

Pamukkale Üniversitesi, Mühendislik Fakültesi  
Bilgisayar Mühendisliği Bölümü

*Lisans Bitirme Tezi*

**Elektrik Sistemlerinin   
Uzaktan Denetimi ve Yönetimi: intelliPWR**

*Tarafından*

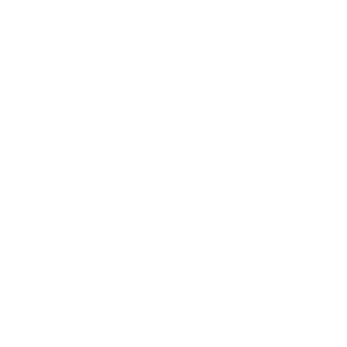
**Veysel Berk ALTUN   
13253004**

*Denetleyen*

**Yrd. Doç. Dr. Elif HAYTAOĞLU**

Mayıs 21, 2018

Denizli



Pamukkale Üniversitesi, Mühendislik Fakültesi  
Bilgisayar Mühendisliği Bölümü

*Lisans Bitirme Tezi*

**Elektrik Sistemlerinin   
Uzaktan Denetimi ve Yönetimi: intelliPWR**

*Tarafından*

**Veysel Berk ALTUN  
13253004**

*Denetleyen*

**Yrd. Doç. Dr. Elif HAYTAOĞLU**

Mayıs 21, 2018

Denizli

**LİSANS TEZİ ONAY FORMU**

Veysel Berk ALTUN tarafından Yrd. Doç. Dr. Elif HAYTAOĞLU yönetiminde hazırlanan “**Elektrik Sistemlerinin Uzaktan Denetimi ve Yönetimi: intelliPWR**” başlıklı tez tarafımızdan okunmuş, kapsamı ve niteliği açısından bir lisans tezi olarak kabul edilmiştir.

Yrd. Doç. Dr. Elif HAYTAOĞLU  
*Danışman*

|  |  |
| --- | --- |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Jüri Üyesi | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Jüri Üyesi |

Pamukkale Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölüm Kurulu’nun   
\_\_\_\_\_ / \_\_\_\_\_ / \_\_\_\_\_ tarih ve \_\_\_\_\_\_\_\_\_\_ sayılı kararıyla onaylanmıştır.

**Prof. Dr. Sezai Tokat***Bölüm Başkanı*

Bu tezin tasarımı, hazırlanması, yürütülmesi, araştırmalarının yapılması ve bulgularının analizlerinde bilimsel etiğe ve akademik kurallara özenle riayet edildiğini; bu çalışmanın doğrudan birincil ürünü olmayan bulguların, verilerin ve materyallerin bilimsel etiğe uygun olarak kaynak gösterildiğini ve alıntı yapılan çalışmalara atfedildiğine beyan ederim.

Veysel Berk ALTUN  
 Mayıs 2018

ÖNSÖZ

Tez çalışmam sırasında kıymetli bilgi, birikim ve tecrübeleri ile bana yol gösterici ve destek olan değerli danışman hocam sayın *Yrd. Doç. Dr. Elif HAYTAOĞLU*’na, ilgisini ve önerilerini göstermekten kaçınmayan Bilgisayar Mühendisliği Bölümü Ana Bilim Dalı Başkanı sayın *Prof. Dr. Sezai Tokat*’a sonsuz teşekkür ve saygılarımı sunarım.

Her türlü maddi ve manevi destekleriyle beni hiçbir zaman yalnız bırakmayan *Gülyesa ALTUN*’a ve aileme sonsuz teşekkür ederim.

Son olarak yardımlarını hiç esirgemeyen değerli arkadaşlarım *Gülşah ARAS*’a, *Osman YILMAZTÜRK*’e, *Egemen AYHAN*’a ve *Ahmet YILMAZ*’a da teşekkürlerimi bir borç bilirim.

Veysel Berk ALTUN  
 Mayıs 2018

**ÖZET**

Elektriğin ulaştığı her alanda hayata geçirilmesi planlanan bu projede günümüzün priz temelleri yeniden ele alınmış ve tasarımı konusunda inovatif bir fikir ortaya atılmıştır. Yeniden tasarlanan bu prizler ile gelecekte üretilecek olan yeni nesil akıllı ev aygıtlarının bu prizler ile haberleşmesi sağlanmış ve bu akıllı ev aygıtlarının prizler yardımıyla uzaktan denetimi ve yönetimi gerçekleştirilmiştir.

Ortaya atılan bu yeni priz tasarımı fikrinde, priz soketlerinin faz ve nötr pimlerini altına 5 adet küçük bağlantı pimi eklenmiş ve bu pimler yardımı ile prize bağlanan aygıtların geçerli priz ile haberleşmesi amaçlanmıştır.

