = '-';
                }
        }

        return pc + _prints (out, s, width, pad);
}

static int _print(char **out, const char *format, va_list args )
{
        register int width, pad;
        register int pc = 0;
        char scr[2];

        for (; *format != 0; ++format) {
                if (*format == '%') {
                    ++format;
                    width = pad = 0;
                    if (*format == '\0') break;
                    if (*format == '%') goto out;
                    if (*format == '-') {
                        ++format;
                        pad = PAD_RIGHT;
                    }
                    while (*format == '0') {
                        ++format;
                        pad |= PAD_ZERO;
                    }
                    for ( ; *format >= '0' && *format <= '9'; ++format) {
                        width *= 10;
                        width += *format - '0';
                    }
                    if( *format == 's' ) {
                        register char *s = (char *)va_arg( args, int );
                        pc += _prints (out, s?s:"(null)", width, pad);
                        continue;
                    }
                    if( *format == 'd' ) {
                        pc += _printi (out, va_arg( args, int ), 10, 1, width, pad, 'a');
                        continue;
                    }
                    if( *format == 'x' ) {
                        pc += _printi (out, va_arg( args, int ), 16, 0, width, pad, 'a');
                        continue;
```

```
                        }
                        if( *format == 'X' ) {
                                pc += _printi (out, va_arg( args, int ), 16, 0, width, pad, 'A');
                                continue;
                        }
                        if( *format == 'u' ) {
                                pc += _printi (out, va_arg( args, int ), 10, 0, width, pad, 'a');
                                continue;
                        }
                        if( *format == 'c' ) {
                                /* char are converted to int then pushed on the stack */
                                scr[0] = (char)va_arg( args, int );
                                scr[1] = '\0';
                                pc += _prints (out, scr, width, pad);
                                continue;
                        }
                }
                else {
                out:
                        _printchar (out, *format);
                        ++pc;
                }
        }
        if (out) **out = '\0';
        va_end( args );
        return pc;
}

int _printf(const char *format, ...)
{
        va_list args;

        va_start( args, format );
        return _print( 0, format, args );
}

int sprintf(char *out, const char *format, ...)
{
        va_list args;

        va_start( args, format );
        return _print( &out, format, args );
}

uint32_t disable_irq()
{
        uint32_t prim = __get_PRIMASK();
        __disable_irq();
```

```
	return prim;
}

void enable_irq(uint32_t prim)
{
	if (!prim)
	{
		__enable_irq();
		__ISB();
	}
}

/*
void Timer3_setup(void) // 50ms
{
 RCC->APBENR1 |= RCC_APBENR1_TIM3EN;      // Enable clock for Timer 5
	TIM3->PSC = 0;
 TIM3->ARR     = 15999;       // Auto Reload value: (72000000Hz/(PSC+1))*0.05ms
 TIM3->CR1    = TIM_CR1_CEN;       // Enable Counting
	TIM3->DIER   = TIM_DIER_UIE;                        // Enable/Disable INTERRUPT
}

 */

static void MX_TIM1_Init(void)
{

	RCC->APB2ENR |= RCC_APB2ENR_TIM1EN;


	//32320
	// configure TIM1
	TIM1->CR1 = 0;		// dir=0 - upcounting
	TIM1->ARR = 1;		// upper bound
	TIM1->PSC = 35999;	// f=72MHz    F= f/((psc+1)*(ARR+1))   1000hz -> 1 ms
	TIM1->EGR = TIM_EGR_UG;		// Generate an update event to reload the Prescaler and the Repetition counter values immediately
	// start communication
	TIM1->SR = 0;      // clear status
	TIM1->DIER = TIM_DIER_UIE;    // Update interrupt enable
	NVIC_EnableIRQ(TIM1_UP_IRQn);

	TIM1->CR1 |= TIM_CR1_CEN;

}

static void MX_GPIO_Init(void)
```

```
{
        GPIO_InitTypeDef GPIO_InitStruct = {0};

        __HAL_RCC_GPIOC_CLK_ENABLE();
        __HAL_RCC_GPIOB_CLK_ENABLE(); // b portu enable yap
        __HAL_RCC_GPIOA_CLK_ENABLE();

        /*Configure GPIO pin Output Level */
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_11, GPIO_PIN_RESET);

        /*Configure GPIO pin : */
////////////////////////////////  C13 input stm üzerindeki  yeşil led
        GPIO_InitStruct.Pin = GPIO_PIN_13;
        GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
        GPIO_InitStruct.Pull = GPIO_NOPULL;
        GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
        HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

