Pamukkale Üniversitesi, Mühendislik Fakültesi
Bilgisayar Mühendisliği Bölümü

Lisans Bitirme Tezi

Elektrik Sistemlerinin Uzaktan Denetimi ve Yönetimi: intelliPWR

Veysel Berk ALTUN
13253004

Danışman

Dr. Öğr. Ü. Elif HAYTAOĞLU

Mayıs 21, 2018

Denizli

Pamukkale Üniversitesi, Mühendislik Fakültesi
Bilgisayar Mühendisliği Bölümü

Lisans Bitirme Tezi

Elektrik Sistemlerinin Uzaktan Denetimi ve Yönetimi: intelliPWR

Veysel Berk ALTUN
13253004

Danışman

Dr. Öğr. Ü. Elif HAYTAOĞLU

Mayıs 21, 2018

Denizli

LİSANS TEZİ ONAY FORMU

Veysel Berk ALTUN tarafından Dr. Öğr. Ü. 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.

Dr. Öğr. Ü. 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
İmza:

ÖNSÖZ

Tez çalışmam sırasında kıymetli bilgi, birikim ve tecrübeleri ile bana yol gösterici ve destek olan değerli danışman hocam sayın *Dr. Öğr. Ü. 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

İÇİNDEKİLER

KISALTMALAR

AVR:	Automatic Voltage Regulator
AX:	Analog X
DX:	Digital X
DHT:	Dew, Humidity & Temperature
HAB:	Home Automation Bus
IDE:	Integrated Development Environment
IO:	Input – Output
IoT:	Internet of Things
LED:	Light Emitting Diode
MHz:	Megahertz
MQ:	Mĭngăn Qĭ lai (Sensitive to Gas)
MQTT:	Message Queuing Telemetry Transport
PWM:	Pulse Width Modulation
RAW:	Read After Write
R&D:	Research and Development
SoC:	System on A Chip
SPI:	Serial Peripheral Interface
SSR:	Solid State Relay
UART:	Universal Asynchronous Receiver & Transmitter
Wi–Fi:	Wireless Fidelity
WPA2:	Wi–Fi Protected Access 2

TABLO LİSTESİ

ŞEKİL LİSTESİ

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.

ABSTRACT

This thesis project is planned to be employed on to every area where the electricity is reached and in this project, today's bases of the sockets reconsidered and an innovative idea have been put forward about the socket's design. With these redesigned sockets, the communication of future generations of intelligent home appliances is provided with sockets and remote control and management of these smart home devices were realized with the help of redesigned sockets.

In this our new socket design include, five small connection pins inserted under the phase and neutral pins of the socket. With the help of these pins, devices connected to the socket are intended to communicate with the current socket.

In the research and development process of the thesis, with the aim of being used in the wired communication processes, three different libraries are written which are compatible with *RaspBerry Pi 3*, *Arduino Mega*, *Arduino Pro Mini* and *NodeMCU* platforms. In the writing stage of all these libraries, *C*, *C++* and *Phyton* languages are preferred; the *GIT* version control system is used for an easier research and development process. In the wireless communication processes of the thesis, *openHAB* system which is an open source code and which is still being developed is preferred.

As a result, within the scope of the whole project, a total of *4400* lines of functional and modular codes were written in accordance with the basic computer engineering ethics and rules, and a new generation socket design suitable for modernization has been put forward.

1. 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 İnterneti – 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.

2. 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

2.1 Arduino

Arduino, Giriş/Çıkışları bulunan fiziksel bir programlama platformundan oluşan güncel bir geliştirme kartıdır. Bu geliştirme kartının uygulama ve geliştirme süreçlerinde *Processing/Wiring* dilinin temel alan bir yazılım kullanılmaktadır.

Arduino kartlarının donanımında bir adet *Atmel AVR* mikrodenetleyici (*ATmega328*, *ATmega2560* veya *ATmega32u4* gibi), bir adet programlayıcı ve diğer devrelere bağlantı için gerekli olan yan devre elemanları bulunur. Her *Arduino* kartında en az bir adet *5V* regüle entegresi ve bir adet *16MHz* kristal osilator devresi bulunur. *Arduino* kartlarını programlamak için harici herhangi bir programlayıcıya ihtiyaç duyulmaz; çünkü bu geliştirme kartının mikro denetleyicisine üretim aşamasında önceden bir *bootloader* programı yazılmıştır.

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

2.2 NodeMCU

NodeMCU, Giriş/Çıkışları bulunan fiziksel bir programlama platformundan oluşan güncel bir geliştirme kartıdır. *Arduino* ile benzer yapıya sahip olsa da üzerine gömülü olan ESP8266 *Wi–Fi SoC* ile kablosuz haberleşme uygulamalarında da kullanılabilir. Sahip olduğu bu farklı donanım yapısından dolayı *IoT* uygulamalarında yoğun olarak tercih edilmektedir.

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

2.3 openHAB

openHAB (open Home Automation Bus), Java SE Runtime Environment dilinde yazılmış olup, bina otomasyonunda kullanılan bileşenleri üretici ve iletişim protokolü gözetmeden bir platformda birleştiren yazılım çözümüdür. Bununla birlikte herhangi bir işletim sistemine bağlı olmaksızın ek bağlantılarla yeni teknolojilerle ve protokollerle uygulanabilir.

openHAB yazılımının *Ubuntu* işletim sisteminde kararlı çalışabilmesi için işletim sistemine yüklü 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

Tablo 2.1: openHAB Cloud Kurulumu

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

2.4 Mosquitto

MQTT (Message Queuing Telemetry Transport) protokolü, internette yaygın olarak kullanılan mesaj tabanlı bir haberleşme protokoldür. Hafif bir haberleşme altyapısına sahip oluşu ve düşük kaynak tüketmesiyle *IoT* ekosistemlerinde sık sık kullanılmaktadır. Hemen hemen tüm *IoT* bulut platformları akıllı nesnelerden veri gönderip almak için *MQTT* protokolünü desteklemektedir.

Bu protokolde, istek–yanıt (*request–response*) yapısına dayalı *HTTP*'ye karşıt olarak yayın–abone (*publish–subscriber*) yapısında *TCP/IP* bağlantısı kurulur. *TCP/IP* protokolünün çalışabildiği *Linux*, *Windows*, *Android*, *iOS* ve *MacOS* gibi bütün işletim sistemleri ile uyumludur. Ayrıca, *MQTT protokolü* asenkron çalışan bir protokoldür; bu da mesajı beklerken istemci aygıtı engellemediği anlamına gelir. *HTTP* protokolünün aksine, esas olarak eşzamanlı bir protokoldür. *MQTT* protokolünün bir başka özelliği, istemcinin ve yayıncının aynı anda internete bağlı olmasını gerektirmemesidir.

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`

Tablo 2.2: Mosquitto Kurulumu

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

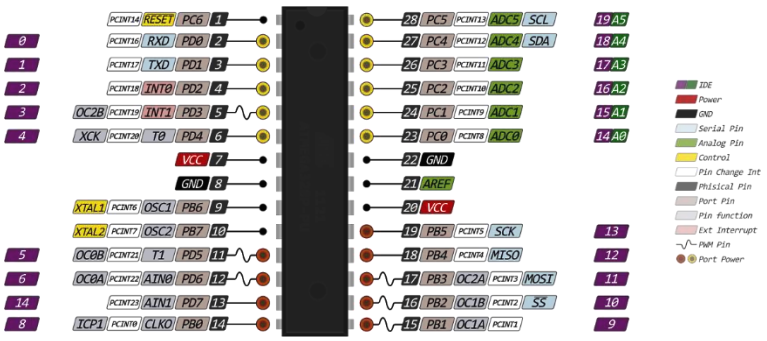
3. DONANIM

Tezin bütün uygulama süreçlerinde toplamda *2* farklı geliştirme kartı ve *5* farklı devre elemanı kullanılmıştır. Aşağıda bu donanımların özellikleri hakkında detaylı bilgiler verilmiştir.

3.1 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.



Şekil 3.1: Arduino Pro Mini Pim Diyagramı

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;

$$f_{\mathrm{SCL}} \leq 100\mathrm{kHz} \rightarrow R_{min} = \frac{V_{\mathrm{CC}} - 0.4\mathrm{V}}{3mA}\,, R_{max} = \frac{1000ns}{C_{bus}}$$

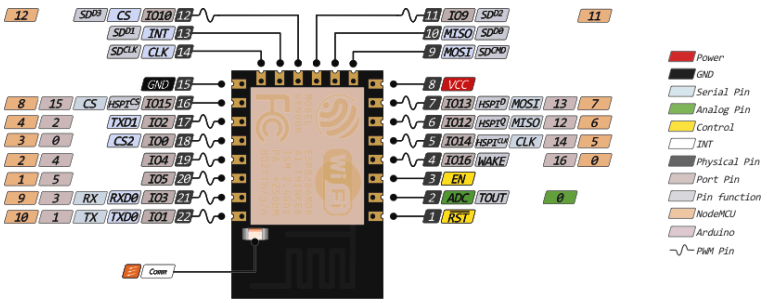
$$f_{\mathrm{SCL}} > 100\mathrm{kHz} \rightarrow R_{min} = \frac{V_{\mathrm{CC}} - 0.4\mathrm{V}}{3mA}\,, R_{max} = \frac{300ns}{C_{bus}}$$

Formüllerde verilen f_{SCL} değeri *SCL* hattının çalışma frekansını, R_{min} değeri *pull–up* direncinin sahip olabileceği en küçük direnç değerini, V_{CC} değeri hattın sahip olduğu gerilim değerini, R_{max} değeri *pull–up* direncinin sahip olabileceği en büyük direnç değerini ve son olarak C_{bus} 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 4.7KΩ olarak baz alınmıştır.