Tezin Ar–Ge sürecinde kablolu haberleşme süreçlerinde kullanılması amacıyla *RaspBerry Pi 3*, *Arduino Mega*, *Arduino Pro Mini* ve *NodeMCU* platformları ile uyumlu 3 farklı kütüphane yazılmıştır. Bütün bu kütüphanelerin yazım aşamasında *C*, *C++* ve *Phyton* dilleri tercih edilmiş; daha kolay bir Ar–Ge süreci için ise *GIT* versiyon kontrol sistemi kullanılmıştır. Tezin kablosuz haberleşme süreçlerinde ise açık kaynak kodlu olan ve halen geliştirilmeye devam edilen *openHAB* sistemi tercih edilmiştir.

Sonuç olarak bütün proje kapsamında temel bilgisayar mühendisliği etiğine ve kurallarına uygun olarak fonksiyonel ve modüler toplam 4400 satır kod yazılmış ve çağımıza uygun yeni nesil bir priz tasarımı ortaya konmuştur.

**GİRİŞ**

Günümüzde insanlığın elektriğe olan ihtiyacını bir bitkinin suya olan ihtiyacı ile benzer şekilde tanımlayabiliriz. Bir bitki su olmadan nasıl en temel faaliyetlerini gerçekleştiremiyorsa, insanlar da artık elektrik olmadan yaşayamıyor ve gündelik faaliyetlerinin çoğunu gerçekleştiremiyorlar. Elektriğin insan hayatında bu derece önemli olduğu 21. Yüzyılda, kullanım alanları ve popülaritesi gün geçtikçe artan *“Nesnelerin Interneti – Iot (Internet of Things)”* ile yapılar içerisindeki elektrik kaynaklarına artık uzaktan erişebilmekte ve yönetebilmekteyiz.

İngilizce karşılığı *Internet of Things* olan *Nesnelerin İnterneti*, ilk kez *1999* yılında Britanyalı teknoloji öncüsü *Kevin Ashton* tarafından ortaya atılmış bir terimdir. En genel tanımıyla fiziksel aygıt, sensör, araç ve elektronik aletlerin, sahip oldukları yazılımsal ve donanımsal kaynaklar ile bir ağ üzerinden birbirleri ile haberleşmeleri olarak tanımlanabilir. Benzer biçimde çeşitli haberleşme protokolleri sayesinde birbirleri ile haberleşen ve birbirine bağlanarak akıllı bir ağ oluşturmuş aygıtlar sistemi olarak da tanımlamak mümkündür.

Nesnelerin İnternetine alternatif olarak günümüzde birçok benzer kavramlar vardır; ancak Nesnelerin İnterneti, bu fenomeni açıklamak için en popüler terimdir.

Günümüzde basit bir *IoT* aygıtının internet ile haberleşmesi aşamasında kullanılan en yaygın yöntem *Wi–Fi* teknolojisi olarak tanımlanabilir. Herhangi bir üreticinin *IoT* kavramına uygun bir aygıt üretmeyi planlaması durumunda, *Wi–Fi* teknolojisi üzerine bir Ar–Ge yapması kaçınılmazdır. Benzer biçimde üretmeyi planladığı yeni nesil bir akıllı ev aygıtına *Wi–Fi* teknoloji destekleyen bir donanım eklemesi gerekmektedir. Bütün bu süreçler ise bir üreticiye ek bir kaynak tüketimi oluşturur.

Tam da bu noktada, üreticilerin kaynak tüketimini daha aza indirmek amacıyla inovatif bir çözüm arayışı doğmuştur. Literatürde yapılan uzun araştırmalar sonucu, donanımlar arası *Wi–Fi* haberleşme altyapısını gruplar halinde gerçekleştirecek bir öneri ortaya atılmıştır.

Bu aşama bir *IoT* aygıtının elektrik olmadan çalışamayacağı varsayılarak, gruplar halinde *Wi–Fi* haberleşme teknolojisini sağlayacak olan yeni nesil priz tasarımı ortaya atılmıştır. Bu prizler ile üreticilerin *Wi–Fi* teknoloji ya da benzer herhangi bir haberleşme altyapısına ihtiyaç duymadan, aynı haberleşme sürecinin daha düşük bir bütçe ile gerçekleştirebilmesi amaçlanmıştır.

**ALTYAPI**

Tezin uygulama süreçlerinde yapılan testlerin ve uygulamaların tamamında *Ubuntu* işletim sisteminin *14.04* sürümü tercih edilmiştir. Bu nedenle örnek bir uygulama sürecinde yapılacak olan bütün uygulamaların, bu işletim sisteminin daha eski sürümlerinde yapılacak olan uygulamalar ile aynı sonucu vereceği kesinlikle teyit edilemez.

Bir diğer önemli konu, *MQTT* haberleşme sürecinde zorunlu olarak gereken aktif internet bağlantısı üzerinedir. Bu aşamada *port blocking* özelliği olmayan *WPA2 Personal* kimlik doğrulamalı bir bireysel ağ yapısı tercih edilmiştir. Tezin uygulama süreçlerinin güvenlik üzerine yeterli bir Ar–Ge çalışması içermemesinden dolayı kamuya açık olan ağların bu tez sürecinde kullanılması önerilemez. Tercihen kablolu internet bağlantısı da alternatif olarak kullanılabilir.