//_____________________________________________________
// INPUTS (GİRİŞLER)
////////////////////////////////  B4 input I0.7
GPIO_InitStruct.Pin = GPIO_PIN_4;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;     // GPIO_PULLDOWN   GPIO_PULLUP
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
////////////////////////////////  B3 input I0.6
GPIO_InitStruct.Pin = GPIO_PIN_3;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;     // GPIO_PULLDOWN   GPIO_PULLUP
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
////////////////////////////////  A15 input I0.5
GPIO_InitStruct.Pin = GPIO_PIN_15;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;     // GPIO_PULLDOWN   GPIO_PULLUP
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
////////////////////////////////  B6 input I0.4
GPIO_InitStruct.Pin = GPIO_PIN_6;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;     // GPIO_PULLDOWN   GPIO_PULLUP
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
////////////////////////////////  A11 input I0.3
GPIO_InitStruct.Pin = GPIO_PIN_11;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;     // GPIO_PULLDOWN   GPIO_PULLUP
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
////////////////////////////////  A10 input I0.2
GPIO_InitStruct.Pin = GPIO_PIN_10;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
```

```
GPIO_InitStruct.Pull = GPIO_NOPULL;     // GPIO_PULLDOWN GPIO_PULLUP
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
//////////////////////////////  A9 input I0.1
GPIO_InitStruct.Pin = GPIO_PIN_9;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;     // GPIO_PULLDOWN GPIO_PULLUP
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
//////////////////////////////  A8 input I0.0
GPIO_InitStruct.Pin = GPIO_PIN_8;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;     // GPIO_PULLDOWN GPIO_PULLUP
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

//_____________________________________________________
// OUTPUTS (ÇIKIŞLAR)
//////////////////////////////  A4 output Q0.7
GPIO_InitStruct.Pin = GPIO_PIN_4;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP; //GPIO_MODE_OUTPUT_OD
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
//////////////////////////////  A5 output Q0.6
GPIO_InitStruct.Pin = GPIO_PIN_5;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP; //GPIO_MODE_OUTPUT_OD
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
//////////////////////////////  A6 output Q0.5
GPIO_InitStruct.Pin = GPIO_PIN_6;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP; //GPIO_MODE_OUTPUT_OD
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
//////////////////////////////  A7 output Q0.4
GPIO_InitStruct.Pin = GPIO_PIN_7;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP; //GPIO_MODE_OUTPUT_OD
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
//////////////////////////////  B0 output Q0.3
GPIO_InitStruct.Pin = GPIO_PIN_0;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP; //GPIO_MODE_OUTPUT_OD
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
//////////////////////////////  B1 output Q0.2
GPIO_InitStruct.Pin = GPIO_PIN_1;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP; //GPIO_MODE_OUTPUT_OD
```

```
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
//////////////////////////////  B10 output Q0.1
GPIO_InitStruct.Pin = GPIO_PIN_10;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP; //GPIO_MODE_OUTPUT_OD
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
//////////////////////////////  B11 output Q0.0
GPIO_InitStruct.Pin = GPIO_PIN_11;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP; //GPIO_MODE_OUTPUT_OD
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
//_____________________________________________________

//RS 232 USB-TTL Seri Haberleşme Dönüştürücü Modülü GPIO
GPIO_InitStruct.Pin = GPIO_PIN_2;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;     // GPIO_PULLDOWN   GPIO_PULLUP
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

GPIO_InitStruct.Pin = GPIO_PIN_3;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;     // GPIO_PULLDOWN   GPIO_PULLUP
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
//_____________________________________________________

        /*
                GPIO_InitStruct.Pin = GPIO_PIN_2;
                GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
                GPIO_InitStruct.Pull = GPIO_NOPULL;
                GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
                HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

                GPIO_InitStruct.Pin = GPIO_PIN_3;
                GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
                GPIO_InitStruct.Pull = GPIO_NOPULL;
                GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
                HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);*/

//////////////////////////////  A12 input
GPIO_InitStruct.Pin = GPIO_PIN_12;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
//////////////////////////////  A0 output
```

```
GPIO_InitStruct.Pin = GPIO_PIN_0;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
//////////////////////////////  A1 output
GPIO_InitStruct.Pin = GPIO_PIN_1;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
//////////////////////////////  C14 output
GPIO_InitStruct.Pin = GPIO_PIN_14;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
//////////////////////////////  C15 output
GPIO_InitStruct.Pin = GPIO_PIN_15;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