3.2 NodeMCU

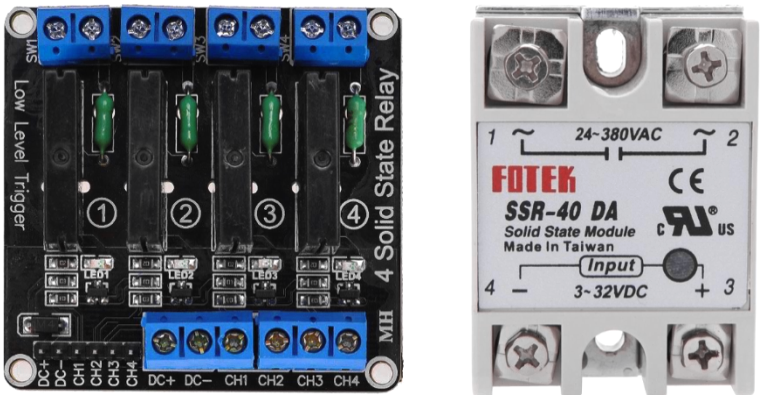
NodeMCU, ESP8266–12 tabanlı bir geliştirme kartıdır. Bu geliştirme kartı üzerinde *17* adet dijital giriş – çıkış pimi (bu pimlerden *9* tanesi *PWM* çıkışı), *1* adet analog giriş – çıkış, bir adet *80 MHz* kristal osilatör, bir adet 802.11 b/g/n *Wi–Fi* modülü ve bir adet *reset* butonu bulundurur.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



Şekil 3.2: NodeMCU Pim Diyagramı

3.3 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.



Şekil 3.3: FOTEK SSR–40DA

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 en yüksek güç değerliğini hesaplayabiliriz. Gücün formülü;

$$P(W) = I(A) \times V(V)$$

Formüllerde verilen $I(A)$ değeri akım değerini, $V(V)$ 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 $220V - 50Hz$ şehir geriliminde bu SSR modülü, ortalama $8880W$ değerliğine kadar kararlı bir şekilde çalışabilecektir.

3.4 MQ–2: Yanıcı Gaz ve MQ–7: Karbonmonoksit Gaz Sensörü

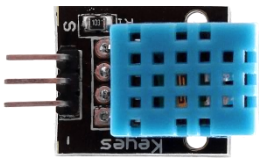
MQ–2 sensörü, ortamda bulunan ve konsantrasyonu *300~10.000PPM* arasında değişen yanıcı gazı algılayan bir sensör modülüdür. *–100~500°C* arasında çalışabilir ve 5V gerilim üzerinde 150mA akım çeker. Analog çıkışı sayesinde algılanan gaz konsantrasyonu kolayca okunabilir.

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.

Şekil 3.4: MQ–2: Yanıcı Gaz ve MQ–7: Karbonmonoksit Gaz Sensörü

3.5 DHT11: Isı 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.



Şekil 3.5: DHT11: Isı Ve Nem Sensörü

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

4. 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.

4.1 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"}
}
```

4.2 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 12 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.

4.3 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.

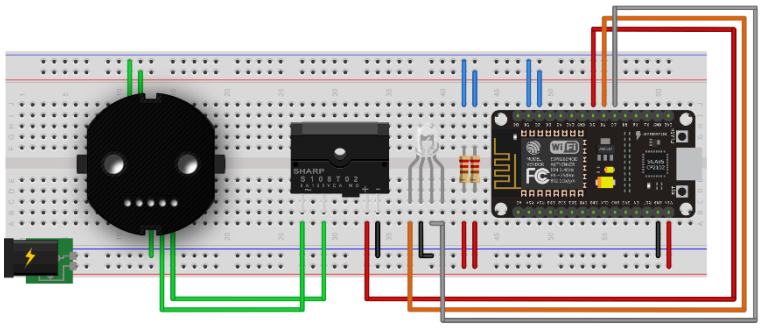
5. UYGULAMA

Uygulama aşamasında bütün prizler *Master*, prize bağlanan çevre aygıtlar *Slave* olarak baz alınmıştır. Örnek bir otomasyon sisteminde herhangi bir çevre aygıtı bu prizlere bağlandığında, bu çevre aygıtlarının bütün fonksiyonları ve üretici bilgileri prizlerin veri tabanına yüklenir. Ardından bu veriler *Wi–Fi* haberleşme altyapısı ile ağın bağlı olduğu *MQTT* sunucusuna gönderilir. Gönderilen bu veriler bu mesaja abone olan bütün aygıtlara iletilerek örnek bir haberleşme süreci tamamlanmış olur. Yukarı anlatan örnek bir otomasyon sisteminin donanımsal ve yazılımsal detaylı altyapısı aşağıda verilmiştir.

5.1 Devre Tasarımları

Devre tasarımında *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.

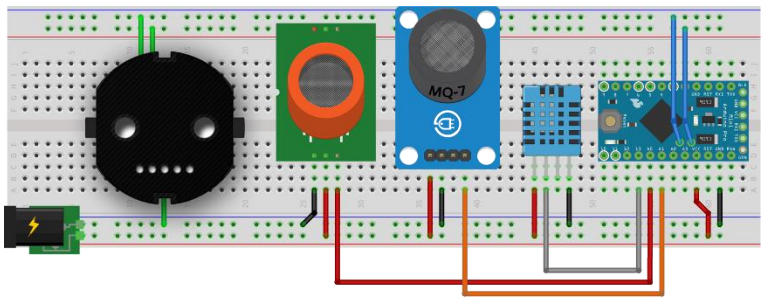


Şekil 5.1: Master devre tasarımı

Verilen devre şemasındaki 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 şemasından 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 pim ise *DHT11* sensörüne rezerve edilmiştir. Bu pimlerden herhangi birisinin değiştirilmesi istendiğinde gerek kod kısmında gerekse yazılım kısmında birtakım değişikliklerin yapılması gerekmektedir.



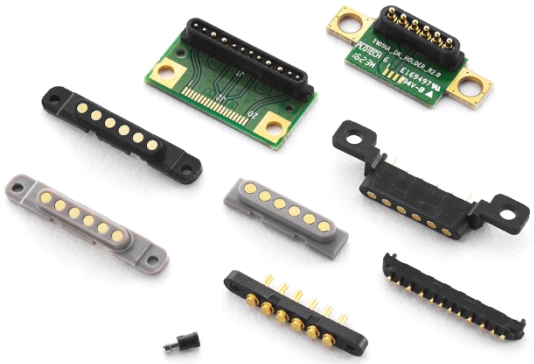
Şekil 5.2: Slave devre tasarımı

5.2 Prototip

Uygulama aşamasına her şeyden önce, ortaya atılan yeni priz tasarımı hakkında örnek bir çizim ve *3* boyutlu prototipleme yapılmıştır. Aşağıda, yapılan bu priz tasarımın *4* farklı açıdan sanal bir modeli verilmiştir.

Şekil 5.3: Örnek Priz Modeli

Yapılan bu protiplere günümüzde sıkça kullanılan pogo pim konnektörleri entegre edilmiş ve son prototip modeline ulaşılmıştır.



Şekil 5.4: Örnek Pogo Pimleri

Şekil 5.5: Prototip

6. SONUÇ

Tezin sonucunda gelecekte popülaritesi artacak olan ev otomasyonu sistemleri için düşük maliyetli, modüler, geliştirmeye açık ve kişi merkezli akıllı priz prototipleri yapılmış ve uygulamalı olarak test edilmiştir. Yapılan bu prototipler, herhangi bir kitleyi ve bölgeyi baz almak yerine evrensel açıdan herkesi baz alacak şekilde tasarlamıştır.

Diğer *IoT* sistemlerinden farklı olarak bu priz prototipleri kurulum gerektirmeyen bir yapıya sahiptir. Prizlere herhangi bir yeni nesil aygıt bağlandığında, bu aygıt için geçerli prize aplikasyon ve haberleşme kurulumu yapılması gerekmez; sisteme bağlanan bu yeni nesil akıllı aygıt evinizin veri tabanına eş zamanlı olarak anında kaydedilir.

Tezin hayata geçirilmesi durumunda, günümüzde yeterliliği daha az olan akıllı ev sistemleri yerine yeterliliği ve kullanılabilirliği daha fazla olan akıllı ev sistemleri inşa edilebilecektir. Benzer şekilde ev elektroniğinde yaşanabilecek olası kazalar azaltılabilecek ve yaşanabilecek herhangi bir kaza önceden tahmin edilebilecektir. Sistemin bütünlük felsefesini benimsemesi nedeniyle kullanım karmaşası da azaltılmış olacaktır.

Düşük maaliyetli oluşundan dolayı yeni tasarlanan bu prizler, yapılardaki eski priz sistemleri ile kolaylıkla değiştirilebilecek ve böylece eski yapıdaki priz sistemleri akıllı priz sistemlerine kolaylıkla dönüştürülebilecektir.

7. KAYNAKÇA

[1] Robert Clemenzi, *"Arduino – Examples and Libraries"*, 2015. [Online]. Bağlantı: http://mc–computing.com/Hardware_Platforms/Arduino/Libraries [Erişim: 01/2018]

[2] Dan Hirschberg, *"Pointers and Memory Allocation"*, 2016. [Online]. Bağlantı: https://www.ics.uci.edu/~dan/class/165/notes/memory [Erişim: 03/2018]

[3] Arduino, *"Manuals and Curriculum"*, 2016. [Online]. Bağlantı: https://playground.arduino.cc/Main/ArduinoPinCurrentLimitations [Erişim: 11/2017]

[4] tutorialspoint, *"C library function void *realloc(void *ptr)"*, 2018. [Online]. Bağlantı: https://www.tutorialspoint.com/c_standard_library/c_function_realloc [Erişim: 12/2017]

[5] tutorialspoint, *"C library function char *strtok(char *str)"*, 2018. [Online]. Bağlantı: https://www.tutorialspoint.com/c_standard_library/c_function_strtok [Erişim: 02/2018]

[6] GitHUB VCS, *"open–source electronics prototyping platform"*, 2018. [Online]. Bağlantı: https://github.com/arduino/Arduino [Erişim: 04/2018]

[7] Codingstreet, *"C INPUT OUTPUT"*, 2013. [Online]. Bağlantı: http://codingstreet.com/c–input–output [Erişim: 12/2017]

[8] armMBED, *"C Data Types"*, 2018. [Online]. Bağlantı: https://os.mbed.com/handbook/C–Data–Types [Erişim: 01/2018]

[9] cplusplus, *"strtok: Split string into tokens"*, 2015. [Online]. Bağlantı: http://www.cplusplus.com/reference/cstring/strtok [Erişim: 02/2018]

[10] cplusplus, *"strchr: Locate first occurrence of character in string"*, 2017. [Online]. Bağlantı: http://www.cplusplus.com/reference/cstring/strchr [Erişim: 01/2018]

[11] cplusplus, *"isalnum: Check if character is alphanumeric"*, 2016. [Online]. Bağlantı: http://www.cplusplus.com/reference/cctype/isalnum [Erişim: 02/2018]

[12] Wikipedia, *"Delimiter"*, 2010. [Online]. Bağlantı: https://en.wikipedia.org/wiki/Delimiter [Erişim: 10/2017]

[13] Wikipedia, *"C0 and C1 control codes"*, 2010. [Online]. Bağlantı: https://en.wikipedia.org/wiki/C0_and_C1_control_codes [Erişim: 03/2018]

[14] aivosto, *"Control characters in ASCII and Unicode"*, 2016. [Online]. Bağlantı: http://www.aivosto.com/vbtips/control–characters [Erişim: 01/2018]

[15] ElectronicWings, *"NodeMCU I2C with Arduino IDE"*, 2016. [Online]. Bağlantı: http://www.electronicwings.com/nodemcu/nodemcu–i2c–with–arduino–ide [Erişim: 01/2018]

[16] openHAB Foundation, *"MQTT Binding"*, 2017. [Online]. Bağlantı: https://docs.openhab.org/addons/bindings/mqtt1/readme [Erişim: 04/2018]

[17] GitHUB VCS, *"The firmware development of intelliPWR"*, 2017. [Online]. Bağlantı: https://github.com/vberkaltun/nodemcu-firmware-development [Erişim: 10/2017]

8. EKLER

Aşağıda tez süresince geliştirilen ve kullanılan kod betikleri verilmiştir. Bu kod betikleri ile örnek bir otomasyonu sistemi yapılabilir.

8.1 Serializer.CPP