*Ubuntu* işletim sisteminin kurulumu hakkında detaylı bilgi için;  
<https://www.ubuntu.com/download>

**ARDUINO**

*Arduino* platformundaki bütün uygulama geliştirme süreçlerinde *Arduino IDE* yazılımının *1.8.5* sürümü tercih edilmiştir.

*Arduino IDE* yazılımının kurulumu hakkında detaylı bilgi için;  
<https://www.arduino.cc/en/Main/Software>

**NodeMCU**

*NodeMCU* platformu her ne kadar *Lua* *Scripting* programlama dilini benimsemiş olsa da uygun yapılandırmalar ile *NodeMCU* platformunun bütün uygulama geliştirme süreçleri *Arduino IDE* yazılımıyla da yapılabilir.

*NodeMCU* platformunun *Arduino IDE* kurulumu hakkında detaylı bilgi için; <https://github.com/esp8266/Arduino>

**openHAB**

*openHAB* yazılımının *Ubuntu* işletim sisteminde kararlı çalışabilmesi için işletim sistemine uygun bir *Java SE Runtime Environment* dağıtımı yazılım gerekmektedir. Buna ek olarak *Java SE Runtime Environment* yazılımın *1.9.x* sürümleri *openHAB* ile kararlı bir şekilde çalışmadığından, bu yazılımın *1.8.x* sürümleri kullanıcılar için resmi olarak önerilmektedir.

Tezin uygulama süreçlerindeki uzak bağlantılar için *openHAB Foundation* tarafından barındırılan *openHAB* *Cloud* altyapısı kullanılmıştır. Bu altyapının, *openHAB* yazılımı ile kararlı çalışabilmesi için bir takım yapılandırma ayarlarının yapılması gerekmektedir.

|  |  |
| --- | --- |
| **Dosya** | **Kurulum** |
| UUID | /var/lib/openhab2/uuid |
| Secret | /var/lib/openhab2/openhabcloud/secret |

*openHAB Cloud* ile kimlik doğrulaması için yerel *openHAB* yazılımı, *openHAB Cloud* servisinin hesap ayarlarınızda girilmesi gereken iki değer üretir. İlki, çalışma zamanınızı tanımlamaya izin veren benzersiz bir kimlik tanımlayıcıdır. İkincisi, şifre olarak hizmet sunan rastgele bir gizli anahtardır. Her iki değer de yerel dosya sistemine yazılır.

Bu dosyaları bir sebepten dolayı hasar görmesi durumunda, *openHAB* yazılımı otomatik olarak yenilerini oluşturur. Daha sonra *UUID* ve *SECRET* değerlerini *openHAB Cloud* hizmetinde yeniden yapılandırmanız gerekecektir.

*openHAB* yazılımının kurulumu hakkında detaylı bilgi için;   
<https://docs.openhab.org/installation>

*openHAB Cloud Connector* hakkında detaylı bilgi için; <https://docs.openhab.org/addons/ios/openhabcloud/readme>

*MQTT Binding* hakkında detaylı bilgi için; [https://docs.openhab.org/addons/bindings/mqtt/readme](https://docs.openhab.org/addons/bindings/mqtt1/readme)

**MQTT**

Günümüzde IoT platformlarında *Mosquitto*, *Kafka*, *RabbitMQ*, *emqttd* ve *VerneMQ* olmak üzere tercih edilebilecek en az *5* adet *MQTT* yazılımı bulunmaktadır. Bu tezin uygulama süreçlerinde kurulumu ve kullanımı en kolay olan *Mosquitto* *MQTT* yazılımı tercih edilmiştir.

*openHAB* yazılımının bir *MQTT* istemcisi olarak çalışmasını sağlamak için bir takım yapılandırma ayarlarının yapılması gerekmektedir. Bu yapılandırmalar *openHAB* yazılımına *MQTT* işlevselliği kazandırmaz, bunun için *Mosquitto* veya bir başka altyapı yazılımının kullanılması gerekmektedir.

|  |  |  |
| --- | --- | --- |
| **Özellik** | **Varsayılan** | **Zorunlu** |
| *<broker>*.url | – | Evet |
| *<broker>*.clientId | Rassal | Hayır |
| *<broker>*.user | – | Evet |
| *<broker>*.pwd | – | Evet |
| *<broker>*.qos | 0 | Hayır |
| *<broker>*.retain | Yanlış | Hayır |
| *<broker>*.async | Doğru | Hayır |
| *<broker>*.keepAlive | 60 | Hayır |
| *<broker>*.allowLongerClientIds | Yanlış | Hayır |

Bir *MQTT* yayıncısına mesaj göndermek veya yayınlamak için, *services/mqtt.cfg* dosyasına kullanılması planlanan tüm aracıların bildirimlerinin yapılması gerekmektedir.