}

void USART2_setup(void)
{
        RCC->APB1ENR |= RCC_APB1ENR_USART2EN;

        USART2->CR1 &= (uint16_t)~((uint32_t)USART_CR1_UE);
        USART2->CR1 = 0x0000;
        USART2->CR2 = 0x0000;
        USART2->CR3 = 0x0000;

   USART2->BRR = 3750;// 1875  3072 512   104.1666667 0x683

        USART2->CR1 |= USART_CR1_TE ;
        USART2->CR1 |= USART_CR1_UE ;

}
//_____________INTERRUPT(KESME)_____________
_Bool re_RTS (_Bool RTS, const int i)
{
        TEMP=0;
        if(RTS==0)
        {
                RED_T[i]=1;
```

```
        }
        else if (RED_T[i]==1)
        {
                TEMP=1;
                RED_T[i] =0;
        }
        else
        {
                TEMP=0;
        }
return TEMP;
}

//____________________ Kontak ve röle temelli fonksiyonlar_________________
void ld(_Bool in)
{
        if (in==0) TEMP=0;
        else TEMP=1;
}

void ld_not(_Bool in)
{
        if (in==0) TEMP=1;
        else TEMP=0;
}
_Bool out (void)
{
        if (TEMP==0) TEMP=0;
        else TEMP=1;
        return TEMP;
}

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

_Bool mid_out (_Bool m_out)
{
        m_out=TEMP;
        return m_out;
}
_Bool mid_out_not (_Bool m_out)
{
        m_out=!TEMP;
        return m_out;
```

```
}

_Bool inv_out (_Bool in)
{
        return (!in);
}

_Bool _set (_Bool out)
{
        if (TEMP) out=1;
        return (out);
}

_Bool _reset (_Bool out)
{
        if (TEMP) out=0;
        return (out);
}

_Bool SR (_Bool S, _Bool R1, _Bool out)
{
        if (R1) out=0;
        else if (S) out=1;
        return (out);
}

_Bool RS (_Bool R, _Bool S1, _Bool out)
{
        if (S1) out=1;
        else if (R) out=0;
        return (out);
}
_Bool r_toggle(const int i, _Bool out)
{
                if(TEMP==0)
                {
                        RED_TOGGLE[i]=1;
                }
                else if (RED_TOGGLE[i]==1)
                {
                        RED_TOGGLE[i] =0;
                        out=!out;
                }
        return out;
}
_Bool f_toggle(const int i, _Bool out)
{
        if(TEMP==1)
```

```
        {
                FED_TOGGLE[i]=1;
        }
        else if (FED_TOGGLE[i]==1)
        {
                FED_TOGGLE[i] =0;
                out=!out;
        }
                return out;
}
void adrs_re(const int i,_Bool adrs)
{
        if(TEMP && (adrs==0) )
        {
                adrs_RED[i]=1;
                TEMP=0;
        }
        if(TEMP && (adrs==0) && (adrs_RED[i]==0))
        {
                TEMP=0;
        }
        if(TEMP && (adrs==1) && (adrs_RED[i]==1))
        {
                adrs_RED[i]=0;
                TEMP=1;
        }
        else  {TEMP=0;
        }
}

void adrs_fe(const int i,_Bool adrs)
{
        if(TEMP && (adrs==1) )
        {
                adrs_FED[i]=1;
                TEMP=0;
        }
        if(TEMP && (adrs==1) && (adrs_FED[i]==0))
        {
                TEMP=0;
        }
        if(TEMP && (adrs==0) && (adrs_FED[i]==1))
        {
                adrs_FED[i]=0;
                TEMP=1;
        }
        else  {TEMP=0;
        }
```

```
}

//_____________________________________
void and (_Bool in)
{
        if (TEMP && in) TEMP=1;
        else TEMP=0;
}

void and_not (_Bool in)
{
        if (TEMP==1 && in==0) TEMP=1;
        else TEMP=0;
}

//_____________________________________
void or (_Bool in)
{
        if (TEMP || in) TEMP=1;
        else TEMP=0;
}

void xor (_Bool in)
{
        if (TEMP != in) TEMP=1;
        else TEMP=0;
}

void xor_not (_Bool in)
{
        in=!in;
        if (TEMP != in) TEMP=1;
        else TEMP=0;
}

void xnor (_Bool in)
{
        if (TEMP == in) TEMP=1;
        else TEMP=0;
}

void or_not (_Bool in)
{
        if (TEMP==1 || in==0) TEMP=1;
        else TEMP=0;
}

void not (void)
```

```
{
        if (TEMP) TEMP=0;
        else TEMP=1;
}

//_____________________________________
void nor (_Bool in)
{
        if (TEMP==0 && in==0) TEMP=1;
        else TEMP=0;
}