```
/*
* Serializer.cpp
* SERIALIZER - 17.04.2018
*
* ========================================================================
*
* The MIT License (MIT)
*
* Copyright (c) 2018 Berk Altun - vberkaltun.com
*
* Permission is hereby granted, free of charge, to any person obtaining a copy
* of this software and associated documentation files (the "Software"), to deal
* in the Software without restriction, including without limitation the rights
* to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
* copies of the Software, and to permit persons to whom the Software is
* furnished to do so, subject to the following conditions:
*
* The above copyright notice and this permission notice shall be included in
* all copies or substantial portions of the Software.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
* AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
* LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
* OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
* SOFTWARE.
*
* ========================================================================
*/

#include "Serializer.h"

// -----

unsigned short Serializer::calculateMaximumLineWidth() {

     // IMPORTANT NOTICE: In this step, We need to calculate the bigger size of
     // one line in given array. When we declare a 2D array, program will allocate
     // each line of this array with same line width. Because of this,
     // for avoiding of overflow, we must calculate the upper bound all given line

     // Store the size of received data at the here
     unsigned short sizeofBuffer = strlen(decodedList.givenData);
```

```
    char *bufferData = (char *) malloc(sizeof (char) * (sizeofBuffer + 1));
    strcpy(bufferData, decodedList.givenData);
    bufferData[sizeofBuffer] = '\0';

    // -----

    // Store the size of received data at the here
    unsigned short maximumLineWidth = 0;

    // Store splitting data at the here
    char *tokenizer = strtok(bufferData, decodedList.delimiter);

    while (tokenizer != NULL) {

        // Store the size of received data at the here
        unsigned short sizeofTokenizer = strlen(tokenizer);

        if (sizeofTokenizer > maximumLineWidth)
            maximumLineWidth = sizeofTokenizer;

        tokenizer = strtok(NULL, decodedList.delimiter);
    }

    // Do not forget, strictly free up memory
    free(bufferData);

    return maximumLineWidth;
}

void Serializer::clearDecodedList(bool isAllData) {

    // IMPORTANT NOTICE: seems so confused, right?
    // When you free up a pointer, you can not use it again anymore
    // But when you make reassigning NULL to a pointer after the freeing up,
    // You can this pointer again very well

    free(decodedList.delimiter);
    decodedList.delimiter = NULL;

    free(decodedList.givenData);
    decodedList.givenData = NULL;

    if (isAllData) {

        for (unsigned short index = 0; index < decodedList.sizeofResultData; index++) {
            free(decodedList.resultData[index]);
            decodedList.resultData[index] = NULL;
        }

        free(decodedList.resultData);
        decodedList.resultData = NULL;
        decodedList.sizeofResultData = 0;
    }

    // -----

    decodedList.sizeofDelimiter = 0;
    decodedList.sizeofGivenData = 0;
}

void Serializer::clearEncodedList(bool isAllData) {

    // IMPORTANT NOTICE: seems so confused, right?
```

```
        // When you free up a pointer, you can not use it again anymore
        // But when you make reassigning NULL to a pointer after the freeing up,
        // You can this pointer again very well

        free(encodedList.delimiter);
        encodedList.delimiter = NULL;

        for (unsigned short index = 0; index < encodedList.sizeofGivenData; index++) {
             free(encodedList.givenData[index]);
             encodedList.givenData[index] = NULL;
        }

        free(encodedList.givenData);
        encodedList.givenData = NULL;

        if (isAllData) {
             free(encodedList.resultData);
             encodedList.resultData = NULL;
             encodedList.sizeofResultData = 0;
        }

        // -----

        encodedList.sizeofDelimiter = 0;
        encodedList.sizeofGivenData = 0;
   }

   void Serializer::fillDecodedList() {

        // Store the size of received data at the here
        unsigned short maximumLineWidth = calculateMaximumLineWidth();

        // Store splitting data at the here
        char *tokenizer = strtok(decodedList.givenData, decodedList.delimiter);

        while (tokenizer != NULL) {

             // Malloc and realloc a sentence, a list of words
             if (decodedList.resultData == NULL)
                  decodedList.resultData = (char **) malloc(sizeof (char *) *
                                           (++decodedList.sizeofResultData));
             else
                  decodedList.resultData = (char **) realloc(decodedList.resultData, sizeof
                                           (char *) * (++decodedList.sizeofResultData));

             // Malloc and realloc a word, a list of characters, after carry it
             decodedList.resultData[decodedList.sizeofResultData - 1] = (char *)
             malloc(sizeof (char) * (maximumLineWidth + 1));
             strcpy(decodedList.resultData[decodedList.sizeofResultData - 1], tokenizer);
             decodedList.resultData[decodedList.sizeofResultData - 1][maximumLineWidth] =
             '\0';

             tokenizer = strtok(NULL, decodedList.delimiter);
        }

        // Do not forget, strictly free up memory
        free(tokenizer);

        // IMPORTANT NOTICE: Probably it is looking unnecessary ...
        // But it has a very important role on memory. With this code,
        // Program will resize the array and put a NULL parameter to tail
        decodedList.resultData = (char **) realloc(decodedList.resultData, sizeof (char *) *
                                 (decodedList.sizeofResultData + 1));
        decodedList.resultData[decodedList.sizeofResultData] = '\0';
```

```
}

void Serializer::fillEncodedList() {

     // Store the size of received data at the here
     int sizeofAbsolute = encodedList.sizeofDelimiter - encodedList.sizeofGivenData;

     // -----

     // Increase total count of data size
     encodedList.sizeofResultData += encodedList.sizeofDelimiter;
     encodedList.resultData = (char *) malloc(sizeof (char) *
                              (encodedList.sizeofResultData + 1));

     // Declare an variable for storing done separator
     unsigned short checkedDelimiter = 0;
     unsigned short currentIndex = 0;

     // If absolute value is bigger than 0, add a delimiter to the first index
     if (sizeofAbsolute >= 0)
          encodedList.resultData[currentIndex++] =
          encodedList.delimiter[checkedDelimiter++];

     for (unsigned short array = 0; array < encodedList.sizeofGivenData; array++) {

          // Store the size of received data at the here
          unsigned short sizeofBuffer = strlen(encodedList.givenData[array]);

          // Get line by line characters and fill result data
          for (unsigned short index = 0; index < sizeofBuffer; index++)
               encodedList.resultData[currentIndex++] =
               encodedList.givenData[array][index];

          if (checkedDelimiter != encodedList.sizeofDelimiter)
               encodedList.resultData[currentIndex++] =
               encodedList.delimiter[checkedDelimiter++];
     }

     // Finally add end null character to tail
     encodedList.resultData[currentIndex++] = '\0';
}

bool Serializer::decodeData() {

     // Declare an variable for storing done separator
     unsigned short checkedDelimiter = 0;

     for (unsigned short index = 0; index < decodedList.sizeofGivenData; index++) {

          // Found status flag, using for to find operate
          bool foundFlag = false;

          for (unsigned short subIndex = 0; subIndex < decodedList.sizeofDelimiter;
               subIndex++) {
               if (decodedList.delimiter[subIndex] == decodedList.givenData[index]) {
                    foundFlag = true;
                    break;
               }
          }

     // Is a delimiter not found in delimiter list, jump to next
     if (!foundFlag)
          continue;
```

```
        if (checkedDelimiter >= decodedList.sizeofDelimiter)
            return false;

        if (decodedList.delimiter[checkedDelimiter++] != decodedList.givenData[index])
            return false;
        }

        if (checkedDelimiter < decodedList.sizeofDelimiter)
            return false;

        // Arrived final function
        fillDecodedList();

        return true;
}

bool Serializer::encodeData() {

        // IMPORTANT NOTICE: The absolute value always must be 0 or zero
        // For example, If size of given data is bigger or smaller than
        // the size of delimiters, we can not have enough delimiters for encoding
        // For this reason, When ABS(s) of delimiters and data is 0 or 1,
        // encoding can be performed very well
        if (abs(encodedList.sizeofGivenData - encodedList.sizeofDelimiter) > 1)
            return false;

        // -----

        for (unsigned short array = 0; array < encodedList.sizeofGivenData; array++) {

            // Store the size of received data at the here
            unsigned short sizeofBuffer = strlen(encodedList.givenData[array]);

            // Check that whether given data includes a delimiters or not
            for (unsigned short index = 0; index < sizeofBuffer; index++)
                for (unsigned short iterator = 0; iterator < encodedList.sizeofDelimiter;
                     iterator++)
                     if (encodedList.givenData[array][index] ==
                         encodedList.delimiter[iterator])
                          return false;

            // Increase total count of data size
            encodedList.sizeofResultData += sizeofBuffer;
        }

        // Arrived final function
        fillEncodedList();

        return true;
}

// -----

/**
* Constructors
*
* @param -
* @return -
*/
Serializer::Serializer() {
}

// -----
```

```
char **Serializer::decode(unsigned short sizeofDelimiter, char delimiter[], unsigned short
                          sizeofGivenData, char givenData[]) {

     // Clear last stored data
     clearDecodedList(true);

     if (delimiter == NULL)
          return NULL;

     if (givenData == NULL)
          return NULL;

     if (sizeofDelimiter == 0)
          return NULL;

     // Store all received data size on lib
     decodedList.sizeofDelimiter = sizeofDelimiter;
     decodedList.sizeofGivenData = sizeofGivenData;

