SOĞUK ZİNCİR LOJİSTİĞİ İÇİN USB ARAYÜZLÜ SICAKLIK VERİ KAYIT CİHAZI TASARIMI VE UYGULAMASI

SEMİH GÜZEL

YÜKSEK LİSANS TEZİ

MEKATRONİK MÜHENDİSLİĞİ ANA BİLİM DALI
Dr. Öğr. Üyesi Levent GÖKREM

Nisan - 2018

T.C.
GAZİOSMANPAŞA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
MEKATRONİK MÜHENDİSLİĞİ ANA BİLİM DALI

YÜKSEK LİSANS TEZİ

SOĞUK ZİNCİR LOJİSTİĞİ İÇİN USB ARAYÜZLÜ SICAKLIK VERİ KAYIT CİHAZI TASARIMI VE UYGULAMASI

SEMİH GÜZEL

TOKAT
Nisan - 2018

Semih GÜZEL tarafından hazırlanan "**Soğuk Zincir Lojistiği İçin USB Arayüzlü Sıcaklık Veri Kayıt Cihazı Tasarımı Ve Uygulaması**" adlı tez çalışmasının savunma sınavı 20 NİSAN 2018 tarihinde yapılmış olup aşağıda verilen Jüri tarafından Oy Birliği ile Gaziosmanpaşa Üniversitesi Fen Bilimleri Enstitüsü MEKATRONİK MÜHENDİSLİĞİ ANA BİLİM DALI 'nda YÜKSEK LİSANS TEZİ olarak kabul edilmiştir.

Jüri Üyeleri	İmza
Danışman Dr. Öğr. Üyesi Levent GÖKREM Gaziosmanpaşa Üniversitesi	.....
Üye Dr. Öğr. Üyesi Mehmet Serhat CAN Gaziosmanpaşa Üniversitesi	.....
Üye Dr. Öğr. Üyesi Uğur ERKAN Karamanoğlu Mehmetbey Üniversitesi	.....

ONAY

Prof. Dr. Ebubekir ALTUNTAŞ
Fen Bilimleri Enstitüsü Müdürü

24/04/2018

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 ederim.

SEMİH GÜZEL

24 Nisan 2018

ÖZET

YÜKSEK LİSANS TEZİ

SOĞUK ZİNCİR LOJİSTİĞİ İÇİN USB ARAYÜZLÜ SICAKLIK VERİ KAYIT CİHAZI TASARIMI VE UYGULAMASI

SEMİH GÜZEL

GAZİOSMANPAŞA ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

MEKATRONİK MÜHENDİSLİĞİ ANA BİLİM DALI

(TEZ DANIŞMANI: DR. ÖĞR. ÜYESİ LEVENT GÖKREM)

Günümüzde ortam sıcaklığı ve nem değerleri birçok sektör için önem arz etmektedir. Bu sektörlerden birisi olan soğuk zincir lojistiğinde, taşıma işlemi yapılan ürünlerin cinsine ve kimyasal yapısına göre sıcaklık ve nem değerlerinin belirli bir aralıkta olması oldukça önemlidir. Sıcaklık ve nem değerlerinin belirli bir değerin altında veya üzerinde olması ürünlerde taşıma esnasında istenmeyen zararlı maddelerin üremesine ve istenmeyen sonuçların oluşmasına sebep olmaktadır. Bu tez çalışmasında soğuk zincir lojistiğinde istenmeyen bu durumların belgelenmesi için USB arayüzlü sıcaklık veri kayıt cihazı tasarımı gerçekleştirilmiştir. Çalışmada baskılı devre kartı Altium Designer programında tasarlanmıştır. Mikrodenetleyici seçiminde enerjinin olabildiğince az tüketilmesi, cihaz çalışmadığında uyku modunda olması durumları göz önüne alınmıştır. Enerjiyi en etkin kullanan STM32 mikrodenetleyicisi seçilmiştir. Kodlar Keil Microvision da derlenmiş ve baskılı devre kartına yüklenmiştir. İstenilen koşulların cihaza tanıtılabilmesi için Qt de arayüz tasarlanmıştır. Bu arayüz sayesinde cihaza veri alma sıklığı, üst-alt limit değerleri, saat ve tarih bilgileri girilerek gerçek zamanlı olarak çalışması sağlanmıştır. Tasarımı yapılan cihazın -30 ^{c}C ile 70 ^{c}C arasında sıcaklık değerlerini kayıt altına aldığı yapılan testlerde görülmüştür. Kayıt altına alınan bu verilerin aynı arayüz sayesinde bilgisayara Excel formatında aktarılması sağlanmıştır.

2018, 158 SAYFA

ANAHTAR KELİMELER: Sıcaklık Kayıt Cihazı, Veri Kaydedici, Sayısal Sıcaklık İzleme

ABSTRACT

MASTER THESIS

USB INTERFACED TEMPERATURE DATA LOGGER FOR COLD CHAIN LOGISTICS DESIGN AND IMPLEMENTATION

SEMİH GÜZEL

GAZIOSMANPASA UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

DEPARTMENT OF MECHATRONICS ENGINEERING

SUPERVISOR: ASST. PROF. DR. LEVENT GÖKREM

Nowadays, ambient temperature and potential moisture have importance for several sectors and cold chain logistics are one of them. In cold chain logistics, it is really significant to keep the values in a certain range in accordance with the type and chemical structure of the product. Temperature, which is either higher or lower than a certain value, causes the growth of harmful substences and unwanted results. In this thesis, temperature logger with USB interface design has been carried out, so that those unwanted results can be documented. In this study, printed circuit board has been designed via Altium Designer program. While choosing the microcontroller, conditions like its using as little energy as possible and activating sleep mode when it is not used have been taken into consideration. Thus, STM32 microcontroller which uses energy most effectively has been chosen. Codes are compiled in Keil Microvision and installed on microcontroller. An interface has been designed in Qt with the aim of introducing designed conditions to the device. By means of this interface, the device has gained the capacity to function in real time by entering frequency of data retrieual, upper – lower limit values, date and time. It has been seen in the tests that designed device has recorded the temperature values between -30 ^{c}C and 70 ^{c}C degrees. Then, those records has been transferred to the computer in the form of Excel sheet.

2018, 158 PAGE

KEYWORDS: Temperature Logger Device, Data Logger, Numerical Temperature Tracking.

ÖNSÖZ

Yüksek lisans çalışmamda destek ve emeğini esirgemeyen değerli hocam Dr. Öğr. Üyesi Levent GÖKREM'e, tez konu seçimimde yardımcı olan Arş. Gör. Mahmut DURGUN'a (Karamanoğlu Mehmetbey Üniversitesi) sonsuz teşekkür ederim.

Bu süreçte yanımda olan tüm arkadaşlarıma, bu günlere gelmemde emeklerini esirgemeyen Ailem'e çok teşekkür ederim.

SEMİH GÜZEL

24 Nisan 2018

İÇİNDEKİLER

Sayfa

SİMGELER VE KISALTMALAR

Simgeler	Açıklama
V	Volt
A	Amper
Ω	Ohm
°C	Santigrat
µA	Mikroamper
µs	Mikrosaniye
mV	Milivolt
µF	Mikrofarad
nF	Nanofarad

Kısaltmalar	Açıklama
ARM	Acord Rısc Machine
RICS	Reduced Instruction Set Computer
CPU	Central Process Unit
CISC	Complex Instruction Set Computer
DMA	Direct Memory Access
RAM	Rondom Access Memory
DSP	Dijital Signal Process
ASSP	Application-Specific Standart Parts
SoC	System On Chip
PC	Personel Computer
GPS	Global Positioning System
EEPROM	Electronically Erasable Programmable Read-Only Memory
LPTIM	Low Power Timer
UART	Universal Asynchronous Receiver Transmitter
PCLK	Peripheral Clock
VLCD	Very Low Calorie Diet
HCLK	Contains The Core Clock

APBx	Advanced Peripheral Bus
AHBx	Advanced High Speed Bus
BCD	Binary-Coded Decimal
PWM	Sinyal Genişlik Modülasyonu
SCL	Serial Clock
SDA	Serial Data
LCD	Liquid-Crystal Display
IRQ	Interrupt ReQuest
HCD	Host Controller Driver
BOR	Brownout Reset
PLL	Phase Locked Loop
PCB	Printed Circuit Board
MPU	Bellek Koruma Birimi
MCU	Micro Controller Unit
EPD	Electronic Paper Display
LED	Light Emitting Diode
OLED	Organic Light Emitting Diode
SPI	Serial Protokol Arayüzü
I2C	Inter Integrated Circuit
RTC	Real Time Clock
USB	Universal Serial Bus
GPIO	General Purpose Input Output
ADC	Analog to Digital Converter
DAC	Digital to Analog Converter
USART	Universal Syncronous/Asyncronous Transmitter/Receiver
FLASH	Non Volatile Storage
RAM	Random Access Memory
ACK	Acknowledge
CPOL	Chip Polarisation
CPHA	Chip Phase

ŞEKİL LİSTESİ

Şekil **Sayfa**

ÇİZELGE LİSTESİ

Çizelge **Sayfa**

1. GİRİŞ

Günümüz şartlarında bozulabilir gıda ürünlerinin tüketimi soğuk zincir uygulamaları sayesinde güvenilir hale gelmektedir. Soğuk zincir teknolojisi ile bozulabilir gıda ürünlerinin ülkeler veya kıtalar arası taşımacılıktan tüketiciye teslimatına kadar kaliteli ve güvenli bir şekilde sevkiyatı sağlanmalıdır. Bu sayede gıda işleme ve paketlemelerdeki ilerleme ile gıda güvenliği uygulamalardaki artan hassasiyet soğuk zincir uygulamalarının gelişmesinde önemli bir etken haline gelmiştir. Bu sebeple gıda ürünleri için soğuk zincir uygulamaları gün geçtikçe daha da önemli bir durum haline gelmektedir. Ürünlerin orjini tüketiciye ulaştırılmak üzere bulunduğu noktadan uzaklaştırılmakta, aktarım işlemi fazlalaşmakta, dolayısıyla taşıma süreleri artmaktadır. Bu bağlamda gıda endüstrisi liderleri tarafından kaliteli ve güvenilir gıda açısından endişeleri artmıştır. Bu durum gıda endüstrisi liderleri ve üreticilerini, ürünün kaynağından tüketiciye gelene dek işleme, depolama aşamalarına kadar soğuk zincirin her aşamasında daha fazla bilgi ve kontrol sağlayabilme süreçlerine yönlendirir (Sarısoy, 2011).

Soğuk zincir uygulamalarının ekonomik boyutundaki artış, sağlık ve gıda gibi iki kritik sektörü ilgilendirmesinden dolayı son zamanlarda araştırmacılardan önemli bir ilgi görmektedir. Shuchang, Yufan ve Radzion (2015) soğuk zincir lojistiğinin zayıf noktaları üzerine bir araştırma yapmışlardır ve zayıf noktalardan biri olarak aktarma sayısı ve aktarma sırasında oluşan sıcak değişimlerinin etkilediğini görmüşlerdir (İzer, 2017).

Goedhals-Gerber, Haasbroek, Freiboth ve Van Dyk (2015), Cape Town limanında meyvelerin kalitesi ve bozulması üzerine bir araştırma yapmışlardır. Bu araştırmada ihracat esnasında yaşanan sorunları incelemiş ve soğuk hava depolu (frigo) gemilerinin konteynerleri kamyon üzerinde dolumu sırasında, gemiye taşınıp güç kaynağına bağlanana kadar geçen süredeki ısı değişiminden kaynaklanan kalitedeki bozulmalar tespit etmişlerdir. Ayrıca yine bu araştırma sonucunda konteynerler içerisinde bulunan soğutma sistemlerinin belirli bir sıcaklığı korumaya yeterli olduğunu ancak gerekli soğutmayı yapamadıklarını tespit etmişlerdir. Bu yüzden meyvelerin yeterli soğuklukta yüklenmesini tavsiye etmişlerdir. İlk soğutmanın ve burada elde edilen ısı derecesinin bütün taşıma esnasındaki önemi bu araştırmada ifade

edilmektedir. Rong, Akkerman, Grunow (2009) yapmış oldukları araştırmalarında raf ömrü ve ürün kalitesi için soğuk zincir uygulamalarında sıcaklığın ana belirleyici etken olduğuna değinmişlerdir. Aung ve Chang, (2013) yılında yapmış oldukları araştırmada soğuk zincir uygulamalarında raf ömrünün azalmasına ve kalitenin bozulmasına yol açan mikrobiyolojik olayların artışında sıcaklığın önemli bir etken olduğu vurgulanmıştır. Rediers, Claes, Peeters ve Willems (2008) yapmış oldukları araştırmalarında soğuk zincirde karşılaşılan sıcaklık dalgalanmaları sonucu koliform bakterilerinde artış olduğunu tespit etmişlerdir. Abad, Palacio, Nuin, Zarate, Juarros, Gomez ve Marco, (2009) yapmış oldukları araştırmalarında Copetown'da yapılan uçak yüklemeleri sırasında sıcaklık ve nem dalgalanmaları olduğunu görmüş ancak buna karşın saklama esnasında sabit olduğunu görmüşlerdir. Aynı araştırmada yükleme sırasında taze balıkların saklama sıcaklığında 5 derecelik bir değişim fark edilmiştir (İzer, 2017).

Gıda güvenliği için farklı ülkelerde bazı düzenlemeler yürürlüğe konulmuştur. Buna göre gıda güvenliği uyumluluğu kalitesinden üreticiler, işleyiciler ve perakendeciler sorumlu tutulmakla birlikte gıda güvenliği kontrolü sağlanmasına dikkat çekmekte ve soğuk zincir boyunca izlenebilirlik ve takip ile ilgili yükümlülükler getirmektedir (Sarısoy, 2011).

İzlenebilirlikle ilgili farklı tanımlar yapılmıştır. Bunun en yaygın olarak kullanılanı Avrupa Birliği Gıda Kondeksi'nin yapmış olduğu tanımdır. Buna göre izlenebilirlik; üretim, işleme ve dağıtım boyunca bütün aşamalarda gıda ürününün takip edilmesi yeteneğidir (Sarısoy, 2011).

Yaş sebze, meyve, süt, süt ürünleri, et ve et ürünleri gibi belirli bir sıcaklık altında kalması gereken ürünlerin taşıma işlemlerinde ürünün kalitesi, sıcaklık değerlerine bağlı olarak önem arz etmektedir. İstenilen sıcaklık değerleri aralığında olmayan ürünlerde bozulmalar ve çürümeler gerçekleşmektedir. Bunun sonucu olarak ürünü satın alan firmalar, ürün taşıma halinden kendilerine teslimatına kadar sıcaklıktaki değişimini kontrol etmek istemektedirler.

Sıcaklık veri kayıt cihazları, kontrol edilmek istenen sıcaklık değerlerini istenilen aralıklarda kayıt altına alan ve bunu da hafızasına kaydederek istemiş

olduğumuz sıcaklık değişim raporunu veren cihazlardır. Bu cihazların maliyet bakımından ucuz, kullanım bakımından rahat olması önemlidir.

Bugün dünya pazarında başarılı olabilmek için taşımacılık büyük önem arz etmektedir. Yaş meyve ve sebze, bahçe ürünleri, deniz yolu, demir yolu, hava yolu ile lojistik olarak taşınmaktadır. Burada önemli bir nokta olan ürün gruplarına göre belirli bir sıcaklık ve nem değerleri altında ürünün varış noktasına ulaşmasıdır (Türk ve Ünverdi, 2015).

Özerden, 2015 yılında yapmış olduğu bir sunumda, meyvelerin soğuk işlem gördükten sonra taşıma işleminin yapılabilmesi için sıcaklığın ölçülmesi gerektiğini ve bu ölçülen değerlerin kayıt altına alınarak Excel ortamında bilgisayara indirmesinin öneminden bahsetmiştir (Özerden, 2015).

Hızlı dondurulmuş gıda ürünlerinin depolanması, muhafazası ve taşınması esnasında ortam sıcaklığı, uygun ölçümlerin yapılması ve kayıt cihazları kullanılarak düzenli ve sık aralıklarla kaydedilmesi gerekmektedir (Anonim, 2014).

Soğuk zincir sevkiyatının en zor noktalarından birisi olan lojistik zincir takibinin istenilen değerleri sağlamasının belgelenmesi sıcaklık kaydedicilerle sağlanmaktadır. Ürünlerin belirli bir değerin altında veya üzerinde olması istenmeyen zararlı maddelerin üremesine yol açmaktadır. Bu da ürüne doğrudan zarar vermektedir. Bu sebeple belirli bir sıcaklık altında taşınması gereken ilaçların tüm sevkiyat sırasında istenilen değerlerde taşınmasının belgelenmesi için sıcaklık kayıt cihazları kullanılmalıdır (Anonim, 2016).

Aşının üretiminden başlanarak uygulanan kişiye kadar bir çok adımı vardır. Aşı teslim alınırken, soğuk ortamlarda depolanırken, bir yerden başka bir yere taşınırken, yani soğuk zincirin bütün aşamalarında sıcaklığın doğru tutulması gerekmektedir. Aşıların üretim merkezlerinden sahaya kadar soğuk zincirin kırılmadan gönderilmeleri bağışıklık vermeleri için son derece önemli bir noktadır. Bu hususlara uyulmadığı durumda aşıların antijenik gücünde azalma meydana gelir veya hiç kalmaz (Anonim, 2017).

Ürünlerin güvenli bir biçimde muhafaza edildiğini anlayabilmek için çeşitli izleme yolları ile ortamın sıcaklık ve nemlerin takip edilmesi gerekmektedir. İzlenen bu verilere göre ürüne müdahale yapılarak doğru bir şekilde muhafazası sağlanmalıdır (Anonim, 2016).

Et, balık, süt, meyve veya aşı gibi kalitesi yüksek olan ürünlerin taşıma ve depolama işlemlerinde sıcaklık ve nem gibi değerlerini sensörler ile ölçen ve hafızasına kayıt eden cihazların olması ürünlerin izlenebilirliğini sağlamaktadır ve ürünlerin fiziksel konumunun saptanmasında kolaylık sağlamaktadır (Aksu, 2010).

Aksoy (2012) de yaptığı tez çalışmasında soğuk zincir içerisinde bulunan ilaçların ısı sensörleri RFID (Radyo Frekansı Tanımlayıcı) etiket taşınmasını incelemiştir. Bu sistem ile soğuk zincir içerisinde bulunan ilaçların ısı sensörlü RFID etiketler ile soğuk zincir aşamalarındaki ısı değerleri etiket üzerinde bulunan mikroçipe kayıt işlemi gerçekleştirilmesinin sağlanması ve bu değerlerin belirli aralıklarda okunmasını sağlayan sistemdir. Bu sistemde kullanılan teknoloji ürünün üretiminden tüketimine kadar gerçekleşen depolama ve dağıtım işlemlerinde maruz kalınan ısının izlenmesini sağlayan, ambalaj içerisinde veya dışarısında kullanılan göstergelerdir. Burada ilaçların maruz kaldığı ısı değerleri saptanmış, istenilen ısılarda tutulup tutulmadığı, istenilen değerlerin dışına çıkmışsa hangi sürelerde çıktığının tespiti yapılmış ve bu yönde süre iyileştirmelerinin yapılabilmesi gibi faydalarda sağlanmıştır. Bu sistem soğuk zincir gerektiren aşı, serum ve biyoteknolojik ürünlerde kullanılabilen bir sistemdir (Aksoy, 2012).

Gıda veya ilaç taşımacılığında uygulanan soğuk zincir uygulamalarında ürünleri istenilen sıcaklık altında tutmak yeterli gelmemekle birlikte sıcaklıklarda yaşanan artış ve azalışların istenilen sınır aralıkları içerisinde kalması veya en aza indirilmesini sağlamak ürün kalitesi için oldukça önemlidir. Meyve hasatı sonrasında soğuk zincirlerin kırılması sonucu dünya genelinde %10 - %20 arası bir kayıp yaşanmaktadır. Yaşanan sıcaklık değişmelerinden dolayı ürünlerin kalitelerinde düşüş, raf ömürlerinde azalma görülmektedir. Ürünlerin özellikle 0 ^{c}C ile +1 ^{c}C derece aralığında belirlenen sıcaklık sınır değerlerinin bu aralığı birkaç derece dahi geçmeleri halinde mikrobiyolojik üremenin başlamasına neden olmaktadır. Bu sebeple ürünlerde bozulmalar oluşmakta, gıda zehirlenme riski artmaktadır. Gıda ürünlerindeki kalite

kaybını ve güvenliğini saklama sıcaklığı üzerinde geçirilen zaman belirlemektedir. Hasat veya üretim sonrası ürün kalitesi ve güvenliği soğuk zincir uygulamalarında ideal sıcaklık koşullarında saklanması ile mümkündür. Bakteri üretimi ve kalite bozulmasını engellemek için sıcaklık kontrolü gerekmektedir. Soğuk zincir uygulamalarında sıcaklık artışlarındaki en büyük risk ürünlerin soğuk hava deposundan araçlara yüklenmesi veya soğuk hava deposundan gemilere ve uçaklara yüklenmesi esnasında yaşanmaktadır. Oluşan bu sorunlar için son yıllarda üreticiler ve nakliyeciler tarafından çalışmalar yapılmaktadır (İzer, 2017).

Soğuk zincir uygulanmış ürünlerde kalite değişikliğinin olmaması için tüm dağıtım kanallarına aynı işlem uygulanmalıdır. Ürünlerin nakliyesi esnasında araçların da soğuk zincir ısısını koruyabilecek özellikte olması ürünün kaliteli bir şekilde üretilmesi kadar önemlidir. Ayrıca, satıcıya ulaşan ürünlerin doğru alanlarda muhafazası ve sergilenmesi tüketiciye sağlıklı bir şekilde ulaşması için önemlidir (Durak ve ark, 2014).

Soğuk zincir boyunca ürünün ısı takibinin sorunsuz bir şekilde yapılması, son zamanlarda gıda sektörü ile ilgili devlet kurumlarının, gıda üreticilerinin, lojistik şirketlerin, perakendecilerin ve son adres olan müşterin ilgi alanı olmuştur (Alanur, 2014).

Gıda maddelerinin soğuk zincirde taşınmasının birçok sebebi vardır. Belirli bir sıcaklık altında olan ürünün kalitesi üst seviyelerde olurken, bazı gıda maddelerinin sıcaklıklarında kısa süreli değişimi olumsuz yönde etkilemektedir. Buna sebep olarak ise gıda kaynaklı hastalıklar meydana gelmektedir. Ürünlerin belirli bir seviyenin üzerinde sıcağa veya soğuğa maruz kalmaları ürünlerin kimyasında ciddi değişikliklere sebep olmaktadır. Ürünün kimyasındaki değişiklik gıda sektöründe ürünün ömrünün değişmesine ilaç sektöründe ise ürünün etkisinin yitirilmesine sebep olmaktadır. Bu nedenlerle ürünlerin taşınmasının ve dağıtımının güvenli bir şekilde yapıldığının güvencesini müşterilere vermenin yolu sıcaklığı doğrulamaktır. Ürünün taşınmasında görev alan kişi tüm veri kayıtlarını teslim etmek, teslim alan kişi ise istenilen sıcaklık değerinde taşıma işleminin gerçekleştirilmesini, sıcaklık değeri aşılmış ise ne kadar süre de aşıldığını kontrol etmek ve paylaşmak zorundadır (Atlatırlar, 2012).

1.1. Veri Kayıt Cihazları

Veri kaydediciler, veri toplama işlemi yapan elektronik ölçüm cihazlarıdır. Sıcaklık, nem, basınç, ışık yoğunluğu gibi sensörlerden gelen veri bilgilerini gerçek zamana bağlı kalarak kayıt altına alırlar (Çiçekdeş, 2011).

Genel olarak veri kaydediciler taşınabilir küçüklükte ve pille çalışırlar. Giriş kanalları mikro denetleyiciye bağlı olarak bir veya daha fazla iletişim protokolüne ve veri depolama özelliğine sahiptirler. Kayıt altına alınan veriler gerçek zaman saatine (RTC) bağlı kalınarak kayıt edilir. Örnekleme sıklıkları genel olarak düşüktür. Çünkü nemlilik gibi yavaş değişen çevre parametrelerini ölçmek, sıcaklık, basınç v.b. düşük veri alma frekansına sahip olunabilmesi, depolama ünitesinin verimli bir şekilde kullanılması için önemlidir.

Veri kaydediciler çoğunlukla pil ile çalışırlar, bu nedenle daha uzun süre kullanımlar için güç tüketimleri olabildiğince optimize edilmelidir. Daha az güç tüketimi için genellikle uyku modunda çalıştırılırlar. Bir sonraki veri alınana kadar uyku modunda kalırlar (Çiçekdeş, 2011).

Veri kayıt cihazlarının çoğunda, verilerin istenilen aralıkta kaydedilmesi ve depolanan verilerin görüntülenebilmesi için bir bilgisayar yazılımı gerektirir. Özellikle EEPROM veya dahili flash bellekleri kullananlar depolanmış verileri aktarabilmek için bir iletişim protokolünü desteklemek zorundadır. Genellikle yaygın olduğu için RS-232 standardı kullanılır (Çiçekdeş, 2011).

1.2. Veri Kayıt Cihazları Kullanım Yerleri

Bilgiler önceden ayarlanmış zaman aralıklarına göre kayıt altına alan cihazlara veri kaydedici (datalogger) denilmektedir. Diğer bir isimleri veri kayıt edicilerdir. Bu cihazlar genellikle saha çalışmalarında, kalite çalışmalarında, nakliye esnasında izlemede, genel araştırma ve eğitim bilimlerinde kullanılırlar. Geniş kullanım alanlarına sahiplerdir. Sıcaklık, nem, basınç gibi değerler istenilen aralıkta dataloggerlar ile kayıt altına alınabilmektedir. Sensör teknolojisinin gelişmesiyle günümüzde ölçülebilen değerler kayıt altına alınabilmektedir (Anonim, 2017).

Farklı lojistik alanlarda sıcaklıkların doğru bir şekilde takibi için sıcaklık veri kayıt cihazlarının çeşitleri vardır. Bunun bir örneği Şekil 1.1'de görülmektedir. Bunlara bakıldığında taşınabilir, sabit ve tek kullanımlık veri kayıt cihazlarıdır. Soğuk hava depolarındaki sıcaklık takibi için sabit kayıt cihazları kullanılırken soğuk zincir taşımacılığında tek kullanımlık kayıt cihazları tercih edilmektedir (Anonim, 2017).

Şekil 1.1. Sıcaklık veri kayıt cihazı

2. TEKNOLOJİK ALT YAPI

Bu bölüm soğuk zincir lojistiğinde veri kayıt cihazları ile ilgili sistem bileşenlerine alt yapı sağlamayı amaçlamaktadır.

2.1. ARM Nedir?

ARM, 1983 yılında Acorn Computer Ltd. şirketi tarafından geliştirme projesi olarak başlatılmıştır. İlk olarak MOS Technology 6502, Rogert Wilson ve arkadaşları tarafından geliştirilmiştir (Türk, 2011).

ARM mimarisi, gömülü tasarımların birçoğunda kullanılmakta olan 32 bit azaltılmış komut seti kütüphanesi (RICS) işlemci mimarisidir. ARM işlemciler düşük güç tüketiminin olduğu birçok uygulama da tercih edilen merkezi işlem birimidir (CPU) (Türk, 2011).

Dünya da 32 bit gömülü işlemcilerin %75'ine yakınını ARM işlemci ailesi oluşturmaktadır. Bu işlemci ailesi genellikle taşınabilir cihazların (hesap makinası, cep telefonu, oyun konsolu) bilgisayar parçaları da dahil olmak üzere tüketici elektroniklerinin bir çoğunda yoğun bir şekilde kullanılmaktadır (Türk, 2011).

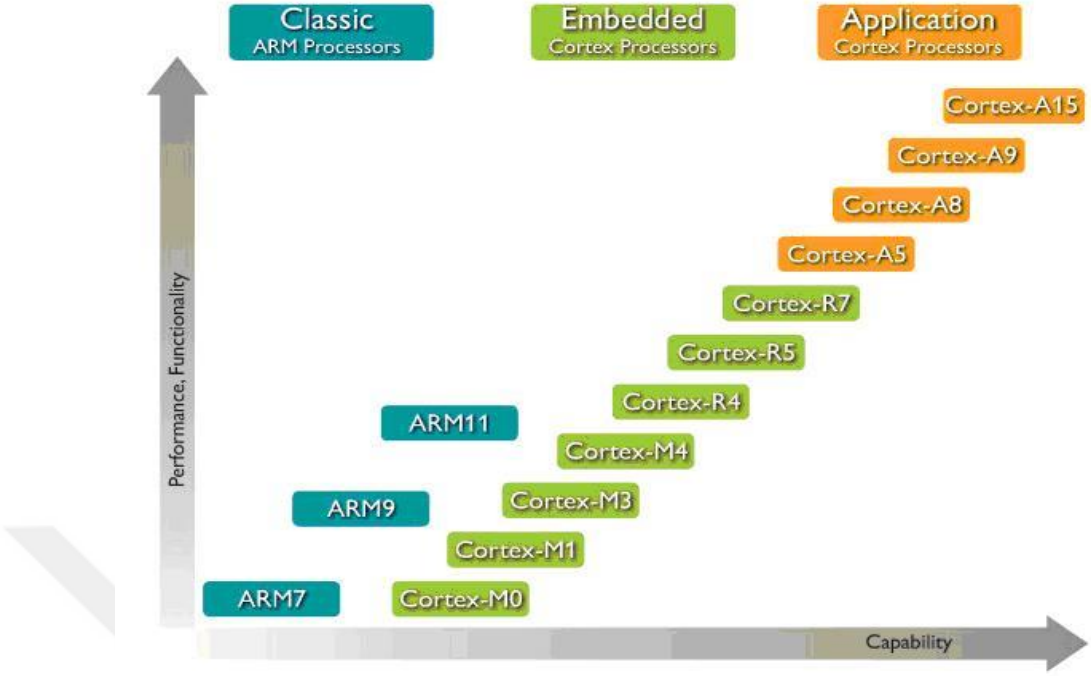
2.2. ARM Mimarisi

Komut setleri kısa olan ARM mimarisi, RICS tabanlı gelişmiş mikrokontrol mimarisidir, işlemci değildir. Kısacası işlemci, chip üreten firmalar için tasarlanan işlemci mimarisi, ARM firmasının geliştirmiş olduğu bir mimaridir (Göl, 2015).

ARM işlemcilerin, 32 ve 64 bit versiyonları vardır, genellikle güç tüketimleri düşüktür. Diğer RICS tabanlı işlemcilere göre performansları yüksektir, x86-x64 işlemcilere göre daha hesaplı olmaları nedeniyle gömülü sistemlerde, taşınabilir cihazlarda genelde ARM işlemciler tercih edilmektedir. ARM firmasında işlemci üretimi yapılmaz, tasarlanır ve lisansı satılır. Aynı jenerasyondaki işlemci karmaşık komut seti kütüphanesi (CISC) tabanlı işlemcilere göre özelleştirilmeleri sebebiyle farklı üreticilerden değiştirilmiş olarak çıkabilir, bu nedenle performans farklılıkları görülebilmektedir. Bu yüzdendir ki ARM referans tasarımı, işlemcilerin jenerasyon ve karakteristikleri incelenirken ele alınır (Cebeci, 2014).

ARM mimari ailesine bakıldığında, gömülü sistem ARM işlemcileri (cortex m serisi), klasik ARM işlemciler (ARM7, ARM9, ARM11), uygulama ARM işlemcileri (cortex a serisi) olmak üzere 3 temel gruptan oluşmaktadır. STM32L0, ARM Cortex M0 mimarisine sahiptir. Bu seri işlemcilerdeki saat darbe (clock) kaynağına bakıldığında kristal osilatör, dahili RC osilatör veya harici bir kaynak kullanılabilmektedir. Çevre birimlerinin clock sinyallerinin tamamen kapatılmasıyla güç tüketiminin kontrol edilebilmesi sağlanmaktadır. Bu işlem ile güç tüketimi oldukça azaltılabilmektedir. Doğrudan veri erişimi (DMA) donanımı sayesinde bu serideki işlemciler sıralı veya tek adresten okunan verileri, bir başka sıralı veya tek adrese CPU gücü kullanılmadan taşınabilmektedir. Bu donanım verileri CPU'dan memory-peripheral, peripheral-memory, memory-memory, peripheral-peripheral yolları sayesinde kaynak adresten hedef adrese taşınmaktadır. CPU üzerine düşen iş yükünde de azalma olmaktadır (Göl, 2015).

Rogert Wilson ve arkadaşları tarafından oluşan takım, 1985 yılında ARM1'i geliştirdi, sonraki yıllarda gerçek ürün olan ARM2'yi geliştirdiler. ARM2'ye bakıldığında en önemli özellikleri 16 adet 32 bitlik yazmaç, 26 bit adres boşluğu, 64 MB adres alanı ve 32 bit veri yolu sağlamasıdır. Yazmaçlardan bir tanesi program sayacı olarak kullanılabilmektedir. Bu sayaca bakıldığında en az 2 biti, en fazla 6 biti durum göstergesini tutar. ARM2 işlemcisi, en basit 32 bit mikroişlemci olması 30.000 transistörü olmasına bağlıdır. Bu basitlik günümüzde de olduğu gibi önbellek içermesinden ve mikrokod bulundurmamasındandır. Bu basitlik İntel 80286 işlemcisine göre daha az güç tüketimi sağlamanın yanında daha da iyi performans göstermektedir. ARM3' de daha fazla performans artışı sağlayabilmek için 4 KB önbellek ile geliştirildi. 1991 yılında ARM6 piyasaya sürüldü. Apple kendi Apple Newton PDA'larında ARM6'yı temel olarak kullandı. Acorn, 1994 yılında kendi PC'lerinde ana CPU olarak ARM6'yı kullanmıştır (Cebeci, 2014). Günümüze bakıldığında ise Şekil 2.1'de görülen mimari kullanılmaktadır.