//_____________________________________
void nand (_Bool in)
{
        if (TEMP && in) TEMP=0;
        else TEMP=1;
}

void in_out (_Bool in)
{
        if (in) TEMP=1;
        else TEMP=0;
}



//_____________________ latch fonksiyonları
_Bool latch(_Bool D, const int i)
{
        if (TEMP) LB[i]=D;
        return LB[i];
}

//_____________________ sr flip-flop fonksiyonları
/*
_Bool sr_ff (_Bool S, _Bool R, const int i)
{
        if ((S)&& (R==0)) SR[i]=1;
        if ((R)&& (S==0)) SR[i]=0;
        return SR[i];
}
*/
//_____________________ Yükselen Kenar Algılama fonksiyonları_____________

void r_edge (const int i)
{
        if(TEMP==0)
```

```
        {
                RED[i]=1;
        }
        else if (RED[i]==1)
        {
                TEMP=1;
                RED[i] =0;
        }
        else
        {
                TEMP=0;
        }
}
//_____________________ Düşen Kenar Algılama fonksiyonları _____________________
void f_edge (const int i)
{
        if(TEMP)
        {
                FED[i]=1;
                TEMP=0;
        }
        else if (FED[i]==1)
        {
                TEMP=1;
                FED[i]=0;
        }
        else
        {
                TEMP=0;
        }
}
//_________________ Karşılaştırma fonksiyonları _____________________
void GT (unsigned int in1, unsigned int in2)
{
        if (TEMP && (in1 > in2)) TEMP=1;
        else TEMP=0;
}
void  GE (unsigned int in1, unsigned int in2)
{
        if (TEMP && (in1 >= in2)) TEMP=1;
        else TEMP=0;
}
void EQ (unsigned int in1, unsigned int in2)
{
        if (TEMP && (in1 == in2)) TEMP=1;
        else TEMP=0;
}
void LT (unsigned int in1, unsigned int in2)
```

```
{
        if (TEMP && (in1 < in2)) TEMP=1;
        else TEMP=0;
}
void LE (unsigned int in1, unsigned int in2)
{
        if (TEMP && (in1 <= in2)) TEMP=1;
        else TEMP=0;
}
void NE (unsigned int in1, unsigned int in2)
{
        if (TEMP && (in1 != in2)) TEMP=1;
        else TEMP=0;
}
//_____ Aritmetik fonksiyonları _____

// 32bit işlemler yapar

 int ADD (unsigned int in1, unsigned int in2)
{
        if (TEMP) TEMP_REG=in1+in2;
        return TEMP_REG;
}
 int SUB (unsigned int in1, unsigned int in2)
{
        if (TEMP) TEMP_REG=in1-in2;
        return TEMP_REG;
}

 int MUL (unsigned int in1, unsigned int in2)
{
        if (TEMP) TEMP_REG=in1*in2;
        return TEMP_REG;
}
 int DIV (unsigned int in1, unsigned int in2)
{
        if (TEMP) TEMP_REG=in1/in2;
        return TEMP_REG;
}
 int INC ( int in1)
{
        if (TEMP) TEMP_REG= in1 + 1;
        else TEMP_REG=in1;
        return TEMP_REG;
}
 int DEC (  int in1)
{
        if (TEMP) TEMP_REG= in1 - 1;
```

```
        else TEMP_REG=in1;
        return TEMP_REG;
}
//____________________ 32 bit 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<33)&& (TEMP)) TEMP_REG = (in << i);
        return TEMP_REG;
}
unsigned int SHIFT_R ( unsigned int in, unsigned int i)
```

```
{
        if ((i<33)&& (TEMP)) TEMP_REG = (in >> i);
        return TEMP_REG;
}
//_____ Zamanlayıcı fonksiyonları _____

void ton (_Bool re_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(re_clk){
                                TON[i]++;
                                if (TON[i]==sbt)
                                        TONQ[i]=1;
                        }
                }
        }
}
void tof (_Bool re_clk, unsigned int sbt,unsigned int i) {
        if (i<64)
        {
                if(TEMP){
                        if (TOFQ[i]==1)         TOF[i]=0;
                        else TOFQ[i]=1;
                }
                else if (TOFQ[i]==1)
                {
                        if(re_clk){
                                TOF[i]++;
                                if (TOF[i]==sbt)
                                        TOFQ[i]=0;
                        }
                }
        }
}
//_____ Sayıcı fonksiyonları _____
void CTU (_Bool CU,_Bool Reset,unsigned int PV,unsigned int i)
{
        if (i<64)
        {
                if(Reset==1)
```