     // -----

     decodedList.delimiter = (char *) malloc(sizeof (char) * (sizeofDelimiter + 1));
     strcpy(decodedList.delimiter, delimiter);
     decodedList.delimiter[sizeofDelimiter] = '\0';

     decodedList.givenData = (char *) malloc(sizeof (char) * (sizeofGivenData + 1));
     strcpy(decodedList.givenData, givenData);
     decodedList.givenData[sizeofGivenData] = '\0';

     // -----

     bool decodeDataFlag = decodeData();

     if (!decodeDataFlag)
          clearDecodedList(true);
     else
          clearDecodedList(false);

     return (decodeDataFlag ? decodedList.resultData : NULL);
}

char *Serializer::encode(unsigned short sizeofDelimiter, char delimiter[], unsigned short
                         sizeofGivenData, char *givenData[]) {

     // Clear last stored data
     clearEncodedList(true);

     if (delimiter == NULL)
          return NULL;

     if (givenData == NULL)
          return NULL;

     if (sizeofDelimiter == 0)
          return NULL;

     // Store all received data size on lib
     encodedList.sizeofDelimiter = sizeofDelimiter;
     encodedList.sizeofGivenData = sizeofGivenData;

     // -----

     encodedList.delimiter = (char *) malloc(sizeof (char) * (sizeofDelimiter + 1));
     strcpy(encodedList.delimiter, delimiter);
```

```
        encodedList.delimiter[sizeofDelimiter] = '\0';

        // IMPORTANT NOTICE: In Clang, there is no way out to copy 2D array
        // If you want to do that, first you must allocate main array memory
        // After that you must to fill each line of main array
        encodedList.givenData = (char **) malloc(sizeof (char *) * (sizeofGivenData + 1));
        for (unsigned short index = 0; index < sizeofGivenData; index++) {
             // Store the size of received data at the here
             unsigned short sizeofArray = strlen(givenData[index]);
             encodedList.givenData[index] = (char *) malloc(sizeof (char) * (sizeofArray +
                                                1));
             strcpy(encodedList.givenData[index], givenData[index]);
        }
        encodedList.givenData[sizeofGivenData] = '\0';

        bool encodeDataFlag = encodeData();

        if (!encodeDataFlag)
             clearEncodedList(true);
        else
             clearEncodedList(false);

        return (encodeDataFlag ? encodedList.resultData : NULL);
   }

   // -----

   /**
   * Preinstantiate Objects
   *
   * @param -
   * @return -
   */
   Serializer Serialization = Serializer();
```

8.2 MasterScanner.CPP

```
   /*
   * MasterScanner.cpp
   * MASTER SCANNER - 23.03.2018
   *
   * ======================================================================
   *
   * The MIT License (MIT)
   *
   * Copyright (c) 2018 Berk Altun - vberkaltun.com
   *
   * Permission is hereby granted, free of charge, to any person obtaining a copy
   * of this software and associated documentation files (the "Software"), to deal
   * in the Software without restriction, including without limitation the rights
   * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
   * copies of the Software, and to permit persons to whom the Software is
   * furnished to do so, subject to the following conditions:
   *
   * The above copyright notice and this permission notice shall be included in
   * all copies or substantial portions of the Software.
   *
   * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
   * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
   * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
   * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
   * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
```

```
* OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
* SOFTWARE.
*
* ========================================================================
*/

#include "MasterScanner.h"

void Scanner::onTriggeredConnected(uint8_t _array[], byte _count) {

     if (_count != 0) {

          // don't bother if user hasn't registered a callback
          if (!onConnected) {
               return;
          }

          onConnected(_array, _count);
     }
}

void Scanner::onTriggeredDisconnected(uint8_t _array[], byte _count) {

     if (_count != 0) {

          // don't bother if user hasn't registered a callback
          if (!onDisconnected) {
               return;
          }

          onDisconnected(_array, _count);
     }
}

bool Scanner::checkRange(uint16_t _intervalMillis) {

     if (_intervalMillis < DEFAULT_INTERVAL_MILLIS_MIN)
          return false;

     if (_intervalMillis > DEFAULT_INTERVAL_MILLIS_MAX)
          return false;

     return true;
}

bool Scanner::checkRange(uint8_t _startAddress, uint8_t _stopAddress) {

     if (_startAddress > _stopAddress)
          return false;

     if (_startAddress < defaultData.startAddress)
          return false;

     if (_stopAddress > defaultData.stopAddress)
          return false;

     return true;
}

void Scanner::cleanRange(uint8_t _startAddress, uint8_t _stopAddress) {

     for (uint8_t address = _startAddress; address <= _stopAddress; address++)
          givenData.connectedSlavesArray[address] = NULL;
}
```

```
uint8_t *Scanner::fillArray(uint8_t _array[], uint8_t _address, byte _count) {

     if (_array == NULL)
          _array = (uint8_t*) malloc(sizeof (uint8_t) * _count);
     else
          _array = (uint8_t*) realloc(_array, sizeof (uint8_t) * _count);

     _array[_count - 1] = _address;

     return _array;
}

// -----

/**
* Constructors
*
* @param -
* @return -
*/
Scanner::Scanner() {
}

// -----

/**
* Changes default scanning range with new range, additional changes delay
*
* @param Delay, start and stop address of range
* @return -
*/
bool Scanner::setRange(uint16_t _intervalMillis, uint8_t _startAddress, uint8_t
                       _stopAddress) {

     bool setRangeFlag = false;

     if (checkRange(_intervalMillis) && checkRange(_startAddress, _stopAddress)) {
          this->setRange(_intervalMillis);
          this->setRange(_startAddress, _stopAddress);
          setRangeFlag = true;
     }

     return setRangeFlag;
}

/**
* Changes default scanning range with new range
*
* @param Delay range
* @return -
*/
bool Scanner::setRange(uint16_t _intervalMillis) {

     bool setRangeFlag = false;

     if (checkRange(_intervalMillis)) {
          givenData.intervalMillis = _intervalMillis;
          setRangeFlag = true;
     }

     return setRangeFlag;
}
```

```
/**
* Changes default scanning range with new range
*
* @param Start and stop address of range
* @return -
*/
bool Scanner::setRange(uint8_t _startAddress, uint8_t _stopAddress) {

     bool setRangeFlag = false;

     if (checkRange(_startAddress, _stopAddress)) {

          if (_startAddress > givenData.startAddress)
               this->cleanRange(givenData.startAddress, _startAddress - 1);

          if (_stopAddress < givenData.stopAddress)
               this->cleanRange(_stopAddress + 1, givenData.stopAddress);

          givenData.startAddress = _startAddress;
          givenData.stopAddress = _stopAddress;
          setRangeFlag = true;
     }

     return setRangeFlag;
}

/**
* Resets current scanning range with default range
*
* @param Start and stop address of range
* @return -
*/
void Scanner::resetRange() {

     this->setRange(defaultData.intervalMillis, defaultData.startAddress,
                    defaultData.stopAddress);
}

// -----

/**
* Scans connected devices with the given range
*
* @param -
* @return -
*/
void Scanner::scanSlaves() {

     uint16_t currentMillis = millis();

     if (currentMillis - previousMillis >= givenData.intervalMillis) {

          // save the last time you blinked the LED
          previousMillis = currentMillis;

          // -----

          uint8_t *currentConnectedSlavesArray = NULL;
          uint8_t *currentDisconnectedSlavesArray = NULL;

          byte currentConnectedSlavesCount = 0;
          byte currentDisconnectedSlavesCount = 0;

          // -----
```

```
          // addresses 0x00 through 0x77
          for (uint8_t currentAddress = givenData.startAddress; currentAddress <=
               givenData.stopAddress; currentAddress++) {


               Wire.beginTransmission(currentAddress);

               if (Wire.endTransmission() == 0) {

                    if (givenData.connectedSlavesArray[currentAddress] == NULL) {
                         givenData.connectedSlavesArray[currentAddress] = currentAddress;
                         givenData.connectedSlavesCount++;

                         currentConnectedSlavesArray = this-
                         >fillArray(currentConnectedSlavesArray, currentAddress,
                         ++currentConnectedSlavesCount);
                    }

               } else {

                    if (givenData.connectedSlavesArray[currentAddress] != NULL) {
                         givenData.connectedSlavesArray[currentAddress] = NULL;
                         givenData.connectedSlavesCount--;

                         currentDisconnectedSlavesArray = this-
                         >fillArray(currentDisconnectedSlavesArray, currentAddress,
                         ++currentDisconnectedSlavesCount);
                    }
               }
          }

          onTriggeredConnected(currentConnectedSlavesArray, currentConnectedSlavesCount);
          onTriggeredDisconnected(currentDisconnectedSlavesArray,
                                  currentDisconnectedSlavesCount);

          // always free at the end
          free(currentConnectedSlavesArray);
          free(currentDisconnectedSlavesArray);
     }
}

// -----

/**
* Triggers when at least one slave is connected
*
* @param A function that has two parameters of array and count
* @return -
*/
void Scanner::onConnectedSlaves(void (*pointer)(uint8_t[], byte)) {

     onConnected = pointer;
}

/**
* Triggers when at least one slave is disconnected
*
* @param A function that has two parameters of array and count
* @return -
*/
void Scanner::onDisconnectedSlaves(void (*pointer)(uint8_t[], byte)) {

     onDisconnected = pointer;
```

```
}

// -----

/**
* Gets start address of range
*
* @param -
* @return Start address of range
*/
uint8_t Scanner::getStartAddress() {

     return givenData.startAddress;
}

/**
* Gets stop address of range
*
* @param -
* @return Stop address of range
*/
uint8_t Scanner::getStopAddress() {

     return givenData.stopAddress;
}

/**
* Gets scanning frequency
*
* @param -
* @return Scanning frequency
*/
uint16_t Scanner::getIntervalMillis() {

     return givenData.intervalMillis;
}

/**
* Gets connected slaves count
*
* @param -
* @return Connected slaves count
*/
byte Scanner::getConnectedSlavesCount() {

     return givenData.connectedSlavesCount;
}

/**
* Checks if specified address is online on bus
*
* @param Address of specified device
* @return Connected or not Connected
*/
bool Scanner::isConnected(uint8_t _address) {

     bool isConnectedFlag = false;