*Mosquitto* yazılımının kurulumu hakkında detaylı bilgi için;   
<https://mosquitto.org/download>

**DONANIM**

**ARDUINO PRO MINI**

*Arduino Pro Mini*, *ATmega328* tabanlı bir *Arduino* geliştirme kartıdır. Bu geliştirme kartı üzerinde *14* adet dijital giriş – çıkış pimi (bu pimlerden *6* tanesi *PWM* çıkışı), *6* adet analog giriş – çıkış, bir adet kristal osilatör (*8 MHz* ya da *16 MHz*) ve bir adet *reset* butonu bulundurur. Sahip olduğu entegre voltaj regülatörü ile *12V* gerilime kadar voltaj beslemelerine izin vermektedir. Regüle edilmemiş voltaj beslemesi *RAW* piminden yapılmaktadır.

*Arduino Pro Mini* geliştirme kartının *3.3V 8 MHz* ve *5V 16 MHz* olmak üzere iki farklı versiyonu bulunmaktadır.



Tezin uygulama süreçlerinde, kablolu haberleşmelerin tamamında *I²C* protokolü kullanılmıştır. *I²C* protokolü, *UART* veya *SPI* protokollerinden farklı olarak *open–drain* yapıya sahiptir. Sıradan bir *open–drain* hattında geçerli pimin çıkış bacağı *P–MOSFET* ile *N–MOSFET* devre elemanları arasında bulunmaktadır. *open–drain* durumuna getirilmiş bir bağlantı hattında *MOSFET* elemanının *source* hattı toprağa bağlıdır, *gate* içeriden sürülmüş durumdadır ve *drain* açıktadır. Bu aşamada faz ile geçerli bu pim arasında sonsuz empedans oluşur. Bu durumda pimi *high* konumuna getirebilmek için dışarıdan uygun değerlikte bir *pull–up* direncinin eklenmesi gerekmektedir. Bu özellikle, çıkışa bağlanacak olan devre elemanının daha fazla akım çekmesi sağlanır ve böylece akım mikroişlemciden değil, *pull–up* direnci üzerinden çekilir.

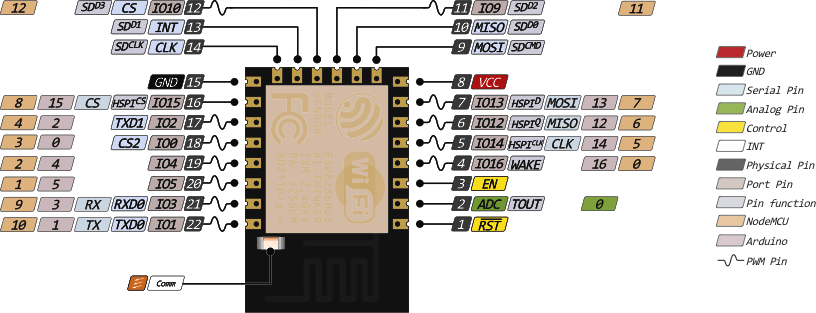
Uygulama sürecinde *I²C* protokolünün bütün *SCL* ve *SDA* hatlarına birer adet *pull–up* direnci bağlanmıştır. Dirençlerin değerleri, yarı iletken üreticisi *Microchip Technology*’nin *ATmega8*, *ATmega168* ve *ATmega328* modelleri için 2016 yılında güncel olarak yayınlamış olduğu kullanım kılavuzundaki formüller ile hesaplanmıştır. Verilen formüller;

Formüllerde verilen değeri *SCL* hattının çalışma frekansını, değeri *pull–up* direncinin sahip olabileceği en küçük direnç değerini, değeri hattın sahip olduğu gerilim değerini, değeri *pull–up* direncinin sahip olabileceği en büyük direnç değerini ve son olarak değeri *I²C* hattında sahip olunan toplam aygıt sayısını temsil eder.

Formüldeki bütün değişkenlerin uygun değerler ile değiştirilmesi sonucu en uygun *pull–up* direnç değeri hesaplanabilir. Bu tezin uygulama süreçlerinde kullanılacak olan bütün *pull–up* direnç değerleri olarak baz alınmıştır.