```
                {
            CV[i]=0;}
                else
                 {
                        if(CU==0) {CURED[i]=1;}
                    if( (CU==1) && (CURED[i]==1) && (CV[i]< 0xFFFFFFFF))
                     { CURED[i]=0;
                       CV[i]++;
                     }
                 }

            if (CV[i]>=PV) CQ[i]=1;
            else CQ[i]=0;
        }
}
void CTD (_Bool CD,_Bool Load,unsigned int PV,unsigned int i)
{
        if (i<64)
        {
                if(Load==1)
                {
                CV[i]=PV;
                }
                else
                        {
                         if(CD==0) {CDRED[i]=1;}
                         if((CD==1) &&  (CDRED[i]==1) && (CV[i]>0))
                           {
                        CDRED[i]=0;
                        CV[i]--;
                        }
              }
   if (CV[i]==0) CQ[i]=1;
   else CQ[i]=0;
}
}
void CTUD (_Bool CU,_Bool CD,_Bool Reset,_Bool Load,unsigned int PV, unsigned
int i)
{
        if (i<64)
        {
                if(Reset==1)
                { CV[i]=0;}
                else
                {
                        if(Load==1) {CV[i]=PV;}
                   else
                   {
```

```
                    if(CU==0) {CURED[i]=1;}
                    if(CD==0) {CDRED[i]=1;}
            //          if(!(  ((CU==1)  &&  (CURED[i]=1))  &&  ((CD==1)  &&
(CDRED[i]=1))
                    if( (CU==1) && (CURED[i]==1) && (CV[i]< 0xFFFFFFFF))
                                        { CURED[i]=0;
                                          CV[i]++;
                                        }
                    if((CD==1) &&  (CDRED[i]==1) && (CV[i]>0))
                                        {
                                       CDRED[i]=0;
                                       CV[i]--;
                                       }
            }
    }
            if (CV[i]>=PV) QU[i]=1;
            else QU[i]=0;

            if (CV[i]==0) QD[i]=1;
            else QD[i]=0;
  }
}

const float alpha = 0.999;  // 0.001 ~ 0.999

void read_and_filter_input(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, int index) {
   _Bool current_state = HAL_GPIO_ReadPin(GPIOx, GPIO_Pin);
   filteredState[index] = ((1-alpha) * (float)current_state) + (alpha * filteredState[index]);

}
unsigned int get_inputs(void)
{
        read_and_filter_input(GPIOA,GPIO_IDR_IDR8,0);
        read_and_filter_input(GPIOA,GPIO_IDR_IDR9,1);
        read_and_filter_input(GPIOA,GPIO_IDR_IDR10,2);
        read_and_filter_input(GPIOA,GPIO_IDR_IDR11,3);
        read_and_filter_input(GPIOB,GPIO_IDR_IDR6,4);
        read_and_filter_input(GPIOA,GPIO_IDR_IDR15,5);
        read_and_filter_input(GPIOB,GPIO_IDR_IDR3,6);
        read_and_filter_input(GPIOB,GPIO_IDR_IDR4,7);

        for(int indx=0;indx<8;indx++){
                if(filteredState[indx]>=0.5f)bit_set(INPUT,BIT(indx));
                else bit_clear(INPUT,BIT(indx));
        }
        /*
        if ((GPIOA->IDR&GPIO_IDR_IDR8)==GPIO_IDR_IDR8) // I0.7
                bit_set(INPUT,BIT(0));
```