Şekil 2.1. Arm mimarisi

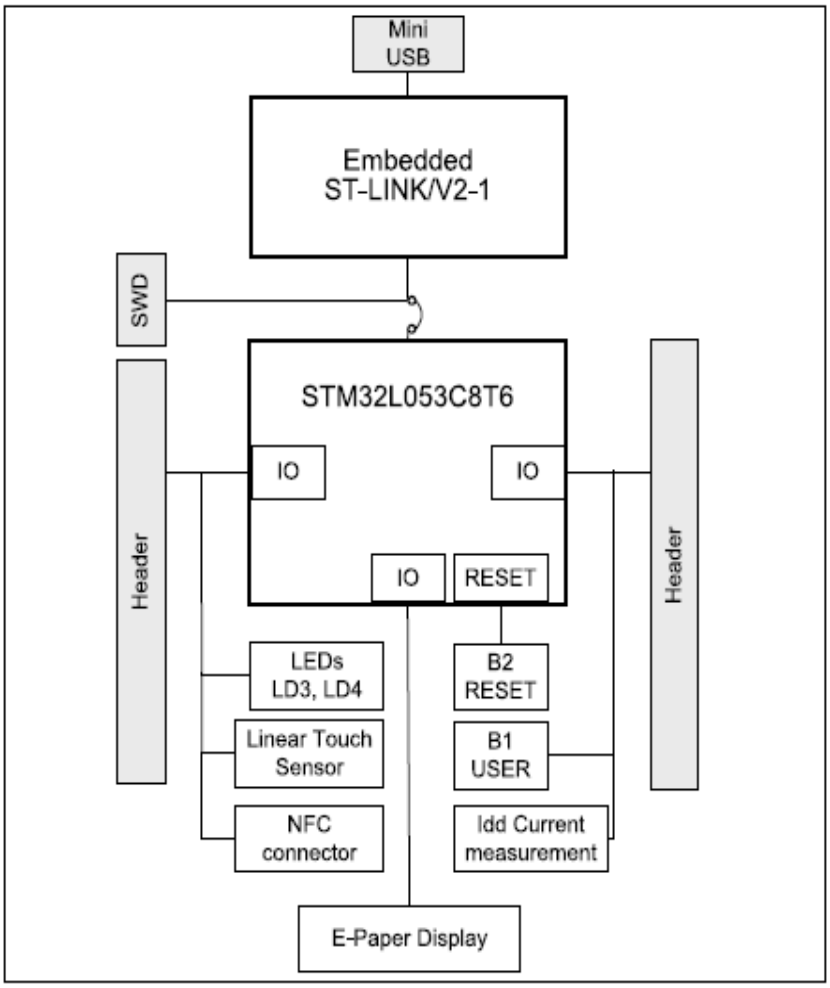
2.3. ARM Kullanım Sebepleri

ARM işlemcilerin diğer işlemcilere göre avantajları aşağıdaki gibi sıralanabilir:

- Güç tüketimi: Bu işlemcilerde güç tüketimi çok azdır. Bunun nedenine bakıldığında tasarımında transistör sayısı düşük tutulmaktadır. Bu sebeple genelde mobil cihazlarda ARM işlemcisinin kullanıldığı görülmektedir.
- İşletim modeli: İntel işlemcilere bakıldığında karar vermek için bir karşılaştırma işlemi yaptırılır, bu karşılaştırma işleminin sonucuna göre programın başka bir kısmına geçilir. Bu geçiş işlemi kafa karıştırmaktadır. İntel işlemcilere bakıldığında bu karışıklığı engellemek için ‘‘dallanma tahmin etme birimi’’ bulunmaktadır. ARM işlemcilere bakıldığında her bir komutun hangi durumlarda çalıştırılacağı işlemin bir parçasıdır. İşlemci, komutlara uymayan komut satırlarını atlamaktadır. Bu sayede bazı algoritmaların daha temiz ve performanslı yazılması sağlanmaktadır.
- RICS mimarisi: Bu özellik sayesinde işlemci için tasarım, üretim, çalışacak kodun üretimini kolaylaştırmaktadır.

- Lisans modeli - Model çeşitliliği: Şirketler, ARM lisansını alarak kendi ARM işlemcilerini üretebilmektedir. Bu sonuç karşısında işlemciler farklı tasarım, fiyat, performans özelliklerinde olsalar da aynı özellikte çalışan ARM işlemciler yer almaktadır.
- RAM'e erişim: RAM'e LOAD/STORE ile erişim sağlanmaktadır. ARM komut setinde bütün işlemler yazmaçlar üzerinden yapılmaktadır. Bu sayede komut setinin daha temiz kalması sağlanmaktadır (Cebeci, 2014).

Şekil 2.2'de STM32L053 işlemcisinin discovery blok diyagramı görülmektedir.

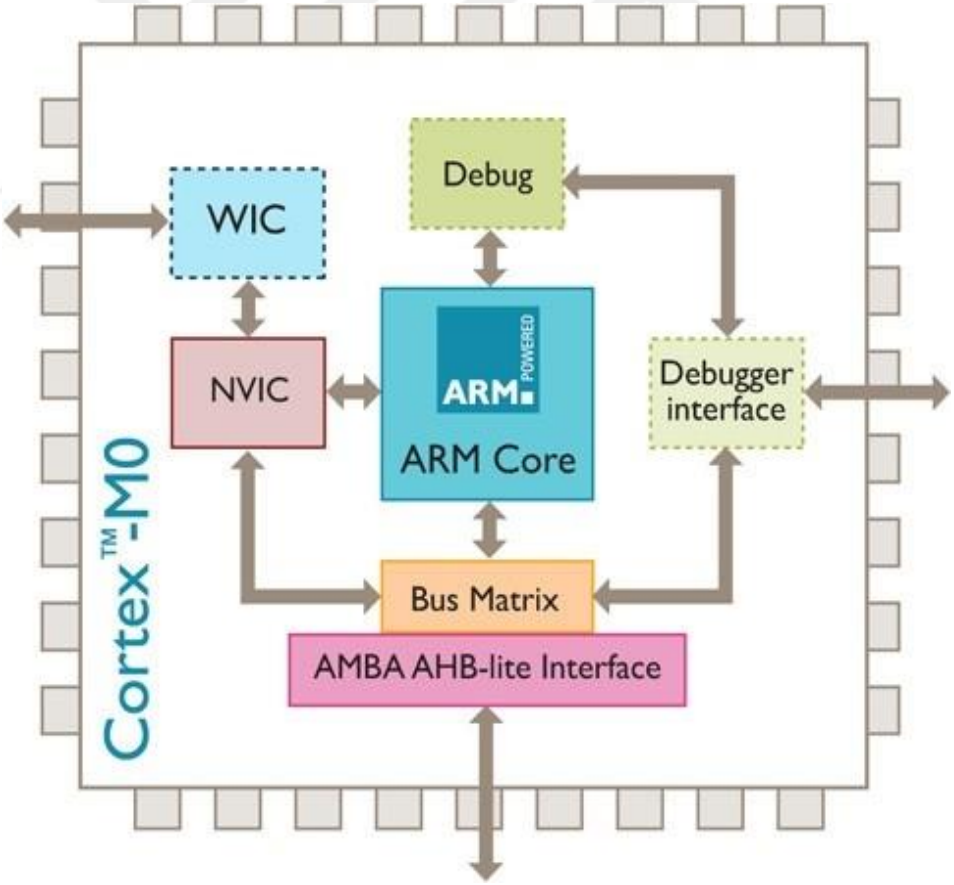


Şekil 2.2. STM32L053 discovery blok diyagramı

2.4. Cortex-M Serisi İşlemci Ailesi

ARM mimarisinin mikrodenetleyici ailesinden biri olan Cortex M serisi, piyasada mevcut bulunan 8 ve 16 bitlik mikrodenetleyicilere rakip olarak ortaya çıkmış 32 bitlik işlemci mimarisine sahip işlemci serisidir. Çok düşük enerji tüketimlerinin yanı sıra maliyetleri de düşüktür. Şekil 2.3'de Cortex M0 işlemcisinin iç yapısı görülmektedir (Anonim, 2017).

Günümüzde Cortex-M3 serisi mikrodenetleyiciler daha yaygın olarak kullanılmaktadır. Bunun nedeni hem daha önce üretilmeleri hem de daha ucuz olmalarıdır. Ancak son olarak tasarlanan M4 serisine eklenen dijital sinyal süreci (DSP) özelliği gelecekte çok daha etkili işlerin yapılabileceğinin bir göstergesidir (Anonim, 2017).



Şekil 2.3. Cortex M0 iç yapısı

2.5. Cortex-M0 Genel Özellikleri

Cortex-M0 işlemci çok çeşitli ürünlerde kullanılır. En çok kullanılan mikrokontrollerdendir. Birçok Cortex-M0 mikrodenetleyici, düşük maliyetlidir ve düşük güç uygulamaları için tasarlanmıştır. Bilgisayar çevre birimleri ve aksesuarları içeren uygulamalarda kullanılabildiği gibi, beyaz eşya, endüstriyel, ısıtma, havalandırma, klima ve ev otomasyonlarında da kullanılırlar (Yiu, 2016).

Cortex-M0 mikrodenetleyicileri kullanılarak, bu ürünlerde daha fazla özellik, çok daha gelişmiş bir kullanıcı arabirimi, daha iyi performans ve genellikle daha iyi enerji verimliliği elde edilebilir. Aynı zamanda, Cortex-M işlemcileri ile yazılım geliştirme, 8-bit ve 16-bit mikrodenetleyicileri kullanmak kadar kolaydır. Cortex-M0 mikrodenetleyicilerinin maliyeti de rekabetçidir (Yiu, 2016).

Cortex-M0 uygulamalarının önemli bir diğer grubu da uygulamaya özel standart ürünler (ASSP'ler) ve sistem-on-chip (SoC)'dir. Karışık sinyal denetleyicileri gibi ASSP'ler için, Cortex-M0'ın düşük kapı sayısı avantajı, geleneksel olarak yalnızca 8-bit veya basit 16-bit işlemcinin kullanılmasına izin veren chip tasarımlarına 32-bit işleme kabiliyetinin dahil edilmesini sağlar. Cortex-M0 mikrodenetleyicisine bir örnek dokunmatik ekran denetleyicileri verebilir (Yiu, 2016).

Karmaşık sistem-on-chip (SoC) için, tasarımlar genellikle bir ana uygulama işlemci sistemi ile I/O denetimleri, iletişim protokolü işleme sistem yönetimi için bir dizi alt sistem ayrılır. Bazı durumlarda, Cortex-M0 işlemcileri ana uygulamadan bazı aktiviteleri boşaltmak için alt sistemlerin bir bölümünde bekleme modunda kullanılır (Yiu, 2016).

2.6. STM32L0 Mikrodenetleyicisi

Çok düşük güç tüketen STM32L0 mikrodenetleyicisi 48 pinden 64 pine kadar farklı paketler halinde sunulmaktadır. Seçilen aygıta bağlı olarak farklı çevre birimlerinde kullanılmaktadır. STM32L0 mikrodenetleyicisinin görünümü Şekil 2.4'de olduğu gibidir. Genel olarak özelliklerine bakıldığında ultra düşük güç tüketen bu

mikrodenetleyicilerin kullanım alanlarının bazıları aşağıda belirtilmiştir (St Microelectronics, 2014).

- Gaz/Su sayaçları ve endüstriyel sensörler
- Uzaktan kumanda ve kullanıcı arabirimi
- PC çevre birimi, oyun, GPS ekipmanları
- Alarm sistemleri, kablolu ve kablosuz sensörler

Şekil 2.4. STM32L0 mikrodenetleyicisi

Yüksek performanslı Arm Cortex-M0 mikrodenetleyicilerine bakıldığında, evrensel seri veri yolu, 32 MHz frekansta çalışan 32 bit RISC çekirdeği, bellek koruma birimi (MPU), yüksek hız gömülü hafıza (64 KB Flash program hafızasına, 2 KB veri EEPROM ve 8 KB RAM) artı gelişmiş I/O'lar ve çevre birimi özellikleri vardır. Bu mikrodenetleyici ailesi geniş performansa sahiptir. Geniş iç ve dış saat kaynakları seçimiyle, dahili gerilim adaptasyonu ve düşük güç modunda çalışmaktadır (St Microelectronics, 2014).

STM32L0 mikrodenetleyicilerin özelliklerine bakıldığında;

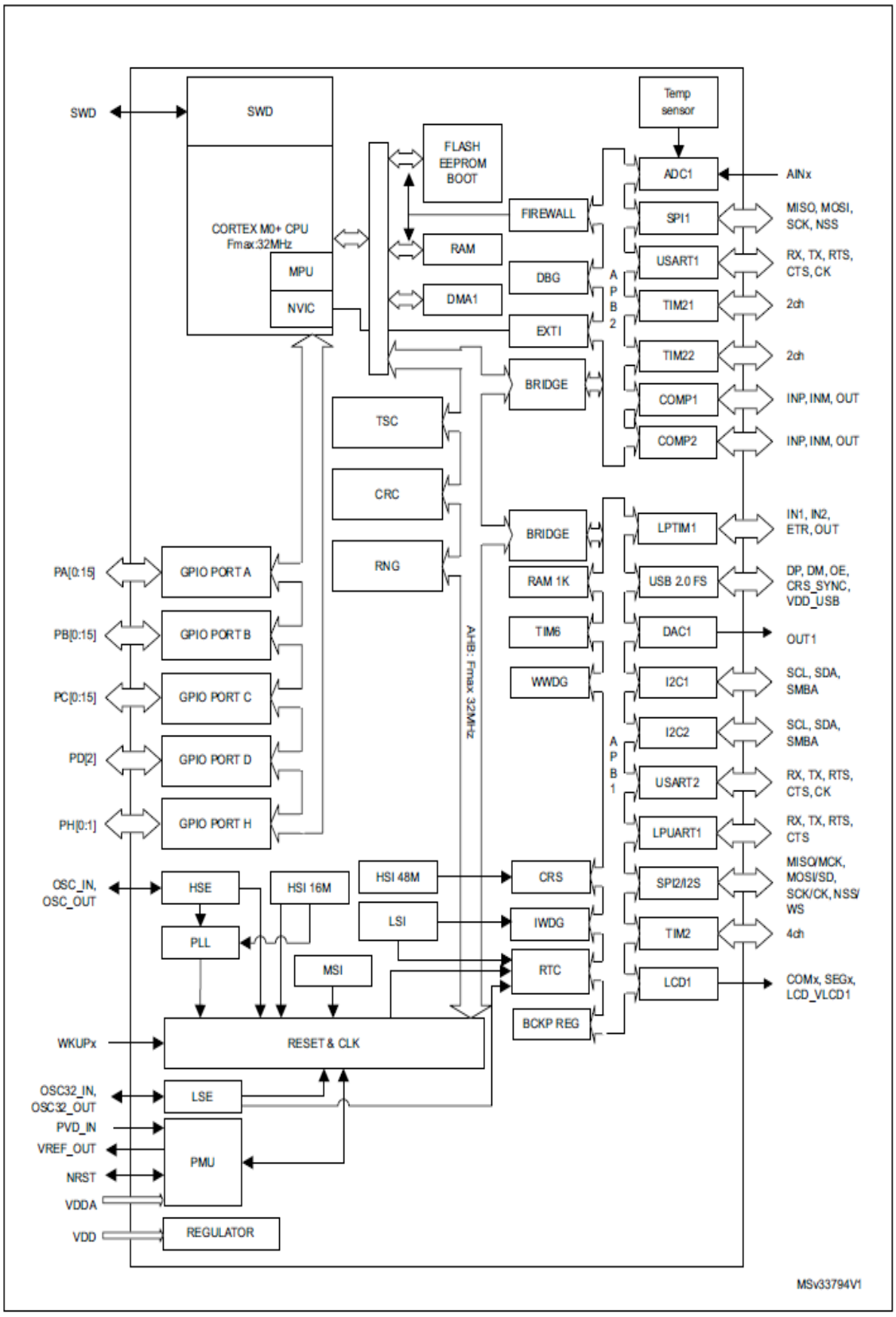
- Donanımlı bir adet 12 bit ADC aşırı örnekleme
- Bir adet DAC
- İki adet ultra düşük güç karşılaştırıcısı
- Zamanlayıcılar

- Bir adet düşük güç zamanlayıcısı (LPTIM)
- Üç adet genel genel amaçlı 16 bit zamanlayıcı
- Bir adet temel zamanlayıcı
- RTC özelliği

STM32L0 mikrodenetleyicisi standart ve gelişmiş iletişim arabirimlerine sahiptir. Bu ara birimler;

- İki adete kadar I2C
- İki adet SPI
- Bir adet I2S
- İki adet USART
- Düşük güçlü bir UART
- Evrensel seri yolu (USB)

Bu mikrodenetleyiciler gerçek zamanlı bir saat ve bir yedek kayıt kümesi içerir. Bekleme modunda açık kalmaya devam etmektedir. Şekil 2.5'de bu işlemcinin blok diyagramı görülmektedir.



Şekil 2.5. STM32L0 blok diyagramı

2.6.1. Analog Dijital Çevirici Genel Özellikleri (ADC)

• Yüksek performans

- 12 bit, 10 bit, 8 bit veya 6 bit çözünürlük yapılandırılabilir.
- ADC dönüşüm süresi: 12 bit çözünürlük (1,14 MHz) için 0,87 µs, 0,81 µs dönüşüm 10 bit çözünürlük zamanı, daha hızlı dönüştürme süreleri indirilerek çözüm elde edilebilir.
- Kendini kalibrasyon edebilme özelliği
- Programlanabilir örnekleme zamanı
- Dahili veri tutarlılığı ile veri hizalama
- DMA desteği

• Düşük güç

- Uygulama da düşük güç çalışması için PCLK frekansını azaltırken çalışmaya devam eder, optimum ADC performansını korur.
- Bekleme modu: Düşük frekanslı PCLK'lı uygulamalarda ADC aşırı yayılmasını önler.
- Otomatik kapanma modu: ADC aktif olması dışında otomatik olarak kapanır. Bu ADC'nin güç tüketimini önemli ölçüde azaltır.

• Analog giriş kanalları

- 16 harici analog giriş
- Dahili sıcaklık sensörü için 1 kanal (VSENSE)
- Dahili referans voltajı için 1 kanal (VREFINT)
- Harici VLCD güç kaynağı pinini izlemek için 1 kanal

• Dönüşüm başlangıcı başlatılabilir:

- Yazılımla
- Yapılandırılabilir polariteye sahip donanım tetikleyicileri (TIM2'den gelen dahili

zamanlayıcı olayları, TIM3, TIM6, TIM21, TIM22 veya GPIO giriş olayları)

• Dönüştürme modları

- Tek bir kanalı dönüştürebilir veya bir dizi kanal tarayabilir.
- Tek mod, tetik başına bir kez seçilen girişleri dönüştürür
- Sürekli mod seçilen girişleri sürekli olarak dönüştürür
- Süreksiz mod

• Analog zamanlayıcı

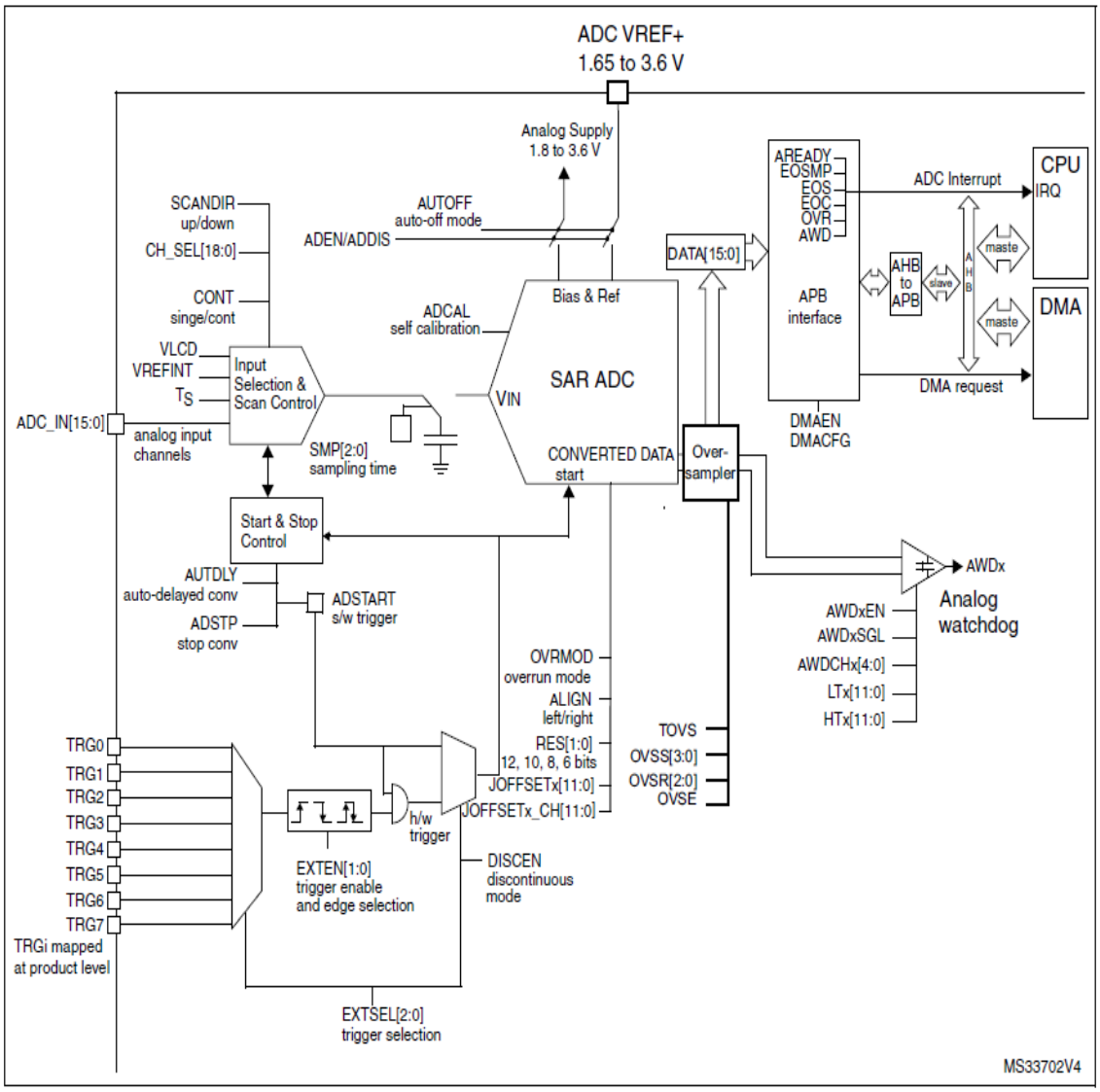
• Yüksek hızda örnekleme

- 16 bit veri kaydı
- 2 - 256x arasında ayarlanabilen aşırı örnekleme oranı
- 8 bit'e kadar programlanabilir veri kayması

• ADC besleme gereksinimleri: 1,65 ila 3,6 V

• ADC giriş aralığı: VSSA ≤ VIN ≤ VDDA

Şekil 2.6'ya bakıldığında analog dijital çevirici blok diyagramı görülmektedir.



Şekil 2.6. ADC blok diyagramı

2.6.2. Düşük Güç Modları

Mikrodenetleyici bir sistemde açılışta sıfırlama işleminden sonra çalışma modundadır. Çalışma modunda CPU çekirdek saati (HCLK) tarafından zamanlanır ve program kodu çalıştırılır. CPU'nun çalışmasına ihtiyaç duyulmadığı anlarda güç tasarrufu sağlamak için düşük güç modları seçilir (Anonim, 2015).

Bu işlemcilerde beş adet düşük güç modu yer almaktadır:

• Düşük güç çalışma modu: düşük güç modunda regülatör, sınırlı saat frekansı, sınırlı

çalışan çevre birimlerinin sayısı yer alır.

• Uyku modu: Cortex-M0 + çekirdek durur, çevre birimleri çalışmaya devam eder.

• Düşük güç uyku modu: Cortex-M0 + çekirdek durur, sınırlı saat frekansı, sınırlı

çalışan çevre birimlerinin sayısı, düşük güç modunda regülatör, flaş durur.

• Durma modu (tüm saatler durdurulur, regülatör düşük güç modunda çalışır.

• Bekleme modu: V_{CORE} alanı kapalı

Ek olarak, çalışma modundaki güç tüketimi aşağıdakilerden biri tarafından azaltılabilir:

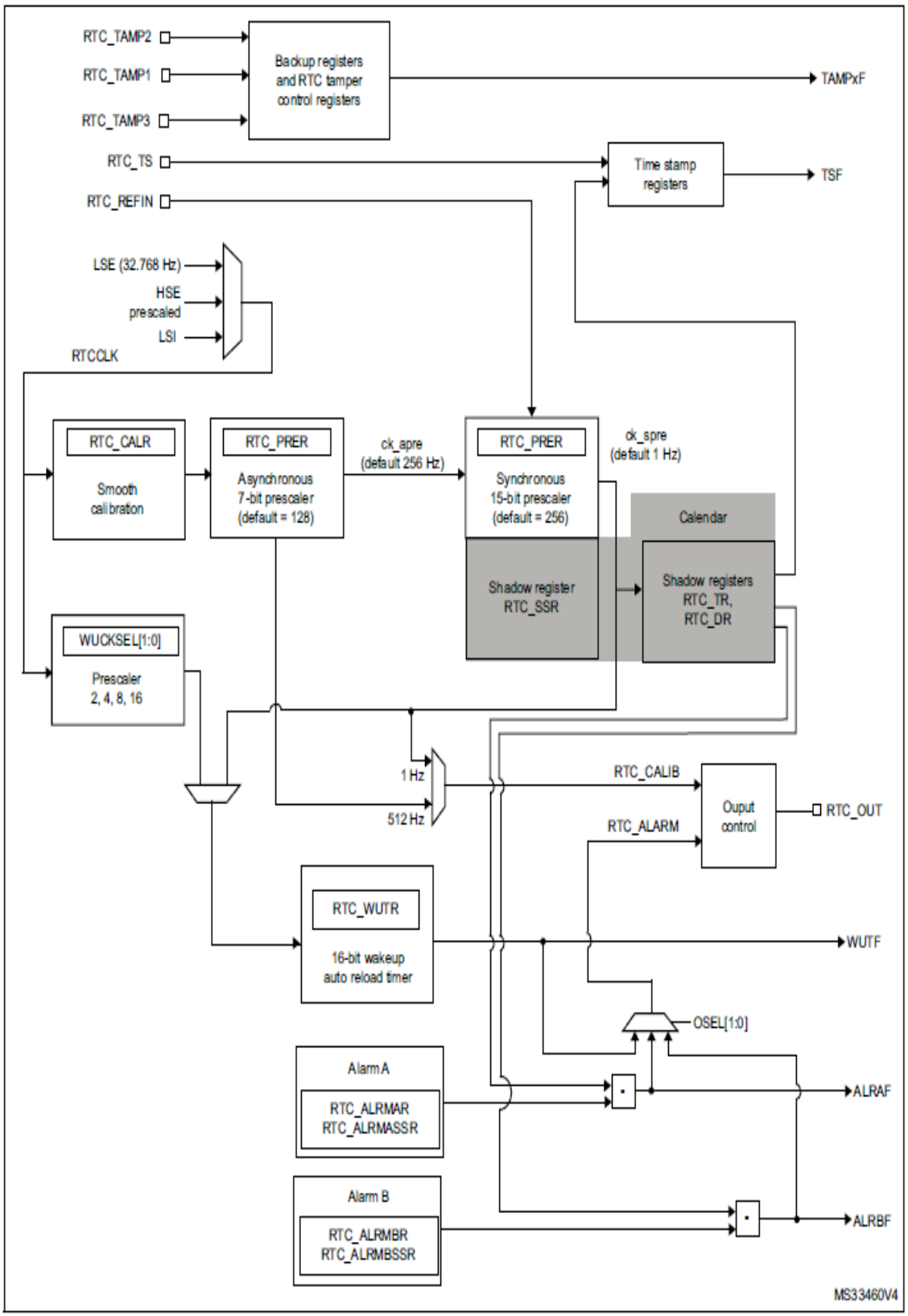
• Sistem saatlerinin yavaşlatılması sonucu

• Saatler kullanılmadığında APBx ve AHBx çevre birimlerinde tutulması sonucu (Anonim, 2015).

2.6.3. Gerçek Zaman Saati

RTC, tüm düşük güç modlarındaki çalışmalar için otomatik bir uyandırma sağlamaktadır. RTC bağımsız bir ikili kodlanmış onluk sayı (BCD) zamanlayıcı/sayaçtır. RTC programlanabilen alarm kesmeleri ile saat, gün ve takvim ayarlaması sağlamaktadır. RTC, kesme özelliğine sahip periyodik olarak programlanabilir bir uyandırma sinyallerini de sahiptir. İki 32 bitlik saniye, dakika, saat (12 veya 24 formatında), gün (gün, hafta), tarih (ayın günü), ay ve yıl, ikili kodlu ondalık formatta kayıt yapabilme özelliğine sahiptir. 28-, 29- (sıçrama yılı), 30-, 31-günlük aylar için otomatik ayarlama yapılabilir.

Kayıtlar için programlanabilir alt-zamanlayıcıları, saniye, dakika, saat, gün ve tarih alarm ayarları yapılabilir. Kristal osilatörlerdeki herhangi bir sapmayı telafi etmek için dijital kalibrasyon özelliği mevcuttur. RTC alan adı sıfırlandıktan sonra, tüm RTC kayıtları olası parazit yazmaya karşı korunur. Cihazın durumuna bakmaksızın besleme gerilimi çalışma aralığında kaldığı sürece, RTC asla durmaz (Anonim, 2015). RTC blok diyagramı Şekil 2.7'de olduğu gibidir.



Şekil 2.7. RTC blok diyagramı

RTC biriminin temel özellikleri şunlardır :

• Alt zaman aşımları, saniye, dakika, saat (12 veya 24 biçiminde), gün, tarih, ay ve yıl yazabilme özelliğine sahiptir.

• Kesme fonksiyonu sayesinde alarm programlanabilir.

• Otomatik uyandırma ünitesi, otomatik uyanmayı tetikleyen periyodik kesme sinyali üretir.

• Referans saati algılama: Takvim duyarlılığını arttırmak için daha hassas ikinci kaynak saati (50 veya 60 Hz) kullanılabilir.

• İkinci derece kaydırma özelliğini kullanarak harici bir saatle doğru senkronizasyon sağlanır.

• Dijital kalibrasyon devresi (periyodik sayaç düzeltme): 0.95 ppm hassasiyet, birkaç saniyede kalibrasyon elde edilebilir.

• Olay kaydetme için zaman damgası fonksiyonu

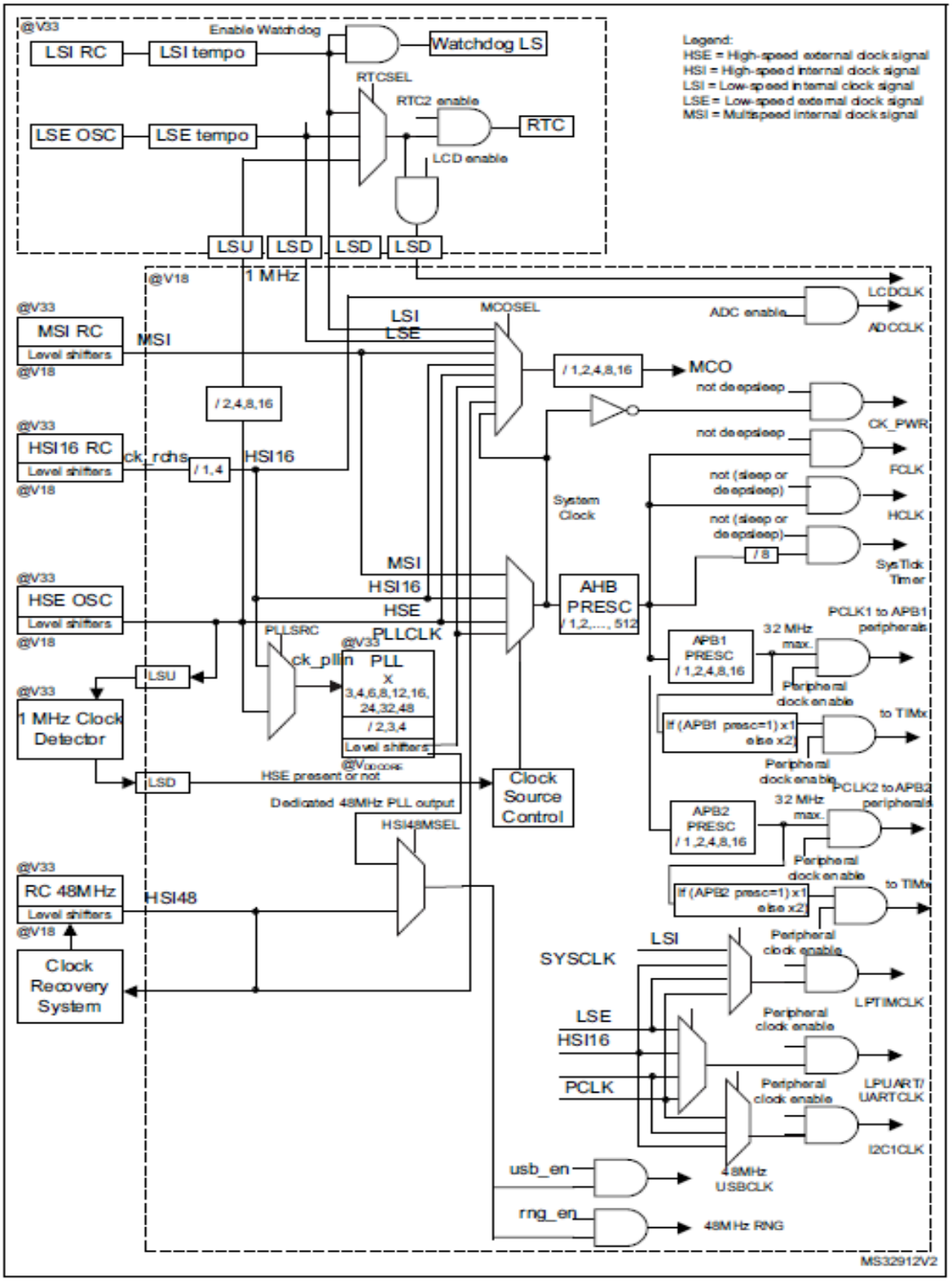
• Ayarlanabilir filtre ve dahili pull-up özellikli tamper algılama olayı

• Maskelenebilir kesmeler / olaylar:

- Alarm A
- Alarm B
- Uyanma kesintisi
- Zaman damgası
- Tamper algılama

• 5 yedek kayıt alma özelliği

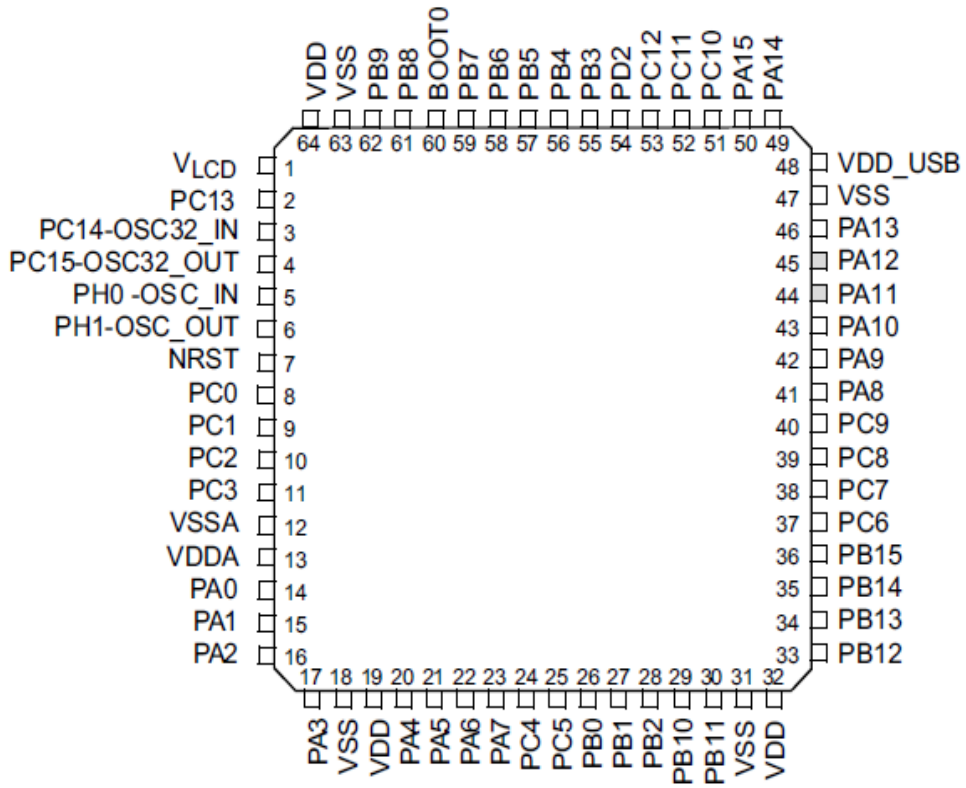
Şekil 2.8'e bakıldığında işlemcinin saat ağacı görülmektedir.