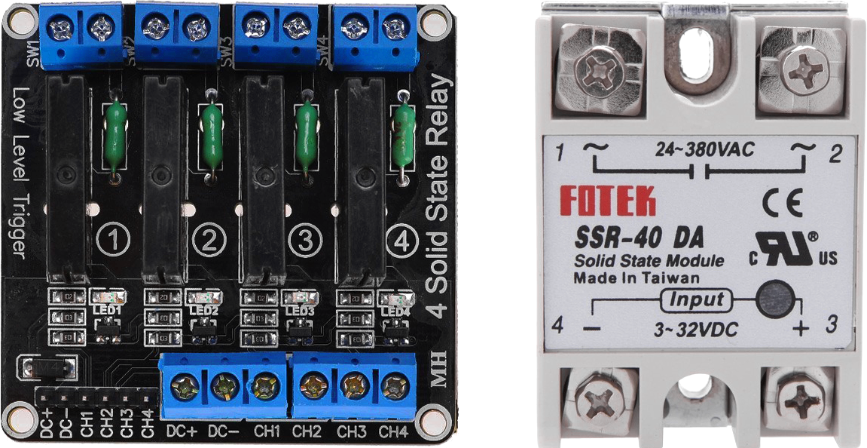
**NodeMCU**

*NodeMCU*, *ESP8266–12* tabanlı bir geliştirme kartıdır. Bu geliştirme kartı üzerinde *17* adet dijital giriş – çıkış pimi (bu pimlerden *9* tanesi *PWM* çıkışı), *1* adet analog giriş – çıkış, bir adet *80 MHz* kristal osilatör, bir adet 802.11 b/g/n *Wi–Fi* modülü ve bir adet *reset* butonu bulundurur. Sahip olduğu entegre voltaj regülatörü ile *12V* gerilime kadar voltaj beslemelerine izin vermektedir.



**FOTEK SSR–40DA**

*SSR (Solid State Relay)*, tamamen elektronik parçalardan oluşturulmuş bir katı hal anahtarlama düzenidir. Klasik röle ve kontaktörle ile aynı işi yapar. Kontaktörler ve röleler gibi kumanda ve güç devrelerine sahiptirler. *SSR*’lar, elektromekanik rölelere göre daha hızlı çalışmaktadır. Anahtarlama zamanları bir *LED*’in açılıp kapanma süresine bağlıdır ve bu süre yaklaşık olarak *1ms* ile *0.5ms* arasındadır. Mekanik bir parçadan oluşmadığı için kullanım ömürleri elektromekanik ve *Reed* rölelere göre daha uzundur. Elektromekanik ve *Reed* rölelerin aksine bağlantıları transistörler ile yapıldığı için *SSR*’ların kontak dirençleri çok daha yüksektir. Ayrıca gelişen teknoloji ile beraber kontak dirençleri sürekli olarak artmaktadır.



Bu tezin uygulama süreçlerinde *FOTEK* tarafından geliştirilen *SSR–40DA* model *SSR* modülü kullanılmıştır. Giriş voltaj değeri *3~32V* ve akım değeri *7.5mA*’dir. Çıkış hattında *40A* değerine kadar çalışabilen bütün devre elemanlarını sürebilir.

Yukarıda verilen bilgiler ışığında, bu tezde oluşturulan elektrik sistemlerinin kaldırabileceği maximum güç değerliğini hesaplayabiliriz. Gücün formülü;

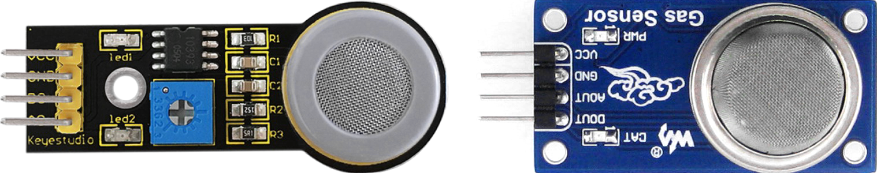
Formüllerde verilen değeri akım değerini, değeri voltaj değerini temsil eder. Formüldeki bütün değişkenlerin uygun değerler ile değiştirilmesi sonucu tez uygulama sürecindeki bir elektrik sisteminin kaldırabileceği maximum güç değerliği hesaplanabilir.

Yapılan hesaplamalar sonucu şehir geriliminde bu SSR modülü, ortalama değerliğe kadar kararlı bir şekilde çalışabilecektir.

**MQ–2: YANICI GAZ VE MQ–7: KARBONMONOKSİT GAZ SENSÖRÜ**

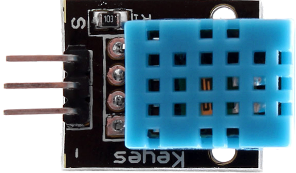
*MQ–2* sensörü, ortamda bulunan ve konsantrasyonu *300~10.000PPM* arasında değişen yanıcı gazı algılayan bir sensör modülüdür. –*100~500°C* arasında çalışabilir ve 5V gerilim üzerinde 150mA akım çeker. Analog çıkışı sayesinde algılanan gaz konsantrasyonu kolayca okunabilir.

*MQ–7* sensörü, ortamda bulunan ve konsantrasyonu *10~10,000PPM* arasında değişen karbonmonoksiti algılayan bir sensör modülüdür. –*10~50°C* arasında çalışabilir ve 5V gerilim üzerinde 150mA akım çeker. Analog çıkışı sayesinde algılanan gaz konsantrasyonu kolayca okunabilir.