```
        else
                bit_clear(INPUT,BIT(0));

        if ((GPIOA->IDR&GPIO_IDR_IDR9)==GPIO_IDR_IDR9) // I0.7
                bit_set(INPUT,BIT(1));
        else
                bit_clear(INPUT,BIT(1));

        if ((GPIOA->IDR&GPIO_IDR_IDR10)==GPIO_IDR_IDR10) // I0.7
                bit_set(INPUT,BIT(2));
        else
                bit_clear(INPUT,BIT(2));

        if ((GPIOA->IDR&GPIO_IDR_IDR11)==GPIO_IDR_IDR11) // I0.7
                bit_set(INPUT,BIT(3));
        else
                bit_clear(INPUT,BIT(3));

        if ((GPIOB->IDR&GPIO_IDR_IDR6)==GPIO_IDR_IDR6) // I0.7
                bit_set(INPUT,BIT(4));
        else
                bit_clear(INPUT,BIT(4));

        if ((GPIOA->IDR&GPIO_IDR_IDR15)==GPIO_IDR_IDR15) // I0.7
                bit_set(INPUT,BIT(5));
        else
                bit_clear(INPUT,BIT(5));

        if ((GPIOB->IDR&GPIO_IDR_IDR3)==GPIO_IDR_IDR3) // I0.7
                bit_set(INPUT,BIT(6));
        else
                bit_clear(INPUT,BIT(6));

        if ((GPIOB->IDR&GPIO_IDR_IDR4)==GPIO_IDR_IDR4) // I0.7
                bit_set(INPUT,BIT(7));
        else
                bit_clear(INPUT,BIT(7));
*/
        return 0;
}


unsigned int send_outputs (void)
{
        if(Q00)GPIOB->BSRR  = (1 << 11);
        else GPIOB->BRR  = (1 << 11);
```

```
	if(Q01)GPIOB->BSRR  = (1 << 10);
	else GPIOB->BRR  = (1 << 10);

	if(Q02)GPIOB->BSRR  = (1 << 1);
	else GPIOB->BRR  = (1 << 1);

	if(Q03)GPIOB->BSRR  = (1 << 0);
	else GPIOB->BRR  = (1 << 0);

	if(Q04)GPIOA->BSRR  = (1 << 7);
	else GPIOA->BRR  = (1 << 7);

	if(Q05)GPIOA->BSRR  = (1 << 6);
	else GPIOA->BRR  = (1 << 6);

	if(Q06)GPIOA->BSRR  = (1 << 5);
	else GPIOA->BRR  = (1 << 5);

	if(Q07)GPIOA->BSRR  = (1 << 4);
	else GPIOA->BRR  = (1 << 4);

	return 0;
}

void Error_Handler(void)
{
	/* USER CODE BEGIN Error_Handler_Debug */
	/* User can add his own implementation to report the HAL error return state */
	__disable_irq();
	while (1)
	{
	}
	/* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
 * @brief  Reports the name of the source file and the source line number
 *         where the assert_param error has occurred.
 * @param  file: pointer to the source file name
 * @param  line: assert_param error line source number
 * @retval None
 */
void assert_failed(uint8_t *file, uint32_t line)
{
	/* USER CODE BEGIN 6 */

	/* User can add his own implementation to report the file name and line number,
```

```
    ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
        /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