Şekil 2.8. Saat Ağacı

2.6.4. Genel Amaçlı I/O

Şekil 2.9'daki mikrodenetleyicide (STM32L053C8T6) 51 adet hızlı I/O pinleri bulunmaktadır.

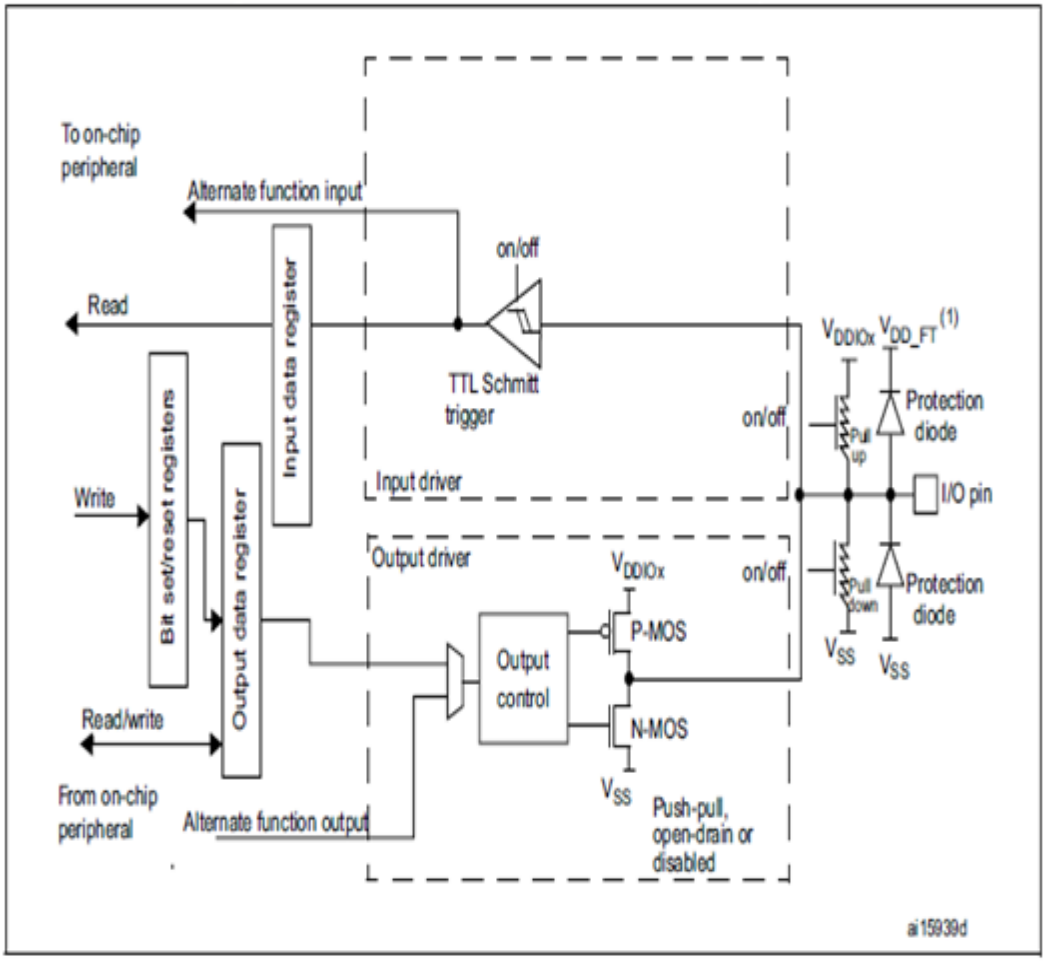


Şekil 2.9. STM32L0C8T6 MCU pin yapısı

I/O bağlantı noktası yapısı Şekil 2.10'da görülmektedir. Bu bağlantı noktasının belirli donanım özelliklerine bağlı olarak, genel amaçlı I/O (GPIO) bağlantı noktalarının her port biti yazılım tarafından çeşitli modlarda ayrı ayrı konfigüre edilebilir:

- Pin çıkışları push-pull/open drain olarak seçilebilir veya pull up/pull down olarak da seçilme özelliğine sahiptir.
- Portlar dijital I/O olarak kullanılabildiği gibi alternatif fonksiyon (PWM, DAC v.b.) olarak da kullanılabilmektedir.

- Portların hızları ayarlanabilmektedir.
- Pull-up / pull-down olarak pin girişleri ayarlanabildiği gibi, analog veya boşta ayarlanabilmektedir.
- Pinler analog da ayarlanabilmektedir.
- GPIOx_BSRR (bit set/reset) kaydedicisi ile GPIOx_ODR port kaydedicilerine bit bazında erişim sağlanabilmektedir.
- Port girişleri GPIOx_IDR kaydedicisi ile alternatif veya dijital fonksiyon olarak yapılabilmektedir.
- Hızlı konum değişimi özelliğine sahiplerdir.
- GPIOx_BSRR ve GPIOx_BRR kayıtları, GPIOx_ODR kayıtlarından herhangi birisine okuma/değiştirme erişimine izin vermektedir (Anonim, 2015).



Şekil 2.10. 5 Volt toleranslı I/O bağlantı noktası temel yapısı

2.7. Sıcaklık Sensörü

Sıcaklık sensörü (T_{SENSE}) sıcaklık ile doğrusal olarak değişen V_{SENSE} gerilimi üretir. Sıcaklık sensörü dahili olarak sensör çıkış voltajını dijital değere dönüştürmek için kullanılan ADC_IN18 giriş kanalına bağlanır (St Microelectronics, 2014).

Sensör doğrusallığı sağlamaktadır, ancak sıcaklık ölçümünün daha iyi bir doğrulukta ölçümü sağlanabilmesi için kalibre edilmesi gerekmektedir. Bu kalibrasyon değerleri Çizelge 2.1'de görülmektedir. Sıcaklık sensörünün bekleme süresi, süreç değişimi nedeniyle çipten çipe değiştiğinden, kalibre edilmemiş dahili sıcaklık sensörü sadece sıcaklık değişimini algılayan uygulamalar için uygundur.

Sıcaklık sensörü ölçümünün doğruluğunu iyileştirmek için, her cihaz ST tarafından ayrı ayrı fabrika da kalibre edilmektedir. Sıcaklık sensörü fabrika kalibrasyon verileri ST tarafından salt okunur modda erişilebilen sistem belleği alanında saklanmaktadır (St Microelectronics, 2014).

Çizelge 2.1. Sıcaklık sensörü kalibrasyon değerleri

Kalibrasyon değeri adı	**Açıklama**	**Bellek Adresi**
TSENSE_CAL1	TS ADC'ye ait işlenmemiş verilerden elde edilen sıcaklık 30°C, V_{DDA} = 3V	0X1FF8 007A - 0X1FF8 007B
TSENSE_CAL2	TS ADC'ye ait işlenmemiş verilerden elde edilen sıcaklık 130°C, V_{DDA} = 3V	0X1FF8 007E - 0X1FF8 007F

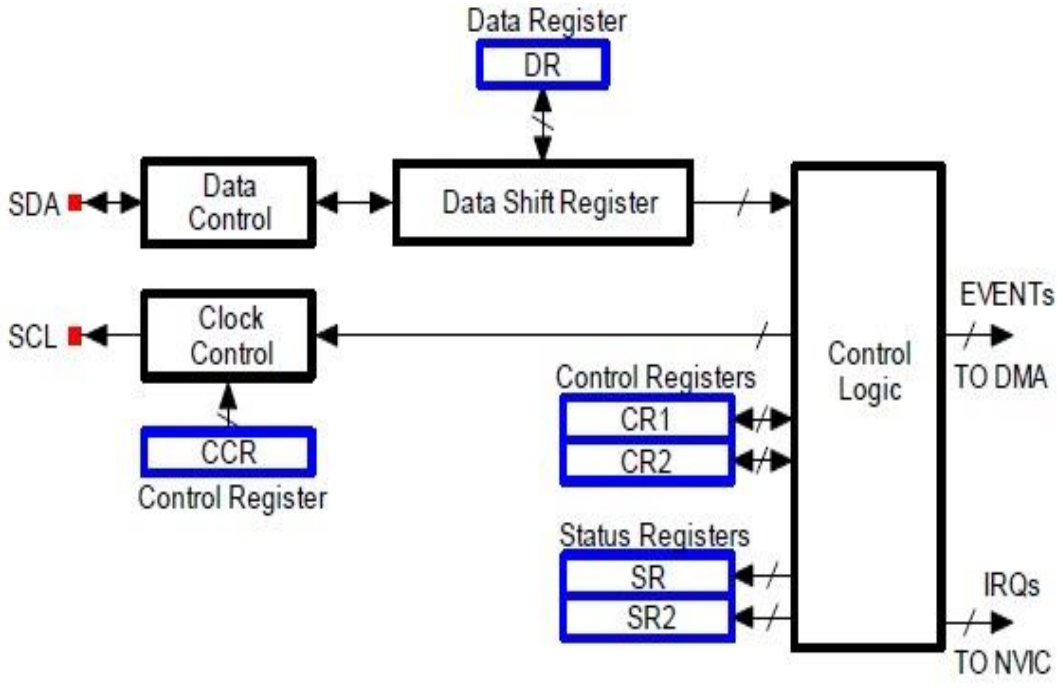
2.8.I2C Veri Yolu

Sersörlerin mikroişlemcilerden uzakta kullanıldığı zamanlarda vardır. ADC'ler bunlar için SPI, I2C veya paralel veri yollarından her hangi birisini kullanarak sensörden okunan analog bilginin dijital değerini gönderir (Kaplan, 2015).

Diğer haberleşme protokollerine göre I2C protokolü hızlı veri aktarımı sağlamaktadır. Birlikte çalışan ve belirli sürelerde haberleşen, çeşitli çevresel cihazların en az donanımla haberleşmesini sağlamaktadır. Düşük bant genişliğine sahiplerdir ve kısa mesafe protokolüdür. Adresleme planları olması sebebiyle birden fazla cihazla haberleştirilebilir (Sural, 2010).

I2C protokolünde SCL ve SCA olmak üzere iki hat vardır. Bu iki hat, veri senkronizasyonunda kullanılan saat darbeleri hattı (SCL) ve veri hattıdır (SDA). I2C protokolünde verilerin gönderilmesi ve okunması aynı hat üzerinde gerçekleşmektedir. SDA hattında transfer işlemi gerçekleştirilir. SDA pini bir pull-up direnciyle bağlı olması gerektiği V_{CC}'ye, hattaki start-stop bitleri, hatta lojik 0 olup olmaması ile anlaşılır (Sural, 2010).

Şekil 2.11'de STM32 mikrodenetleyicisine ait I2C blok yapısı görülmektedir. I2C protokolünde, genelde mikrodenetleyicilerde kullanılan ana kontrol cihazı 'Master' ve 'Slave' denilen bir cihaz mevcuttur. I2C protokolünde veri yoluna bağlı slave cihazları en fazla 7 bitten oluşur ve her birinin kendine ait adresi vardır. Bu cihazlar LCD, EEPROM veya bir sayısal sistem olabilir. Veri yolu da iki yönlü haberleşme sağlanabilir (Sural, 2010).



Şekil 2.11. STM32 I2C basitleştirilmiş blok yapısı

2.9.USB (Evrensel Veri Yolu)

Evrensel veri yolu (USB), tak çalıştır gibi çok önemli özelliğinden dolayı, bilgisayar ve çevre birimleri arası veri alış verişi sağlayan, yaygın bir şekilde kullanılan seri haberleşme teknolojisidir. Bu teknoloji birçok aygıt için ortak kullanım arabirimi sağlamakta ve diğer arabirimlerde oluşan kablo karışıklığının da önüne geçmektedir. USB haberleşme teknolojisi, evrenselleşme ve diğer portların yerini almak için geliştirilmiştir (Çimenli, 2013).

USB, kullanıcılara sağlamış olduğu kolaylıklar ve avantajları ile günümüzde vazgeçilmez veri yolu standartlarından birisi haline gelmiştir. USB gelişim sürecine bakıldığında Çizelge 2.2'de görüldüğü gibi, Kasım 1994'de 0.7 sürümü ile gelişime başlayan standart, Nisan 2000'de 2.0 sürümü ile devam etmiş ve Temmuz 2013'de 3.1 sürümünü geliştirmiştir (Kiremitci, 2007).

Çizelge 2.2. USB tarihi gelişim süreci

Sürüm	Tarih
0.7	Kasım 1994
0.8	Aralık 1994
0.9	Nisan 1995
0.99	Ağustos 1995
1.0 FDR	Kasım 1995
1.0	Ocak 1996
1.1	Eylül 1998
2.0 (0.79 tasarısı)	Ekim 1999
2.0 (0.9 tasarısı)	Aralık 1999
2.0	Nisan 2000
3.0	Kasım 2008
3.1	Temmuz 2013

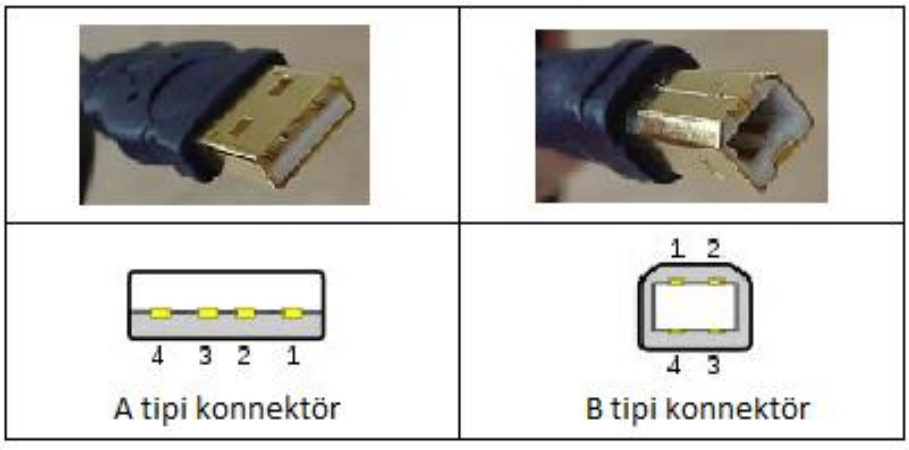
USB'nin sağlamış olduğu avantajlar:

- Tek bir PC'ye 127 adete kadar cihaz bağlayabilme

- Hiçbir sürücüye, IRQ ayarlarına, DMA kanallarına ve I/O adreslerine, genişleme yuvalarına gerek duymadan kolay kurulum
- Çevre cihazları için PC'yi kapatıp açmadan tak ve çalıştır fonksiyonelliği
- Bütün cihazlar için tek tip konnektör
- PC'yi kapatmadan cihaz ekleme ve kaldırma özelliği
- Düşük maliyetli olması
- Güç tasarrufu sağlaması
- Güvenilir ve hızlı veri akışının sağlanması

2.9.1. USB Konnektörleri

USB'de, Şekil 2.12'de görüleceği üzere A serisi ve B serisi olmak üzere iki tip konnektör kullanılmaktadır. Kullanıcıların yanlış yapmalarını önleyebilmek ve veri yolunu uygun olmayan topolojilerden koruyabilmek için bu kadar az tipte kullanılmaktadır. Bu durum karşısında kullanıcı doğru konnektörü, portlara ve çevre birimlerine kolaylıkla bağlanmasını sağlamaktadır. Çizelge 2.3'de ise her bir pin ucunun sinyali gösterilmektedir (Buldu, 2003).



Şekil 2.12. A ve B tipi konnektör yapısı

Çizelge 2.3. USB pin numara ve sinyal sıralaması

Pin No	Sinyal
1	V_{CC}
2	Data -
3	Data +
4	Toprak

2.9.2. USB Protokolü

Ev sahibi (Host) ile aygıtların haberleşmeleri için USB bir protokol sağlamaktadır. Host, belirli bir zaman aralığında birden fazla haberleşme sağlayamaz, yani tek bir aygıtla haberleşmesi söz konusudur. USB yolu üzerinde veri akışı belirli bir zaman aralığında tek yönlü taşınmaktadır (Çimenli, 2013).

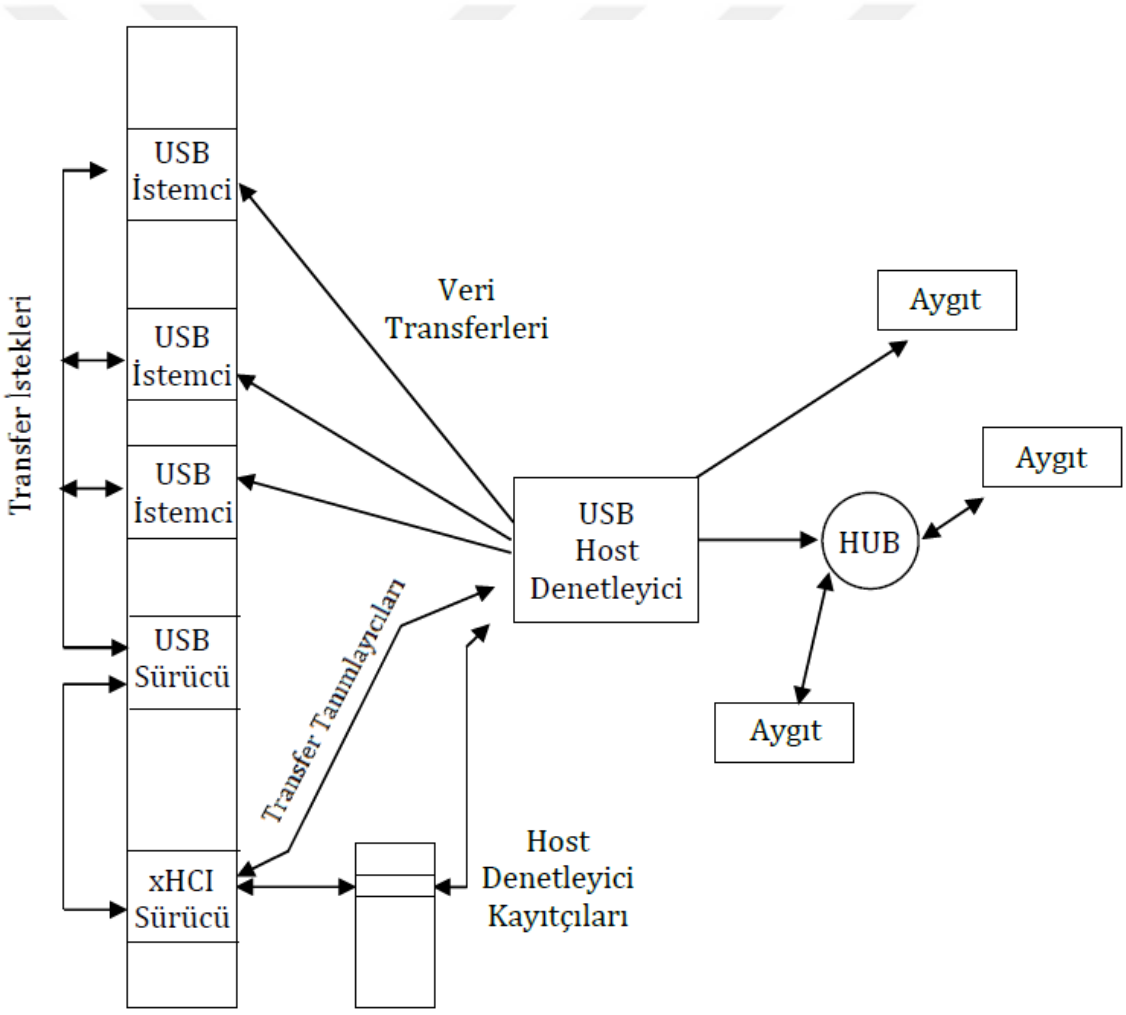
Sistemden çekilen kaynakları, USB aygıtları doğrudan tüketmemektedir. USB aygıtları tarafından ihtiyaç duyulan kaynak aynı zamanda bu aygıtların kullandığı yazılım hafıza alanıdır. Bu sayede USB istemci, transfer isteğinde bulunmasını istendiği zaman USB sistem yazılımının çağırılması yeterlidir. USB aygıtları üzerinden USB istemci sürücüleri transfer işlemini gerçekleştirirken depolama yapabilmek için tampon bir bellek sağlamaktadır. Transfer işlemleri, aygıt konfigürasyonu sırasında USB sistem yazılımının oluşturduğu iletişim kanalı sayesinde gerçekleşir. USB sistem yazılımları gelen transfer isteklerini aygıt band genişliğine uygun olarak farklı işlemlere ayırmaktadır (Çimenli, 2013).

Bu protokolün avantajları :

- Protokol işleminin az olmasına karşılık yüksek verimler elde edilmektedir.
- Farklı hızlardaki veri transferi sağlanmaktadır.
- Farklı paket boyutundaki veri transferi sağlanmaktadır.
- Akış kontrolü protokol tarafından sağlanmaktadır.
- Güvenlik açısından istenildiği zaman takılıp çıkartılabilmektedir.

- Protokolün uygulanması oldukça kolaydır.
- Bilgisayarlardaki tak-çalıştır özelliğine uyum sağlamaktadır.
- İşletim sistemleri ile uyum içerisinde çalışmaktadır.
- Sunuculara kolay uygulanmaktadır.
- Düşük fiyatlı aygıtların üretimine olanak sağlamaktadır.
- Bağlantı elemanları düşük fiyatlıdır (Dinçbakır, 2002).

USB temel iletişim akışı Şekil 2.13'de görülmektedir.

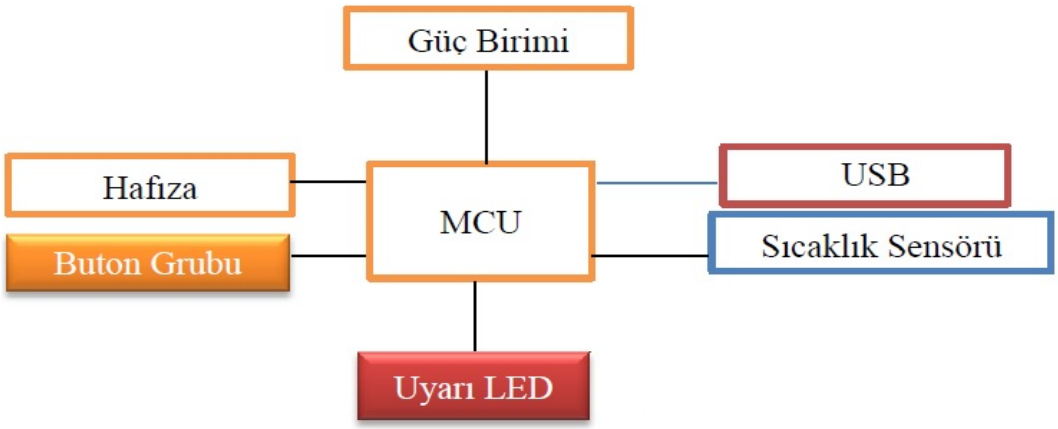


Şekil 2.13. USB genel iletişim modeli (Anderson, 2001)

3. MATERYAL VE YÖNTEM

3.1. Donanım Seçim Kriterleri

Bu tez çalışmasında kullanılan ana donanımların listesi aşağıda görülebilmektedir. Sıcaklık veri kayıt cihazı blok diyagramı Şekil 3.1'de verilmiştir.



Şekil 3.1. Sıcaklık veri kayıt cihazı blok diyagramı

- MCU – Mikrodenetleyici
- Sensörler – Sıcaklık & Nem Sensörü
- Güç Kaynağı
- Hafıza
- Bağlantı Konnektörleri
- PCB

Bu listede belirtilen elemanların seçim kriterlerini aşağıdaki alt başlıklarda incelenmektedir.

3.1.1. Mikrodenetleyici

Mikrodenetleyici seçimi yapılırken öncelikle düşük güç tüketimine sahip bir mikrodenetleyici seçilmesi gerekmektedir. Bunun sebebi sistemin küçük bir buton pil ile çalıştırılacak olması ilk öncelikli gelmektedir. Diğer taraftan düşük güçte çalışan bir

mikrodenetleyici, çalışma durumunda anlık ve devamlı kullanılan akım değerleri kullanılabilecek olan pillerin anlık ve devamlı verebildiği akım değerlerini geçmemesi gerekmektedir. Mikrodenetleyicinin ölçüm sıcaklık aralığında çalışabilmesi diğer bir parametredir.

Mikrodenetleyicinin kullanılacağı frekans kullandığı gücü değiştirmektedir. Bu nedenle düşük frekans da çalışabilen bir mikrodenetleyici seçilmesi gerekmektedir. Seçilen mikrodenetleyicinin değişik güç modlarının bulunması, mikrodenetleyicinin istenilen performansta çalışabileceğinin bir kanıtı olarak görülebilir.

Seçilen mikrodenetleyicide iki adet SPI – Serial Protochol Interface, bir adet I2C – Inter Integrated Circuit, USB – Universal Serial Bus, RTC – Real Time Clock, en az üç adet timer, en az altı adet GPIO – General Purpose Input Output bulunmalıdır.

Kullanım kolaylığı düşünüldüğünde mikrodenetleyicinin kendisinin osilatör ve USB arabilirimi bulundurması önem oluşturmaktadır. Son olarak düşük güç tüketimi yakalanabilmesi için mikrodenetleyicileri herhangi bir işlem yapmadığı zamanlarda uyku veya düşük uyku modunda bekletilmesi gerekmektedir. Sensörlerden verilerin okunup değerlendirilebilmesi için mikrodenetleyicinin uyku modundan çıkıp çalışma moduna gelmesi gerekmektedir. Burada uyku modu ile çalışma modu arasındaki geçiş süreleri verilerin tatmin edici bir şekilde alınması için önem oluşturmaktadır ve bu nedenle bu sürenin mikrosaniyeler mertebesinde olması gerekmektedir.

Bu kriterler göz önünde bulundurulduğunda aşağıdaki listedeki firmaların mikrodenetleyicilerini bulunabilmektedir.

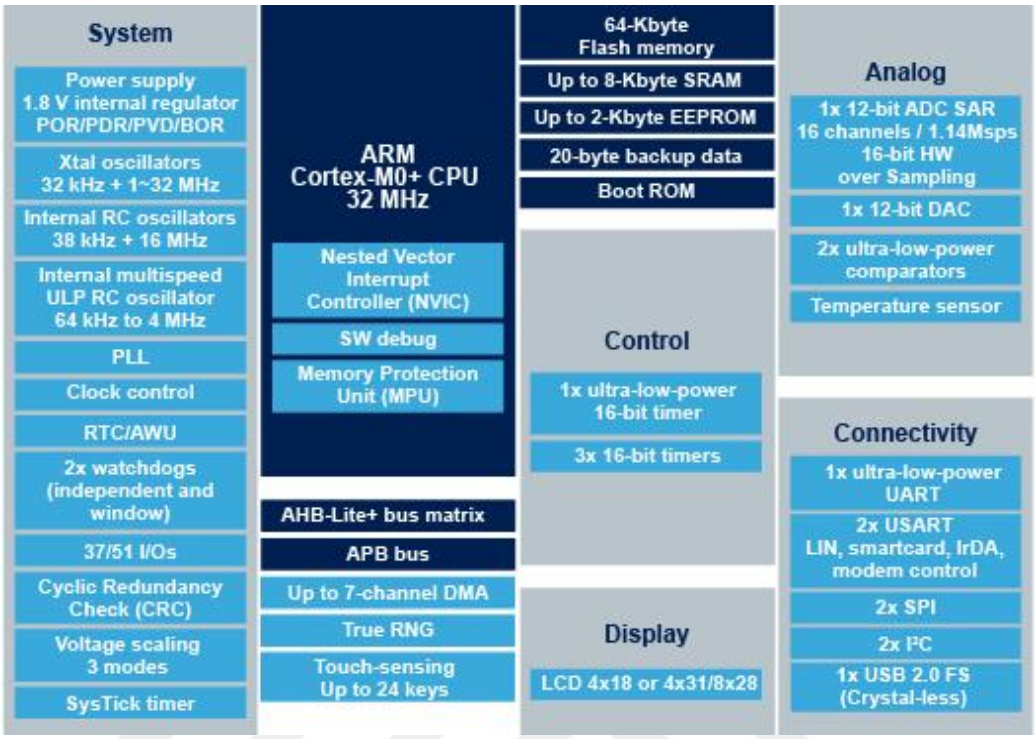
- TI Ultra Low Power MCU serisi
- Renesans RX100
- Freescale Arm Cortex M0
- ST Microelectronics STM32L serisi
- Atmel picoPower serisi
- Silicon Labs C8051 serisi
- Microchip PIC serisi

Bu listedeki işlemci ailelerine bakıldığında istenilen özellikleri sağlayan mikrodenetleyiciler bulunmaktadır. Bulunan bu mikrodenetleyiciler arasında çok ufak sayılabilecek farklılıklar ortaya çıkmaktadır. Burada birincil olarak mikrodenetleyicinin güç tüketimi göz önüne alındığında ST Micro Electronics firmasının ürünleri ön plana çıkmaktadır. Diğer taraftan bu mikrodenetleyicilerin 32 bitlik ARM Cortex M0 tabanlı olmasının avantajları bulunmaktadır.

Bu nedenden dolayı sistemde kullanılacak işlemci olarak ST Microelectronics firmasının STM32L053C8T6 ürünü seçilmiştir. Bu mikrodenetleyiciye ait devre diyagramı Şekil 3.2'de görülmektedir. Seçilen bu mikrodenetleyicinin başlıca özellikleri aşağıda belirtilmektedir.

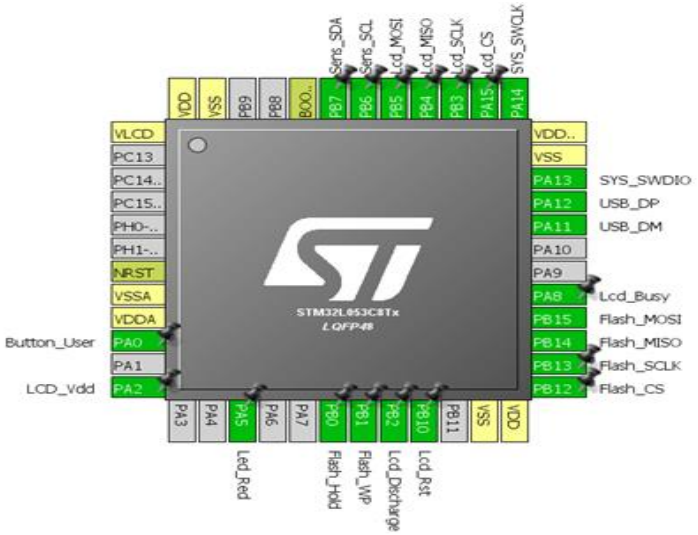
- Çok Düşük Güç Platformu
  - 1.65 V to 3.6 V güç kaynağı
  - -40 to 125 °C sıcaklık aralığı
  - 0.27 µA bekleme modu
  - 0.4 µA durma modu
  - 0.8 µA durma modu RAM aktif
  - 88 µA/MHz çalışma modunda
  - 3.5 µs RAM den uyanma süresi
  - 5 µs FLASH dan uyanma süresi
- ARM® 32-bit Cortex®-M0
  - 32 kHz den 32 MHz
  - 0.95 DMIPS/MHz
- Reset ve Güç Yönetimi
  - Korumalı, düşük güç BOR (brownout reset)
  - Çok düşük güç POR/PDR
  - Programlanabilir gerilim dedektörü
- Clock
  - 1 den 25 MHz crystal oscillator
  - 32 kHz oscillator
  - Dahili Yüksek Hız 16 MHz RC
  - Dahili 37 kHz RC

    - Dahili farklı hız 65 kHz - 4.2 MHz RC
    - CPU clock için PLL
- Önceden yüklenmiş bootloader
    - USART, SPI
- Geliştirme
    - Serial wire debug desteklenir
- 51 adete kadar hızlı I/O
- Hafıza
    - 64 KB Flash
    - 8 KB RAM
    - 2 KB EEPROM
    - 20 byte backup register
    - Yazma okumada sektör koruma
- LCD sürücü, maksimum 8×28 segment
    - Kontrast Ayarlama
    - Blink Modu
    - Dahilli Voltaj Yükseltici
- Analog Çevre Birimleri
    - 12-bit ADC 1.14 Msps 16 kanal
    - 12-bit 1 kanal DAC
    - 2x düşük güçlü komparatör
- 24 kanala kadar dokunmatik sensör
- 7-kanal DMA kontrolcü, desteklenen ADC, SPI, I2C, USART, DAC, Timers
- 8x çevrebirimi iletişim
    - 1x USB 2.0
    - 2x USART (ISO 7816, IrDA), 1x UART
    - 4x SPI 16 Mbits/s
    - 2x I2C (SMBus/PMBus)
- 9x timers: 1x 16-bit 4 channels, 2x 16-bit with, 1x 16-bit timer, 1x SysTick, 1x RTC, 1x 16-bit DAC, and 2x watchdogs
- CRC hesap birimi, 96-bit unique ID
- True RNG



Şekil 3.2. Mikrodenetleyici devre diyagramı

Mikrodenetleyicinin pin bağlantı diyagramı Şekil 3.3'de gösterilmektedir.