     // addresses 0x00 through 0x77
     for (uint8_t currentAddress = givenData.startAddress; currentAddress <=
          givenData.stopAddress; currentAddress++) {
          if (givenData.connectedSlavesArray[currentAddress] == _address) {

               isConnectedFlag = true;
```

```
                break;
            }
        }

        return isConnectedFlag;
}

// -----

/**
* Preinstantiate Objects
*
* @param -
* @return -
*/
Scanner MasterScanner = Scanner();
```

8.3 TimerQueue.CPP

```
/*
* TimerQueue.cpp
* TIMER QUEUE - 28.04.2018
*
* ============================================================================
*
* The MIT License (MIT)
*
* Copyright (c) 2018 Berk Altun - vberkaltun.com
*
* Permission is hereby granted, free of charge, to any person obtaining a copy
* of this software and associated documentation files (the "Software"), to deal
* in the Software without restriction, including without limitation the rights
* to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
* copies of the Software, and to permit persons to whom the Software is
* furnished to do so, subject to the following conditions:
*
* The above copyright notice and this permission notice shall be included in
* all copies or substantial portions of the Software.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
* AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
* LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
* OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
* SOFTWARE.
*
* ============================================================================
*/

#include "TimerQueue.h"

bool Timer::isRegistered(void (*pointer)(void)) {

    bool isRegisteredFlag = false;

    if (timerQueueArray != NULL) {
        for (char currentQueueCount = 0; currentQueueCount < timerQueueCount;
                currentQueueCount++) {

            if (pointer == timerQueueArray[currentQueueCount].pointer) {
                isRegisteredFlag = true;
```

```
                    break;
                }
            }
        }

        return isRegisteredFlag;
}

bool Timer::checkRange(unsigned long intervalMillis) {

        if (intervalMillis < DEFAULT_INTERVAL_MILLIS_MIN)
            return false;

        if (intervalMillis > DEFAULT_INTERVAL_MILLIS_MAX)
            return false;

        return true;
}

// -----

/**
* Constructors
*
* @param -
* @return -
*/
Timer::Timer() {

}

// -----

bool Timer::attach(void (*pointer)(void)) {

        return this->attach(pointer, DEFAULT_INTERVAL_MILLIS, DEFAULT_STATUS);
}

bool Timer::attach(void (*pointer)(void), unsigned long intervalMillis) {

        return this->attach(pointer, intervalMillis, DEFAULT_STATUS);
}

bool Timer::attach(void (*pointer)(void), bool enabledStatus) {

        return this->attach(pointer, DEFAULT_INTERVAL_MILLIS, enabledStatus);
}

bool Timer::attach(void (*pointer)(void), unsigned long intervalMillis, bool
                    enabledStatus) {

        bool attachFlag = false;

        if (!isRegistered(pointer) && checkRange(intervalMillis) && ++timerQueueCount <
            DEFAULT_QUEUE_SIZE) {

            timerQueueArray[timerQueueCount - 1].pointer = pointer;
            timerQueueArray[timerQueueCount - 1].intervalMillis = intervalMillis;
            timerQueueArray[timerQueueCount - 1].previousMillis = millis();
            timerQueueArray[timerQueueCount - 1].enabledStatus = enabledStatus;

            attachFlag = true;
        }
```

```
    return attachFlag;
}

// -----

void Timer::start() {

    for (char currentQueueCount = 0; currentQueueCount < timerQueueCount;
         currentQueueCount++)
         timerQueueArray[currentQueueCount].previousMillis = millis();

    timerEnabledStatus = true;
}

void Timer::loop() {

    while (timerEnabledStatus && timerQueueArray != NULL) {

        for (char currentQueueCount = 0; currentQueueCount < timerQueueCount;
             currentQueueCount++) {
             if (timerQueueArray[currentQueueCount].enabledStatus == true) {

                 unsigned long currentMillis = millis();

                 if (currentMillis - timerQueueArray[currentQueueCount].previousMillis
                     >= timerQueueArray[currentQueueCount].intervalMillis) {

                      // save the last time you blinked the LED
                      timerQueueArray[currentQueueCount].previousMillis =
                      currentMillis;
                      timerQueueArray[currentQueueCount].pointer();

                      // Additional for NodeMCU
                      yield();
                 }
             }
        }
    }
}

void Timer::stop() {

    timerEnabledStatus = false;
}

// -----

bool Timer::startProcess(void (*pointer)(void)) {

    bool startProcessFlag = false;

    if (isRegistered(pointer)) {

        for (char currentQueueCount = 0; currentQueueCount < timerQueueCount;
             currentQueueCount++) {

             // save the last time you blinked the LED
             if (timerQueueArray[currentQueueCount].pointer == pointer) {
                 timerQueueArray[currentQueueCount].previousMillis = millis();
                 timerQueueArray[currentQueueCount].enabledStatus = true;

                 startProcessFlag = true;
                 break;
             }
```

```
            }
        }
        return startProcessFlag;
}

bool Timer::stopProcess(void (*pointer)(void)) {

        bool stopProcessFlag = false;

        if (isRegistered(pointer)) {

            for (char currentQueueCount = 0; currentQueueCount < timerQueueCount;
            currentQueueCount++) {

                // save the last time you blinked the LED
                if (timerQueueArray[currentQueueCount].pointer == pointer) {
                    timerQueueArray[currentQueueCount].enabledStatus = false;

                     stopProcessFlag = true;
                     break;
                }
            }
        }

        return stopProcessFlag;
}

// -----

/**
* Preinstantiate Objects
*
* @param -
* @return -
*/
Timer TimerQueue = Timer();
```

8.4 Master.INO

```
#include <Wire.h>
#include <Serializer.h>
#include <QueueList.h>
#include <MasterScanner.h>
#include <TimerQueue.h>

#include <PubSubClient.h>
#include <ESP8266WiFi.h>
#include <ArduinoOTA.h>

// IMPORTANT NOTICE: These all constant is related with your
// MQTT server and WiFi protocol. In additional, at the here, we are
// Using cloudMQTT server for communication

// IMPORTANT NOTICE: These all constant is depending on your protocol
// As you can see, this protocol delimiter was declared in this scope
// That's mean, all function will use this delimiter constant on
// Communication between two or more devices
#define DEVICE_BRAND "intelliPWR Incorporated"
#define DEVICE_MODEL "MasterCore.X1"
#define DEVICE_VERSION "VER 1.0.0"

// // IMPORTANT NOTICE: We must redeclare our bus range because of
```

```
// Subslave of slave device. At the here, we do not need to scan all
// These device on the bus. In any case, slave device will scan their
// Own slave device on the bus
#define I2C_BUS_SDA 5
#define I2C_BUS_SCL 4
#define I2C_START_ADDRESS 0x20
#define I2C_STOP_ADDRESS 0x65

// IMPORTANT NOTICE: On I2C bus, You can send up to 32 bits on
// Each transmission. Therefore, if there is more data than 32 bits
// We should split it. Then we can send our data to master
#define DIVISOR_NUMBER 25

// IMPORTANT NOTICE: Based on buffer size of I2C bus and maximum
// Range of your device. At the here, we declared it with 32 bit
// Because of buffer size of Arduino but if you have a bigger buffer
// Than 32 bit. you can upgrade and speed up your buffers
#define BUFFER_SIZE 32

// IMPORTANT NOTICE: IMPORTANT NOTICE: If buffer size is not looking
// Enough for you, you can extend or shrink your data with this variable.
// Due to lack of resources on memory, we were setted it as 8 but if you
// Have more memory on your device, bigger value can be compatible
#define MINIMIZED_BUFFER_SIZE 16

// Outside protocol delimiters
#define PROTOCOL_DELIMITERS ""
#define PROTOCOL_DELIMITERS_SIZE 3

// Inside protocol delimiters, we called it data delimiters
#define DATA_DELIMITER ""
#define DATA_DELIMITER_SIZE 1

// Start and end type of protocol delimiters
#define IDLE_SINGLE_START 0x15
#define IDLE_MULTI_START 0x16
#define IDLE_MULTI_END 0x17

// -----

enum handshakeData {Unknown, Ready};
enum communicationData {Idle, Continue, End};
enum notifyData {Online, Offline, Confirmed, Unconfirmed};

struct vendorData {
     char Brand[BUFFER_SIZE];
     char Model[BUFFER_SIZE];
     char Version[BUFFER_SIZE];
};

struct functionData {
     char Name[BUFFER_SIZE];
     bool Request = false;
     unsigned short Listen = false;
};

struct deviceData {
     struct vendorData vendorList;
     QueueList<functionData> functionList;
     enum handshakeData handshake = Unknown;
     char address = NULL;
};

QueueList<deviceData> deviceList;
```

```
enum communicationData communicationFlag = Idle;
enum notifyData notifyFlag = Offline;

// -----

// Do not change default value of this variable
char receivedBuffer[BUFFER_SIZE * BUFFER_SIZE * MINIMIZED_BUFFER_SIZE];
unsigned short sizeofReceivedBuffer = 0;

// Do not change default value of this variable
char givenBuffer[MINIMIZED_BUFFER_SIZE][BUFFER_SIZE];
unsigned short sizeofGivenBuffer = 0;
unsigned short indexofGivenBuffer = 0;

char sizeofFunctionList = 2;
char *functionList[] = {"getVendors",
                        "getFunctionList"
                       };

// -----

// MQTT client objects, required for MQTT
void callBack(char* topic, byte* payload, unsigned int length);
WiFiClient wifiClient;
PubSubClient mqttClient(MQTT_SERVER, MQTT_PORT, callBack, wifiClient);

// TEMPORARILY, will be delete on release
#define WIRE_BEGIN 0x01

#define BLINK_FREQUENCY 1000
#define BLINK_RANGE 200
#define BLINK_R D6
#define BLINK_GB D7
#define SSR D5

void setup() {

     // Set PWM frequency 500, default is 1000
     analogWriteFreq(BLINK_FREQUENCY);

     // Set range 0 ~ 100, default is 0 ~ 1023
     analogWriteRange(BLINK_RANGE);

     // Initialize RGB port and device listen port
     pinMode(BLINK_R, OUTPUT);
     pinMode(BLINK_GB, OUTPUT);
     pinMode(SSR, OUTPUT);

     // Reset pins
     analogWrite(BLINK_R, BLINK_RANGE);
     analogWrite(BLINK_GB, BLINK_RANGE);
     digitalWrite(SSR, HIGH);

     // -----

     // Initialize communication on wire and serial protocol
     Serial.begin(9600);

     // Initialize communication on Wire protocol
     Wire.begin(I2C_BUS_SDA, I2C_BUS_SCL);
     Wire.begin(WIRE_BEGIN);

     // A name when discovering it as a port in ARDUINO IDE
     ArduinoOTA.setHostname(DEVICE_MODEL);
```