**DHT11: ISI VE NEM SENSÖRÜ**

Dijital bir sıcaklık ve nem sensör modülüdür. Çevresindeki havayı ölçmek için içerisindeki kapasitif nem sensörünü ve termistörü kullanır. Bu sensörün okunan verilerini dijital çıkış pimine aktarır. Ek olarak bu sensör, her *2* saniyede bir çıkış verir.



*3~5V* gerilim aralığında, maksimum *2.5mA* akım değerinde *0~50°C* derece sıcaklık için *±2°C* hassasiyet ile çalışmaktadır.

**KÜTÜPHANE**

Tezin uygulama süreçlerinde *C++* dilinde *3* adet kütüphane yazılmıştır. Bu üç kütüphanenin hepsi, haberleşme aşamalarındaki işlem süreçlerini kısaltmak ve kolaylaştırmak amacıyla tezin birçok yazılımsal bölümünde kullanılmıştır.

**SERIALIZER**

*Serializer* kütüphanesi, geliştirme kartları için yazılmış modüler bir çoklu veri paketleme kütüphanesidir. Bu kütüphane, verilen bir veri dizisi önceden belirtilen parametreler ile böler ve çıktı olarak bütün veri dizisinin birleşiminden oluşan tek bir veri dizisi oluşturur. Benzer şekilde bu işlemin tam tersini de gerçekleştirebilir. Kütüphanenin herhangi bir platform da kullanılabilmesi için, projenin ana kod dosyasına ilk olarak *Serializer* kütüphanesinin bildirilmesi gerekir. Bu adım için uygulanacak olan işlem aşağıdaki gibidir;

// Kütüphaneyi sisteme bildir   
#include <Serializer.h>

Kütüphanede kullanıcıların kullanabileceği *2* adet fonksiyonu vardır. Her iki fonksiyon da parametre olarak sırası ile ayırıcı boyutunu, ayırıcı dizisini, girdi dizisi boyutunu ve girdi dizisini alır. Bu fonksiyonlar;

// Verilen bir serileştirilmiş veriyi çözümle  
char \*\*decode(uint16\_t sizeofDelim, char delim[], uint16\_t sizeofData, char givenData[]);

// Verilen bir serileştirilmemiş veriyi birleştir  
char \*encode(uint16\_t sizeofDelim, char delim[], uint16\_t sizeofData, char \*givenData[]);

Aşağıda bu kütüphanenin *decode()* ve *encode()* fonksiyonları ile *Arduino* platformunda çalışan örnek bir kullanımı gösterilmiştir;

void setup() {  
 // Ayırıcı ve veri listesi   
 char \*delim = "\_+";  
 char \*data[] = {"Hello", "World", "Serializer"};

// Birleştir ve sakla  
 char \*resultofEncode = Serialization.encode(2, delim, 3, data);

// Çıktı --->  
 // "Hello\_World+Serializer"  
}

Benzer şekilde *data* verisinin serileştirilmiş çıktısını olan *resultofEncode* verisini *decode()* fonksiyonu ile tekrardan sıralı dizi şeklinde çözümleyebiliriz.

void setup() {  
 // Çözümleme ve sakla  
 char \*\*resultofDecode = Serialization.decode(2, delim, 24, resultofEncode);

// Çıktı --->  
 // {"Hello\_World", "Serializer"}  
}

**MASTERSCANNER**

*MasterScanner* kütüphanesi, *I²C* protokolü destekleyen geliştirme kartları için yazılmış bir veri yolu tarayıcısı kütüphanesidir. Bu kütüphane, bir *I²C* ağındaki *Master* özelliğine sahip olan bir aygıtın, aynı ağ üzerindeki *Slave* aygıtlarını taramasını sağlar ve ağda herhangi bir *Slave* aygıtta değişiklik meydana geldiğinde bu değişikliği kullanıcıya bildirir.

Kütüphanenin herhangi bir platform da kullanılabilmesi için, projenin ana kod dosyasına ilk olarak *MasterScanner* kütüphanesinin bildirilmesi gerekir. Bu adım için uygulanacak olan işlem aşağıdaki gibidir;

// Kütüphaneyi sisteme bildir   
#include <MasterScanner.h>

Kütüphanede kullanıcıların kullanabileceği 1*2* adet fonksiyonu vardır. Bu fonksiyonların kullanımı hakkında kısa bir bilgi aşağıda verilmiştir;

// Tarama aralığı, başlangıç adresi ve bitiş adresi değişkenleri  
bool setRange(uint16\_t intervalMillis, uint8\_t startAddress, uint8\_t stopAddress);  
bool setRange(uint16\_t intervalMillis);  
bool setRange(uint8\_t startAddress, uint8\_t stopAddress);

// Başlangıç adresi ve bitiş adresi aralığını sıfırla  
void resetRange();

// Köle aygıtları tara  
void scanSlaves();