Şekil 3.3. MCU pin bağlantıları

Bu bağlantılara ait açıklama Çizelge 3.1'de görülmektedir.

Çizelge 3.1. MCU pin bağlantı açıklama tablosu

Pozisyon	Adı	Tipi	Sinyal	Etiket	Açıklama
1	VLCD	Power			Güç
2	PC13	I/O			Bağlantı yok
3	PC14-OSC32_IN	I/O			Bağlantı yok
4	PC15-OSC32_OUT	I/O			Bağlantı yok
5	PH0-OSC_IN	I/O			Bağlantı yok
6	PH1-OSC_OUT	I/O			Bağlantı yok
7	NRST	Reset			Reset Bağlantısı
8	VSSA	Power			Toprak Bağlantısı
9	VDDA	Power			Güç
10	PA0	Input	GPIO_Input	Button_User	Buton Bağlantı
11	PA1	I/O			Bağlantı yok
12	PA2	Output	GPIO_Output	LCD_Vdd	Ekran Güç Kontrol
13	PA3	I/O			Bağlantı yok
14	PA4	I/O			Bağlantı yok
15	PA5	Output	GPIO_Output	Led_Red	Uyarı LED Bağlantısı
16	PA6	I/O			Bağlantı yok
17	PA7	I/O			Bağlantı yok
18	PB0	Output	GPIO_Output	Flash_Hold	Hafıza Hold bağlantı
19	PB1	Output	GPIO_Output	Flash_WP	Hafıza WP Bağlantı
20	PB2	Output	GPIO_Output	Lcd_Discharge	Ekran Discharge Bağlantı
21	PB10	Output	GPIO_Output	Lcd_Rst	Ekran Reset Bağlantı
22	PB11	I/O			Bağlantı yok
23	VSS	Power			Toprak Bağlantısı

Çizelge 3.1. (Devam) MCU pin bağlantı açıklama tablosu

24	VDD	Power			Güç
25	PB12	Output	GPIO_Output	Flash_CS	Hafıza SPI bağlantı
26	PB13	I/O	SPI2_SCK	Flash_SCLK	Hafıza SPI bağlantı
27	PB14	I/O	SPI2_MISO	Flash_MISO	Hafıza SPI bağlantı
28	PB15	I/O	SPI2_MOSI	Flash_MOSI	Hafıza SPI bağlantı
29	PA8	Input	GPIO_Input	Lcd_Busy	Ekran İşlem Durum
30	PA9	I/O			Bağlantı yok
31	PA10	I/O			Bağlantı yok
32	PA11	I/O	USB_DM		USB Data Bağlantı
33	PA12	I/O	USB_DP		USB Data Bağlantı
34	PA13	I/O	SYS_SWDIO		MCU Programlama
35	VSS	Power			Toprak Bağlantısı
36	VDD_USB	Power			Güç
37	PA14	I/O	SYS_SWCLK		MCU Programlama
38	PA15	Output	GPIO_Output	Lcd_CS	Ekran SPI
39	PB3	I/O	SPI1_SCK	Lcd_SCLK	Ekran SPI
40	PB4	I/O	SPI1_MISO	Lcd_MISO	Ekran SPI
41	PB5	I/O	SPI1_MOSI	Lcd_MOSI	Ekran SPI
42	PB6	I/O	I2C1_SCL	Sens_SCL	Sensör I2C
43	PB7	I/O	I2C1_SDA	Sens_SDA	Sensör I2C
44	BOOT0	Boot			Açılış Dizin
45	PB8	I/O			Bağlantı yok
46	PB9	I/O			Bağlantı yok
47	VSS	Power			Toprak Bağlantısı
48	VDD	Power			Güç

MCU bağlantılarında güç bacaklarına 100 nF'lık kapasitörler stabilizasyon amaçlı takılmıştır. Düşük güç mikrodenetleyicisinin VDDA pinine bir adet ferrit boncuk (ferrite bead) kullanarak gücün verilmesi sonucunda oluşabilecek gürültü durumlarından kurtulmuş olunur.

USB bağlantısından gelen güç hattı üzerinde bir adet schotky diyot kullanılmıştır. Böylece USB bağlantısına geri bir akım oluşması engellenmiştir. Bu diyotdan sonra sistemin kullanacağı voltaj değerine bir regülatör yardımı ile indirilmiştir. Regülatörün düşük gürültülü bir şekilde çalışabilmesi için giriş ve çıkışına birer adet 4.7 µF'lık kapasitörler bağlanmıştır.

Pil besleme hattına bir adet schotky diyot konularak sistem hem USB bağlantısı, hem de pil ile beslenirken ters akımların oluşup pile zarar vermesi engellenmiştir. Üreticinin önerdiği şekilde sensörün I2C hattına pull-up dirençleri bağlanmıştır. Diğer taraftan uyarı ledine 1 KΩ'luk bir direnç takılarak akım sınırlandırılması yapılmıştır. Hafıza chipinin güç beslemesine bir adet stabil çalışması için 100 nF'lık kapasitör takılmıştır.

Mikrodenetleyicinin programlanabilmesi için gerekli pinler 2.54 mm pin aralığına sahip bir konnektörle çıkarılmıştır. Buton bağlantısında iki adet 10 KΩ'luk direnç ve 100 nF kapasitör sayesinde elektrik akımındaki küçük dalgalanmaları kaldırma (debounce) özelliği kazandırılmıştır.

USB bağlantısı baskılı devre kartının kendisi üzerinden olacağından dolayı, çizim programının kütüphanesine bir adet komponent tanımlanmış ve USB erkek olarak çizilmiştir. Bu devre şemasında kullanılan devre elemanlarıyla ilgili ayrıntılara Ek-2 deki listede görülebilir.

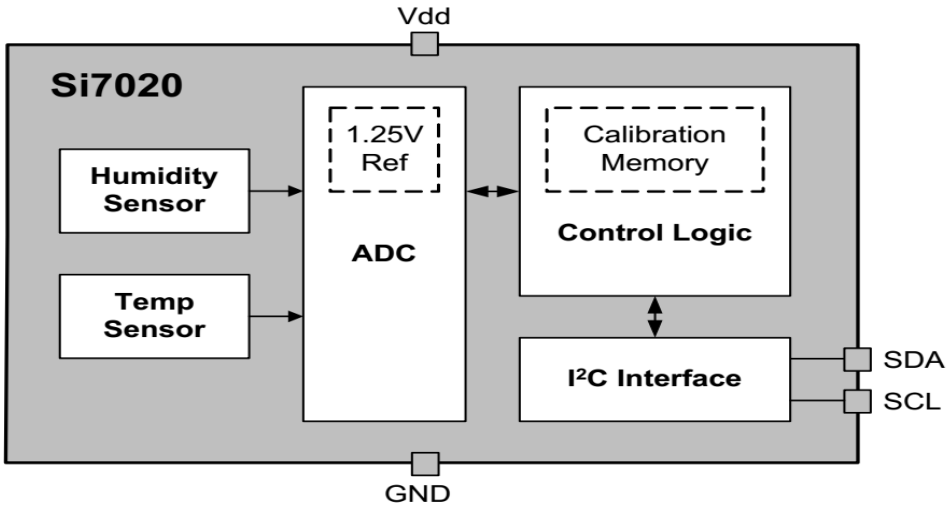
3.1.2. Sensörler

Kullanılacak olan sıcaklık ve nem sensörleri belirtilen -30 °C ile +70 °C derece arasında ölçüm yapabilmesi gerekmektedir. Diğer taraftan mikrodenetleyici ile I2C bağlantısı üzerinden veri iletişimi yapabilmelidir. Bu iletişimin seçilmesinin ana sebeplerinden birisi üretimden kalibreli bir şekilde çıkan sensörlerin bu veri iletişim protokolünü kullanmasıdır. Böylece tekrardan belirli sürelerde kalibrasyon yapılma ihtiyacı ortadan kalkmış olmaktadır. Diğer taraftan bu veri iletişim metodunun güvenilir

olması seçim kriterleri açısından önemlidir. Şekil 3.4'de sıcaklık sensörünün görünümü verilmiştir.

Şekil 3.4. Sıcaklık sensör görünümü

Seçilen sensörün çalışma gerilim aralığının 1.9 V ile 3 V olması gerekmektedir. Burada seçilen gerilim değerleri kullanılan pilden kaynaklanmaktadır. Bu gerilim aralığında sensörün düşük güç tüketimli olması sistemin kararlı bir şekilde çalışabilmesi için gereklidir. Nem ölçümleri için sensörün içinde ısıtıcının bulunması gerekmektedir. Mekanik olarak sistemin gerekli inceliğinin yakalanması için olabildiğince boyutsal olarak küçük bir sensör olması gerekmektedir. Sensöre ait blok diyagramı Şekil 3.5'de görülmektedir.



Şekil 3.5. Si7020 Sıcaklık sensörü blok diyagramı

Bu seçim kriteleri göz önünde bulundurulduğunda piyasada bulunabilen sensörler arasında yapılan seçim Silicon Labs firmasının Si7020 modelidir.

Seçilen bu sensörün başlıca özellikleri aşağıda verilmiştir:

- ±4% RH
- ±0.4 °C
- 0 – 100 % RH ölçüm aralığı
- -40 °C ile +125 °C çalışma aralığı
- 150 µA aktif akım
- Fabrika kalibrasyonlu
- I2C arayüzü
- Dahili ısıtıcı
- 3x3 mm DFN paket
- 1.9 V – 3.6 V çalışma gerilim aralığı

3.1.3. Güç Kaynağı

Sistemin genel yapısı itibari ile seçilecek olan pil gerekli güce, küçük boyuta ve istenilen sıcaklık değerlerinde çalışabilme özelliğine sahip olması gerekmektedir. Seçilen pilin gerilim değeri 3 V olmalıdır. Polimer tabanlı piller istenilen sıcaklık aralığında çalışmadığından dolayı alternatifler arasından çıkmaktadır.

Yapılan araştırma neticesinde sisteme uygun piller arasına CR2032 ve CR2450 serisi piller girmektedir. Burada CR2450 nin kapasitesi daha fazla olması bir avantaj olmasına rağmen fiziksel boyutlarından dolayı CR2032 seçilmiştir.

CR2032 seri pilin başlıca özellikleri aşağıda belirtilmiştir:

- Nominal Voltaj 3 V
- Nominal Kapasite 225 mAh
- Devamlı yük 0.2 mA
- Çap 20 mm

- Yükseklik 3.2 mm olarak ortaya çıkmaktadır.

Sistemin USB bağlantısından ve pil ile optimum bir şekilde çalışabilmesi için gerekli güç elemanları seçiminde doğrusal regülatör ve diyot kullanılmaktadır. Doğrusal regülatör USB bağlantısından gelen 5 V'luk gerilimi 3.3 V seviyesine gürültüsüz bir şekilde çekilmesini sağlayacaktır. Seçilen doğrusal regülatörün akım ve voltaj değerlerinin yanı sıra istenilen sıcaklık değerlerinde çalışması gereklidir. Regülatörün çıkışında pilden herhangi bir akımın USB bağlantısına akışının olmamasından dolayı diyot konulması bu sistem için yeterli olacaktır. Bu diyotun gerilim düşümünün düşük olması ve gerekli akım değerlerinde çalışabilmesi gerekmektedir. Aynı şekilde pil beslemesinin önüne bir diyot konulması gerekmektedir. Böylece USB bağlantısı sağlandıktan sonra pile bir akış olmayacak ve pil korunmuş olacaktır.

Bu sistem için seçilen regülatör AP2114H-3.3TRG1 dir. Bu regülatörün özellikleri aşağıda belirtilmiştir.

- Giriş voltajı maksimum 6.5 V
- Çıkış voltajı 3.3 V
- Çıkış akımı 1 A
- Çıkış voltaj doğruluğu ±1.5 %
- Voltaj düşümü 450 mV

Bu regülatörün giriş ve çıkışına birer adet 4.7 µF'lık kapasitörler bağlanılarak kararlı olarak çalışması sağlanmaktadır.

Seçilen diyot 10BQ030 dur. Bu seçim yapılırken voltaj düşümü etkileyici bir faktör olarak ortaya çıkmaktadır. Bu diyotun bazı özellikleri aşağıda belirtilmektedir.

- Maksimum voltaj 30 V
- Maksimum akım 1 A
- Gerilim düşümü 0.42 V
- Maksimum ters akım 1 mA

3.1.4. Hafıza

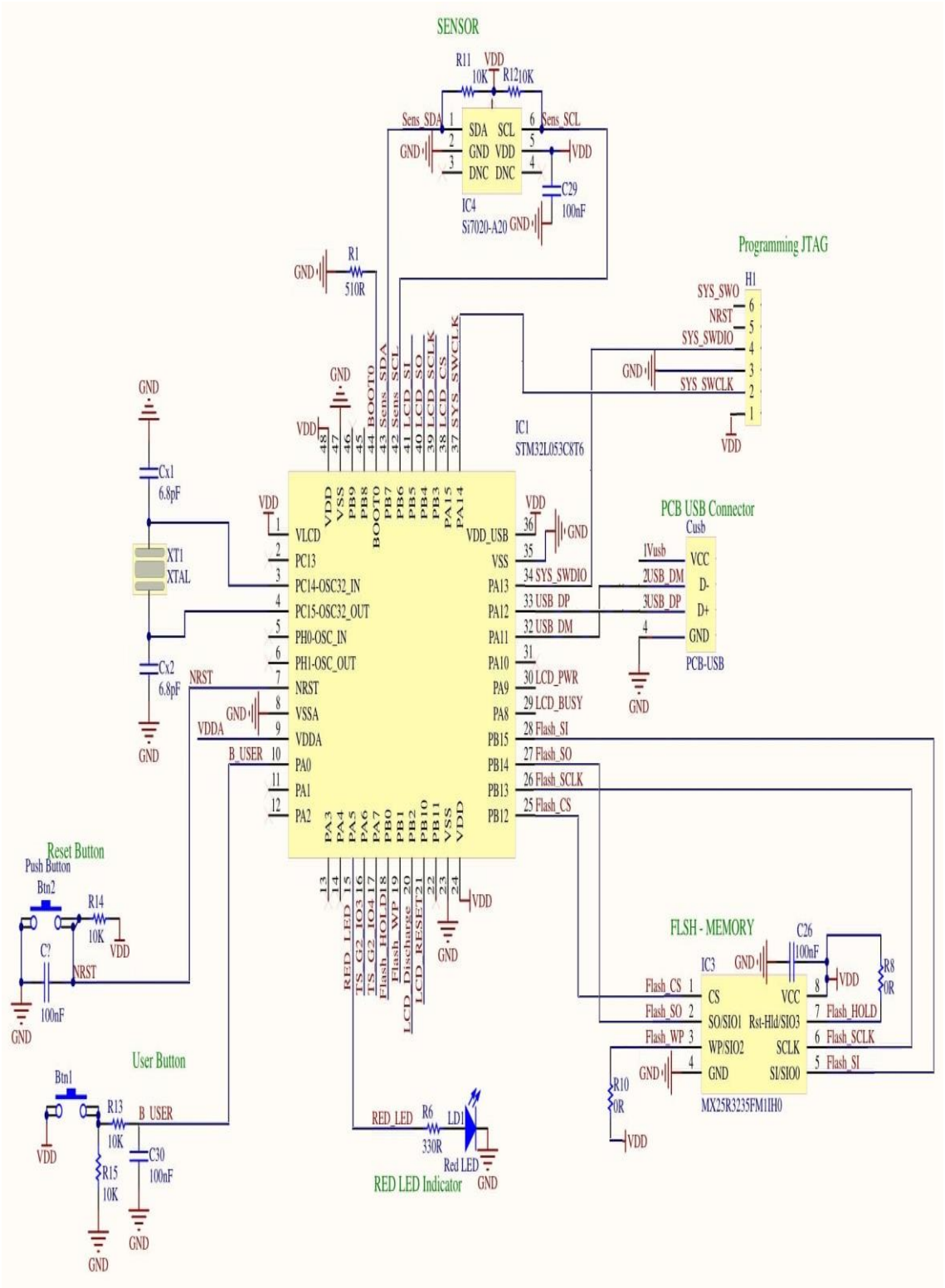
Sensörlerden alınan verilerin güvenli bir şekilde saklanması için gerekli kapasiteye ve koruma faktörlerine sahip bir hafıza biriminin kullanılması gerekmektedir. Diğer taraftan diğer komponentlerde olduğu gibi ölçüm yapılacak olan sıcaklık aralığında optimum şekilde çalışması gerekir. Tasarlanan sistemin bütün bileşenleri gibi bu bileşeninde düşük güç tüketimine sahip olması gerekmektedir. Sistem kurgulanan gerilim değerleri arasında çalışmalıdır.

İstenilenler doğrultusunda seçilen FLASH hafıza Macronix firmasının MX25R3235F modelidir. Bu ürünle ilgili özellikler aşağıda belirtilmektedir.

- 32 M-BIT depolama alanı
- 1.65 V ile 3.6 V gerilimler arasında çalışma
- Minimum 100000 yazma silme garantisi
- 20 yıl veri saklama özelliği
- -40 °C ile +85 °C arasında çalışabilme özelliği

3.2. Donanım Tasarımı

Tasarlanan sistemin devre şeması Ek-1'de görülebilir. Bu tasarımlar gerçekleştirilirken Altium Designer devre tasarım programı kullanılmıştır. Şekil 3.6'da bulunan mikrodenetleyici bağlantı şemasına bakıldığında USB konnektörü, programlama bağlantısı, pil bağlantısı, voltaj regülatörü bağlantıları, uyarı led bağlantısı, FLASH hafıza bağlantısı, kullanıcı butonu bağlantısı ve sensör bağlantılarıyla beraber yan elemanların bağlantıları görülmektedir.

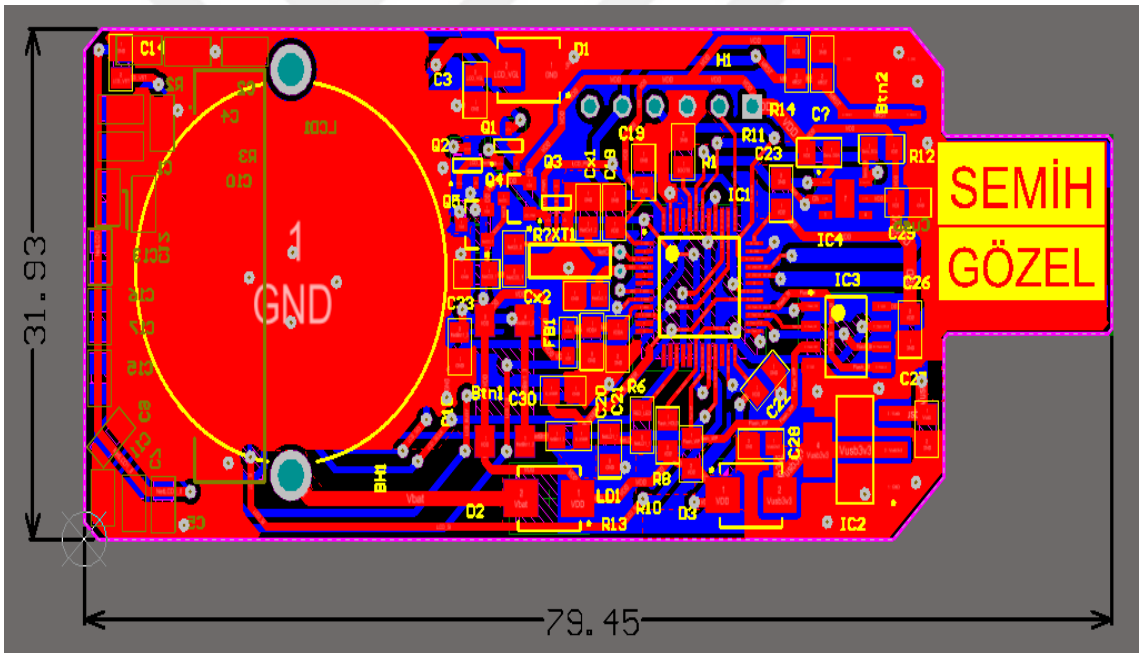


Şekil 3.6. Mikrodenetleyici bağlantı şeması

3.3. Baskılı Devre Kartı Tasarımı

Tasarımda kullanılan baskılı devrenin imal edildiği malzemenin termal özellikleri önemli bir seçim kriteridir. Sistemin düşük sıcaklıklarda çalışabilmesi için uygun bir baskılı devre malzemesi seçilmesi gerekmektedir. Bu nedenle tasarımda FR406 ürünü seçilmiştir.

Sıcaklık veri kayıt cihazının oluşturulmasında ilk olarak şematik tasarım Ek-1'de verildiği gibi yapılmıştır. Ardından bu şematik tasarıma uygun baskı devre tasarımı gerçekleştirilmiştir. Bu tasarımlar Altium Designer devre tasarım programı ile gerçekleştirilmiştir. Tasarlanan PCB nin görseli Şekil 3.7'de görülmektedir.



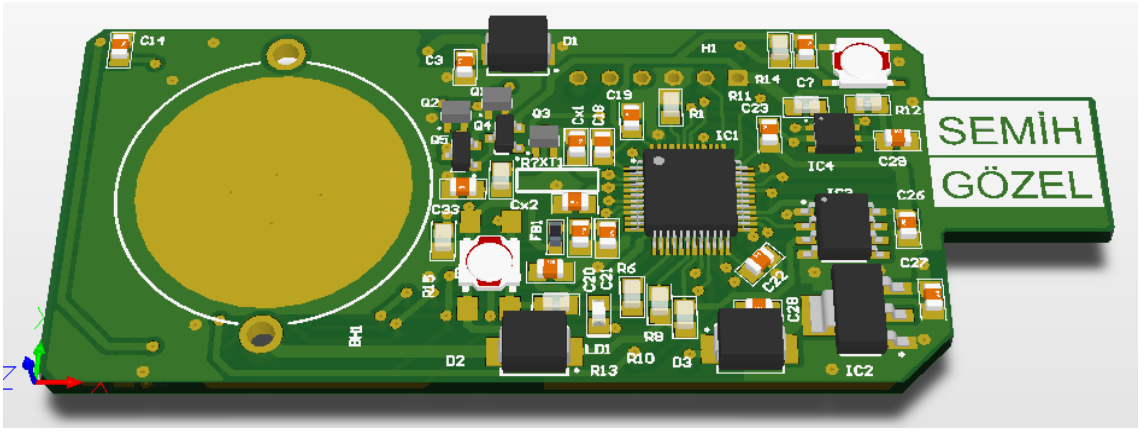
Şekil 3.7. PCB tasarımı

PCB'nin boydan boya uzunlukları 35.73 mm x 91.97 mm dir. PCB'nin üzerinde dahili bir erkek USB bağlantısı bulunmaktadır. PCB'nin enine kesitinin gösterimi (stack up) bilgileri Çizelge 3.2'deki gibidir.

Çizelge 3.2. Board stack report

Stack Up		Layer Stack			
Layer	Board Layer Stack	Name	Material	Thickness	Constant
1		Top Paste			
2		Top Overlay			
3		Top Solder	Solder Resist	0.010mm	3.5
4		Top Layer	Copper	0.036mm	
5		Dielectric 1	FR-4	1.500mm	4.8
6		Bottom Layer	Copper	0.036mm	
7		Bottom Solder	Solder Resist	0.010mm	3.5
8		Bottom Overlay			
9		Bottom Paste			
	Height : 1.591mm				

Bu tabloda verilen bilgilere göre toplam kalınlığı yaklaşık olarak 1.59 mm olan PCB, iki katlı olarak tasarlanmıştır. Dielektrik malzeme FR-4 grubundan seçilmiştir ve kalınlığı 1.5 mm dir. Tasarım yapılırken gerekli su yolu kalınlıkları ve mesafeler, delik çapları gibi bilgiler bu rapordaki değerler göz önünde bulundurularak işlemler gerçekleştirilmiştir. Şekil 3.8'de baskı devre tasarımına ait 3D görünümü verilmiştir.



Şekil 3.8. PCB 3D görünümü

3.4. Test Prosedürleri

Baskılı devre üzerinde kısa devre veya açık devre kontrolü multimetre ile yapılmıştır. Yapılan bu kontrolün ardından baskılı devre üzerinde malzeme dizimi gerçekleştirilmiştir.

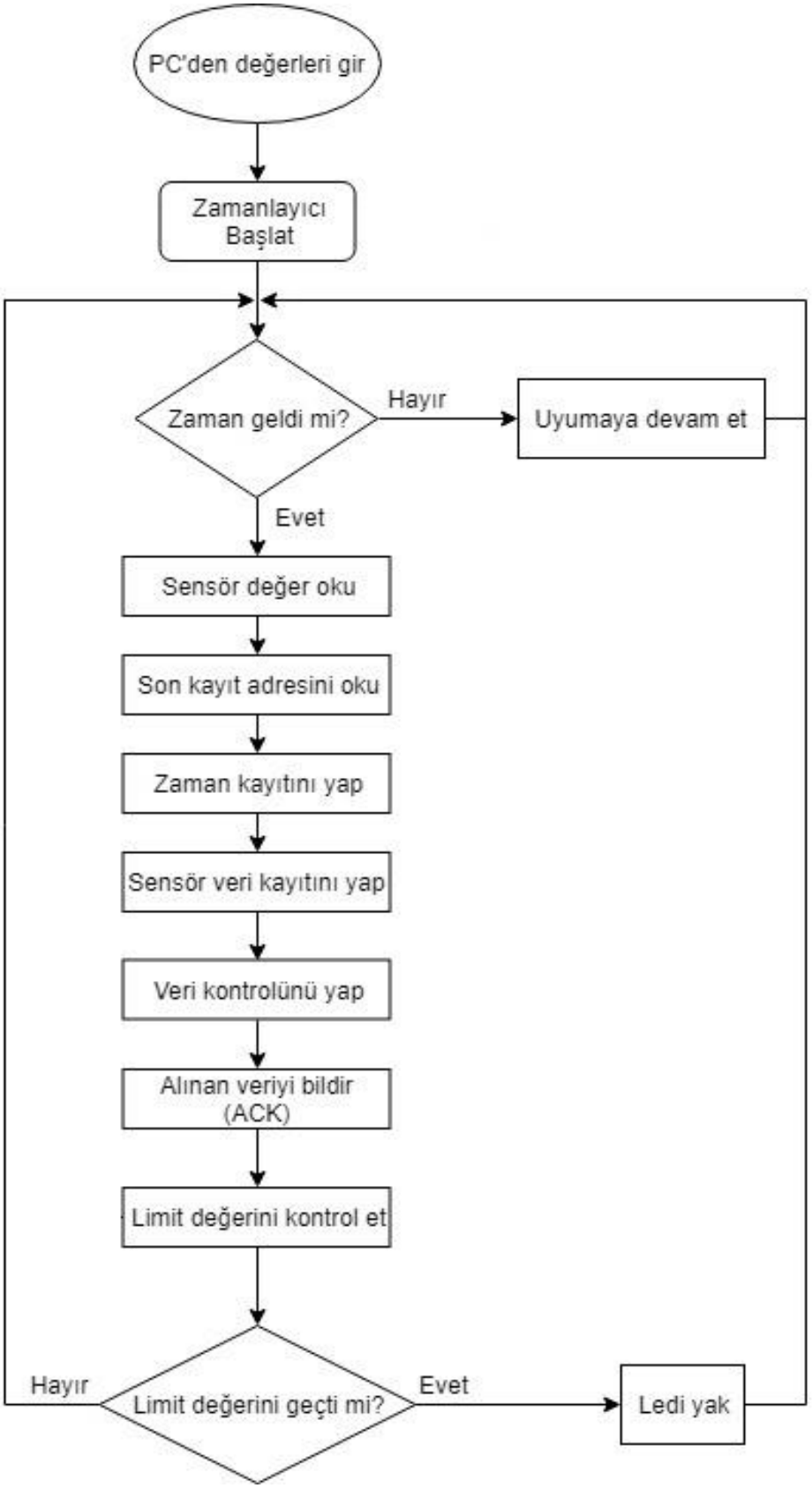
Baskılı devrenin dizgi işlemi bittikten sonra, önce gözle bütün devre elemanlarının doğru bir şekilde dizildiğinin kontrolü gerçekleştirilmiştir. Sonrasında devreye güç verilerek gerilim noktaları multimetre ile kontrol edilmiştir. Eğer gerilim noktalarında istenilen sonuçlar ortaya çıkmamışsa tekrardan kısa devre analizi ve güç hattı analizi yapılması gerekmektedir. Gerilimlerde bir problem olmadığı durumda mikrodenetleyici bilgisayara hata ayıklayıcı (debugger) ile bağlanarak çalışması kontrol edilir. Çalışmasında herhangi bir problem görünmüyorsa geliştirilen gömülü yazılım mikrodenetleyiciye yüklenir ve hata (debug) işlemine başlanır. Sensör ve hafızanın ID'lerinin alınıp alınmadığı kontrol edilir. ID'lerin alındığı bilgisi ile gelen veriler kontrol edilir. Bu kontrolün amacı bileşenlerin uygun bir şekilde baskılı devreye dizildiğinin ve herhangi bir soğuk lehimin olmadığının tespiti içindir. Bu işlemlerin hepsi tamamlandıktan sonra test işlemi tamamlanmış olur. Burada seçilen sensör fabrika kalibrasyonlu olduğundan herhangi bir kalibrasyon yapmaya gerek duyulmaz.

3.5. Bağlantı Konnektörleri

Sistemde bir adet bağlantı konnektörü bulunmaktadır. Bu bağlantı konnektörü sistemin USB bağlantı konnektörüdür. Burada bir konnektör kullanmak yerine, baskılı devre üzerine erkek USB bağlantı şekli yapılmıştır ve baskılı devrenin uygun ölçülerle yapılması sayesinde USB konnektör olarak kullanılabilmektedir.

3.6. Sistem Yazılımı

Cihazın başlatılması ile sıcaklık değerlerini kayıt altına almaya başlamaktadır. Belirtilen sıklıklarda kayıt alan cihaz diğer durumlarda uyku moduna geçmektedir. İstenen limit değerlerini geçmesi sonucu uyarı verip tekrar kayıt almaya devam etmektedir. Geliştirilen gömülü sistem kodlarının genel yapısı Şekil 3.9'daki gibidir.



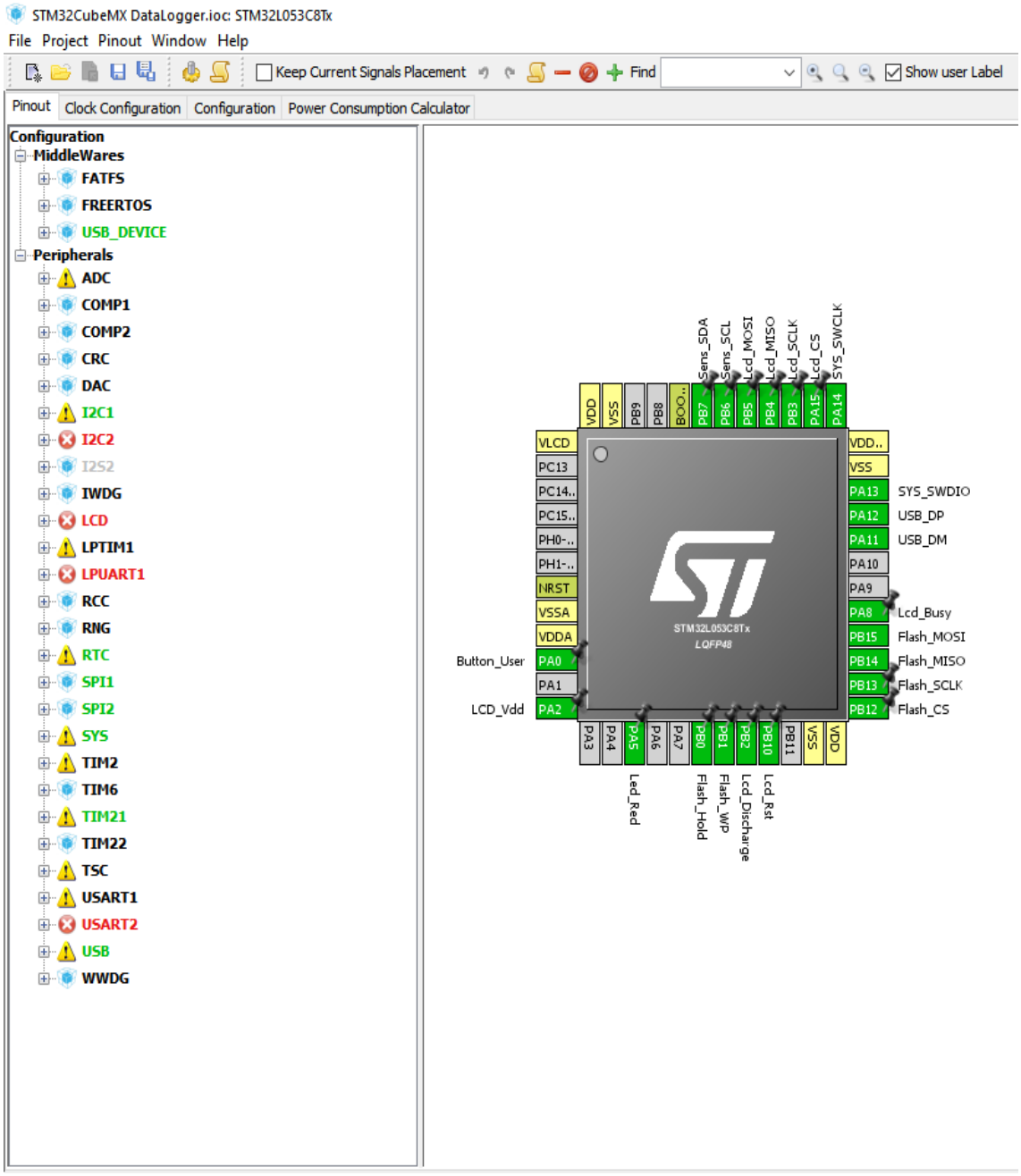
Şekil 3.9. Sıcaklık veri kayıt cihazı akış diyagramı

Sistemin gömülü yazılımı Keil Microvision derleyicisi ile geliştirilmiştir. Mikrodenetleyicinin port, clock ve konfigrasyon alt yapısı ise CubeMX programı ile gerçekleştirilmiştir.