```
    ArduinoOTA.begin();

    // Start wifi subsystem
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
    connectWiFi();

    // -----

    // IMPORTANT NOTICE: Due to our I2C scanner lib, When a new device
    // Connected or disconnected to master, our I2C scanner lib decides
    // Which one is to be triggered
    MasterScanner.setRange(I2C_START_ADDRESS, I2C_STOP_ADDRESS);
    MasterScanner.onConnectedSlaves(connectedSlaves);
    MasterScanner.onDisconnectedSlaves(disconnectedSlaves);

    // Attach functions to lib and after run main lib
    TimerQueue.attach(listenSlave, (unsigned long)1000);
    TimerQueue.attach(listenFunction, (unsigned long)10);
    TimerQueue.attach(listenWiFi, (unsigned long)20);
    TimerQueue.start();
}

void loop() {

    // IMPORTANT NOTICE: As you can see, we do not use delay function
    // In all lib. Delay function is a non-blocking function in Arduino
    // Core. So solving this, we are using MILLIS()
    TimerQueue.loop();
}

void listenSlave() {
    MasterScanner.scanSlaves();
}

void listenFunction() {

    // IMPORTANT NOTICE: Device registering is more priority than others
    // Step, When new device(s) were connected to master device, firstly
    // Register these device(s) to system, after continue what you do
    for (unsigned short index = 0; index < deviceList.size(); index++) {

        // If current index is empty, go next
        if (deviceList[index].functionList.size() == 0)
            continue;

        // If handshake is not ok, that's mean probably function is also not ok
        if (deviceList[index].handshake != Ready)
            continue;

        // Listen functions on connected device(s)
        for (unsigned short subindex = 0; subindex <
            deviceList[index].functionList.size(); subindex++) {

            if (!deviceList[index].functionList[subindex].Listen)
                continue;

            // Store func name at here, we can not use it directly
            char internalData1[BUFFER_SIZE];
            sprintf(internalData1, "%s",
                    deviceList[index].functionList[subindex].Name);

            char *encodeDelimiter1 = generateDelimiterBuffer(DATA_DELIMITER, 1);
            char *resultBuffer1[] = {internalData1, "NULL"};
            char *result1 = Serialization.encode(1, encodeDelimiter1, 2,
```

```
                                   resultBuffer1);

                    // Write internal function list to connected device, one-by-one
                    writeData(deviceList[index].address, result1);

                    // We will do this till decoding will return false
                    while (true) {

                         // More info was given at inside of this function
                         // Actually, that's not worst case
                         if (!readData(deviceList[index].address))
                              break;

                         // IMPORTANT NOTICE: At the here, We are making output control.
                         // The code that given at above changes global flag output(s). So,
                         // For next operations, We need to check this output(s) and in this
                         // Way detect our current status
                         if (communicationFlag != Continue)
                              break;
                    }

                    // If it is still IDLE, that's mean data is corrupted (Not END)
                    if (communicationFlag == Idle) {
                         unknownEvent(sizeofReceivedBuffer, receivedBuffer);
                         break;
                    }

                    // -----

                    // Decode last given data from slave, after we will publish it
                    char *subDelimiter = generateDelimiterBuffer(DATA_DELIMITER, 1);
                    char **subBuffer = Serialization.decode(1, subDelimiter,
                                        sizeofReceivedBuffer, receivedBuffer);

                    // Null operator check
                    if (subBuffer == NULL)
                         break;

                    // -----

                    // Store func name at here, we can not use it directly
                    char internalData2[BUFFER_SIZE];
                    sprintf(internalData2, "0x%2x", deviceList[index].address);

                    // Looking good, inline if-loop
                    char *encodeDelimiter2 = generateDelimiterBuffer("/", 2);
                    char *resultBuffer2[] = {internalData2, internalData1};
                    char *result2 = Serialization.encode(2, encodeDelimiter2, 2,
                                        resultBuffer2);

                    // Publish last received buffer to MQTT broker
                    mqttClient.publish(result2, subBuffer[1]);
               }
          }
     }

     void listenWiFi() {

          // Reconnect if connection is lost
          if (!mqttClient.connected() && WiFi.status() == 3)
               connectWiFi();

          // Maintain MQTT connection
          mqttClient.loop();
```

```
     // Allow Wi-Fi functions to run
     ArduinoOTA.handle();
}

// -----

void connectedSlaves(uint8_t data[], byte sizeofData) {

     // IMPORTANT NOTICE: Before the calling internal functions,
     // Last stored data must be removed on memory. Otherwise, we can not sent
     // Last stored data to master device. And additional, data removing will refresh
     // the size of data in memory. This is most important thing ...
     for (char index = 0; index < sizeofGivenBuffer; index++)
          for (char subindex = 0; subindex < BUFFER_SIZE; subindex++)
               givenBuffer[index][subindex] = '\0';

     sizeofGivenBuffer = 0;
     indexofGivenBuffer = 0;

     // Free up out-of-date buffer data
     receivedBuffer[0] = '\0';
     sizeofReceivedBuffer = 0;

     // -----

     Serial.print("Done! ");
     Serial.print(sizeofData, DEC);
     Serial.println(" slave device(s) connected to I2C bus. ID(s): ");

     for (unsigned short index = 0; index < sizeofData; index++) {
          Serial.print("\t ID: 0x");
          Serial.println(data[index], HEX);
     }
     Serial.println("\t ---");

     // Notify end user, status is device online
          notifyBlink(0, Online);

     // -----

     // IMPORTANT NOTICE: Device registering is more priority than others
     // Step, When new device(s) were connected to master device, firstly
     // Register these device(s) to system, after continue what you do
     for (unsigned short index = 0; index < sizeofData; index++) {

          // Register last added device to system
          for (unsigned short subindex = 0; subindex < sizeofFunctionList; subindex++) {

               char *encodeDelimiter = generateDelimiterBuffer(DATA_DELIMITER, 1);
               char *resultBuffer[] = {functionList[subindex], "NULL"};
               char *result = Serialization.encode(1, encodeDelimiter, 2, resultBuffer);

               // Write internal function list to connected device, one-by-one
               writeData(data[index], result);

               // We will do this till decoding will return false
               while (true) {

                    // More info was given at inside of this function
                    // Actually, that's not worst case
                    if (!readData(data[index]))
                         break;
```

```
                    if (communicationFlag != Continue)
                         break;
               }

               if (communicationFlag == Idle) {
                    unknownEvent(sizeofReceivedBuffer, receivedBuffer);
                    break;
               }

               if (!fillConfigurationData(subindex, data[index]))
               break;

               // -----

               // Subscribe all function of current slave to MQTT broker
               subscribeTopic(data[index], true);
          }
     }
}

void disconnectedSlaves(uint8_t data[], byte sizeofData) {

     // IMPORTANT NOTICE: Before the calling internal functions,
     // Last stored data must be removed on memory. Otherwise, we can not sent
     // Last stored data to master device. And additional, data removing will refresh
     // the size of data in memory. This is most important thing ...
     for (char index = 0; index < sizeofGivenBuffer; index++)
          for (char subindex = 0; subindex < BUFFER_SIZE; subindex++)
               givenBuffer[index][subindex] = '\0';

     sizeofGivenBuffer = 0;
     indexofGivenBuffer = 0;

     // Free up out-of-date buffer data
     receivedBuffer[0] = '\0';
     sizeofReceivedBuffer = 0;

     // -----

     Serial.print("Done! ");
     Serial.print(sizeofData, DEC);
     Serial.println(" slave device(s) disconnected from I2C bus.");

     for (unsigned short index = 0; index < sizeofData; index++) {
          Serial.print("\t ID: 0x");
          Serial.println(data[index], HEX);

          // Unsubscribe all function of current slave to MQTT broker
          subscribeTopic(data[index], false);
     }
     Serial.println("\t ---");

     // Notify end user, status is device online
     notifyBlink(0, Offline);

     // -----

     // IMPORTANT NOTICE: At the here, firstly, we will make a search about
     // Disconnected device. When we find it, We will delete index of this
     // Device, and after we will add all temporarily popped device again
     // If disconnected count is equal to size of device data, delete all
     if (sizeofData >= deviceList.size())
          deviceList.clear();
     else {
```

```
          for (unsigned short index = 0; index < sizeofData; index++) {

               // Just to be on the safe side, check that queue is empty or not
               if (deviceList.size() == 0)
                    break;

               if (data[index] == deviceList[0].address) {
                    deviceList.popFront();
                    continue;
               }

               if (data[index] == deviceList[deviceList.size() - 1].address) {
                    deviceList.popBack();
                    continue;
               }

               for (unsigned short subindex = 1; subindex < deviceList.size() - 1;
                    subindex++) {
                    if (deviceList[subindex].address == data[index]) {

                         copyConfigurationData(0, subindex);
                         deviceList.popFront();

                         // We found, stop the current session
                         break;
                    }
               }
          }
     }
}

// -----

void unknownEvent(unsigned short sizeofData, char data[]) {

     // IMPORTANT NOTICE: Before the calling internal functions,
     // Last stored data must be removed on memory. Otherwise, we can not sent
     // Last stored data to master device. And additional, data removing will refresh
     // the size of data in memory. This is most important thing ...
     for (char index = 0; index < sizeofGivenBuffer; index++)
          for (char subindex = 0; subindex < BUFFER_SIZE; subindex++)
               givenBuffer[index][subindex] = '\0';

     sizeofGivenBuffer = 0;
     indexofGivenBuffer = 0;

     // Free up out-of-date buffer data
     receivedBuffer[0] = '\0';
     sizeofReceivedBuffer = 0;

     // -----

     // Notify user
     Serial.print("Error! Unexpected <");
     Serial.print(data);
     Serial.print(">[");
     Serial.print(sizeofData);
     Serial.println("] data received.");

     // At the end, free up out-of-date buffer data
     receivedBuffer[0] = '\0';
     sizeofReceivedBuffer = 0;
}
```

```
bool subscribeTopic(char address, bool type) {

     // Make sure we are connected to WIFI before attemping to reconnect to MQTT
     if (WiFi.status() != WL_CONNECTED)
          return false;

     // Make sure we are connected to the MQTT server
     if (!mqttClient.connected())
          return false;

     // -----

     // When we found given device in device list, generate MQTT vendor(s)
     for (unsigned short index = 0; index < deviceList.size(); index++) {
          if (address == deviceList[index].address) {

               // Store func name at here, we can not use it directly
               char internalData[BUFFER_SIZE];
               sprintf(internalData, "0x%2x", address);