// Köle aygıtlardaki değişiklikleri event-driven olarak dinle  
void onConnectedSlaves(void (\*pointer)(uint8\_t[], byte));  
void onDisconnectedSlaves(void (\*pointer)(uint8\_t[], byte));

// Başlangıç adresini, bitiş adresini veya tarama aralığını döndür  
uint8\_t getStartAddress();  
uint8\_t getStopAddress();  
uint8\_t getIntervalMillis();

// Toplam bağlı aygıt sayısını hesapla  
byte getConnectedSlavesCount();

// Herhangi bir adresin dolu olup olmadığını kontrol et  
bool isConnected(uint8\_t address);

Aşağıda bu kütüphanenin *Arduino* platformunda çalışan örnek bir kullanımı gösterilmiştir;

void setup() {  
 // Arduino aygıtına event-driven tetikleme fonksiyonları bildir  
 MasterScanner.onConnectedSlaves(connectedSlaves);  
 MasterScanner.onDisconnectedSlaves(disconnectedSlaves);  
}

void loop() {  
 // I2C veri yolu hattını tara  
 MasterScanner.scanSlaves();  
}

// I2C veri yoluna herhangi bir yeni aygıt bağlandığında tetiklenir   
void connectedSlaves(uint8\_t data[], byte sizeofData) { }

// I2C veri yolundan herhangi bir aygıt kaldırıldığında tetiklenir  
void disconnectedSlaves(uint8\_t data[], byte sizeofData) { }

*Arduino* ve benzeri geliştirme kartları genellikle *event-driven* olmayan yapıda tasarlanmış geliştirme kartlarıdır. Bu geliştirme kartlarında herhangi bir olayı birkaç istisnai durum dışında kendi başınıza tetikleyemezsiniz. Bu nedenle *Arduino* ve benzeri geliştirme kartlarında herhangi *Slave* aygıttın gelen verileri takip etmek istiyorsanız, Arduino’nun loop*()* yapısı içinde sürekli olarak takip etmek istediğiniz *Slave* aygıtların yazılımsal kontrolünü yapmalısınız.

Yukarıda verilen *MasterScanner* kütüphanesi, *Arduino*’nun *event-driven* olmayan yapısında *event-driven* yapıda çalışacak şekilde yazılmıştır. Yani bir kullanıcı herhangi bir *I²C* ağındaki *Master* özelliğine sahip olan bir aygıta, herhangi bir *Slave* özelliğine sahip aygıt bağladığında ilgili fonksiyonlar bu yazılımda artık tetiklenebilecektir.

Başka bir açıdan bu kütüphanenin benzeri diğer kütüphanelerden en büyük ayırt edici özelliği, *event-driven* olarak yazılmış olmasıdır.

**TIMERQUEUE**

*TimerQueue* kütüphanesi, geliştirme kartları için yazılmış kuyruk yapısına sahip bir zamanlayıcı kütüphanesidir. Belli zaman aralıkları ile çalıştırmak istediğiniz fonksiyonları bu kütüphaneye bildirerek sistem tarafından çalıştırılmasını sağlayabilirsiniz. *TimerQueue* kütüphanesi de *MasterScanner* kütüphanesine benzer bir şekilde *Arduino* ve benzeri geliştirme kartlarında *event-driven* yapıda çalışacak şekilde yazılmıştır.

Kütüphanenin herhangi bir platform da kullanılabilmesi için, projenin ana kod dosyasına ilk olarak *TimerQueue* kütüphanesinin bildirilmesi gerekir. Bu adım için uygulanacak olan işlem aşağıdaki gibidir;

// Kütüphaneyi sisteme bildir   
#include <TimerQueue.h>

Kütüphanede kullanıcıların kullanabileceği 9 adet fonksiyonu vardır. Bu fonksiyonların kullanımı hakkında kısa bir bilgi aşağıda verilmiştir;

// Kütüphaneye zaman gecikmesi ve durum değerleri ile fonksiyon ekle  
bool attach(void (\*pointer)(void));  
bool attach(void (\*pointer)(void), uint16\_t intervalMillis);  
bool attach(void (\*pointer)(void), bool enabledStatus);  
bool attach(void (\*pointer)(void), uint16\_t intervalMillis, bool enabledStatus);

// Kütüphanenin ana fonksiyonunu başlat, dönüye al veya durdur  
void start();  
void loop();  
void stop();

// Kütüphanenin ana fonksiyonuna kayıtlı bir işlemi başlat veya durdur  
bool startProcess(void (\*pointer)(void));  
bool stopProcess(void (\*pointer)(void));

Aşağıda bu kütüphanenin *Arduino* platformunda çalışan örnek bir kullanımı gösterilmiştir;

void setup() {  
 // Kütüphaneye 1 saniye aralıklarla çalışan bir fonksiyon ekle  
 TimerQueue.attach(listenFunction, (uint8\_t)1000);  
 // Kütüphanenin ana fonksiyonunu başlat  
 TimerQueue.start();  
}

void loop() {  
 // Kütüphanenin ana fonksiyonunu dönüye al  
 TimerQueue.loop();  
}