Port ayarları CubeMx de aşağıdaki gibi ayarlanmıştır.

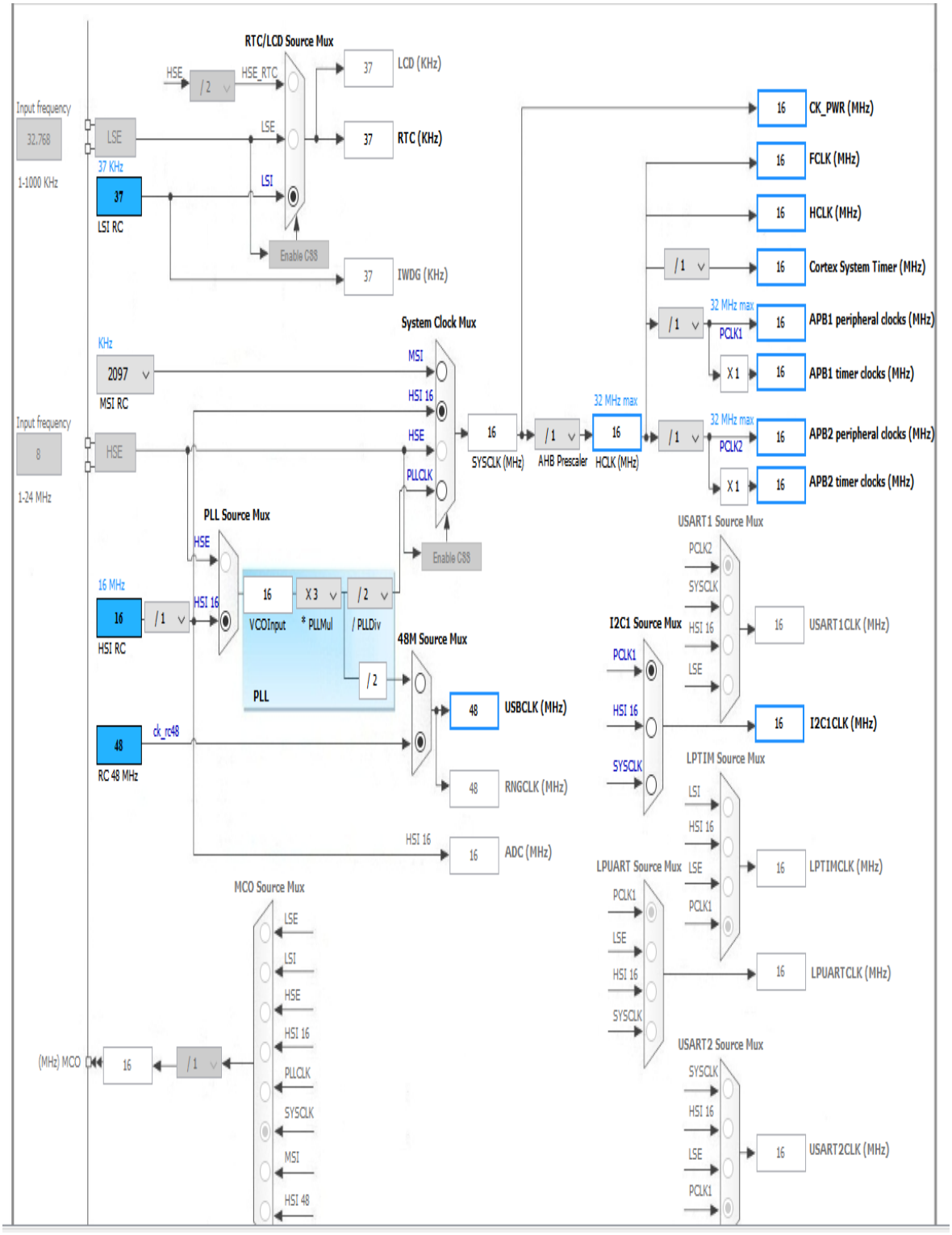
- USB_DEVICE : Communication Device Class
- ADC : not activated
- COMP1 ve COMP2 : not activated
- CRC: not activated
- DAC: not activated
- I2C1 : I2C
- I2C2: not activated
- IWDG: not activated
- LCD: not activated
- LPTIM1: not activated
- LPUART1: not activated
- RCC: all disabled
- RNG : not activated
- RTC: clock source activated
- SPI1: Full dublex Master
- SPI2: Full Dublex Master
- SYS: Debug Serial Wire
- TIM21 : Channel 1: Input Capture Direc mode
- USB: Device FS

Bu ayarlar Şekil 3.10'da görüldüğü gibi yapılmıştır.



Şekil 3.10. MCU port ayarı

Port ayarları Şekil 3.10'da olduğu gibi yapıldıktan sonra saat darbesi ayarları Şekil 3.11 'de gösterildiği gibi yapılır. Burada dahili saat darbesi (internal clock) kullanılmıştır. USB için 48 MHz lik saat darbesi kullanımı gerçekleştirilmiştir.



Şekil 3.11. MCU saat darbesi ayar tablosu

MCU'nun configrasyonları aşağıdaki gibi tanımlanmaktadır.

SPI2: FLASH

Frame Format: Motorola

Data Size : 8 Bits

First Bit: MSB First

Prescalar: 2

CPOL: LOW

CPHA: 1 Edge

Port GPIO Mode: AlternateFunction Push Pull

Port Maximum Output Speed : Very High

I2C: SENSOR

Speed Mode : Fast Mode

Frequency : 300 kHz

Rise Time: 120 ns

Fall Time: 120 ns

Analog Filter : Enabled

Clock No Strech Mode: Disabled

General Call Adress Detection : Disabled

Primary Address Lenght selection: 7-bit

Dual Address Acknowledge: Disabled

Primary Slave Adress: 0x00

USB

Speed : 12 Mbit/s

Endpoint: 64 Bytes

Physical Interface: Internal

RCC

VDD: 3 V

HSI calibration: 16

MSI calibration: 0

RTC

Hour Format: 24

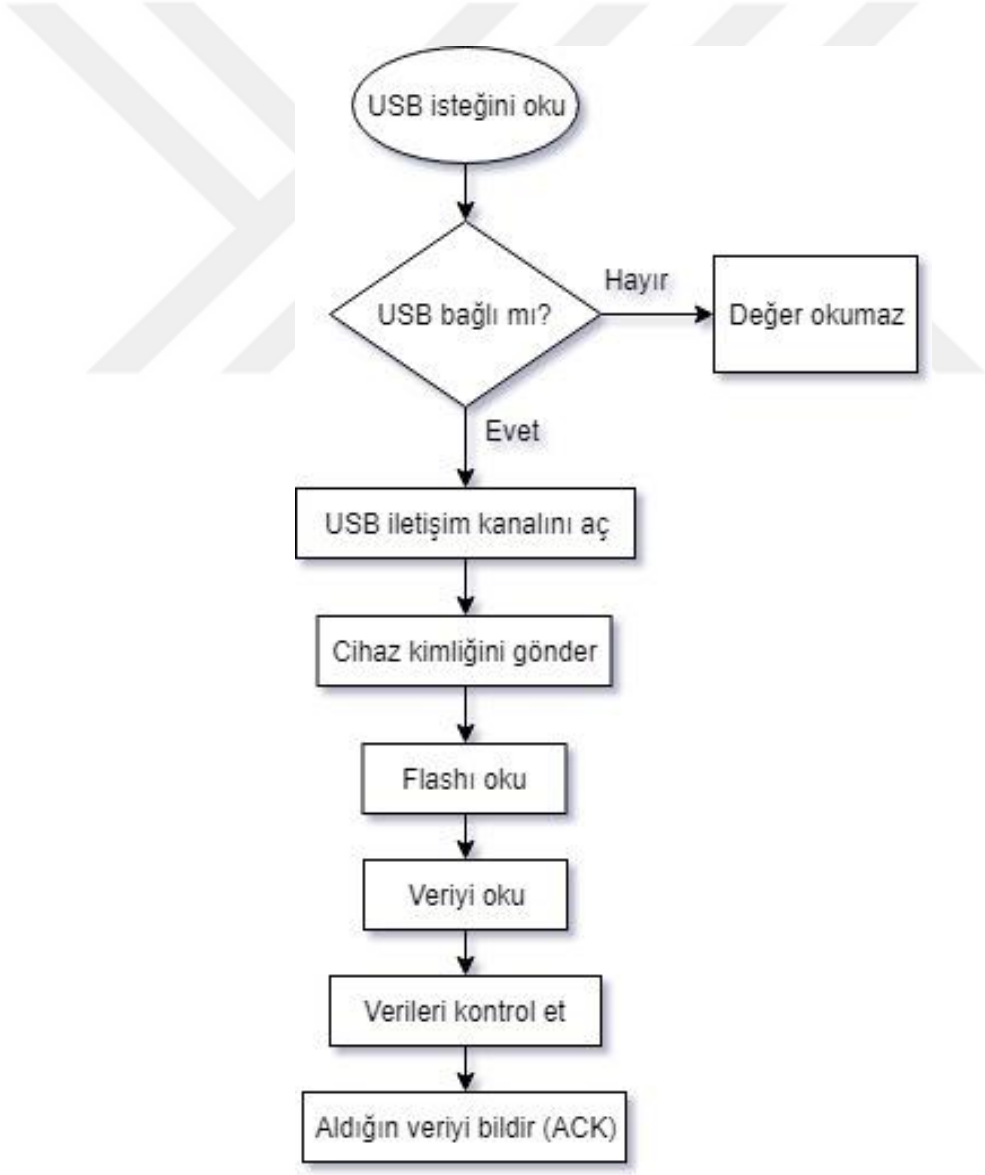
Asynchronous Predivider: 127

Synchronous Predivider: 255

Bu ayarlar yapıldıktan sonra kod oluşturulur ve Keil Microvision da açılır.

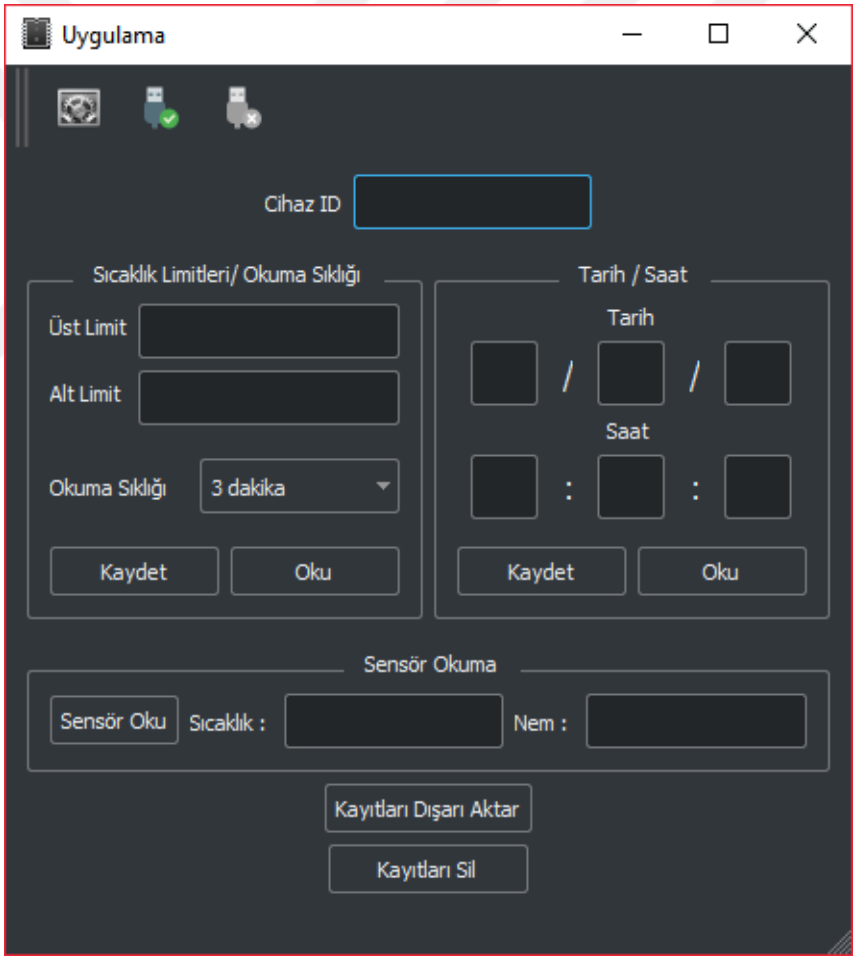
3.7. PC Yazılımı

PC yazılımı ile sistemin veri alma sıklığı, tarih, saat ve sıcaklık limitleri belirlenebilmektedir. Diğer taraftan yapılmış olan kayıtlar bu yazılım sayesinde bilgisayar ortamına aktarılabilmektedir. Bu yazılım bilgisayarın USB portuna bağlı olan cihazı gönderdiği komut ile hangi porta takılı olduğunu bulur ve sonrasında cihaz ile iletişime geçer. Bilgiler bu yazılım sayesinde cihaza kaydedilmektedir. Cihazın kullanım işlemi tamamlandıktan sonra veriler yine bu yazılım sayesinde bilgisayar ortamına alınır. Bu yazılım geliştirilirken Qt kullanılmıştır. Qt'nin kod akış şeması Şekil 3.12'de görülmektedir.



Şekil 3.12. Qt akış diyagramı

Qt, gömülü yazılım için geliştirilmiş önemli bir programdır. Grafik geliştirme araç takımı olan bu program, birden fazla platformu desteklemesi önemli bir özelliğidir. Gelişmiş kütüphanesi sayesinde Grafiksel kullanıcı arayüzü (GUI) bileşenlerinin yanı sıra birçok araç içermektedir. Diğer taraftan Qt'nin C++ kullanması ve bu programlama diliyle komut satırlarının istenildiği gibi yazılması diğer önemli özelliklerinden birisidir. Qt'nin kurulumuyla birçok önemli özellik beraberinde gelmektedir. Qt'nin açık kaynak projeleri için ücretsiz olması ve Excel dosya biçimini kullanması bu çalışma için önemli bir kullanım sebebidir. Kullanım öncesi yapılan ayarların ve verinin bilgisayar ortamına alınmasını sağlayan bu programın arayüzü Şekil 3.13'de görülmektedir.

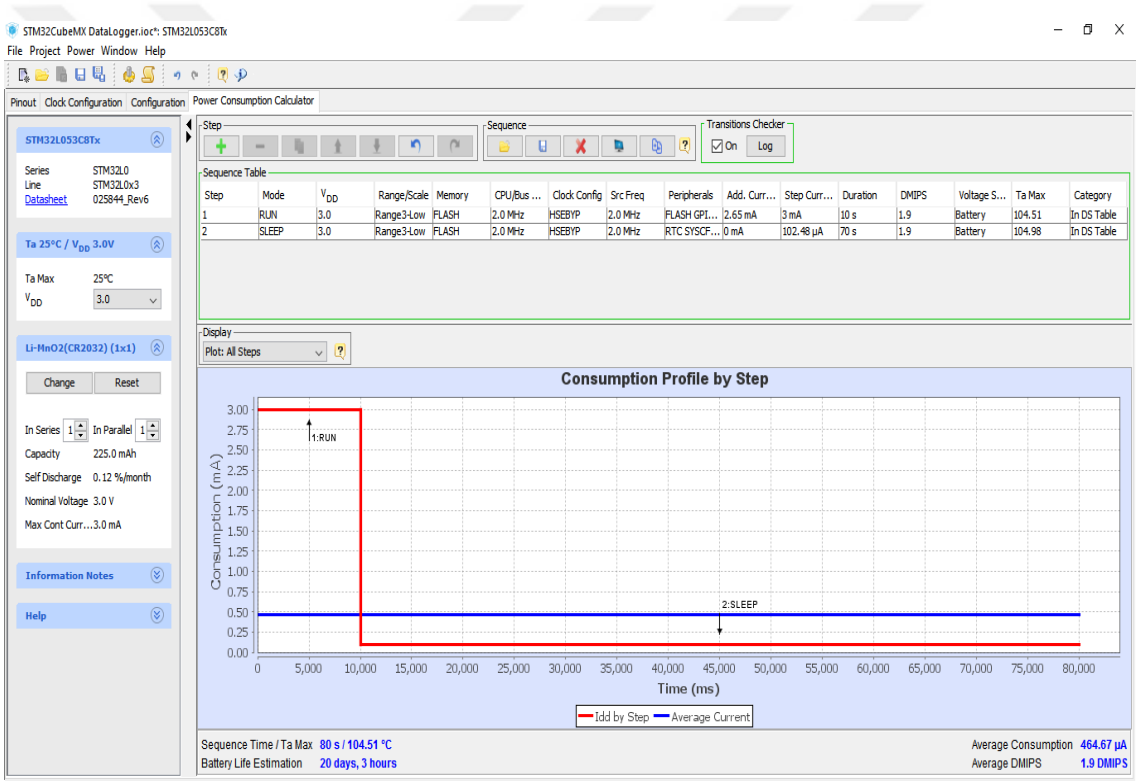


Şekil 3.13. Qt arayüzü görünümü

4. BULGULAR

4.1. Güç Analizi

Sistemin güç analizi mikrodenetleyici firmasının sunmuş olduğu CubeMX aracı ile yapılmıştır. Burada mikrodenetleyicinin çalışma değer bilgileri ile uyku modu bilgileri girilmiştir. Sonuç olarak Şekil 4.1'de görüleceği gibi ortalama akım tüketimi 464,67 µA olan cihazın, kullanmış olduğu batarya 20 gün boyunca sistemin çalışır durumda olmasını sağlamaktadır.

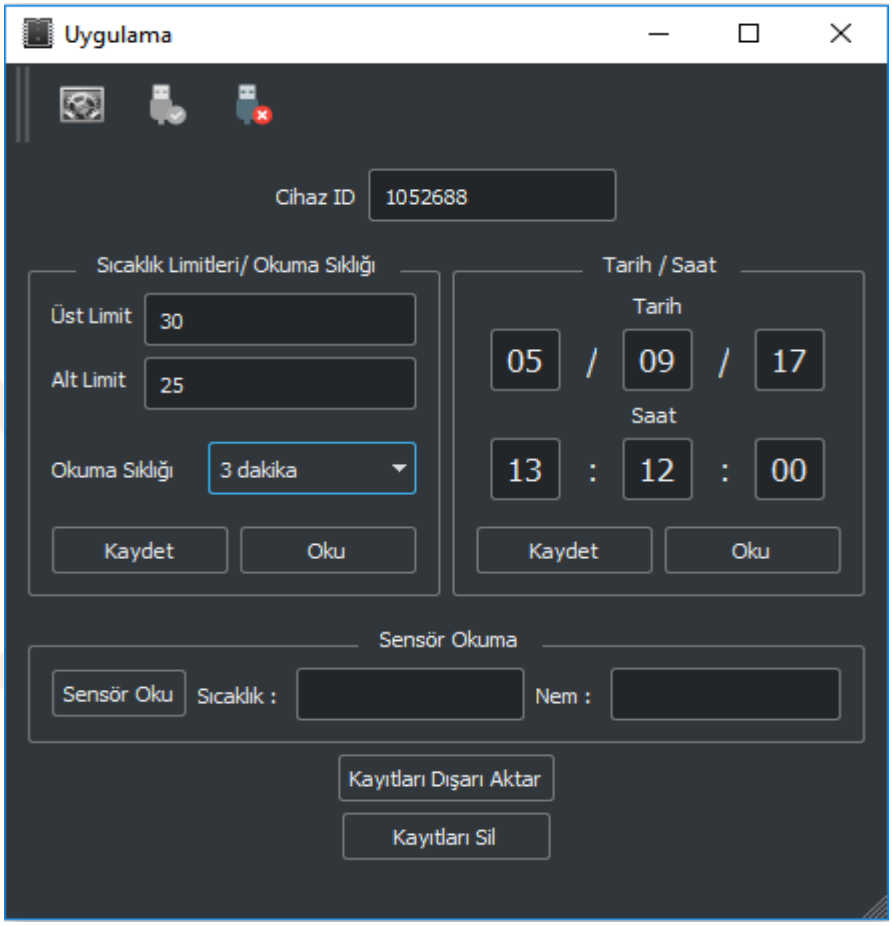


Şekil 4.1. Sistem güç analizi

4.2. Sıcaklık Veri Kayıt Cihazı ile Yapılan Çalışma

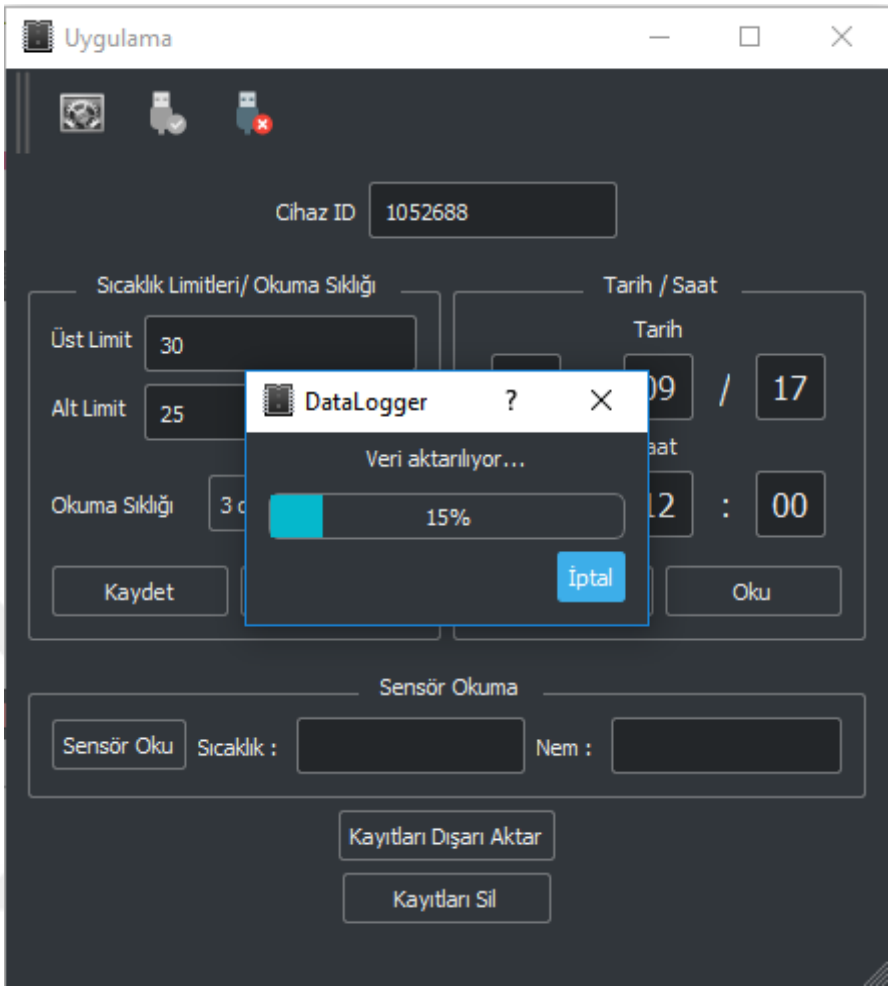
Cihazın doğruluğunu görebilmek için sıcaklık-nem ölçümü yapabilen başka bir cihazla birlikte aynı ortamdan veri alınmıştır. Yapılan örnek çalışmada meyve ve sebzelerin bozulmadan kalabilecekleri bir ortam tercih edilmiştir. İlk olarak cihazımız bilgisayarımızın USB portuna bağlanmıştır. Qt ile oluşturduğumuz arayüz açılmıştır ve

cihaz bilgisayarımıza tanıtılmıştır. Şekil 4.2'de görüleceği gibi arayüzde bulunan saat, tarih, okuma sıklığı aralığı ve üst-alt limit değerleri girilerek cihaza kayıt işlemi gerçekleştirilmiştir.



Şekil 4.2. Değerlerin cihaza tanıtılması

Bu işlem gerçekleştirildikten sonra cihaz bilgisayarın USB portundan çıkartılmıştır ve sıcaklık-nem kaydının alınacağı ortama yerleştirilmiştir. Cihaz çalıştırılmıştır ve veri alma işlemi bitene kadar beklenmiştir. Veri alma işlemi bittikten sonra cihaz üzerinde bulunan pil çıkartılarak kayıt işlemi durdurulabildiği gibi cihaz bilgisayarın USB portuna bağlanılarak da kayıt işlemi durdurulabilmektedir. Bilgisayarımıza bağlanan cihaz tekrardan bilgisayara tanıtılmıştır. Tanıtım işlemi gerçekleştirildikten sonra ''Kayıtları Dışarı Aktar'' butonuna tıklayarak Şekil 4.3'de görüldüğü gibi alınmış olunan veriler cihazdan bilgisayara aktarılmıştır.



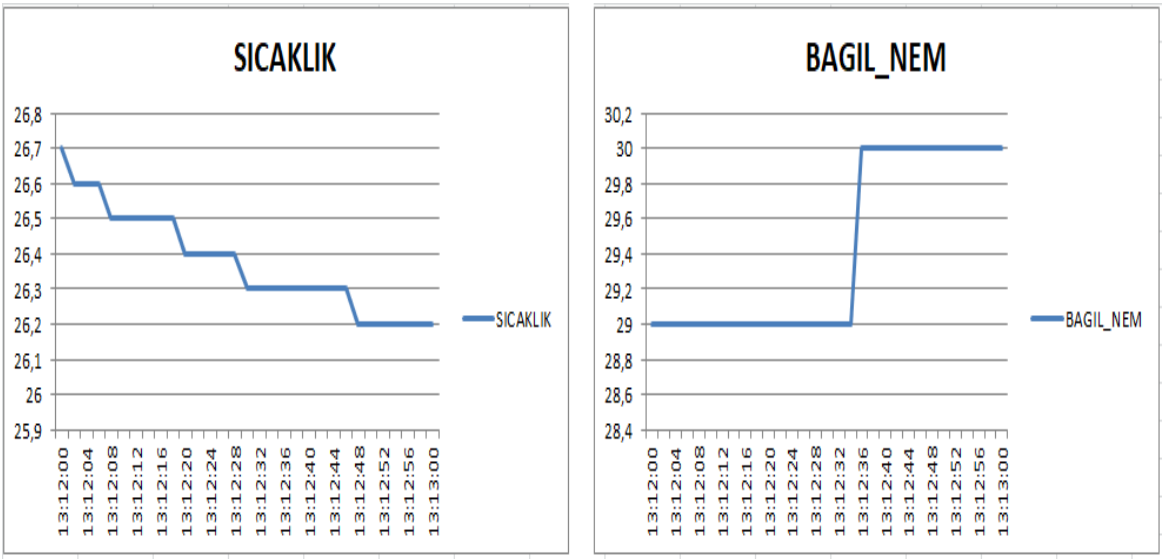
Şekil 4.3. Kaydedilen değerlerin bilgisayara aktarılması

Veri aktarım işlemi tamamlandıktan sonra Excel dosyası olarak indirilmiş dosya Şekil 4.4'de görüldüğü gibi açılır ve burada alınan veriler tarih, saat, sıcaklık ve nem başlıkları altında alındığı görülmektedir. Şekil 4.5'de ise alınan bu verilerin grafiksel görünümü yer almaktadır. Ayrıca Şekil 4.6'da başka bir cihazla aynı ortamdan alınmış sıcaklık-nem verileri ve Şekil 4.7'de ise grafiksel görünümü yer almaktadır. Böylelikle cihazın doğruluğuda test edilmiştir.

R38

	A	B	C	D	E	F
1	KAYIT_NO	TARIH	SAAT	SICAKLIK	BAGIL_NEM	
2	1	5.09.2017	13:12:00	26,7	29	
3	2	5.09.2017	13:12:02	26,6	29	
4	3	5.09.2017	13:12:04	26,6	29	
5	4	5.09.2017	13:12:06	26,6	29	
6	5	5.09.2017	13:12:08	26,5	29	
7	6	5.09.2017	13:12:10	26,5	29	
8	7	5.09.2017	13:12:12	26,5	29	
9	8	5.09.2017	13:12:14	26,5	29	
10	9	5.09.2017	13:12:16	26,5	29	
11	10	5.09.2017	13:12:18	26,5	29	
12	11	5.09.2017	13:12:20	26,4	29	
13	12	5.09.2017	13:12:22	26,4	29	
14	13	5.09.2017	13:12:24	26,4	29	
15	14	5.09.2017	13:12:26	26,4	29	
16	15	5.09.2017	13:12:28	26,4	29	
17	16	5.09.2017	13:12:30	26,3	29	
18	17	5.09.2017	13:12:32	26,3	29	
19	18	5.09.2017	13:12:34	26,3	29	
20	19	5.09.2017	13:12:36	26,3	30	
21	20	5.09.2017	13:12:38	26,3	30	
22	21	5.09.2017	13:12:40	26,3	30	
23	22	5.09.2017	13:12:42	26,3	30	
24	23	5.09.2017	13:12:44	26,3	30	
25	24	5.09.2017	13:12:46	26,3	30	
26	25	5.09.2017	13:12:48	26,2	30	
27	26	5.09.2017	13:12:50	26,2	30	
28	27	5.09.2017	13:12:52	26,2	30	
29	28	5.09.2017	13:12:54	26,2	30	
30	29	5.09.2017	13:12:56	26,2	30	
31	30	5.09.2017	13:12:58	26,2	30	

Şekil 4.4. Sıcaklık veri kayıt cihazından alınan veri

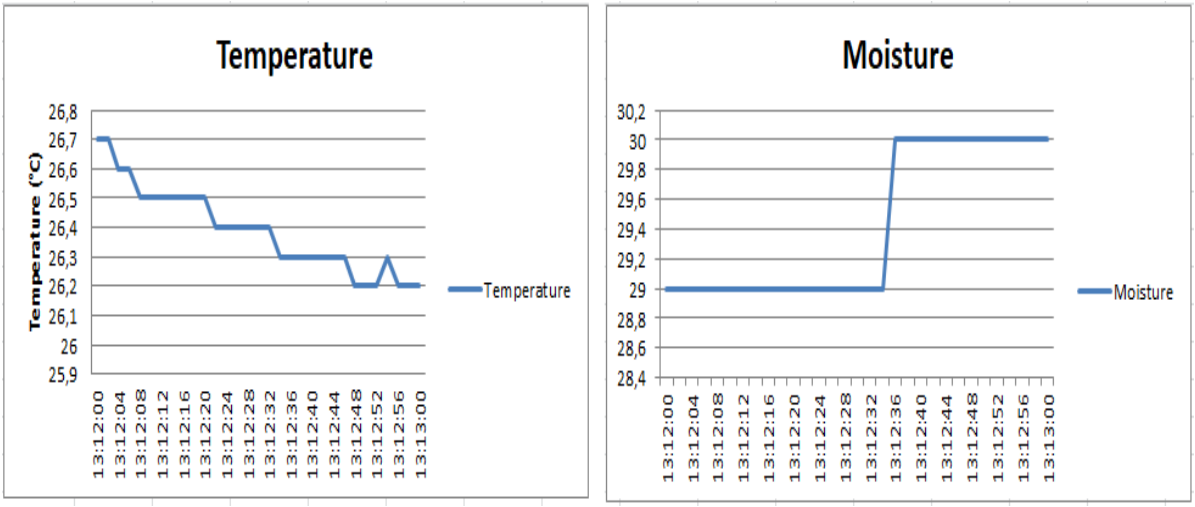


Şekil 4.5. Alınan verilerin grafiksel görünümü

M44

	A	B	C	D	E	F	G	H
1	Place	Data	Time	Moisture	Unit	Temperature	Unit	
2	1	05.09.17	13:12:00	29	%RH	26,7	Degree C	
3	2	05.09.17	13:12:02	29	%RH	26,7	Degree C	
4	3	05.09.17	13:12:04	29	%RH	26,6	Degree C	
5	4	05.09.17	13:12:06	29	%RH	26,6	Degree C	
6	5	05.09.17	13:12:08	29	%RH	26,5	Degree C	
7	6	05.09.17	13:12:10	29	%RH	26,5	Degree C	
8	7	05.09.17	13:12:12	29	%RH	26,5	Degree C	
9	8	05.09.17	13:12:14	29	%RH	26,5	Degree C	
10	9	05.09.17	13:12:16	29	%RH	26,5	Degree C	
11	10	05.09.17	13:12:18	29	%RH	26,5	Degree C	
12	11	05.09.17	13:12:20	29	%RH	26,5	Degree C	
13	12	05.09.17	13:12:22	29	%RH	26,4	Degree C	
14	13	05.09.17	13:12:24	29	%RH	26,4	Degree C	
15	14	05.09.17	13:12:26	29	%RH	26,4	Degree C	
16	15	05.09.17	13:12:28	29	%RH	26,4	Degree C	
17	16	05.09.17	13:12:30	29	%RH	26,4	Degree C	
18	17	05.09.17	13:12:32	29	%RH	26,4	Degree C	
19	18	05.09.17	13:12:34	29	%RH	26,3	Degree C	
20	19	05.09.17	13:12:36	30	%RH	26,3	Degree C	
21	20	05.09.17	13:12:38	30	%RH	26,3	Degree C	
22	21	05.09.17	13:12:40	30	%RH	26,3	Degree C	
23	22	05.09.17	13:12:42	30	%RH	26,3	Degree C	
24	23	05.09.17	13:12:44	30	%RH	26,3	Degree C	
25	24	05.09.17	13:12:46	30	%RH	26,3	Degree C	
26	25	05.09.17	13:12:48	30	%RH	26,2	Degree C	
27	26	05.09.17	13:12:50	30	%RH	26,2	Degree C	
28	27	05.09.17	13:12:52	30	%RH	26,2	Degree C	
29	28	05.09.17	13:12:54	30	%RH	26,3	Degree C	
30	29	05.09.17	13:12:56	30	%RH	26,2	Degree C	
31	30	05.09.17	13:12:58	30	%RH	26,2	Degree C	

Şekil 4.6. Bir başka cihazdan alınan veri



Şekil 4.7. Bir başka cihazdan alınan veri grafiği

5. SONUÇ

Yapılan tez çalışmasında USB arayüzlü cihazın tasarımı ve uygulaması gerçekleştirilmiştir. Tasarımda Altium Designer devre tasarım programı kullanılmıştır. Tasarlanan baskılı devre kartı ile sistem 20 gün boyunca veri kaydedebildiği bulgular bölümünde de görülebilmektedir. Ayrıca minimum düzeyde enerji harcanması için en uygun mikrodenetleyici olan STM32 tercih edilmiş, bu sayede enerji etkin bir şekilde kullanılmıştır.