               // Looking good, inline if-loop
               char *encodeDelimiter1 = generateDelimiterBuffer("/", 2);
               char *resultBuffer1[] = {internalData, "isConnected"};
               char *result1 = Serialization.encode(2, encodeDelimiter1, 2,
                              resultBuffer1);
               type ? mqttClient.subscribe(result1) : mqttClient.unsubscribe(result1);
               mqttClient.publish(result1, type ? "1" : "0");

               char *encodeDelimiter2 = generateDelimiterBuffer("/", 2);
               char *resultBuffer2[] = {internalData, "brand"};
               char *result2 = Serialization.encode(2, encodeDelimiter2, 2,
                              resultBuffer2);
               type ? mqttClient.publish(result2, deviceList[index].vendorList.Brand) :
                 NULL;

               char *encodeDelimiter3 = generateDelimiterBuffer("/", 2);
               char *resultBuffer3[] = {internalData, "model"};
               char *result3 = Serialization.encode(2, encodeDelimiter3, 2,
                              resultBuffer3);
               type ? mqttClient.publish(result3, deviceList[index].vendorList.Model) :
                 NULL;

               char *encodeDelimiter4 = generateDelimiterBuffer("/", 2);
               char *resultBuffer4[] = {internalData, "version"};
               char *result4 = Serialization.encode(2, encodeDelimiter4, 2,
                              resultBuffer4);
               type ? mqttClient.publish(result4, deviceList[index].vendorList.Version) :
                 NULL;

               // -----

               for (unsigned short subindex = 0; subindex <
                    deviceList[index].functionList.size(); subindex++) {

                    if (!deviceList[index].functionList[subindex].Request)
                         continue;

                    // Store func name at here, we can not use it directly
                    char subInternalData0[BUFFER_SIZE * MINIMIZED_BUFFER_SIZE];
                    sprintf(subInternalData0, "0x%2x", deviceList[index].address);

                    // Store address at here, we can not use it directly
                    char subInternalData1[BUFFER_SIZE * MINIMIZED_BUFFER_SIZE];
```

```
                    sprintf(subInternalData1, "%s",
                            deviceList[index].functionList[subindex].Name);

                    char *subEncodeDelimiter = generateDelimiterBuffer("/", 2);
                    char *subResultBuffer[] = {subInternalData0, subInternalData1};
                    char *subResult = Serialization.encode(2, subEncodeDelimiter, 2,
                                    subResultBuffer);

                    // Looking good, inline if-loop
                    type ? mqttClient.subscribe(subResult) :
                           mqttClient.unsubscribe(subResult);
                }

                // Do not need to search all data, we are OK now
                break;
            }
        }
}

void callBack(char* topic, byte * payload, unsigned int length) {

     // Convert topic to string to make it easier to work with
     String stringofTopic = topic;
     unsigned short sizeofTopic = stringofTopic.length();

     // Print out some debugging info
     Serial.print("Done! Callback updated on <");
     Serial.print(stringofTopic);
     Serial.print(">[");
     Serial.print(length);
     Serial.println("].");

     // Decode payload to char array
     char payloadData[length + 1];
     for (unsigned short subindex = 0; subindex < sizeofTopic; subindex++)
          payloadData[subindex] = (char)payload[subindex];
     payloadData[length] = '\0';

     // -----

     // Generate a delimiter data and use in with decoding function
     char *decodeDelimiter = generateDelimiterBuffer("/", 2);
     char **decodeBuffer = Serialization.decode(2, decodeDelimiter, sizeofTopic, topic);

     // Null operator check
     if (decodeBuffer != NULL) {

          char internalData0[BUFFER_SIZE];
          sprintf(internalData0, "%s", decodeBuffer[0]);
          unsigned short sizeofInternalData0 = strlen(internalData0);

          char internalData1[BUFFER_SIZE];
          sprintf(internalData1, "%s", decodeBuffer[1]);
          unsigned short sizeofInternalData1 = strlen(internalData1);

          // -----

          // Declare a flag, related with device status
          bool deviceModelFlag = true;

          // Compare internal data(s) with device model, higher priority
          for (unsigned short index = 0; index < strlen(DEVICE_MODEL); index++) {
               if (internalData0[index] != DEVICE_MODEL[index]) {
                    deviceModelFlag = false;
```

```
                    break;
               }
          }

          // IMPORTANT NOTICE: If we can arrive there, that's mean the payload
          // Data is consisted of status data. Otherwise, we can say the payload
          // Data is not consisted of status data. Worst case, Go other state
          // Of if, that's mean also it is a function data(s)
          if (deviceModelFlag && isNumeric(1, payloadData))
               digitalWrite(SSR, (bool)(payload[0] - '0'));

          // -----

          // Compare internal data(s) with registered function list
          for (unsigned short index = 0; index < deviceList.size(); index++) {

               // Store func name at the here, we can not use it directly
               char internalData2[BUFFER_SIZE];
               sprintf(internalData2, "0x%2x", deviceList[index].address);

               // Check thar is it same or not
               for (unsigned short subindex = 0; subindex < sizeofInternalData0;
                    subindex++)
                    if (internalData0[subindex] != internalData2[subindex])
                         break;

               // -----

               char *encodeDelimiter = generateDelimiterBuffer(DATA_DELIMITER, 1);
               char *resultBuffer[] = {internalData1, payloadData};
               char *result = Serialization.encode(1, encodeDelimiter, 2, resultBuffer);

               // Write internal data list to connected device, one-by-one
               writeData(deviceList[index].address, result);
          }
     }
}

void connectWiFi() {

     // Attempt to connect to the wifi if connection is lost
     if (WiFi.status() != WL_CONNECTED) {

          // Notify user
          Serial.print("Connecting to <");
          Serial.print(WIFI_SSID);
          Serial.print("> ...");

          // Loop while we wait for connection
          while (WiFi.status() != WL_CONNECTED) {
               delay(500);
               Serial.print(".");
          }

          // Notify user
          Serial.println("");
          Serial.print("Done! WiFi connection is OK. IP address was given as <");
          Serial.print(WiFi.localIP());
          Serial.println(">.");
     }

     // If we are not connected to server, try to connect server
     while (!mqttClient.connected()) {
          if (mqttClient.connect((char*) DEVICE_MODEL, MQTT_USER, MQTT_PASSWORD)) {
```

```
                // IMPORTANT NOTICE: First of all, we need to subscribe main device
                // We call it like XXXX/status and this broker is related with SSR
                // State of device. We will listen something about this and execute it
                char *encodeDelimiter = generateDelimiterBuffer("/", 2);
                char *resultBuffer[] = {DEVICE_MODEL, "status"};
                char *result = Serialization.encode(2, encodeDelimiter, 2, resultBuffer);
                mqttClient.subscribe(result);
            }
        }
}

void notifyBlink(unsigned short port, enum notifyData status) {

     switch (status) {
     case Online:
          for (int index = 0; index < 3; index++) {
               for (int subindex = BLINK_RANGE; subindex > 0; subindex--) {
                    analogWrite(BLINK_R, subindex);
                    analogWrite(BLINK_GB, subindex);
                    delay(1);
               }
               delay(400);
               for (int subindex = 0; subindex < BLINK_RANGE; subindex++) {
                    analogWrite(BLINK_R, subindex);
                    analogWrite(BLINK_GB, subindex);
                    delay(2);
               }
               delay(800);
          }
          for (int index = BLINK_RANGE; index > 0; index--) {
               analogWrite(BLINK_R, index);
               analogWrite(BLINK_GB, index);
               delay(1);
          }
          // Update last stored notify flag
          notifyFlag = Online;
          break;

     case Offline:
          switch (notifyFlag) {
          case Online:
               for (int index = 0; index < BLINK_RANGE; index++) {
                    analogWrite(BLINK_R, index);
                    analogWrite(BLINK_GB, index);
                    delay(1);
               }
               break;

          case Confirmed:
               for (int index = 0; index < BLINK_RANGE; index++) {
                    analogWrite(BLINK_GB, index);
                    delay(1);
               }
               break;

          case Unconfirmed:
               for (int index = 0; index < BLINK_RANGE; index++) {
                    analogWrite(BLINK_R, index);
                    delay(1);
               }
               break;

          default:
```

```
                analogWrite(BLINK_R, BLINK_RANGE);
                analogWrite(BLINK_GB, BLINK_RANGE);
                break;
          }
          // Update last stored notify flag
          notifyFlag = Offline;
          break;

     case Confirmed:
          for (int index = 0; index < BLINK_RANGE; index++) {
                analogWrite(BLINK_R, index);
                delay(1);
          }
          // Update last stored notify flag
          notifyFlag = Confirmed;
          break;

     case Unconfirmed:
          for (int index = BLINK_RANGE; index > 0; index--) {
                analogWrite(BLINK_GB, index);
                delay(1);
          }
          // Update last stored notify flag
          notifyFlag = Unconfirmed;
          break;

     default:
          analogWrite(BLINK_R, BLINK_RANGE);
          analogWrite(BLINK_GB, BLINK_RANGE);
          // Update last stored notify flag
          notifyFlag = Offline;
          break;
     }
}

// -----

bool writeData(char address, char *data) {

     // Free up out-of-date buffer data, top level clearing
     receivedBuffer[0] = '\0';
     sizeofReceivedBuffer = 0;

     // Encode given data for bus communication
     encodeData(strlen(data), data);

     // Send all remainder data to newly registered slave
     while (indexofGivenBuffer < sizeofGivenBuffer) {
          Wire.beginTransmission(address);
          Wire.write(givenBuffer[indexofGivenBuffer++]);
          Wire.endTransmission();

          // Maybe not need, right?
          delay(15);
     }

     // IMPORTANT NOTICE: Due to decoding of slave device, we need to Wait
     // A little bit. Otherwise, Master device will request data From slave
     // Device too early and slave cannot send it. Additional, when more
     // Devices are connected, we need to downgrade delay time. Already,
     // It will take the same time during roaming
     delay(100);

     // If everything goes well, we will arrive here and return true
```

```
        return true;
}

bool readData(char address) {

        unsigned short indexofNewReceivedBuffer = 0;
        char newReceivedBuffer[BUFFER_SIZE];

        // Request data from slave
        Wire.requestFrom(address, BUFFER_SIZE);
        while (Wire.available()) {

                // Store new received data at the here
                char currentBuffer = Wire.read();
                newReceivedBuffer[indexofNewReceivedBuffer++] = (char)currentBuffer;