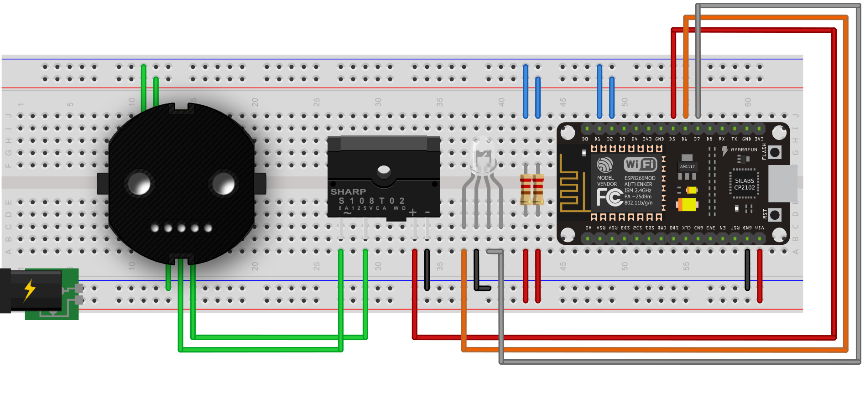
// Kütüphaneye kaydedilen fonksiyonun bildirimi  
void listenFunction() { }

Yukarıda verilen örnek kullanımda, *TimerQueue* kütüphanesine *listenFunction()* adında bir fonksiyon iliştirilmiştir. Bu işlemin ardından kütüphane çalıştırılmış ve döngüye alınmıştır. Bu noktadan sonra *TimerQueue* kütüphanesi geliştirme kartının sahip olduğu entegre osilatör ile sürekli olarak zaman hesaplaması yapacak ve sisteme kaydedilen bu fonksiyonu 1’er saniye zaman aralıklarında tetikleyerek çalıştıracaktır.

**UYGULAMA**

Uygulama aşamasında *NodeMCU* geliştirme kartı ile *FOTEK SSR–40DA* elektronik anahtarı kullanılmış, *Arduino Pro Mini* geliştirme kartı ile *DHT11*, *MQ–2* ve *MQ–7* sensörleri kullanılmıştır. *Arduino Pro Mini* ile geliştirilen çoklu sensör devresi, akıllı priz sistemleri için örnek bir aygıt olarak tasarlanmıştır. Kullanıcılar bu aşamada amaç ve istekleri doğrultusunda farklı türde cihazlar geliştirebilir ve sisteme bağlayabilirler.

Aşağıdaki şemada, *I²C* protokolünün *Master* bölümüne ait devre şeması verilmiştir. Bu devre şemasında *D1* ve *D2* pimleri *I²C* bağlantılarına, *D6* ve *D7* pimleri *LED* bağlantılarına ve son olarak *D5* pimi ise *SSR* elektronik anahtar bağlantısına rezerve edilmiştir. Bu pimlerden herhangi birisinin değiştirilmesi istendiğinde gerek kod kısmında gerekse yazılım kısmında birtakım değişikliklerin yapılması gerekmektedir.



Verilen devre şemasında akıllı prize herhangi bir aygıt bağlandığında, priz ile aygıt arasında ilk olarak *handshake* işlemi, yani el sıkışma işlemi yapılır. Ardından *I²C* ağında *Master* özelliğine sahip olan *NodeMCU* aygıtı *Slave* aygıttan sahip olduğu fonksiyonlarının özelliklerini talep eder. Bu talep işlemi ile *Slave* aygıtı sahip olduğu harici bütün fonksiyonları *Serializer* kütüphanesi ile serileştirir ve *I²C* ağı üzerinden *Master* aygıta yollar. Bütün bu işlem ve süreçler sonucu prize bağlanan *Slave* aygıtının haberleşme için gerekli olan bütün özellikleri *Master* aygıtın veri tabanına kaydedilir ve böylece *Slave* aygıt haberleşme süreci için hazır konuma geçmiş olur.

Bir sonraki adımda *Master* aygıt, *Slave* aygıtın sahip olduğu fonksiyonlardan ilgili verileri belirli zaman aralıklarla talep eder ve dönen verileri *MQTT* altyapısı ile *openHAB* yazılımına bildirir.

Yukarıda verilen devre şemadan farklı olarak aşağıdaki verilen şema, *I²C* protokolünün *Slave* bölümüne ait devre şemasıdır. Bu devre şemasında *A0* pimi *MQ–2* sensörüne, *A1* pimi *MQ–7* sensörüne ve son olarak *13* numaralı pimi *DHT11* sensörüne rezerve edilmiştir. Bu pimlerden herhangi birisinin değiştirilmesi istendiğinde gerek kod kısmında gerekse yazılım kısmında birtakım değişikliklerin yapılması gerekmektedir.