İlerisi için ticarileşme göz önünde tutularak baskılı devre kartı olabildiğince küçük ve ince olması hesaplanmıştır. Burada; baskılı devre kartı kalınlığı 0,4 mm, elektronik malzemeler ile birlikte kalınlık 0,7 mm olarak tasarlanmıştır. Mekanik kasa tasarımının ise 0,3 mm olacağı düşünülmüş ve cihazın toplam kalınlığı 1 cm olarak hesaplanmıştır. Ayrıca cihazın önemli özelliklerinden birisi de USB arayüzü baskılı devre kartı üzerinde olması sağlanmış ve bu sayede harici bir haberleşme kablosuna ihtiyaç ortadan kaldırılmıştır.

Yapılan çalışmada Qt programı ile kullanıcıların rahatlıkla kullanabileceği PC arayüz tasarımı gerçekleştirilmiştir. Bu arayüz sayesinde mikrodenetleyiciye tarih, saat, okuma sıklığı gibi ayarlar tanıtılmış, alınan bu değerlerin PC'ye yüklenmesi sağlanmıştır. Bu programı kullanımdaki en önemli nedenlerden birisi ise Excel dosyası biçimini kullanıyor olabilmesidir.

Soğuk zincir için tasarımı yapılan sistemden alınan veriler ayrı bir sıcaklık ve nem kayıt cihazıyla karşılaştırılmış, sensörlerin birbirleriyle aynı değer okuduğu görülmüştür. Dolayısıyla sistemin doğruluğu kanıtlanmıştır.

Tasarımı yapılan sisteme alternatif olarak geliştirilebilecek sistemler için seçilebilecek devre elemanları olarak en başta mikrodenetleyici gelmektedir. Harici bir hafızaya ihtiyaç duymadan daha büyük hafızalı bir mikrodenetleyici seçimi yapılabilir, fakat bu durumda sistem güç tüketimi artmaktadır. Sisteme küresel mobil iletişim sistemi (GSM) üzerinden uzaktan bağlantı kurularak da anlık veri alınması sağlanabilir.

6. KAYNAKLAR

Aksoy, T., 2012. Tokat İl Merkezindeki Eczanelerde Soğuk Zincir Uygulamalarının İncelenmesi. (Yüksek Lisans Tezi), Halk Sağlığı Ana Bilim Dalı, Tokat.

Aksu, H., 2010. Gıda Perakendeciliği ve Lojistik. http://cdn.istanbul.edu.tr/FileHandler2.ashx?f=gıda-perakendeciliği-lojistik.pdf-(02.12.2017).

Alanur, H., 2014. Soğuk Zincir Lojistik Yönetiminde Dış Kaynak Kullanımının İşletme Performansı Üzerine Etkisi: Gıda Tedarik Zincirine Yönelik Bir Alan Çalışması. (Yüksek Lisans Tezi),Dokuz Eylül Üniversitesi. Denizcilik İşletmeleri Yönetimi Anabilim Dalı, İzmir.

Anderson, D., 2001. Usb System Architecture. Addıson-Wesley Developer's Press, 506, United States of America.

Anonim, 2014. Resmi Gazete (18 Ekim 2014), Sayı No: 29149.

Anonim, 2015. Ultra-low-power STM32L0x3 advanced ARM based 32-bit Mcus Online Dökümanı. https://studio.segger.com/packages/STM32L0xx/CMSIS/Documents/DM00095744.pdf - (05.12.2017).

Anonim, 2016. Nem Sıcaklık Kayıt Cihazları ve Daha Fazlası: Datalogger Saveris. Dijital Dergi. http://www.otomasyondergisi.com.tr/arsiv/yazi/nem-sicaklik-kayit-cihazlari-ve-daha-fazlasi-datalogger-saveris, - (25.11.2017).

Anonim, 2017.Arm Mimarisi ve Uygulamaları. http://www.ktu.edu.tr/dosyalar/bilgisayar_a7670.pdf-(05.12.2017).

Anonim, 2017. Evd Seti Deney Föyü. http://gazi.edu.tr/posts/download?id=160661 – (02.12.2017).

Anonim, 2017. Soğuk Zincir Hakkında Detaylı Bilgi. https://www.zeo.org.tr/rehber-40 - (25.11.2017).

Atlatır, T., 2012. Soğuk Zincir Ürünlerinin Sıcaklıklarının Kontrolü ve Doğrulaması. Lojistik, http://www.lojistikdunyasi.com/soguk-zincir-urunlerinin-sicakliklarinin-kontrolu-ve-dogrulanmasi.

Buldu, A., 2003. Mikrodenetleyicili Usb Eğitimi Sistemi. (Doktora Tezi), Marmara Üniversitesi. Elektronik-Bilgisayar Eğitimi Anabilim Dalı, İstanbul.

Cebeci, H. Y., 2014. Düşük Maliyetli, Bilgisayar Tabanlı Data Logger Özellikli Osiloskop Tasarımı ve Gerçekleştirilmesi. (Yüksek Lisans Tezi), Gazi Üniversitesi. Bilgisayar Eğitimi Anabilim Dalı, Ankara.

Çiçekdeş, C., 2011. Arm Microcontroller Based Wireless Data Logger. (Yüksek Lisans Tezi), Dokuz Eylul Universıty. Electrical Electronics Engineering Program, İzmir.

Çimenli, O., 2013. İnternet Erişimli Deney Modüllerinin Usb Tabanlı Kontrolü. (Yüksek Lisans Tezi), Süleyman Demirel Üniversitesi. Elektronik-Bilgisayar Eğitimi Anabilim Dalı, Isparta.

Dinçbakır, C., 2002. Usb Arayüz İçin Fiziksel Katman Tasarımı. (Yüksek Lisans Tezi), Yıldız Teknik Üniversitesi. Elektronik ve Haberleşme Mühendisliği Anabilim Dalı, İstanbul.

Durak, M. G., Ünverdi, İ., 2014. Dondurulmuş Gıda Lojistiğinde Maliyet Bilgisinin Kullanımı. Business and Economics Research Journal., vol. 5, no. 4, pp. 19-41.

Göl, G,. 2015. Arm Nedir?. http://www.gokhangol.com/arm/arm-nedir/ - (01.12.2017).

İzer, D., 2017. Soğuk Zincir Lojistiği İçinde Risklerin Azaltılmasında Yeni Teknolojiler. 6. Ulusal Lojistik ve Tedarik Zinciri Kongresi, Antalya.

Kaplan, H., 2015. STM32F407 I2C Kullanımı. http://hakkikaplan.blogspot.com.tr/p/cogu-zaman-sensorler-mikroislemcilerden.html - (02.12.2017).

Kiremitci, A. F., 2007. PIC18F4550 Mikrodenetleyicisi İle Usb-Pc Veri Aktarım Arabirimi Gerçeklenmesi. (Yüksek Lisans Tezi), Atatürk Üniversitesi. Elektrik ve Elektronik Mühendisliği Anabilim Dalı, Erzurum.

Özerden, 2015. http://zkm.tarim.gov.tr/mersin/Belgeler/Haberler/Tar%C4%B1m%20Dan%C4%B1%20E%C4%9Fitimi%20%20Mart%202015/So%C4%9Fuk%20%C4%B0%C5%9F1em%20-Uygulamalar%C4%B1_3.ppt - (25.11.2017).

Sarısoy, G., 2011. Gıdaların Soğuk Zincir Lojistiği. (Yüksek Lisans Tezi), Bahçeşehir Üniversitesi. Tedarik Zinciri ve Lojistik Yönetimi Bölümü, İstanbul.

St Microelectronics, 2014. Ultra-low-power 32-bit Mcu Arm Based Cortex M0+, up to 64 Kb Flash, 8 Kb Sram, 2 Kb Eeprom, Lcd, Usb, Adc, Dac Online Dökümanı, http://www.st.com/content/ccc/resource/technical/document/datasheet/8a/f4/9d/d7/61/1b/46/b4/DM00105960.pdf/files/DM00105960.pdf/jcr:content/translations/en.DM00105960.pdf - (05.12.2017).

Sural, R., 2010. I2C Haberleşme Protokolü. http://ramazansural.blogspot.com.tr/2010/11/i2c-haberlesme-protokolu.html - (17.11.2017)

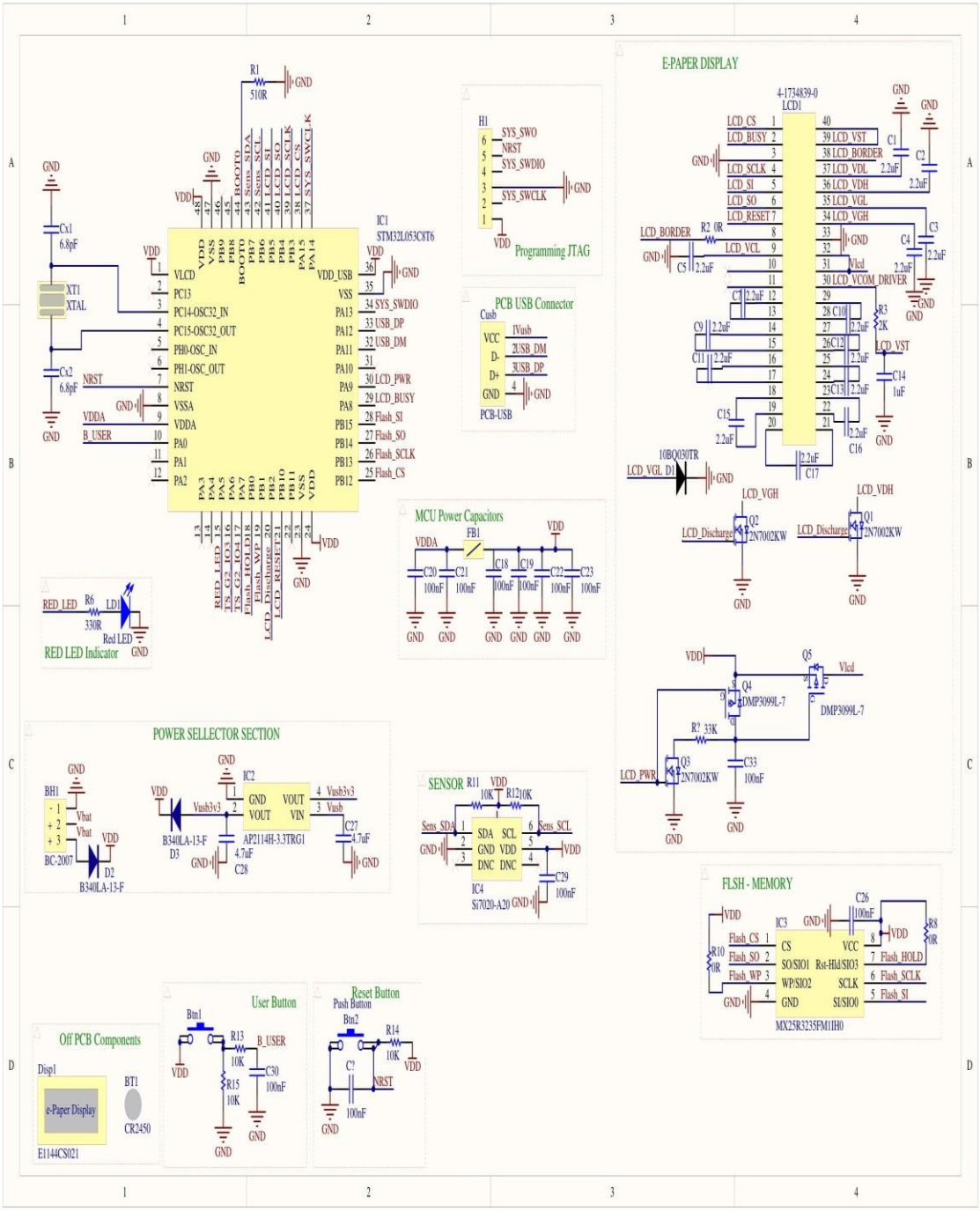
Türk, Ö., 2011. Arm İşlemciler İle Tek Kullanımlık Şifre Uygulamasının Gerçekleştirilmesi. (Yüksek Lisans Tezi), Fırat Üniversitesi. Bilgisayar Mühendisliği, Elazığ.

Türk ve ark, 2015. Türk, R., Yıldırım, I., İkat, D., 2015. Meyve ve Sebzelerin Muhafazasında Soğuk Depoların Kalite ve Kantiteye Etkileri. Tesisat Mühendisliği Dergisi, 1 (148), 75-81.

Yiu, J,. 2016. An overview of the ARM Cortex-M processor family and comparison. https://community.arm.com/cfs-file/__key/telligent-evolution-components-attachments/01-2142-00-00-00-00-52-96/White-Paper-_2D00_-Cortex_2D00_M-for-Beginners-_2D00_-2016-_2800_final-v3_2900_.pdf - (18.11.2017).

7. EKLER

7.1. Ek-1. Elektronik Devre Şeması



7.2. Ek-2. Elektronik Devre Elemanları Listesi

Komponent Listesi

Source Data From: SemihGozel.PrjPcb
Project: SemihGozel.PrjPcb
Variant: None

Creation Date: 4/7/2017
Print Date: 12-May-17

Comment	Manufacturer	MPN	Designator	Description	Quantity
CR2032 Holder	Memory Protection Devices	BK-915-TR	BH	Coin Cell Battery Holder	1
CR2032	Panasonic	CR2032	BT1	Coin Cell Battery	1
Push Button	E-Switch	TL3315NF100Q	Btn1, Btn2	Push Button	2
Cap.Cer.2u2F.25V.%1.Y5V	Samsung	CL21F225ZAFNNNE	C1, C2, C3, C4, C5, C7, C9, C10, C11, C12, C13, C15, C16, C17	Ceramic Capacitor	14
Cap.Cer.1uF.25V.%10.Y5V	Yageo	CC0805ZRY5V8BB105	C14	Ceramic Capacitor	1
Cap.Cer.100nF.50V.%10.X7R	AVX	08055C104KAT2A	C18, C19, C20, C21, C22, C23, C26, C29, C30, C31	Ceramic Capacitor	10
Cap.Cer.4u7F.10V.%10.X5R	Samsung	CL21A475KPFNNNG	C25, C27, C28	Ceramic Capacitor	3
10BQ030TR	SMC Diode Solutions	10BQ030TR	D1, D2, D3, D4	Schotky Diode	4
E1144CS021	Pervasive Displays	E1144CS021	Disp1	e-Paper Display	1
Bead.600Ohm.1A.0603	Bourns	MH1608-601Y	FB1	Ferrite Bead	1
STM32L053C8T6	ST	STM32L053C8T6	IC1	MCU	1
AP2114H-3.3TRG1	Diodes Incorporated	AP2114H-3.3TRG1	IC2	3V3 LDO 1A	1
MX25R3235FM1IH0	Macronics	MX25R3235FM1IH0	IC3	Flash Memory	1
Si7020-A20	SiLabs	Si7020-A20-GM1	IC4	Temperature Humidity Sensor	1
4-1734839-0	TE Connectivity	4-1734839-0	LCD1	E-paper Display	1
Red LED	OSRAM	LS R976-NR-1	LD1	SMD LED	1
2N7002KW	Fairchild/ON Semiconductor	2N7002KW	Q1, Q2	N Channel Mosfet	2
Res.510R.%1.1/8W	Yageo	RC0805FR-07510RL	R1	Chip Resistor	1
Res.0R.%1.1/8W	Yageo	RC0805JR-070RL	R2	Chip Resistor	1
Res.2K.%1.1/8W	Yageo	RC0805FR-072KL	R3	Chip Resistor	1
Res.10K.%1.1/8W	Yageo	RC0805FR-0710KL	R5, R7, R11, R12, R13, R14, R15, R16	Chip Resistor	8
Res.1K.%1.1/8W	Yageo	RC0805FR-071KL	R6, R9	Chip Resistor	2
					59

7.3. Ek-3. Sistem Yazılımı

Main.c

```
#include "main.h"
#include "stm32l0xx_hal.h"
#include "usb_device.h"
#include "application.h"
ADC_HandleTypeDef hadc;
DMA_HandleTypeDef hdma_adc;
I2C_HandleTypeDef hi2c1;
DMA_HandleTypeDef hdma_i2c1_rx;
RTC_HandleTypeDef hrtc;
SPI_HandleTypeDef hspi1;
SPI_HandleTypeDef hspi2;
DMA_HandleTypeDef hdma_spi2_rx;
TIM_HandleTypeDef htim21;
void SystemClock_Config(void);
void Error_Handler(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_SPI2_Init(void);
static void MX_SPI1_Init(void);
static void MX_I2C1_Init(void);
static void MX_TIM21_Init(void);
static void MX_ADC_Init(void);
static void MX_RTC_Init(void);
```

```
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_SPI2_Init();
  MX_USB_DEVICE_Init();
  MX_SPI1_Init();
  MX_I2C1_Init();
  MX_TIM21_Init();
  MX_ADC_Init();
  MX_RTC_Init();
  InitApplication();
  while (1)
  {
     RunApplication();
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  RCC_PeriphCLKInitTypeDef PeriphClkInit;
__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
  RCC_OscInitStruct.OscillatorType =
RCC_OSCILLATORTYPE_HSI|RCC_OSCILLATORTYPE_LSI
```

```
|RCC_OSCILLATORTYPE_LSE|RCC_OSCILLATORTYPE_HSI48;
  RCC_OscInitStruct.LSEState = RCC_LSE_ON;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = 16;
  RCC_OscInitStruct.LSIState = RCC_LSI_ON;
  RCC_OscInitStruct.HSI48State = RCC_HSI48_ON;
  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();
  }
  PeriphClkInit.PeriphClockSelection =
RCC_PERIPHCLK_I2C1|RCC_PERIPHCLK_RTC
```

```
                              |RCC_PERIPHCLK_USB;
  PeriphClkInit.I2c1ClockSelection = RCC_I2C1CLKSOURCE_PCLK1;
  PeriphClkInit.RTCClockSelection = RCC_RTCCLKSOURCE_LSE;
  PeriphClkInit.UsbClockSelection = RCC_USBCLKSOURCE_HSI48;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }
  __HAL_RCC_LSEDRIVE_CONFIG(RCC_LSEDRIVE_LOW);
  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);
  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}
static void MX_ADC_Init(void)
{
  ADC_ChannelConfTypeDef sConfig;
  hadc.Instance = ADC1;
  hadc.Init.OversamplingMode = DISABLE;
  hadc.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV1;
  hadc.Init.Resolution = ADC_RESOLUTION_12B;
  hadc.Init.SamplingTime = ADC_SAMPLETIME_160CYCLES_5;
  hadc.Init.ScanConvMode = ADC_SCAN_DIRECTION_FORWARD;
  hadc.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc.Init.ContinuousConvMode = ENABLE;
  hadc.Init.DiscontinuousConvMode = DISABLE;
  hadc.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
```

```
  hadc.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc.Init.DMAContinuousRequests = DISABLE;
  hadc.Init.EOCSelection = ADC_EOC_SEQ_CONV;
  hadc.Init.Overrun = ADC_OVR_DATA_PRESERVED;
  hadc.Init.LowPowerAutoWait = ENABLE;
  hadc.Init.LowPowerFrequencyMode = DISABLE;
  hadc.Init.LowPowerAutoPowerOff = DISABLE;
  if (HAL_ADC_Init(&hadc) != HAL_OK)
  {
    Error_Handler();
  }
  sConfig.Channel = ADC_CHANNEL_TEMPSENSOR;
  sConfig.Rank = ADC_RANK_CHANNEL_NUMBER;
  if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
}
static void MX_I2C1_Init(void)
{
  hi2c1.Instance = I2C1;
  hi2c1.Init.Timing = 0x00504150;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
```

```
  hi2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1, I2C_ANALOGFILTER_ENABLE) !=
HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 2) != HAL_OK)
  {
    Error_Handler();
  }
}
static void MX_RTC_Init(void)
{
  hrtc.Instance = RTC;
  hrtc.Init.HourFormat = RTC_HOURFORMAT_24;
  hrtc.Init.AsynchPrediv = 127;
  hrtc.Init.SynchPrediv = 255;
  hrtc.Init.OutPut = RTC_OUTPUT_DISABLE;
  hrtc.Init.OutPutRemap = RTC_OUTPUT_REMAP_NONE;
  hrtc.Init.OutPutPolarity = RTC_OUTPUT_POLARITY_HIGH;
```

```
  hrtc.Init.OutPutType = RTC_OUTPUT_TYPE_OPENDRAIN;
  if (HAL_RTC_Init(&hrtc) != HAL_OK)
  {
    Error_Handler();
  }
}
static void MX_SPI1_Init(void)
{
  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_2LINES;
  hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi1.Init.NSS = SPI_NSS_SOFT;
  hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_32;
  hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi1.Init.CRCPolynomial = 7;
  if (HAL_SPI_Init(&hspi1) != HAL_OK)
  {
    Error_Handler();
  }
}
static void MX_SPI2_Init(void)
```

```
{
  hspi2.Instance = SPI2;
  hspi2.Init.Mode = SPI_MODE_MASTER;
  hspi2.Init.Direction = SPI_DIRECTION_2LINES;
  hspi2.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi2.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi2.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi2.Init.NSS = SPI_NSS_SOFT;
  hspi2.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
  hspi2.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi2.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi2.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi2.Init.CRCPolynomial = 7;
  if (HAL_SPI_Init(&hspi2) != HAL_OK)
  {
    Error_Handler();
  }
}
static void MX_TIM21_Init(void)
{
  TIM_ClockConfigTypeDef sClockSourceConfig;
  TIM_MasterConfigTypeDef sMasterConfig;
  TIM_IC_InitTypeDef sConfigIC;
  htim21.Instance = TIM21;
  htim21.Init.Prescaler = 0;
  htim21.Init.CounterMode = TIM_COUNTERMODE_UP;
```

```
  htim21.Init.Period = 0xFFFF;
  htim21.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  if (HAL_TIM_Base_Init(&htim21) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim21, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_IC_Init(&htim21) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim21, &sMasterConfig) !=
HAL_OK)
  {
    Error_Handler();
  }
  sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
  sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
  sConfigIC.ICPrescaler = TIM_ICPSC_DIV8;
  sConfigIC.ICFilter = 0;
```

```
  if (HAL_TIM_IC_ConfigChannel(&htim21, &sConfigIC, TIM_CHANNEL_1) !=
HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIMEx_RemapConfig(&htim21, TIM21_TI1_LSI) != HAL_OK)
  {
    Error_Handler();
  }
}
static void MX_DMA_Init(void)
{
  __HAL_RCC_DMA1_CLK_ENABLE();
  HAL_NVIC_SetPriority(DMA1_Channel1_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel1_IRQn);
  HAL_NVIC_SetPriority(DMA1_Channel2_3_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel2_3_IRQn);
  HAL_NVIC_SetPriority(DMA1_Channel4_5_6_7_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel4_5_6_7_IRQn);
}
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct;
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
```

```
  HAL_GPIO_WritePin(GPIOA, Led_Red_Pin|Lcd_Power_Pin|Lcd_CS_Pin,
GPIO_PIN_RESET);

  HAL_GPIO_WritePin(GPIOB,
Flash_Hold_Pin|Flash_WP_Pin|Lcd_Discharge_Pin|Lcd_Rst_Pin

                    |Flash_CS_Pin, GPIO_PIN_RESET);

  GPIO_InitStruct.Pin = Button_User_Pin|Lcd_Busy_Pin;

  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;

  GPIO_InitStruct.Pull = GPIO_NOPULL;

  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  GPIO_InitStruct.Pin = Led_Red_Pin|Lcd_Power_Pin|Lcd_CS_Pin;

  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;

  GPIO_InitStruct.Pull = GPIO_NOPULL;

  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_MEDIUM;

  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  GPIO_InitStruct.Pin =
Flash_Hold_Pin|Flash_WP_Pin|Lcd_Discharge_Pin|Lcd_Rst_Pin

|Flash_CS_Pin;

  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;

  GPIO_InitStruct.Pull = GPIO_NOPULL;

  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_MEDIUM;

  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

}

void Error_Handler(void)

{

  while(1)

  {

  }
```

```
}
#ifdef USE_FULL_ASSERT
  void assert_failed(uint8_t* file, uint32_t line)
{
}
```

Application.c

```
#include "application.h"
#include "callbacks.h"
#include "main.h"
#include "stm32l0xx_hal.h"
#include "usb_device.h"
#include "usbd_core.h"
#include "usbd_desc.h"
#include "usbd_cdc.h"
#include "usbd_cdc_if.h"
#include "display.h"
#include "spi_flash.h"
#include "temp_sensor.h"
#include "eeprom_parameters.h"
#include "data_log.h"
#include "serial_cmd.h"
#include "math.h"
extern RTC_HandleTypeDef hrtc;
extern TIM_HandleTypeDef htim21;
extern I2C_HandleTypeDef hi2c1;
```

```
extern ADC_HandleTypeDef hadc;
void getLSIClockFreq(void);
void calibrateRTC(void);
void reinitRTC(void);
void heartBeat(void);
int32_t readTemperature(void);
void serialProcess(void);
void saveLogData(void);
LogDataTypeDef logData;
RTC_TimeTypeDef currTime;
RTC_DateTypeDef currDate;
uint32_t asyncPredv;
uint32_t syncPredv;
uint32_t adcBuffer[ADC_BUFFER_SIZE];
uint8_t i2cRxBuffer[30];
void InitApplication(void)
{
        HAL_ADCEx_Calibration_Start(&hadc,ADC_SINGLE_ENDED);
        HAL_ADCEx_EnableVREFINTTempSensor();
        USBD_CDC_SetRxBuffer(&hUsbDeviceFS,usbRxBuffer);
        USBD_CDC_ReceivePacket(&hUsbDeviceFS);
        if(!FlashInit())
        {
                while(1);
        }
 if(!TempSensorInit())
```

```
        {
                while(1);
        }
        if(!DataLogInit())
        {
                while(1);
        }
}
void RunApplication(void)
{
        heartBeat();
        serialProcess();
}
void getLSIClockFreq(void)
{
        HAL_TIM_IC_Start_IT (&htim21,TIM_CHANNEL_1);
        while(uwCaptureNumber != 2)
  {
  }
          HAL_TIM_IC_Stop_IT(&htim21, TIM_CHANNEL_1);
  uwCaptureNumber = 0;
}
void calibrateRTC(void)
{
        getLSIClockFreq();
        asyncPredv = hrtc.Init.AsynchPrediv;
```

```
        syncPredv = (uwLsiFreq / (asyncPredv+1))-1;
        reinitRTC();
}
void reinitRTC(void)
{
        HAL_RTC_DeInit(&hrtc);
  hrtc.Instance = RTC;
  hrtc.Init.HourFormat = RTC_HOURFORMAT_24;
  hrtc.Init.AsynchPrediv = asyncPredv;
  hrtc.Init.SynchPrediv = syncPredv;
  hrtc.Init.OutPut = RTC_OUTPUT_DISABLE;
  hrtc.Init.OutPutRemap = RTC_OUTPUT_REMAP_NONE;
  hrtc.Init.OutPutPolarity = RTC_OUTPUT_POLARITY_HIGH;
  hrtc.Init.OutPutType = RTC_OUTPUT_TYPE_OPENDRAIN;
  if (HAL_RTC_Init(&hrtc) != HAL_OK)
  {
  }
}
void heartBeat(void)
{
        static uint32_t lastTime;
        uint32_t currentTime = HAL_GetTick();

        if(currentTime - lastTime >= HEART_BEAT_PERIOD)
        {
                lastTime = currentTime;
```

```
                int32_t cTemp = readTemperature();
                saveLogData();
        }
}
int32_t readTemperature(void)
{
        int32_t temperature;
        int32_t tempSensDataTotal = 0;
        int32_t loopCounter = 0;
        adcDataReady = false;
        HAL_ADC_Start_DMA(&hadc,adcBuffer,ADC_BUFFER_SIZE);
        while(!adcDataReady);
        for(loopCounter = 0; loopCounter < ADC_BUFFER_SIZE; loopCounter++)
        {
                tempSensDataTotal = tempSensDataTotal + adcBuffer[loopCounter];
        }
        tempSensDataTotal =(int32_t) tempSensDataTotal /
((int32_t)ADC_BUFFER_SIZE);
        temperature = ((tempSensDataTotal * VDD_APPLI / VDD_CALIB) - (int32_t)
*TEMP30_CAL_ADDR );
        temperature = temperature * (int32_t)(130 - 30);
        temperature = temperature / (int32_t)(*TEMP130_CAL_ADDR -
*TEMP30_CAL_ADDR);
        temperature = temperature + 30;
        return temperature;
}
void serialProcess(void)
```

```
{
}
void saveLogData(void)
{
  TempSensorMeasureRH();
        TempSensorReset();
        HAL_RTC_GetTime(&hrtc,&currTime,FORMAT_BIN);
        HAL_RTC_GetDate(&hrtc,&currDate,FORMAT_BIN);
        logData.date.day = currDate.Date;
        logData.date.month = currDate.Month;
        logData.date.year = currDate.Year;
        logData.time.hour = currTime.Hours;
        logData.time.minute = currTime.Minutes;
        logData.time.second = currTime.Seconds;
        logData.humidity = (uint8_t)roundf(sensorDev.rhValue);
  logData.temperature = (uint16_t)roundf((sensorDev.tempValue + 40) * 10);
        DataLogCalculateChecksum(&logData);
        DataLogAddNewLog(&logData);
}
```

Callbacks.c

```
#include "callbacks.h"
extern TIM_HandleTypeDef htim21;
extern ADC_HandleTypeDef hadc;
int16_t tmpCCTIM_CHANNEL_1[2] = {0, 0};
__IO uint32_t uwLsiFreq = 0;
```

```
__IO uint32_t uwCaptureNumber = 0;

__IO uint32_t uwPeriodValue = 0;

bool adcDataReady = false;

void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)

{

	if(htim == &htim21)

	{

		tmpCCTIM_CHANNEL_1[uwCaptureNumber++] =
HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1);

		if (uwCaptureNumber >= 2)

		{

			if ( tmpCCTIM_CHANNEL_1[0] >
tmpCCTIM_CHANNEL_1[1] )

			{

				uwPeriodValue = (uint16_t)(0xFFFF -
tmpCCTIM_CHANNEL_1[0] + tmpCCTIM_CHANNEL_1[1] + 1);

			}

			else

			{

				uwPeriodValue =
(uint16_t)(tmpCCTIM_CHANNEL_1[1] - tmpCCTIM_CHANNEL_1[0]);

			}

			uwLsiFreq = (uint32_t) HAL_RCC_GetHCLKFreq() /
uwPeriodValue;

			uwLsiFreq *= 8;

		}

	}

}
```

```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *adc)
{
	if(adc == &hadc)
	{
		adcDataReady = true;
	}
}
```

Data_log.c

```
#include "data_log.h"
#include "spi_flash.h"
#include "eeprom_parameters.h"
#include "stm32l0xx_hal.h"
uint8_t logReadBuffer[SINGLE_LOG_SIZE];
uint8_t logWriteBuffer[SINGLE_LOG_SIZE];
uint8_t logProgramBuffer[SINGLE_LOG_SIZE];
bool DataLogInit(void)
{
	bool result = true;
	result = EepromInit();
	if(Device.logCounter == 0)
	{
	}
	return result;
}
```

```
bool DataLogIsDataExist(void)
{
	bool result = false;
	if(Device.logCounter > 0)
	{
		result = true;
	}
	return result;
}
bool DataLogAddNewLog(LogDataTypeDef *log)
{
	bool result = true;
	uint32_t logAddress = Device.logCounter * SINGLE_LOG_SIZE;
	uint16_t counter = 0;
	uint16_t index = 0;
	logWriteBuffer[LOG_INDEX_DAY] = log->date.day;
	logWriteBuffer[LOG_INDEX_MONTH] = log->date.month;
	logWriteBuffer[LOG_INDEX_YEAR] = log->date.year;
	logWriteBuffer[LOG_INDEX_HOUR] = log->time.hour;
	logWriteBuffer[LOG_INDEX_MINUTE] = log->time.minute;
	logWriteBuffer[LOG_INDEX_SECOND] = log->time.second;
	logWriteBuffer[LOG_INDEX_HUMIDITY] = log->humidity;
	logWriteBuffer[LOG_INDEX_TEMPERATURE] = (log->temperature &
0xFF);
	logWriteBuffer[LOG_INDEX_TEMPERATURE+1] = ((log->temperature >> 8)
& 0xFF);
```

```
logWriteBuffer[LOG_INDEX_CHECKSUM] = log->checkSum;
for(counter = 0; counter < SINGLE_LOG_SIZE; counter++)
{
        logProgramBuffer[index] = logWriteBuffer[counter];
        index++;
        if(((logAddress+counter+1) & 0xFF ) == 0x00)
        {
                if(FlashPageProgram(logAddress,logProgramBuffer,index))
                {
                        index = 0;
                        logAddress = logAddress+counter+1;
                }
                else
                {
                        result = false;
                }
        }
        else if(counter == SINGLE_LOG_SIZE-1)
        {
                if(!FlashPageProgram(logAddress,logProgramBuffer,index))
                {
                        result = false;
                }
        }
}
if(result == true)
```

```
        {
                Device.logCounter++;
                EepromSaveLogCounter();
        }
        return result;
}
bool DataLogReadLog(uint32_t index,LogDataTypeDef *log)
{
        bool result = false;
        uint32_t logAddress = 0x00;
        if(index < Device.logCounter)
        {
                logAddress = index * SINGLE_LOG_SIZE;
                FlashReadData(logAddress,logReadBuffer,SINGLE_LOG_SIZE);
                log->date.day = logReadBuffer[LOG_INDEX_DAY];
                log->date.month = logReadBuffer[LOG_INDEX_MONTH];
                log->date.year = logReadBuffer[LOG_INDEX_YEAR];
                log->time.hour = logReadBuffer[LOG_INDEX_HOUR];
                log->time.minute = logReadBuffer[LOG_INDEX_MINUTE];
                log->time.second = logReadBuffer[LOG_INDEX_SECOND];
                log->humidity = logReadBuffer[LOG_INDEX_HUMIDITY];
                log->temperature = logReadBuffer[LOG_INDEX_TEMPERATURE+1];
                log->temperature = (log->temperature << 8) |
logReadBuffer[LOG_INDEX_TEMPERATURE];
                log->checkSum = logReadBuffer[LOG_INDEX_CHECKSUM];
                if(DataLogCalculateChecksum(log) == log->checkSum)
```

```
            {
                result = true;
            }
        }
        return result;
}
void DataLogEraseLogs(void)
{
        Device.logCounter = 0;
        EepromSaveLogCounter();
        FlashChipErase();
}
uint8_t DataLogCalculateChecksum(LogDataTypeDef *log)
{
        uint8_t checksum = 0xFF;
        checksum ^= log->date.day;
        checksum ^= log->date.month;
        checksum ^= log->date.year;
        checksum ^= log->time.hour;
        checksum ^= log->time.minute;
        checksum ^= log->time.second;
        checksum ^= log->humidity;
        checksum ^= (log->temperature & 0xFF);
        checksum ^= ((log->temperature >> 8) & 0xFF);
        log->checkSum = checksum;
```

```
	return checksum;
}
```