DEFINITION BÖLÜMÜ;

```
#ifndef __DEFINITION_H__
#define __DEFINITION_H__ 1

void USART2_setup(void);
int _printf(const char *format, ...);

#define uint_8t unsigned char

unsigned int T_CNT1;
unsigned int T_CNT2;
unsigned int T_CNT3;

_Bool T_10ms;
_Bool T_100ms;
_Bool T_1s;

_Bool re_T10ms;
_Bool re_T100ms;
_Bool re_T1s;

/*
#define SET_BIT(REG, BIT)     ((REG) |= (BIT))
#define CLEAR_BIT(REG, BIT)   ((REG) &= ~(BIT))
#define READ_BIT(REG, BIT)    ((REG) & (BIT))
#define CLEAR_REG(REG)        ((REG) = (0x0))
#define WRITE_REG(REG, VAL)   ((REG) = (VAL))
#define READ_REG(REG)         ((REG))
#define  MODIFY_REG(REG,  CLEARMASK,  SETMASK)      WRITE_REG((REG),
(((READ_REG(REG)) & (~(CLEARMASK))) | (SETMASK)))
 */

_Bool RED_TOGGLE[64];
_Bool FED_TOGGLE[64];
_Bool RED[64];
_Bool FED[64];
_Bool LB[64];
//_Bool SR[64];
```

```
_Bool RED_T[3];

_Bool TEMP_TR;
_Bool adrs_RED[64];
_Bool adrs_FED[64];
#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 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

_Bool setresetTemp;

//void _ISR _T1Interrupt (void);
unsigned int get_inputs (void);
unsigned int send_outputs (void);
void ld (_Bool in);
void ld_not(_Bool in);
_Bool out_not (void);
_Bool out (void);
void not (void);
void in_out (_Bool in);
_Bool inv_out (_Bool in);
_Bool mid_out (_Bool m_out);
_Bool mid_out_not (_Bool m_out);
_Bool _set (_Bool out);
_Bool _reset (_Bool out);
_Bool SR (_Bool S, _Bool R1, _Bool out);
_Bool RS (_Bool R, _Bool S1, _Bool out);
void r_edge (const int i);
void f_edge (const int i);
_Bool r_toggle(const int i, _Bool out);
_Bool f_toggle(const int i, _Bool out);
void adrs_re(const int i,_Bool adrs);
void adrs_fe(const int i,_Bool adrs);
_Bool latch(_Bool D, const int i);

unsigned int AND ( unsigned int in1, unsigned int in2);
unsigned int OR ( unsigned int in1, unsigned int in2);
unsigned int XOR ( unsigned int in1, unsigned int in2);
unsigned int NOT ( unsigned int in);
```

```
unsigned int NAND ( unsigned int in1, unsigned int in2);
unsigned int NOR ( unsigned int in1, unsigned int in2);
unsigned int XNOR ( unsigned int in1, unsigned int in2);
unsigned int SHIFT_L ( unsigned int in, unsigned int i);
unsigned int SHIFT_R ( unsigned int in, unsigned int i);

_Bool re_RTS (_Bool RTS, const int i);

void or (_Bool in);
void or_not (_Bool in);
void and (_Bool in);
void and_not (_Bool in);
void xor (_Bool in);
void xnor (_Bool in);
void xor_not (_Bool in);
void nor (_Bool in);
void nand (_Bool in);


void ton (_Bool re_clk, unsigned int sbt, unsigned int i);
void tof (_Bool re_clk, unsigned int sbt,unsigned int i);

void CTU (_Bool CU,_Bool Reset,unsigned int PV,unsigned int i);
void CTD (_Bool CD,_Bool Load,unsigned int PV,unsigned int i);
void CTUD (_Bool CU,_Bool CD,_Bool Reset,_Bool Load,unsigned int PV, unsigned
int i);

void GT (unsigned int in1, unsigned int in2);
void GE (unsigned int in1, unsigned int in2);
void EQ (unsigned int in1, unsigned int in2);
void LT (unsigned int in1, unsigned int in2);
void LE (unsigned int in1, unsigned int in2);
void NE (unsigned int in1, unsigned int in2);

int add ( unsigned int in1, unsigned int in2);
int sub ( unsigned int in1, unsigned int in2);
int mul ( unsigned int in1, unsigned int in2);
int div ( unsigned int in1, unsigned int in2);
int inc ( int in);
int dec ( int in);


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;
        };
};
//-----------------------------------------------------------------------
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];
_Bool CQ[64];
_Bool QU[64];
_Bool QD[64];
_Bool CURED[64];
_Bool CDRED[64];
unsigned int TON[64];
_Bool TONQ[64];
_Bool TONRED[64];
unsigned int TOF[64];
_Bool TOFQ[64];
_Bool TOFRED[64];


int TIMER_Val;
#endif /* !__DEFINITION_H__ */
```