Eeprom_parameters.c

```
#include "eeprom_parameters.h"
#include "stm32l0xx_hal.h"
#include "spi_flash.h"
DeviceParametersTypeDef Device;
bool EepromInit(void)
{
	bool result = true;
	Device.eepromInitIdentifier =
*(uint32_t*)__GET_EEPROM_ADDRES(EEPROM_INDEX_IDENTIFIER);
	if(Device.eepromInitIdentifier != EEPROM_INIT_IDENTIFIER)
	{
		HAL_FLASHEx_DATAEEPROM_Unlock();
		for(int i = 0; i < 10; i++)
		{
	HAL_FLASHEx_DATAEEPROM_Erase(DATA_EEPROM_BASE+(i*4));
		}
	HAL_FLASHEx_DATAEEPROM_Program(FLASH_TYPEPROGRAMDATA
_WORD,__GET_EEPROM_ADDRES(EEPROM_INDEX_IDENTIFIER),(uint32_t)E
EPROM_INIT_IDENTIFIER);
HAL_FLASHEx_DATAEEPROM_Program(FLASH_TYPEPROGRAMDATA_WOR
D,__GET_EEPROM_ADDRES(EEPROM_INDEX_TMP_LOW_LIMIT),(uint32_t)D
EFAULT_LOWER_LIMIT);
HAL_FLASHEx_DATAEEPROM_Program(FLASH_TYPEPROGRAMDATA_WOR
D,__GET_EEPROM_ADDRES(EEPROM_INDEX_TMP_UP_LIMIT),(uint32_t)DEF
AULT_UPPER_LIMIT);
```

```
        HAL_FLASHEx_DATAEEPROM_Program(FLASH_TYPEPROGRAMDATA
_WORD,__GET_EEPROM_ADDRES(EEPROM_INDEX_DEVICE_ID),(uint32_t)D
EFAULT_DEVICE_ID);

        HAL_FLASHEx_DATAEEPROM_Program(FLASH_TYPEPROGRAMDATA
_WORD,__GET_EEPROM_ADDRES(EEPROM_INDEX_LOG_COUNTER),(uint32
_t)DEFAULT_LOG_COUNTER);

        HAL_FLASHEx_DATAEEPROM_Program(FLASH_TYPEPROGRAMDATA
_WORD,__GET_EEPROM_ADDRES(EEPROM_INDEX_DATA_PERIOD),(uint32_
t)DEFAULT_DATA_PERIOD);

                Device.tempUpperLimit =
*(uint32_t*)(__GET_EEPROM_ADDRES(EEPROM_INDEX_TMP_UP_LIMIT));

                Device.tempLowerLimit =
*(uint32_t*)(__GET_EEPROM_ADDRES(EEPROM_INDEX_TMP_LOW_LIMIT));

                Device.deviceId =
*(uint32_t*)(__GET_EEPROM_ADDRES(EEPROM_INDEX_DEVICE_ID));

                Device.logCounter =
*(uint32_t*)(__GET_EEPROM_ADDRES(EEPROM_INDEX_LOG_COUNTER));

                Device.eepromInitIdentifier =
*(uint32_t*)(__GET_EEPROM_ADDRES(EEPROM_INDEX_IDENTIFIER));

                Device.dataPeriod =
*(uint32_t*)(__GET_EEPROM_ADDRES(EEPROM_INDEX_DATA_PERIOD));

                if(Device.tempLowerLimit != DEFAULT_LOWER_LIMIT

                        || Device.deviceId != DEFAULT_DEVICE_ID

                        || Device.eepromInitIdentifier != EEPROM_INIT_IDENTIFIER

                  || Device.logCounter != DEFAULT_LOG_COUNTER

                  || Device.tempUpperLimit != DEFAULT_UPPER_LIMIT

                  || Device.dataPeriod != DEFAULT_DATA_PERIOD

                        )

                {

                        result = false;
```

```
            }

            HAL_FLASHEx_DATAEEPROM_Lock();

            FlashChipErase();

        }

 else

 {

            Device.tempUpperLimit =
*(uint32_t*)(__GET_EEPROM_ADDRES(EEPROM_INDEX_TMP_UP_LIMIT));

            Device.tempLowerLimit =
*(uint32_t*)(__GET_EEPROM_ADDRES(EEPROM_INDEX_TMP_LOW_LIMIT));

            Device.deviceId =
*(uint32_t*)(__GET_EEPROM_ADDRES(EEPROM_INDEX_DEVICE_ID));

            Device.logCounter =
*(uint32_t*)(__GET_EEPROM_ADDRES(EEPROM_INDEX_LOG_COUNTER));

            Device.dataPeriod =
*(uint32_t*)(__GET_EEPROM_ADDRES(EEPROM_INDEX_DATA_PERIOD));

        }

        return result;

}

bool EepromSaveLimits(void)

{

        bool result = true;

        HAL_FLASHEx_DATAEEPROM_Unlock();

        HAL_FLASHEx_DATAEEPROM_Erase(DATA_EEPROM_BASE+EEPROM
_INDEX_TMP_UP_LIMIT);

        HAL_FLASHEx_DATAEEPROM_Erase(DATA_EEPROM_BASE+EEPROM
_INDEX_TMP_LOW_LIMIT);

        HAL_FLASHEx_DATAEEPROM_Erase(DATA_EEPROM_BASE+EEPROM
_INDEX_DATA_PERIOD);
```

```
        HAL_FLASHEx_DATAEEPROM_Program(FLASH_TYPEPROGRAMDATA
_WORD,__GET_EEPROM_ADDRES(EEPROM_INDEX_DATA_PERIOD),Device.d
ataPeriod);

HAL_FLASHEx_DATAEEPROM_Program(FLASH_TYPEPROGRAMDATA_WOR
D,__GET_EEPROM_ADDRES(EEPROM_INDEX_TMP_LOW_LIMIT),Device.temp
LowerLimit);

HAL_FLASHEx_DATAEEPROM_Program(FLASH_TYPEPROGRAMDATA_WOR
D,__GET_EEPROM_ADDRES(EEPROM_INDEX_TMP_UP_LIMIT),Device.tempU
pperLimit);

        HAL_FLASHEx_DATAEEPROM_Lock();

        return result;

}

bool EepromSaveLogCounter(void)

{

        bool result = true;

        HAL_FLASHEx_DATAEEPROM_Unlock();

        HAL_FLASHEx_DATAEEPROM_Erase(DATA_EEPROM_BASE+EEPROM
_INDEX_LOG_COUNTER);

        HAL_FLASHEx_DATAEEPROM_Program(FLASH_TYPEPROGRAMDATA
_WORD,__GET_EEPROM_ADDRES(EEPROM_INDEX_LOG_COUNTER),Device.
logCounter);

        HAL_FLASHEx_DATAEEPROM_Lock();

        return result;

}

void EepromReadDeviceId(void)

{

        Device.deviceId =
*(uint32_t*)(__GET_EEPROM_ADDRES(EEPROM_INDEX_DEVICE_ID));

}
```

```
void EepromReadLimits(void)
{
	Device.tempUpperLimit =
*(uint32_t*)(__GET_EEPROM_ADDRES(EEPROM_INDEX_TMP_UP_LIMIT));
	Device.tempLowerLimit =
*(uint32_t*)(__GET_EEPROM_ADDRES(EEPROM_INDEX_TMP_LOW_LIMIT));
	Device.dataPeriod =
*(uint32_t*)(__GET_EEPROM_ADDRES(EEPROM_INDEX_DATA_PERIOD));
}
void EepromReadLogCounter(void)
{
	Device.logCounter =
*(uint32_t*)(__GET_EEPROM_ADDRES(EEPROM_INDEX_LOG_COUNTER));
}
```

Serial_cmd.c

```
#include "serial_cmd.h"
#include "main.h"
#include "stm32l0xx_hal.h"
#include "usb_device.h"
#include "usbd_core.h"
#include "usbd_desc.h"
#include "usbd_cdc.h"
#include "usbd_cdc_if.h"
#include "eeprom_parameters.h"
#include "temp_sensor.h"
#include "data_log.h"
```

```
extern RTC_HandleTypeDef hrtc;
RTC_TimeTypeDef timeHandler;
RTC_DateTypeDef dateHandler;
uint8_t usbTxBuffer[CDC_DATA_FS_IN_PACKET_SIZE];
uint8_t usbRxBuffer[CDC_DATA_FS_OUT_PACKET_SIZE];
uint8_t cmdBuffer[CMD_BUFFER_SIZE];
msgFlagsTypeDef msgFlags =
{
        .rFlag = false,
        .nFlag = false,
        .startFlag = false,
        .sizeFlag = false,
        .size = 0
};
void processMessage(unsigned char *data,int size)
{
        uint32_t handler;
        uint8_t humidity;
        uint16_t temperature;
        LogDataTypeDef logObj;
  uint32_t logIndex = 0;
        switch(data[0])
        {
                case GET_DATE_TIME:
                        HAL_RTC_GetTime(&hrtc,&timeHandler,FORMAT_BIN);
```

```
HAL_RTC_GetDate(&hrtc,&dateHandler,FORMAT_BIN);
        cmdBuffer[0] = GET_DATE_TIME;
        cmdBuffer[1] = dateHandler.Date;
        cmdBuffer[2] = dateHandler.Month;
        cmdBuffer[3] = dateHandler.Year;
        cmdBuffer[4] = timeHandler.Hours;
        cmdBuffer[5] = timeHandler.Minutes;
        cmdBuffer[6] = timeHandler.Seconds;
        sendData(cmdBuffer,7);
        break;
    case SET_DATE_TIME:
        if(size == 7)
        {
            dateHandler.Date = data[1];
            dateHandler.Month = data[2];
            dateHandler.Year = data[3];
            timeHandler.Hours = data[4];
            timeHandler.Minutes = data[5];
            timeHandler.Seconds = data[6];
HAL_RTC_SetDate(&hrtc,&dateHandler,FORMAT_BIN);
HAL_RTC_SetTime(&hrtc,&timeHandler,FORMAT_BIN);
        }
        break;
    case ERASE_DATA:
        break;
    case GET_DATA:
```

```
        for(int i = 0; i < 4; i++)
        {
            handler = data[1+i];
            logIndex  = logIndex | (handler << (i*8));
        }
        if(DataLogReadLog(logIndex,&logObj))
        {
            cmdBuffer[0] = GET_DATA;
            for(int i = 0; i < 4; i++)
            {
                cmdBuffer[i+1] = (logIndex >> (i*8)) & 0xFF;
            }
            cmdBuffer[5] = logObj.date.day;
            cmdBuffer[6] = logObj.date.month;
            cmdBuffer[7] = logObj.date.year;
            cmdBuffer[8] = logObj.time.hour;
            cmdBuffer[9] = logObj.time.minute;
            cmdBuffer[10] = logObj.time.second;
            cmdBuffer[11] = logObj.humidity;
            cmdBuffer[12] = logObj.temperature & 0xFF;
            cmdBuffer[13] = (logObj.temperature >> 8) & 0xFF;
            sendData(cmdBuffer,14);
        }
        break;
    case GET_DATA_COUNTER:
```

```
                EepromReadLogCounter();
                cmdBuffer[0] = GET_DATA_COUNTER;
                for(int i = 0; i< 4; i++)
                {
                        cmdBuffer[i+1] = (Device.logCounter >> (i*8)) & 0xFF;
                }
                sendData(cmdBuffer,5);
                break;
        case GET_LIMITS:
                cmdBuffer[0] = GET_LIMITS;
                for(int i = 0; i< 4; i++)
                {
                        cmdBuffer[i+1] = (Device.dataPeriod >> (i*8)) & 0xFF;
                }
                for(int i = 0; i < 4; i++)
                {
                        cmdBuffer[i+5] = (Device.tempUpperLimit >> (i*8)) &
0xFF;
                }
                for(int i = 0; i < 4; i++)
                {
                        cmdBuffer[i+9] = (Device.tempLowerLimit>> (i*8)) &
0xFF;
                }
                sendData(cmdBuffer,13);
                break;
        case SET_LIMITS:
```

```
                Device.dataPeriod = 0;
                Device.tempLowerLimit = 0;
                Device.tempUpperLimit = 0;
                if(size == 13)
                {
                        for(int i = 0; i < 4; i++)
                        {
                                handler = data[1+i];
                                Device.dataPeriod  = Device.dataPeriod | (handler
<< (i*8));
                                handler = data[5+i];
                                Device.tempUpperLimit = Device.tempUpperLimit
| (handler << (i*8));
                                handler = data[9+i];
                                Device.tempLowerLimit =
Device.tempLowerLimit |(handler << (i*8));
                        }

                        EepromSaveLimits();
                        EepromReadLimits();
                }
                break;
        case GET_DATA_PERIOD:
                break;
        case SET_DATA_PERIOD:
                break;
        case SET_DEV_ID:
```

```
                    break;
                case GET_DEV_ID:
                    cmdBuffer[0] = GET_DEV_ID;
                    for(int i = 0; i < 4; i++)
                    {
                        cmdBuffer[i+1] = (Device.deviceId >> (i*8)) & 0xFF;
                    }
                    sendData(cmdBuffer,5);
                    break;
                case GET_SENSOR_DATA:
                    cmdBuffer[0] = GET_SENSOR_DATA;
                    humidity = sensorDev.rhValue;
                    temperature = (sensorDev.tempValue + 40) * 10;
                    cmdBuffer[1] = humidity;
                cmdBuffer[2] = temperature & 0xFF;
    cmdBuffer[3] = (temperature >> 8) & 0xFF;
                    sendData(cmdBuffer,4);
                    break;
        }
}
void readData(unsigned char *data,int size)
{
        int cmdIndex = 0;
        for(int i=0; i < size; i++)
        {
            if(msgFlags.startFlag == false && data[i] == STX)
```

```
		{
			msgFlags.startFlag = true;
		}
		else if(msgFlags.startFlag == true && msgFlags.sizeFlag == false)
		{
			msgFlags.sizeFlag = true;
			msgFlags.size = data[i];
			memset(cmdBuffer,'\0',CMD_BUFFER_SIZE);
			cmdIndex = 0;
		}
		else if(msgFlags.startFlag == true && msgFlags.sizeFlag == true &&
msgFlags.size > 0)
		{
			cmdBuffer[cmdIndex] = data[i];
			msgFlags.size--;
			cmdIndex++;
		}
		else if(msgFlags.startFlag == true && msgFlags.sizeFlag == true &&
msgFlags.size == 0 && msgFlags.rFlag == false)
		{
			if(data[i] == '\r')
			{
				msgFlags.rFlag = true;
			}
			else
			{
				msgFlags.nFlag = false;
```

```
                msgFlags.rFlag = false;
                msgFlags.sizeFlag = false;
                msgFlags.startFlag = false;
                msgFlags.size = 0;
                memset(cmdBuffer,'\0',CMD_BUFFER_SIZE);
                cmdIndex = 0;
            }
        }
        else if(msgFlags.startFlag == true && msgFlags.rFlag == true)
        {
            if(data[i] == '\n')
            {
                processMessage(cmdBuffer,cmdIndex);
            }
            else
            {
            }

            msgFlags.nFlag = false;
            msgFlags.rFlag = false;
            msgFlags.sizeFlag = false;
            msgFlags.startFlag = false;
            msgFlags.size = 0;
            memset(cmdBuffer,'\0',CMD_BUFFER_SIZE);
        }
}
```

```
}
void sendData(unsigned char *data,int size)
{
	int txIndex = 0;
	usbTxBuffer[txIndex] = STX;
	txIndex++;
	usbTxBuffer[txIndex] = size;
	txIndex++;
	for(int i = 0; i < size; i++)
	{
		usbTxBuffer[txIndex] = data[i];
		txIndex++;
	}
	usbTxBuffer[txIndex] = '\r';
	txIndex++;
	usbTxBuffer[txIndex] = '\n';
	txIndex++;
	USBD_CDC_SetTxBuffer(&hUsbDeviceFS,usbTxBuffer,txIndex);
	USBD_CDC_TransmitPacket(&hUsbDeviceFS);
}
```

Spi_flash.c

```
#include "spi_flash.h"
#include "main.h"
#include "stm32l0xx_hal.h"
#include "string.h"
```

```
FlashTypeDef flashDevice;
extern SPI_HandleTypeDef hspi2;
void FlashEnableWP(bool state)
{
	if(state)
	{
	HAL_GPIO_WritePin(Flash_WP_GPIO_Port,Flash_WP_Pin,GPIO_PIN_SET);
	}
	else
	{
	HAL_GPIO_WritePin(Flash_WP_GPIO_Port,Flash_WP_Pin,GPIO_PIN_RESE
T);
	}
}
void FlashEnableCS(bool state)
{
	if(state)
	{
	HAL_GPIO_WritePin(Flash_CS_GPIO_Port,Flash_CS_Pin,GPIO_PIN_RESET
);
	}
	else
	{
	HAL_GPIO_WritePin(Flash_CS_GPIO_Port,Flash_CS_Pin,GPIO_PIN_SET);
	}
}
```

```
bool FlashIsHoldON(void)
{
	bool result = false;
	if(HAL_GPIO_ReadPin(Flash_Hold_GPIO_Port,Flash_Hold_Pin) ==
GPIO_PIN_RESET)
	{
		result = true;
	}
	return result;
}
void FlashReadID(uint32_t *data)
{
	uint8_t rxData[3];
	uint8_t cmd = FLASH_CMD_RDID;
	FlashEnableCS(true);
	if(HAL_SPI_Transmit(&hspi2,&cmd,1,100) == HAL_OK)
	{
		if(HAL_SPI_Receive(&hspi2,rxData,3,200) != HAL_OK)
		{
			while(1);
		}
	}
	FlashEnableCS(false);
	*data = 0;
	*data |= rxData[0];
	*data = (*data << 8) | rxData[1];
```

```
        *data = (*data << 8) | rxData[2];
}
bool FlashInit(void)
{
        uint8_t retry = 5;
        bool result = false;
        while(retry--)
        {
                FlashReadID(&flashDevice.id);
                if(flashDevice.id == FLASH_DEFAULT_ID)
                {
                        result = true;
                        break;
                }
        }
        return result;
}
void FlashWriteEnable(bool state)
{
        uint8_t cmd;
        if(state == true)
        {
                cmd = FLASH_CMD_WREN;
        }
        else
        {
```

```
            cmd = FLASH_CMD_WRDI;
        }
        FlashEnableCS(true);
        HAL_SPI_Transmit(&hspi2,&cmd,1,100);
        FlashEnableCS(false);
}
void FlashReadStatusRegister(uint32_t *data)
{
        uint8_t cmd = FLASH_CMD_RDSR;
        uint8_t rxData[2];
        FlashEnableCS(true);
        if(HAL_SPI_Transmit(&hspi2,&cmd,1,100) == HAL_OK)
        {
            if(HAL_SPI_Receive(&hspi2,rxData,2,200) != HAL_OK)
            {
                while(1);
            }
        }
        FlashEnableCS(false);
        *data = rxData[0];
        *data = (*data << 8) | rxData[1];
}
void FlashReadSecurityRegister(uint8_t *data)
{
        uint8_t cmd = FLASH_CMD_RDSCUR;
```

```
	FlashEnableCS(true);
	HAL_SPI_Transmit(&hspi2,&cmd,1,100);
	HAL_SPI_Receive(&hspi2,&flashDevice.securityReg,1,100);
	FlashEnableCS(false);
}
bool FlashIsWELON(void)
{
	bool result = false;
	FlashReadStatusRegister(&flashDevice.statusReg);
	if((flashDevice.statusReg & FLASH_WEL_MASK) == FLASH_WEL_MASK)
	{
		result = true;
	}
	return result;
}
bool FlashIsWIPON(void)
{
	bool result = false;
	FlashReadStatusRegister(&flashDevice.statusReg);
	if((flashDevice.statusReg & FLASH_WIP_MASK) == FLASH_WIP_MASK)
	{
		result = true;
	}
	return result;
}
bool FlashIsProgramFail(void)
```

```
{
	bool result = false;
	FlashReadSecurityRegister(&flashDevice.securityReg);
	if((flashDevice.securityReg & PROGRAM_FAIL_MASK) ==
PROGRAM_FAIL_MASK)
	{
		result = true;
	}
	return result;
}
bool FlashIsProgramSuspended(void)
{
	bool result = false;
	FlashReadSecurityRegister(&flashDevice.securityReg);
	if((flashDevice.securityReg & PROGRAM_SUSPENDED_MASK) ==
PROGRAM_SUSPENDED_MASK)
	{
		result = true;
	}
	return result;
}
bool FlashIsEraseSuspended(void)
{
	bool result = false;
	FlashReadSecurityRegister(&flashDevice.securityReg);
	if((flashDevice.securityReg & ERASE_SUSPENDED_MASK) ==
ERASE_SUSPENDED_MASK)
```

```
	{
		result = true;
	}
	return result;
}
bool FlashIsEraseFail(void)
{
	bool result = false;
	FlashReadSecurityRegister(&flashDevice.securityReg);
	if((flashDevice.securityReg & ERASE_FAIL_MASK) ==
ERASE_FAIL_MASK)
	{
		result = true;
	}
	return result;
}
void FlashReadData(uint32_t startAddress,uint8_t *dataBuffer,uint32_t size)
{
	uint8_t cmd[4];
	cmd[0] = FLASH_CMD_READ;
	cmd[1] = (startAddress >> 16) & 0xFF;
	cmd[2] = (startAddress >> 8) & 0xFF;
	cmd[3] = startAddress & 0xFF;
	FlashEnableCS(true);
	if(HAL_SPI_Transmit(&hspi2,cmd,4,100) == HAL_OK)
	{
```

```
            if(HAL_SPI_Receive(&hspi2,dataBuffer,size,1000) != HAL_OK)
            {
                while(1);
            }
        }
        FlashEnableCS(false);
}
void FlashSectorErase(uint32_t sector)
{
        uint8_t cmd[4];
        uint32_t address;

        if(sector < NUMBER_OF_SECTORS)
        {
            address = sector * SECTOR_SIZE;
            cmd[0] = FLASH_CMD_SE;
            cmd[1] = (address >> 16) & 0xFF;
            cmd[2] = (address >> 8) & 0xFF;
            cmd[3] = address & 0xFF;
            while(!FlashIsWELON())
            {
                FlashWriteEnable(true);
            }
            FlashEnableWP(false);
            FlashEnableCS(true);
```

```
            if(HAL_SPI_Transmit(&hspi2,cmd,4,100) != HAL_OK)
            {
                while(1);
            }

            FlashEnableCS(false);

            while(FlashIsWIPON());

            while(FlashIsEraseSuspended());

            if(FlashIsEraseFail())
            {
                while(1);
            }
            while(FlashIsWELON())
            {
                FlashWriteEnable(false);
            }
            FlashEnableWP(true);
        }
}
void FlashChipErase(void)
{
        uint8_t cmd = FLASH_CMD_CE;
```

```
        while(!FlashIsWELON())
        {
                FlashWriteEnable(true);
        }
        while(FlashIsHoldON());
        FlashEnableWP(false);
        FlashEnableCS(true);
        if(HAL_SPI_Transmit(&hspi2,&cmd,1,100) != HAL_OK)
        {
                while(1);
        }
        FlashEnableCS(false);
        while(FlashIsWIPON());
  while(FlashIsEraseSuspended());
        if(FlashIsEraseFail())
        {
                while(1);
        }
        while(FlashIsWELON())
        {
                FlashWriteEnable(false);
        }
        FlashEnableWP(true);
}
bool FlashPageProgram(uint32_t address,uint8_t *dataBuffer, uint32_t size)
{
```

```
	bool result = false;
	uint8_t cmd[4];
	if(address < SPI_FLASH_SIZE && size <= PAGE_SIZE)
	{
		cmd[0] = FLASH_CMD_PP;
		cmd[1] = (address >> 16) & 0xFF;
		cmd[2] = (address >> 8) & 0xFF;
		cmd[3] = address & 0xFF;
		while(FlashIsWIPON());
		while(!FlashIsWELON())
		{
			FlashWriteEnable(true);
		}
		FlashEnableWP(false);
		FlashEnableCS(true);
		if(HAL_SPI_Transmit(&hspi2,cmd,4,100) == HAL_OK)
		{
			if(HAL_SPI_Transmit(&hspi2,dataBuffer,size,1000) !=
HAL_OK)
			{
				while(1);
			}
		}
		FlashEnableCS(false);
		while(FlashIsWIPON());
  while(FlashIsProgramSuspended());
```

```
            if(FlashIsProgramFail())
            {
                while(1);
            }
            while(FlashIsWELON())
            {
                FlashWriteEnable(false);
            }
            FlashEnableWP(true);
            result = true;
        }
        return result;
}
bool FlashIs4ByteMode(void)
{
        bool result = false;
        FlashReadSecurityRegister(&flashDevice.securityReg);
        if((flashDevice.securityReg & FLASH_4BYTE_MASK) ==
FLASH_4BYTE_MASK)
        {
            result = true;
        }
        return result;
}
uint32_t FlashTest(void)
```

```
{
        uint32_t errorCounter;
        return errorCounter;
}
```

Stm32I0xx_hal_msp.c

```
#include "stm32l0xx_hal.h"
extern DMA_HandleTypeDef hdma_adc;
extern DMA_HandleTypeDef hdma_i2c1_rx;
extern DMA_HandleTypeDef hdma_spi2_rx;
extern void Error_Handler(void);
void HAL_MspInit(void)
{
  __HAL_RCC_SYSCFG_CLK_ENABLE();
  __HAL_RCC_PWR_CLK_ENABLE();
  HAL_NVIC_SetPriority(SVC_IRQn, 0, 0);
  HAL_NVIC_SetPriority(PendSV_IRQn, 0, 0);
  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}
void HAL_ADC_MspInit(ADC_HandleTypeDef* hadc)
{
  if(hadc->Instance==ADC1)
  {
    __HAL_RCC_ADC1_CLK_ENABLE();
    hdma_adc.Instance = DMA1_Channel1;
    hdma_adc.Init.Request = DMA_REQUEST_0;
```

```
    hdma_adc.Init.Direction = DMA_PERIPH_TO_MEMORY;
    hdma_adc.Init.PeriphInc = DMA_PINC_DISABLE;
    hdma_adc.Init.MemInc = DMA_MINC_ENABLE;
    hdma_adc.Init.PeriphDataAlignment = DMA_PDATAALIGN_HALFWORD;
    hdma_adc.Init.MemDataAlignment = DMA_MDATAALIGN_WORD;
    hdma_adc.Init.Mode = DMA_NORMAL;
    hdma_adc.Init.Priority = DMA_PRIORITY_MEDIUM;
    if (HAL_DMA_Init(&hdma_adc) != HAL_OK)
    {
      Error_Handler();
    }
    __HAL_LINKDMA(hadc,DMA_Handle,hdma_adc);
    HAL_NVIC_SetPriority(ADC1_COMP_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(ADC1_COMP_IRQn);
  }
}
void HAL_ADC_MspDeInit(ADC_HandleTypeDef* hadc)
{
  if(hadc->Instance==ADC1)
  {
    __HAL_RCC_ADC1_CLK_DISABLE();
    HAL_DMA_DeInit(hadc->DMA_Handle);
    HAL_NVIC_DisableIRQ(ADC1_COMP_IRQn);
  }
}
```

```
void HAL_I2C_MspInit(I2C_HandleTypeDef* hi2c)
{
  GPIO_InitTypeDef GPIO_InitStruct;
  if(hi2c->Instance==I2C1)
  {
    GPIO_InitStruct.Pin = Sens_SCL_Pin|Sens_SDA_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_OD;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF1_I2C1;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
    HAL_I2CEx_EnableFastModePlus(I2C_FASTMODEPLUS_PB6);
    HAL_I2CEx_EnableFastModePlus(I2C_FASTMODEPLUS_PB7);
    __HAL_RCC_I2C1_CLK_ENABLE();
    hdma_i2c1_rx.Instance = DMA1_Channel3;
    hdma_i2c1_rx.Init.Request = DMA_REQUEST_6;
    hdma_i2c1_rx.Init.Direction = DMA_PERIPH_TO_MEMORY;
    hdma_i2c1_rx.Init.PeriphInc = DMA_PINC_DISABLE;
    hdma_i2c1_rx.Init.MemInc = DMA_MINC_ENABLE;
    hdma_i2c1_rx.Init.PeriphDataAlignment = DMA_PDATAALIGN_BYTE;
    hdma_i2c1_rx.Init.MemDataAlignment = DMA_MDATAALIGN_BYTE;
    hdma_i2c1_rx.Init.Mode = DMA_CIRCULAR;
    hdma_i2c1_rx.Init.Priority = DMA_PRIORITY_LOW;
    if (HAL_DMA_Init(&hdma_i2c1_rx) != HAL_OK)
    {
      Error_Handler();
```

```
    }
    __HAL_LINKDMA(hi2c,hdmarx,hdma_i2c1_rx);
    HAL_NVIC_SetPriority(I2C1_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(I2C1_IRQn);
  }
}
void HAL_I2C_MspDeInit(I2C_HandleTypeDef* hi2c)
{
  if(hi2c->Instance==I2C1)
  {
    __HAL_RCC_I2C1_CLK_DISABLE();
      HAL_GPIO_DeInit(GPIOB, Sens_SCL_Pin|Sens_SDA_Pin);
    HAL_DMA_DeInit(hi2c->hdmarx);
    HAL_NVIC_DisableIRQ(I2C1_IRQn);
  }
}
void HAL_RTC_MspInit(RTC_HandleTypeDef* hrtc)
{
  if(hrtc->Instance==RTC)
  {
    __HAL_RCC_RTC_ENABLE();
  }
}
void HAL_RTC_MspDeInit(RTC_HandleTypeDef* hrtc)
{
  if(hrtc->Instance==RTC)
```

```
  {
    __HAL_RCC_RTC_DISABLE();
  }
}
void HAL_SPI_MspInit(SPI_HandleTypeDef* hspi)
{
  GPIO_InitTypeDef GPIO_InitStruct;
  if(hspi->Instance==SPI1)
  {
    __HAL_RCC_SPI1_CLK_ENABLE();
    GPIO_InitStruct.Pin = Lcd_SCLK_Pin|Lcd_MISO_Pin|Lcd_MOSI_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF0_SPI1;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
  }
  else if(hspi->Instance==SPI2)
  {
    __HAL_RCC_SPI2_CLK_ENABLE();
    GPIO_InitStruct.Pin = Flash_SCLK_Pin|Flash_MISO_Pin|Flash_MOSI_Pin;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF0_SPI2;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
```

```
    hdma_spi2_rx.Instance = DMA1_Channel4;
    hdma_spi2_rx.Init.Request = DMA_REQUEST_2;
    hdma_spi2_rx.Init.Direction = DMA_PERIPH_TO_MEMORY;
    hdma_spi2_rx.Init.PeriphInc = DMA_PINC_DISABLE;
    hdma_spi2_rx.Init.MemInc = DMA_MINC_ENABLE;
    hdma_spi2_rx.Init.PeriphDataAlignment = DMA_PDATAALIGN_BYTE;
    hdma_spi2_rx.Init.MemDataAlignment = DMA_MDATAALIGN_BYTE;
    hdma_spi2_rx.Init.Mode = DMA_NORMAL;
    hdma_spi2_rx.Init.Priority = DMA_PRIORITY_LOW;
    if (HAL_DMA_Init(&hdma_spi2_rx) != HAL_OK)
    {
      Error_Handler();
    }
    __HAL_LINKDMA(hspi,hdmarx,hdma_spi2_rx);

    HAL_NVIC_SetPriority(SPI2_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(SPI2_IRQn);
  }
}
void HAL_SPI_MspDeInit(SPI_HandleTypeDef* hspi)
{
  if(hspi->Instance==SPI1)
  {
    __HAL_RCC_SPI1_CLK_DISABLE();
```

```
    HAL_GPIO_DeInit(GPIOB, Lcd_SCLK_Pin|Lcd_MISO_Pin|Lcd_MOSI_Pin);
  }
  else if(hspi->Instance==SPI2)
  {
    __HAL_RCC_SPI2_CLK_DISABLE();
    HAL_GPIO_DeInit(GPIOB, Flash_SCLK_Pin|Flash_MISO_Pin|Flash_MOSI_Pin);
    HAL_DMA_DeInit(hspi->hdmarx);
    HAL_NVIC_DisableIRQ(SPI2_IRQn);
  }
}
void HAL_TIM_Base_MspInit(TIM_HandleTypeDef* htim_base)
{
  if(htim_base->Instance==TIM21)
  {
    __HAL_RCC_TIM21_CLK_ENABLE();
    HAL_NVIC_SetPriority(TIM21_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(TIM21_IRQn);
  }
}
void HAL_TIM_Base_MspDeInit(TIM_HandleTypeDef* htim_base)
{
  if(htim_base->Instance==TIM21)
  {
    __HAL_RCC_TIM21_CLK_DISABLE();
    HAL_NVIC_DisableIRQ(TIM21_IRQn);
  }
```

```
}
```

Stm32I0xx_it.c

```
#include "stm32l0xx_hal.h"
#include "stm32l0xx.h"
#include "stm32l0xx_it.h"
extern PCD_HandleTypeDef hpcd_USB_FS;
extern DMA_HandleTypeDef hdma_adc;
extern ADC_HandleTypeDef hadc;
extern DMA_HandleTypeDef hdma_i2c1_rx;
extern I2C_HandleTypeDef hi2c1;
extern DMA_HandleTypeDef hdma_spi2_rx;
extern SPI_HandleTypeDef hspi2;
extern TIM_HandleTypeDef htim21;
void NMI_Handler(void)
{
}
void HardFault_Handler(void)
{
  while (1)
  {
  }
}
void SVC_Handler(void)
{
}
```

```
void PendSV_Handler(void)
{
}
void SysTick_Handler(void)
{
  HAL_IncTick();
  HAL_SYSTICK_IRQHandler();
}
void DMA1_Channel1_IRQHandler(void)
{
  HAL_DMA_IRQHandler(&hdma_adc);
}
void DMA1_Channel2_3_IRQHandler(void)
{
  HAL_DMA_IRQHandler(&hdma_i2c1_rx);
}
void DMA1_Channel4_5_6_7_IRQHandler(void)
{
  HAL_DMA_IRQHandler(&hdma_spi2_rx);
}
*/
void ADC1_COMP_IRQHandler(void)
{
  HAL_ADC_IRQHandler(&hadc);
}
```

```
void TIM21_IRQHandler(void)
{
  HAL_TIM_IRQHandler(&htim21);
}
void I2C1_IRQHandler(void)
{
  if (hi2c1.Instance->ISR & (I2C_FLAG_BERR | I2C_FLAG_ARLO |
I2C_FLAG_OVR)) {
    HAL_I2C_ER_IRQHandler(&hi2c1);
  } else {
    HAL_I2C_EV_IRQHandler(&hi2c1);
  }
}
void SPI2_IRQHandler(void)
{
  HAL_SPI_IRQHandler(&hspi2);
}
void USB_IRQHandler(void)
{
  HAL_PCD_IRQHandler(&hpcd_USB_FS);
}
```

Temp_sensor.c

```
#include "temp_sensor.h"
#include "main.h"
#include "stm32l0xx_hal.h"
```

```
extern I2C_HandleTypeDef hi2c1;
SensorTypeDef sensorDev =
{
        .i2c = &hi2c1
  .address = I2C_ADDR << 1
};
bool TempSensorInit(void)
{
        bool result = false;
        TempSensorReadFwRev();
        TempSensorReadID();
        if(sensorDev.firmwareRev == 0xFF || sensorDev.firmwareRev == 0x20)
        {
                switch((sensorDev.serialNumber2 >> 24)&0xFF)
                {
                        case 0x0D:
                                result = true;
                                break;
                        case 0x14:
                                result = true;
                                break;
                        case 0x15:
                          result = true;
                                break;
                        default:
```

```
                result = false;
                break;
            }
        }
        return result;
}
void TempSensorReadFwRev(void)
{
        uint8_t cmd[2];
        cmd[0] = CMD_R_FW_VERSION_1;
        cmd[1] = CMD_R_FW_VERSION_2;
        if(HAL_I2C_Master_Transmit(sensorDev.i2c,sensorDev.address,cmd,2,100) ==
HAL_OK)
        {
        HAL_I2C_Master_Receive(sensorDev.i2c,sensorDev.address,&sensorDev.firm
wareRev,1,100);
        }
}
void TempSensorReadID(void)
{
        uint8_t cmd[4];
        cmd[0] = CMD_R_ELC_ID_1BYTE_1;
        cmd[1] = CMD_R_ELC_ID_1BYTE_2;
        if(HAL_I2C_Master_Transmit(sensorDev.i2c,sensorDev.address,cmd,2,100) ==
HAL_OK)
        {
        HAL_I2C_Master_Receive(sensorDev.i2c,sensorDev.address,cmd,4,100);
```

```
        }
        sensorDev.serialNumber1 = (0x000000FF & cmd[0]) << 24 ;
        sensorDev.serialNumber1 |= ((0x000000FF & cmd[1]) << 16);
        sensorDev.serialNumber1 |= ((0x000000FF & cmd[2]) << 8);
        sensorDev.serialNumber1 |= (0x000000FF & cmd[3]);
  cmd[0] = CMD_R_ELC_ID_2BYTE_1;
        cmd[1] = CMD_R_ELC_ID_2BYTE_2;
        if(HAL_I2C_Master_Transmit(sensorDev.i2c,sensorDev.address,cmd,2,100) ==
HAL_OK)
        {
        HAL_I2C_Master_Receive(sensorDev.i2c,sensorDev.address,cmd,4,100);
        }
        sensorDev.serialNumber2 = (0x000000FF & cmd[0]) << 24 ;
        sensorDev.serialNumber2 |= ((0x000000FF & cmd[1]) << 16);
        sensorDev.serialNumber2 |= ((0x000000FF & cmd[2]) << 8);
        sensorDev.serialNumber2 |= (0x000000FF & cmd[3]);
}
void TempSensorMeasureRH(void)
{
        uint8_t cmd[2];
        cmd[0] = CMD_MEASURE_RH_HM;
        sensorDev.tempData = 0;
        sensorDev.tempValue = 0;
        sensorDev.rhData = 0;
        sensorDev.rhValue = 0;
```

```
        if(HAL_I2C_Master_Transmit(sensorDev.i2c,sensorDev.address,cmd,1,100) ==
HAL_OK)

        {

        if(HAL_I2C_Master_Receive(sensorDev.i2c,sensorDev.address,cmd,2,500) ==
HAL_OK)

                {

                        sensorDev.rhData = (cmd[0] & 0x00FF) << 8;

                        sensorDev.rhData |= (cmd[1] & 0x00FF);

                        sensorDev.rhValue =
TempSensorCalculteRH(sensorDev.rhData);

                        cmd[0] = CMD_READ_TEMP_OF_RH;
        if(HAL_I2C_Master_Transmit(sensorDev.i2c,sensorDev.address,cmd,1,100) ==
HAL_OK)
                        {

        if(HAL_I2C_Master_Receive(sensorDev.i2c,sensorDev.address,cmd,2,500) ==
HAL_OK)
                                {

                                        sensorDev.tempData = (cmd[0] & 0x00FF) << 8;

                                        sensorDev.tempData |= (cmd[1] & 0x00FF);

                                        sensorDev.tempValue =
TempSensorCalculateTemp(sensorDev.tempData);

                                }

                        }

                }

        }

}

void TempSensorMeasureTemp(void)

{

        uint8_t cmd[2];
```

```
	cmd[0] = CMD_MEASURE_TEMP_HM;
	sensorDev.tempData = 0;
	sensorDev.tempValue = 0;
	if(HAL_I2C_Master_Transmit(sensorDev.i2c,sensorDev.address,cmd,1,100) ==
HAL_OK)
	{
	if(HAL_I2C_Master_Receive(sensorDev.i2c,sensorDev.address,cmd,2,500) ==
HAL_OK)
		{
			sensorDev.tempData = (cmd[0] & 0x00FF) << 8;
			sensorDev.tempData |= (cmd[1] & 0x00FF);
			sensorDev.tempValue =
TempSensorCalculateTemp(sensorDev.tempData);
		}
	}
}
void TempSensorReset(void)
{
	uint8_t cmd;
	cmd = CMD_RESET;
	if(HAL_I2C_Master_Transmit(sensorDev.i2c,sensorDev.address,&cmd,1,100)
== HAL_OK)
	{
	}
}
float TempSensorCalculteRH(uint16_t rhData)
```

```
{
	float result;
	result = rhData;
	result = ((125*result)/65536.0) - 6;
	return result;
}
float TempSensorCalculateTemp(uint16_t tempData)
{
	float result;
	result = tempData;
	result = ((175.72*result)/65536.0) - 46.85;
	return result;
}
```

Usb_device.c

```
#include "usb_device.h"
#include "usbd_core.h"
#include "usbd_desc.h"
#include "usbd_cdc.h"
#include "usbd_cdc_if.h"
USBD_HandleTypeDef hUsbDeviceFS;
void MX_USB_DEVICE_Init(void)
{
  USBD_Init(&hUsbDeviceFS, &FS_Desc, DEVICE_FS);
  USBD_RegisterClass(&hUsbDeviceFS, &USBD_CDC);
  USBD_CDC_RegisterInterface(&hUsbDeviceFS, &USBD_Interface_fops_FS);
```

```
  USBD_Start(&hUsbDeviceFS);
}
```

Usbd_cdc_if.c

```
#include "usbd_cdc_if.h"
#include "serial_cmd.h"
#define APP_RX_DATA_SIZE  4
#define APP_TX_DATA_SIZE  4
uint8_t UserRxBufferFS[APP_RX_DATA_SIZE];
uint8_t UserTxBufferFS[APP_TX_DATA_SIZE];
extern USBD_HandleTypeDef hUsbDeviceFS;
static int8_t CDC_Init_FS     (void);
static int8_t CDC_DeInit_FS   (void);
static int8_t CDC_Control_FS  (uint8_t cmd, uint8_t* pbuf, uint16_t length);
static int8_t CDC_Receive_FS  (uint8_t* pbuf, uint32_t *Len);
USBD_CDC_ItfTypeDef USBD_Interface_fops_FS =
{
  CDC_Init_FS,
  CDC_DeInit_FS,
  CDC_Control_FS,
  CDC_Receive_FS
};
static int8_t CDC_Init_FS(void)
{
  USBD_CDC_SetTxBuffer(&hUsbDeviceFS, UserTxBufferFS, 0);
  USBD_CDC_SetRxBuffer(&hUsbDeviceFS, UserRxBufferFS);
```

```
  return (USBD_OK);
static int8_t CDC_DeInit_FS(void)
{
  return (USBD_OK);
}
static int8_t CDC_Control_FS  (uint8_t cmd, uint8_t* pbuf, uint16_t length)
{
  switch (cmd)
  {
  case CDC_SEND_ENCAPSULATED_COMMAND:
    break;
  case CDC_GET_ENCAPSULATED_RESPONSE:
    break;
  case CDC_SET_COMM_FEATURE:
    break;
  case CDC_GET_COMM_FEATURE:
    break;
  case CDC_CLEAR_COMM_FEATURE:
    break;
  case CDC_SET_LINE_CODING:
    break;
  case CDC_GET_LINE_CODING:
    break;
  case CDC_SET_CONTROL_LINE_STATE:
    break;
case CDC_SEND_BREAK:
```

```
    break;
  default:
    break;
  }
  return (USBD_OK);
}
static int8_t CDC_Receive_FS (uint8_t* Buf, uint32_t *Len)
{
        readData(Buf,*Len);
  USBD_CDC_SetRxBuffer(&hUsbDeviceFS, &Buf[0]);
  USBD_CDC_ReceivePacket(&hUsbDeviceFS);
  return (USBD_OK);
}
uint8_t CDC_Transmit_FS(uint8_t* Buf, uint16_t Len)
{
  uint8_t result = USBD_OK;
  USBD_CDC_HandleTypeDef *hcdc =
(USBD_CDC_HandleTypeDef*)hUsbDeviceFS.pClassData;
  if (hcdc->TxState != 0){
    return USBD_BUSY;
  }
  USBD_CDC_SetTxBuffer(&hUsbDeviceFS, Buf, Len);
  result = USBD_CDC_TransmitPacket(&hUsbDeviceFS);
  return result;
}
```

Usbd_desc.c

```
#include "usbd_core.h"

#include "usbd_desc.h"

#include "usbd_conf.h"

#define USBD_VID     1155

#define USBD_LANGID_STRING     1033

#define USBD_MANUFACTURER_STRING     "STMicroelectronics"

#define USBD_PID_FS     22336

#define USBD_PRODUCT_STRING_FS     "STM32 Virtual ComPort"

#define USBD_SERIALNUMBER_STRING_FS     "00000000001A"

#define USBD_CONFIGURATION_STRING_FS     "CDC Config"

#define USBD_INTERFACE_STRING_FS     "CDC Interface"

uint8_t *     USBD_FS_DeviceDescriptor( USBD_SpeedTypeDef speed , uint16_t
*length);

uint8_t *     USBD_FS_LangIDStrDescriptor( USBD_SpeedTypeDef speed , uint16_t
*length);

uint8_t *     USBD_FS_ManufacturerStrDescriptor ( USBD_SpeedTypeDef speed ,
uint16_t *length);

uint8_t *     USBD_FS_ProductStrDescriptor ( USBD_SpeedTypeDef speed , uint16_t
*length);

uint8_t *     USBD_FS_SerialStrDescriptor( USBD_SpeedTypeDef speed , uint16_t
*length);

uint8_t *     USBD_FS_ConfigStrDescriptor( USBD_SpeedTypeDef speed , uint16_t
*length);

uint8_t *     USBD_FS_InterfaceStrDescriptor( USBD_SpeedTypeDef speed , uint16_t
*length);

#ifdef USB_SUPPORT_USER_STRING_DESC

uint8_t *     USBD_FS_USRStringDesc (USBD_SpeedTypeDef speed, uint8_t idx ,
uint16_t *length);
```

```
USBD_DescriptorsTypeDef FS_Desc =
{
  USBD_FS_DeviceDescriptor,
  USBD_FS_LangIDStrDescriptor,
  USBD_FS_ManufacturerStrDescriptor,
  USBD_FS_ProductStrDescriptor,
  USBD_FS_SerialStrDescriptor,
  USBD_FS_ConfigStrDescriptor,
  USBD_FS_InterfaceStrDescriptor,
};
#if defined ( __ICCARM__ )
  #pragma data_alignment=4
#endif
__ALIGN_BEGIN uint8_t USBD_FS_DeviceDesc[USB_LEN_DEV_DESC]
__ALIGN_END =
  {
    0x12,
    USB_DESC_TYPE_DEVICE,
    0x00,
    0x02,
    0x02,
    0x02,
    0x00,
    USB_MAX_EP0_SIZE,
    LOBYTE(USBD_VID),
```

```
    HIBYTE(USBD_VID),
    LOBYTE(USBD_PID_FS),
    HIBYTE(USBD_PID_FS),
    0x00,
    0x02,
    USBD_IDX_MFC_STR,
    USBD_IDX_PRODUCT_STR,
    USBD_IDX_SERIAL_STR,
    USBD_MAX_NUM_CONFIGURATION
  } ;
#if defined ( __ICCARM__ )
  #pragma data_alignment=4
#endif
__ALIGN_BEGIN uint8_t USBD_LangIDDesc[USB_LEN_LANGID_STR_DESC]
__ALIGN_END =
{
     USB_LEN_LANGID_STR_DESC,
     USB_DESC_TYPE_STRING,
     LOBYTE(USBD_LANGID_STRING),
     HIBYTE(USBD_LANGID_STRING),
};
#if defined ( __ICCARM__ )
  #pragma data_alignment=4
#endif
__ALIGN_BEGIN uint8_t USBD_StrDesc[USBD_MAX_STR_DESC_SIZ]
__ALIGN_END;
```

```
uint8_t *  USBD_FS_DeviceDescriptor( USBD_SpeedTypeDef speed , uint16_t
*length)

{

  *length = sizeof(USBD_FS_DeviceDesc);

  return USBD_FS_DeviceDesc;

}

uint8_t *  USBD_FS_LangIDStrDescriptor( USBD_SpeedTypeDef speed , uint16_t
*length)

{

  *length =  sizeof(USBD_LangIDDesc);

  return USBD_LangIDDesc;

}

uint8_t *  USBD_FS_ProductStrDescriptor( USBD_SpeedTypeDef speed , uint16_t
*length)

{

  if(speed == 0)

  {

    USBD_GetString (USBD_PRODUCT_STRING_FS, USBD_StrDesc, length);

  }

  else

  {

    USBD_GetString (USBD_PRODUCT_STRING_FS, USBD_StrDesc, length);

  }

  return USBD_StrDesc;

}

uint8_t *  USBD_FS_ManufacturerStrDescriptor( USBD_SpeedTypeDef speed ,
uint16_t *length)
```

```
{
  USBD_GetString (USBD_MANUFACTURER_STRING, USBD_StrDesc, length);
  return USBD_StrDesc;
}
uint8_t *  USBD_FS_SerialStrDescriptor( USBD_SpeedTypeDef speed , uint16_t
*length)
{
  if(speed  == USBD_SPEED_HIGH)
  {
    USBD_GetString (USBD_SERIALNUMBER_STRING_FS, USBD_StrDesc,
length);
  }
  else
  {
    USBD_GetString (USBD_SERIALNUMBER_STRING_FS, USBD_StrDesc,
length);
  }
  return USBD_StrDesc;
}
uint8_t *  USBD_FS_ConfigStrDescriptor( USBD_SpeedTypeDef speed , uint16_t
*length)
{
  if(speed  == USBD_SPEED_HIGH)
  {
    USBD_GetString (USBD_CONFIGURATION_STRING_FS, USBD_StrDesc,
length);
  }
  else
```

```
  {
    USBD_GetString (USBD_CONFIGURATION_STRING_FS, USBD_StrDesc,
length);
  }
  return USBD_StrDesc;
}
uint8_t *  USBD_FS_InterfaceStrDescriptor( USBD_SpeedTypeDef speed , uint16_t
*length)
{
  if(speed == 0)
  {
    USBD_GetString (USBD_INTERFACE_STRING_FS, USBD_StrDesc, length);
  }
  else
  {
    USBD_GetString (USBD_INTERFACE_STRING_FS, USBD_StrDesc, length);
  }
  return USBD_StrDesc;
}
```

Usbd_conf.c

```
#include "stm32l0xx.h"
#include "stm32l0xx_hal.h"
#include "usbd_def.h"
#include "usbd_core.h"
#include "usbd_cdc.h"
```

```
PCD_HandleTypeDef hpcd_USB_FS;
void Error_Handler(void);
static void SystemClockConfig_Resume(void);
/* USER CODE END 1 */
void HAL_PCDEx_SetConnectionState(PCD_HandleTypeDef *hpcd, uint8_t state);
extern void SystemClock_Config(void);
void HAL_PCD_MspInit(PCD_HandleTypeDef* pcdHandle)
{
  if(pcdHandle->Instance==USB)
  {
    __HAL_RCC_USB_CLK_ENABLE();

    HAL_NVIC_SetPriority(USB_IRQn, 0, 0);
    HAL_NVIC_EnableIRQ(USB_IRQn);
  }
}
void HAL_PCD_MspDeInit(PCD_HandleTypeDef* pcdHandle)
{
  if(pcdHandle->Instance==USB)
  {
    __HAL_RCC_USB_CLK_DISABLE();
    HAL_NVIC_DisableIRQ(USB_IRQn);
  }
}
void HAL_PCD_SetupStageCallback(PCD_HandleTypeDef *hpcd)
{
```

```
  USBD_LL_SetupStage((USBD_HandleTypeDef*)hpcd->pData, (uint8_t *)hpcd-
>Setup);

}

void HAL_PCD_DataOutStageCallback(PCD_HandleTypeDef *hpcd, uint8_t epnum)

{

  USBD_LL_DataOutStage((USBD_HandleTypeDef*)hpcd->pData, epnum, hpcd-
>OUT_ep[epnum].xfer_buff);

}

void HAL_PCD_DataInStageCallback(PCD_HandleTypeDef *hpcd, uint8_t epnum)

{

  USBD_LL_DataInStage((USBD_HandleTypeDef*)hpcd->pData, epnum, hpcd-
>IN_ep[epnum].xfer_buff);

}

void HAL_PCD_SOFCallback(PCD_HandleTypeDef *hpcd)

{

  USBD_LL_SOF((USBD_HandleTypeDef*)hpcd->pData);

}

void HAL_PCD_ResetCallback(PCD_HandleTypeDef *hpcd)

{

  USBD_SpeedTypeDef speed = USBD_SPEED_FULL;

  switch (hpcd->Init.speed)

 {

  case PCD_SPEED_FULL:

    speed = USBD_SPEED_FULL;

    break;

  default:

    speed = USBD_SPEED_FULL;
```

```
    break;
  }
  USBD_LL_SetSpeed((USBD_HandleTypeDef*)hpcd->pData, speed);
  USBD_LL_Reset((USBD_HandleTypeDef*)hpcd->pData);
}
void HAL_PCD_SuspendCallback(PCD_HandleTypeDef *hpcd)
{
  USBD_LL_Suspend((USBD_HandleTypeDef*)hpcd->pData);
  if (hpcd->Init.low_power_enable)
  {
    SCB->SCR |= (uint32_t)((uint32_t)(SCB_SCR_SLEEPDEEP_Msk |
SCB_SCR_SLEEPONEXIT_Msk));
  }
}
void HAL_PCD_ResumeCallback(PCD_HandleTypeDef *hpcd)
{
  if (hpcd->Init.low_power_enable)
  {
    SCB->SCR &= (uint32_t)~((uint32_t)(SCB_SCR_SLEEPDEEP_Msk |
SCB_SCR_SLEEPONEXIT_Msk));
    SystemClockConfig_Resume();
  }
  USBD_LL_Resume((USBD_HandleTypeDef*)hpcd->pData);
}
void HAL_PCD_ISOOUTIncompleteCallback(PCD_HandleTypeDef *hpcd, uint8_t
epnum)
{
```

```
  USBD_LL_IsoOUTIncomplete((USBD_HandleTypeDef*)hpcd->pData, epnum);
}
void HAL_PCD_ISOINIncompleteCallback(PCD_HandleTypeDef *hpcd, uint8_t
epnum)
{
  USBD_LL_IsoINIncomplete((USBD_HandleTypeDef*)hpcd->pData, epnum);
}
void HAL_PCD_ConnectCallback(PCD_HandleTypeDef *hpcd)
{
  USBD_LL_DevConnected((USBD_HandleTypeDef*)hpcd->pData);
}
void HAL_PCD_DisconnectCallback(PCD_HandleTypeDef *hpcd)
{
  USBD_LL_DevDisconnected((USBD_HandleTypeDef*)hpcd->pData);
}
USBD_StatusTypeDef  USBD_LL_Init (USBD_HandleTypeDef *pdev)
{
  hpcd_USB_FS.pData = pdev;
  pdev->pData = &hpcd_USB_FS;
  hpcd_USB_FS.Instance = USB;
  hpcd_USB_FS.Init.dev_endpoints = 8;
  hpcd_USB_FS.Init.speed = PCD_SPEED_FULL;
  hpcd_USB_FS.Init.ep0_mps = DEP0CTL_MPS_64;
  hpcd_USB_FS.Init.phy_itface = PCD_PHY_EMBEDDED;
  hpcd_USB_FS.Init.low_power_enable = DISABLE;
  hpcd_USB_FS.Init.lpm_enable = DISABLE;
```

```
  hpcd_USB_FS.Init.battery_charging_enable = DISABLE;
  if (HAL_PCD_Init(&hpcd_USB_FS) != HAL_OK)
  {
    Error_Handler();
  }
  HAL_PCDEx_PMAConfig((PCD_HandleTypeDef*)pdev->pData , 0x00 ,
PCD_SNG_BUF, 0x18);
  HAL_PCDEx_PMAConfig((PCD_HandleTypeDef*)pdev->pData , 0x80 ,
PCD_SNG_BUF, 0x58);
  HAL_PCDEx_PMAConfig((PCD_HandleTypeDef*)pdev->pData , 0x81 ,
PCD_SNG_BUF, 0xC0);
  HAL_PCDEx_PMAConfig((PCD_HandleTypeDef*)pdev->pData , 0x01 ,
PCD_SNG_BUF, 0x110);
  HAL_PCDEx_PMAConfig((PCD_HandleTypeDef*)pdev->pData , 0x82 ,
PCD_SNG_BUF, 0x100);
  return USBD_OK;
}
USBD_StatusTypeDef  USBD_LL_DeInit (USBD_HandleTypeDef *pdev)
{
  HAL_StatusTypeDef hal_status = HAL_OK;
  USBD_StatusTypeDef usb_status = USBD_OK;
  hal_status = HAL_PCD_DeInit(pdev->pData);
  switch (hal_status) {
    case HAL_OK :
      usb_status = USBD_OK;
    break;
    case HAL_ERROR :
      usb_status = USBD_FAIL;
```

```
    break;
    case HAL_BUSY :
      usb_status = USBD_BUSY;
    break;
    case HAL_TIMEOUT :
      usb_status = USBD_FAIL;
    break;
    default :
      usb_status = USBD_FAIL;
    break;
  }
  return usb_status;
}
USBD_StatusTypeDef  USBD_LL_Start(USBD_HandleTypeDef *pdev)
{
  HAL_StatusTypeDef hal_status = HAL_OK;
  USBD_StatusTypeDef usb_status = USBD_OK;
  hal_status = HAL_PCD_Start(pdev->pData);
  switch (hal_status) {
    case HAL_OK :
      usb_status = USBD_OK;
    break;
    case HAL_ERROR :
      usb_status = USBD_FAIL;
    break;
    case HAL_BUSY :
```

```
    usb_status = USBD_BUSY;
   break;
   case HAL_TIMEOUT :
    usb_status = USBD_FAIL;
   break;
   default :
    usb_status = USBD_FAIL;
   break;
  }
  return usb_status;
}
USBD_StatusTypeDef  USBD_LL_Stop (USBD_HandleTypeDef *pdev)
{
  HAL_StatusTypeDef hal_status = HAL_OK;
  USBD_StatusTypeDef usb_status = USBD_OK;
  hal_status = HAL_PCD_Stop(pdev->pData);
  switch (hal_status) {
   case HAL_OK :
    usb_status = USBD_OK;
   break;
   case HAL_ERROR :
    usb_status = USBD_FAIL;
   break;
   case HAL_BUSY :
    usb_status = USBD_BUSY;
   break;
```

```
    case HAL_TIMEOUT :
      usb_status = USBD_FAIL;
    break;
    default :
      usb_status = USBD_FAIL;
    break;
  }
  return usb_status;
}
USBD_StatusTypeDef  USBD_LL_OpenEP  (USBD_HandleTypeDef *pdev,
                                     uint8_t  ep_addr,
                                     uint8_t  ep_type,
                                     uint16_t ep_mps)
{
  HAL_StatusTypeDef hal_status = HAL_OK;
  USBD_StatusTypeDef usb_status = USBD_OK;
  hal_status = HAL_PCD_EP_Open(pdev->pData,
                               ep_addr,
                               ep_mps,
                               ep_type);
  switch (hal_status) {
    case HAL_OK :
      usb_status = USBD_OK;
    break;
    case HAL_ERROR :
      usb_status = USBD_FAIL;
```

```
    break;
    case HAL_BUSY :
      usb_status = USBD_BUSY;
    break;
    case HAL_TIMEOUT :
      usb_status = USBD_FAIL;
    break;
    default :
      usb_status = USBD_FAIL;
    break;
  }
  return usb_status;
}
USBD_StatusTypeDef  USBD_LL_CloseEP (USBD_HandleTypeDef *pdev, uint8_t
ep_addr)
{
  HAL_StatusTypeDef hal_status = HAL_OK;
  USBD_StatusTypeDef usb_status = USBD_OK;
  hal_status = HAL_PCD_EP_Close(pdev->pData, ep_addr);
  switch (hal_status) {
    case HAL_OK :
      usb_status = USBD_OK;
    break;
    case HAL_ERROR :
      usb_status = USBD_FAIL;
    break;
```

```
    case HAL_BUSY :
      usb_status = USBD_BUSY;
    break;
    case HAL_TIMEOUT :
      usb_status = USBD_FAIL;
    break;
    default :
      usb_status = USBD_FAIL;
    break;
  }
  return usb_status;
}
USBD_StatusTypeDef  USBD_LL_FlushEP (USBD_HandleTypeDef *pdev, uint8_t
ep_addr)
{
  HAL_StatusTypeDef hal_status = HAL_OK;
  USBD_StatusTypeDef usb_status = USBD_OK;
  hal_status = HAL_PCD_EP_Flush(pdev->pData, ep_addr);
  switch (hal_status) {
    case HAL_OK :
      usb_status = USBD_OK;
    break;
    case HAL_ERROR :
      usb_status = USBD_FAIL;
    break;
    case HAL_BUSY :
```

```
    usb_status = USBD_BUSY;
   break;
   case HAL_TIMEOUT :
    usb_status = USBD_FAIL;
   break;
   default :
    usb_status = USBD_FAIL;
   break;
  }
  return usb_status;
}
USBD_StatusTypeDef  USBD_LL_StallEP (USBD_HandleTypeDef *pdev, uint8_t
ep_addr)
{
  HAL_StatusTypeDef hal_status = HAL_OK;
  USBD_StatusTypeDef usb_status = USBD_OK;
  hal_status = HAL_PCD_EP_SetStall(pdev->pData, ep_addr);
  switch (hal_status) {
   case HAL_OK :
    usb_status = USBD_OK;
   break;
   case HAL_ERROR :
    usb_status = USBD_FAIL;
   break;
   case HAL_BUSY :
    usb_status = USBD_BUSY;
```

```
    break;
    case HAL_TIMEOUT :
      usb_status = USBD_FAIL;
    break;
    default :
      usb_status = USBD_FAIL;
    break;
  }
  return usb_status;
}
USBD_StatusTypeDef  USBD_LL_ClearStallEP (USBD_HandleTypeDef *pdev,
uint8_t ep_addr)
{
  HAL_StatusTypeDef hal_status = HAL_OK;
  USBD_StatusTypeDef usb_status = USBD_OK;
  hal_status = HAL_PCD_EP_ClrStall(pdev->pData, ep_addr);
  switch (hal_status) {
    case HAL_OK :
      usb_status = USBD_OK;
    break;
    case HAL_ERROR :
      usb_status = USBD_FAIL;
    break;
    case HAL_BUSY :
      usb_status = USBD_BUSY;
    break;
```

```
    case HAL_TIMEOUT :
      usb_status = USBD_FAIL;
    break;
    default :
      usb_status = USBD_FAIL;
    break;
  }
  return usb_status;
}
uint8_t USBD_LL_IsStallEP (USBD_HandleTypeDef *pdev, uint8_t ep_addr)
{
  PCD_HandleTypeDef *hpcd = (PCD_HandleTypeDef*) pdev->pData;
  if((ep_addr & 0x80) == 0x80)
  {
    return hpcd->IN_ep[ep_addr & 0x7F].is_stall;
  }
  else
  {
    return hpcd->OUT_ep[ep_addr & 0x7F].is_stall;
  }
}
USBD_StatusTypeDef  USBD_LL_SetUSBAddress (USBD_HandleTypeDef *pdev,
uint8_t dev_addr)
{
  HAL_StatusTypeDef hal_status = HAL_OK;
  USBD_StatusTypeDef usb_status = USBD_OK;
```

```
  hal_status = HAL_PCD_SetAddress(pdev->pData, dev_addr);
  switch (hal_status) {
    case HAL_OK :
      usb_status = USBD_OK;
    break;
    case HAL_ERROR :
      usb_status = USBD_FAIL;
    break;
    case HAL_BUSY :
      usb_status = USBD_BUSY;
    break;
    case HAL_TIMEOUT :
      usb_status = USBD_FAIL;
    break;
    default :
      usb_status = USBD_FAIL;
    break;
  }
  return usb_status;
}
USBD_StatusTypeDef  USBD_LL_Transmit (USBD_HandleTypeDef *pdev,
                                      uint8_t  ep_addr,
                                      uint8_t  *pbuf,
                                      uint16_t  size)
{
  HAL_StatusTypeDef hal_status = HAL_OK;
```

```
  USBD_StatusTypeDef usb_status = USBD_OK;
  hal_status = HAL_PCD_EP_Transmit(pdev->pData, ep_addr, pbuf, size);
  switch (hal_status) {
    case HAL_OK :
      usb_status = USBD_OK;
    break;
    case HAL_ERROR :
      usb_status = USBD_FAIL;
    break;
    case HAL_BUSY :
      usb_status = USBD_BUSY;
    break;
    case HAL_TIMEOUT :
      usb_status = USBD_FAIL;
    break;
    default :
      usb_status = USBD_FAIL;
    break;
  }
  return usb_status;
}
USBD_StatusTypeDef  USBD_LL_PrepareReceive(USBD_HandleTypeDef *pdev,
                                           uint8_t  ep_addr,
                                           uint8_t  *pbuf,
                                           uint16_t  size)
{
```

```
  HAL_StatusTypeDef hal_status = HAL_OK;
  USBD_StatusTypeDef usb_status = USBD_OK;
  hal_status = HAL_PCD_EP_Receive(pdev->pData, ep_addr, pbuf, size);
  switch (hal_status) {
    case HAL_OK :
      usb_status = USBD_OK;
    break;
    case HAL_ERROR :
      usb_status = USBD_FAIL;
    break;
    case HAL_BUSY :
      usb_status = USBD_BUSY;
    break;
    case HAL_TIMEOUT :
      usb_status = USBD_FAIL;
    break;
    default :
      usb_status = USBD_FAIL;
    break;
  }
  return usb_status;
}
uint32_t USBD_LL_GetRxDataSize  (USBD_HandleTypeDef *pdev, uint8_t  ep_addr)
{
  return HAL_PCD_EP_GetRxCount((PCD_HandleTypeDef*) pdev->pData, ep_addr);
}
```

```
void  USBD_LL_Delay (uint32_t Delay)
{
  HAL_Delay(Delay);
}
void *USBD_static_malloc(uint32_t size)
{
  static uint32_t mem[(sizeof(USBD_CDC_HandleTypeDef)/4)+1];/* On 32-bit boundary */
  return mem;
}
void USBD_static_free(void *p)
{
}
static void SystemClockConfig_Resume(void)
{
  SystemClock_Config();
}
void HAL_PCDEx_SetConnectionState(PCD_HandleTypeDef *hpcd, uint8_t state)
{
  if (state == 1)
  {
  }
  else
  {
  }
}
```

7.4. Ek-4. PC Yazılımı

```
QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets serialport

TARGET = DataLogger
TEMPLATE = app
CONFIG += c++11

SOURCES += main.cpp\
        mainwindow.cpp \
    settingsdialog.cpp

HEADERS  += mainwindow.h \
    settingsdialog.h

FORMS    += mainwindow.ui \
    settingsdialog.ui

RESOURCES += \
    resources/resources.qrc \
    qdarkstyle/style.qrc

DISTFILES += \
    myapp.rc

RC_FILE = myapp.rc
```

8. ÖZGEÇMİŞ

Kişisel Bilgiler

Adı Soyadı : Semih GÜZEL
Uyruğu : Türkiye Cumhuriyeti
Medeni Hali : Bekar
Telefon : 0530 821 09 95
E-posta : semih.gozel@hotmail.com.tr
Yabancı dil : İngilizce

Eğitim

Derece	Alan	Eğitim Birimi	Mezuniyet
Yüksek Lisans	Mekatronik Mühendisliği A.B.D.	Gaziosmanpaşa Üniversitesi - Fen Bilimleri Enstitüsü	2018
Lisans	Mekatronik Mühendisliği	Gaziosmanpaşa Üniversitesi - Mühendislik ve Doğa Bilimleri Fakültesi	2014
Lise	Fen Bilimleri	Çardak Anadolu Lisesi	2010

İş Deneyimi

Yıl	Yer	Görev
2017-	Mustafa Kemal Üniversitesi, Hassa Meslek Yüksekokulu, Elektronik ve Otomasyon Bölümü, Biyomedikal Cihaz Teknolojisi Pr.	Öğretim Görevlisi
2015-2017	Arma Metal Dış Tic. Ltd. Şti.	Mekatronik Mühendisi
2014-2015	Buğlem Asansör Kontrol Muayene Belgelendirme Eğitim ve Gözetim Ltd. Şti.	Mekatronik Mühendisi