INTERRUPT BÖLÜMÜ;

```
/* USER CODE BEGIN Header */
/**
******************************************************************************
  * @file    stm32f1xx_it.c
  * @brief   Interrupt Service Routines.
******************************************************************************
  * @attention
  *
  * Copyright (c) 2023 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f1xx_it.h"
//#include "definitions.h"

extern int TIMER_Val;
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN TD */

/* USER CODE END TD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */
```

```
/* Private variables ---------------------------------------------------------*/
/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/* External variables --------------------------------------------------------*/
/* USER CODE BEGIN EV */

/* USER CODE END EV */
/******************************************************************************
******/
/*           Cortex-M3 Processor Interruption and Exception Handlers          */
/******************************************************************************
******/
/**
  * @brief This function handles Non maskable interrupt.
  */
void NMI_Handler(void)
{
  /* USER CODE BEGIN NonMaskableInt_IRQn 0 */
  /* USER CODE END NonMaskableInt_IRQn 0 */
  /* USER CODE BEGIN NonMaskableInt_IRQn 1 */
  while (1)
  {
  }
  /* USER CODE END NonMaskableInt_IRQn 1 */
}
/**
  * @brief This function handles Hard fault interrupt.
  */
void HardFault_Handler(void)
{
  /* USER CODE BEGIN HardFault_IRQn 0 */

  /* USER CODE END HardFault_IRQn 0 */
  while (1)
  {
    /* USER CODE BEGIN W1_HardFault_IRQn 0 */
```

```
    /* USER CODE END W1_HardFault_IRQn 0 */
  }
}
/**
  * @brief This function handles Memory management fault.
  */
void MemManage_Handler(void)
{
  /* USER CODE BEGIN MemoryManagement_IRQn 0 */

  /* USER CODE END MemoryManagement_IRQn 0 */
  while (1)
  {
    /* USER CODE BEGIN W1_MemoryManagement_IRQn 0 */
    /* USER CODE END W1_MemoryManagement_IRQn 0 */
  }
}
/**
  * @brief This function handles Prefetch fault, memory access fault.
  */
void BusFault_Handler(void)
{
  /* USER CODE BEGIN BusFault_IRQn 0 */

  /* USER CODE END BusFault_IRQn 0 */
  while (1)
  {
    /* USER CODE BEGIN W1_BusFault_IRQn 0 */
    /* USER CODE END W1_BusFault_IRQn 0 */
  }
}
/**
  * @brief This function handles Undefined instruction or illegal state.
  */
void UsageFault_Handler(void)
{
  /* USER CODE BEGIN UsageFault_IRQn 0 */

  /* USER CODE END UsageFault_IRQn 0 */
  while (1)
  {
    /* USER CODE BEGIN W1_UsageFault_IRQn 0 */
    /* USER CODE END W1_UsageFault_IRQn 0 */
  }
}
/**
  * @brief This function handles System service call via SWI instruction.
  */
```

```
void SVC_Handler(void)
{
  /* USER CODE BEGIN SVCall_IRQn 0 */

  /* USER CODE END SVCall_IRQn 0 */
  /* USER CODE BEGIN SVCall_IRQn 1 */

  /* USER CODE END SVCall_IRQn 1 */
}
/**
  * @brief This function handles Debug monitor.
  */
void DebugMon_Handler(void)
{
  /* USER CODE BEGIN DebugMonitor_IRQn 0 */

  /* USER CODE END DebugMonitor_IRQn 0 */
  /* USER CODE BEGIN DebugMonitor_IRQn 1 */

  /* USER CODE END DebugMonitor_IRQn 1 */
}
/**
  * @brief This function handles Pendable request for system service.
  */
void PendSV_Handler(void)
{
  /* USER CODE BEGIN PendSV_IRQn 0 */

  /* USER CODE END PendSV_IRQn 0 */
  /* USER CODE BEGIN PendSV_IRQn 1 */

  /* USER CODE END PendSV_IRQn 1 */
}
/**
  * @brief This function handles System tick timer.
  */
void SysTick_Handler(void)
{
  /* USER CODE BEGIN SysTick_IRQn 0 */

  /* USER CODE END SysTick_IRQn 0 */
  HAL_IncTick();
  /* USER CODE BEGIN SysTick_IRQn 1 */

  /* USER CODE END SysTick_IRQn 1 */
}
extern unsigned int T_CNT1;
extern unsigned int T_CNT2;
```

```
extern unsigned int T_CNT3;

extern _Bool T_10ms;
extern _Bool T_100ms;
extern _Bool T_1s;

void TIM1_UP_IRQHandler(void)
{
	if (TIM1->SR&TIM_SR_UIF)
	{
		TIM1->SR &= ~TIM_SR_UIF;
		T_CNT1 = T_CNT1+1;
		T_CNT2 = T_CNT2+1;
		T_CNT3 = T_CNT3+1;

		if(T_CNT1>=5){
			T_10ms=!T_10ms;
			T_CNT1 =0;
		}

		if(T_CNT2>=50){


			T_100ms=!T_100ms;
			T_CNT2 =0;
		}

		if(T_CNT3>=500){
			HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_13);
			T_1s=!T_1s;
			T_CNT3 =0;
		}

		TIMER_Val++;
		if(TIMER_Val>65534)TIMER_Val=0;
	}
	TIM1->SR = 0;
}
/******************************************************************************/
/* STM32F1xx Peripheral Interrupt Handlers                                    */
/* Add here the Interrupt Handlers for the used peripherals.                  */
/* For the available peripheral interrupt handler names,                      */
/* please refer to the startup file (startup_stm32f1xx.s).                    */
/******************************************************************************/
/* USER CODE BEGIN 1 */

/* USER CODE END 1 */
```

T.C.
YOZGAT BOZOK ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