                // Don't forget to add end-of-line char to end
                if (currentBuffer == (char)PROTOCOL_DELIMITERS[2]) {
                        newReceivedBuffer[indexofNewReceivedBuffer] = '\0';
                        break;
                }
        }

        // Maybe not need, right?
        delay(15);

        // Decode last given data
        if (!decodeData(indexofNewReceivedBuffer, newReceivedBuffer))
                return false;

        // IMPORTANT NOTICE: Actually When we arrived this point, we arrived
        // Worst case point even though It was TRUE. If you came there, program will
        // Run till communication flag will be END or IDLE type. Otherwise, this
        // Point is related with CONTINUE status
        return true;
}

// -----

void copyConfigurationData(char fromAddress, char toAddress) {

        // Worst case, We do not have another solution
        for (unsigned short index = 0; index < BUFFER_SIZE; index++) {
                if (deviceList[fromAddress].vendorList.Brand[index] == NULL) {
                        deviceList[toAddress].vendorList.Brand[index] = '\0';
                        break;
                }
                deviceList[toAddress].vendorList.Brand[index] =
                deviceList[fromAddress].vendorList.Brand[index];
        }

        for (unsigned short index = 0; index < BUFFER_SIZE; index++) {
                if (deviceList[fromAddress].vendorList.Model[index] == NULL) {
                        deviceList[toAddress].vendorList.Model[index] = '\0';
                        break;
                }
                deviceList[toAddress].vendorList.Model[index] =
                deviceList[fromAddress].vendorList.Model[index];
        }

        for (unsigned short index = 0; index < BUFFER_SIZE; index++) {
                if (deviceList[fromAddress].vendorList.Version[index] == NULL) {
                        deviceList[toAddress].vendorList.Version[index] = '\0';
                        break;
```

```
            }
            deviceList[toAddress].vendorList.Version[index] =
            deviceList[fromAddress].vendorList.Version[index];
        }

        // -----

        // Worst case, We do not have another solution
        for (unsigned short index = 0; index < deviceList[fromAddress].functionList.size();
            index++) {

            for (unsigned short subindex = 0; subindex < BUFFER_SIZE; subindex++) {
                if (deviceList[fromAddress].functionList[index].Name[subindex] == NULL) {
                    deviceList[toAddress].functionList[index].Name[subindex] = '\0';
                    break;
                }
                deviceList[toAddress].functionList[index].Name[subindex] =
                deviceList[fromAddress].functionList[index].Name[subindex];
            }

            deviceList[toAddress].functionList[index].Request =
            deviceList[fromAddress].functionList[index].Request;
            deviceList[toAddress].functionList[index].Listen =
            deviceList[fromAddress].functionList[index].Listen;
        }

        deviceList[toAddress].handshake = deviceList[fromAddress].handshake;
        deviceList[toAddress].address = deviceList[fromAddress].address;
    }

    bool fillConfigurationData(unsigned short index, char address) {

        // IMPORTANT NOTICE: Before decoding inside data, we need firstly
        // Calculate count of data delimiter. This result gives us that how
        // Many different data included in this inside data
        unsigned short countofCharacter = howManyCharacter(DATA_DELIMITER,
                                          sizeofReceivedBuffer, receivedBuffer);

        // Generate a delimiter data and use in with decoding function
        char *decodeDelimiter = generateDelimiterBuffer(DATA_DELIMITER, countofCharacter);
        char **newReceivedBuffer = Serialization.decode(countofCharacter, decodeDelimiter,
                                   sizeofReceivedBuffer, receivedBuffer);

        // Null operator check
        if (newReceivedBuffer == NULL)
            return false;

        // IMPORTANT NOTICE: In this program, we have two data type, one of them
        // Is VENDOR data, other one is FUNCTION data. Because of this we have two
        // Statement in switch case
        switch (index) {
        case 0:
            if (!fillVendorData(address, countofCharacter, newReceivedBuffer))
                return false;

            // Notify user
            Serial.print("Done! The vendors of [0x");
            Serial.print(address, HEX);
            Serial.println("] address were saved successfully to system.");

            Serial.print("\t BRAND: ");
            Serial.println(deviceList[0].vendorList.Brand);
            Serial.print("\t MODEL: ");
            Serial.println(deviceList[0].vendorList.Model);
```

```
            Serial.print("\t VERSION: ");
            Serial.println(deviceList[0].vendorList.Version);
            Serial.println("\t ---");
            break;

        case 1:
            if (!fillFunctionData(address, countofCharacter, newReceivedBuffer)) {
                // Notify end user, status is device online
                notifyBlink(0, Unconfirmed);
                return false;
            }

            // Notify end user, status is device online
            notifyBlink(0, Confirmed);

            // Notify user
            for (unsigned short index = 0; index < deviceList[0].functionList.size();
                index++) {
                Serial.print("\t FUNCTION: ");
                Serial.println(deviceList[0].functionList[index].Name);
            }
            Serial.println("\t ---");
            break;

        default:
            return false;
        }

        // If everything goes well, we will arrive here and return true
        return true;
    }

    bool fillVendorData(char address, unsigned short sizeofData, char **data) {

        // Worst case, if vendor list size is not equal to default vendor list size
        if (sizeofData != 2) {

            struct deviceData newDeviceData;
            newDeviceData.handshake = Unknown;
            newDeviceData.address = address;

            // Major code for device list
            deviceList.pushFront(newDeviceData);
            unknownEvent(sizeofReceivedBuffer, receivedBuffer);

            return false;
        }

        // IMPORTANT NOTICE: When a new device is registered to master,
        // We are decoding all vendor Data at the here. When we are doing
        // All of these also we need temp variable
        struct deviceData newDeviceData;

        // Best case, we have it all
        for (unsigned short index = 0; index < BUFFER_SIZE; index++) {
            if (data[0][index] == NULL) {
                newDeviceData.vendorList.Brand[index] = '\0';
                break;
            }
            newDeviceData.vendorList.Brand[index] = data[0][index];
        }

        for (unsigned short index = 0; index < BUFFER_SIZE; index++) {
            if (data[1][index] == NULL) {
```

```
                newDeviceData.vendorList.Model[index] = '\0';
                break;
            }
            newDeviceData.vendorList.Model[index] = data[1][index];
        }

        for (unsigned short index = 0; index < BUFFER_SIZE; index++) {
            if (data[2][index] == NULL) {
                newDeviceData.vendorList.Version[index] = '\0';
                break;
            }
            newDeviceData.vendorList.Version[index] = data[2][index];
        }

        // -----

        // Major code for device list
        deviceList.pushFront(newDeviceData);

        // If everything goes well, we will arrive here and return true
        return true;
    }

    bool fillFunctionData(char address, unsigned short sizeofData, char **data) {

        // Worst case, function list size is equal to char - 1 size
        if ((sizeofData + 1) % 3 != 0)
            return false;

        for (unsigned short index = 0; index < sizeofData + 1; index += 3) {

            char internalData0[BUFFER_SIZE];
            sprintf(internalData0, "%s", data[index]);
            unsigned short sizeofInternalData0 = strlen(internalData0);
            if (!isAlphanumeric(sizeofInternalData0, internalData0))
                continue;

            char internalData1[BUFFER_SIZE];
            sprintf(internalData1, "%s", data[index + 1]);
            unsigned short sizeofInternalData1 = strlen(internalData1);
            if (!isNumeric(sizeofInternalData1, internalData1))
                continue;

            char internalData2[BUFFER_SIZE];
            sprintf(internalData2, "%s", data[index + 2]);
            unsigned short sizeofInternalData2 = strlen(internalData2);
            if (!isNumeric(sizeofInternalData2, internalData2))
                continue;

            // -----

            // IMPORTANT NOTICE: When a new device is registered to master,
            // We are decoding all function Data at the here. When we are doing
            // All of these also we need temp variable
            struct functionData newFunctionData;

            for (unsigned short subindex = 0; subindex < BUFFER_SIZE; subindex++) {
                if (internalData0[subindex] == NULL) {
                    newFunctionData.Name[subindex] = '\0';
                    break;
                }
                newFunctionData.Name[subindex] = internalData0[subindex];
            }
```

```
            newFunctionData.Request = (bool)atoi(internalData1);
            newFunctionData.Listen = (bool)atoi(internalData2);

            // If we can arrive there, we can say function data is ok
            deviceList[0].functionList.pushFront(newFunctionData);
      }

      // Set other important externals data to internals list
      deviceList[0].handshake = Ready;
      deviceList[0].address = address;

      // If everything goes well, we will arrive here and return true
      return true;
}

// -----

bool decodeData(unsigned short sizeofData, char data[]) {

      if (sizeofData == 0 || data == NULL)
            return false;

      // Decode given data, Calculate up-of-date and needed buffer size
      char **newReceivedBuffer = Serialization.decode(PROTOCOL_DELIMITERS_SIZE,
                                 PROTOCOL_DELIMITERS, sizeofData, data);

      // Null operator check
      if (newReceivedBuffer == NULL)
      return false;

      // Null operator check
      if (newReceivedBuffer[0] == NULL || newReceivedBuffer[1] == NULL)
            return false;

      // -----

      // IMPORTANT NOTICE: As I said before, NodeMCU do not support lots of
      // C++ library. This one is one of these, also. At the here, we can not
      // Use directly strlen function. For solving, first we need to get inside data,
      // After we need to calculate size of the given data
      char internalData0[BUFFER_SIZE];
      sprintf(internalData0, "%s", newReceivedBuffer[0]);
      unsigned short sizeofInternalData0 = strlen(internalData0);

      char internalData1[BUFFER_SIZE];
      sprintf(internalData1, "%s", newReceivedBuffer[1]);
      unsigned short sizeofInternalData1 = strlen(internalData1);

      // -----

      switch (internalData1[0]) {
      case IDLE_SINGLE_START:
      case IDLE_MULTI_END:
            communicationFlag = End;
            break;

      case IDLE_MULTI_START:
            communicationFlag = Continue;
      break;

      default:
            communicationFlag = Idle;

            // Free up out-of-date buffer data
```

```
            receivedBuffer[0] = '\0';
            sizeofReceivedBuffer = 0;

            return false;
      }

      for (unsigned short index = 0; index < sizeofInternalData0; index++)
            receivedBuffer[sizeofReceivedBuffer + index] = internalData0[index];

      // Add an endofline character to tail
      receivedBuffer[sizeofReceivedBuffer + sizeofInternalData0] = '\0';
      sizeofReceivedBuffer += sizeofInternalData0;

      // If everything goes well, we will arrive here and return true
      return true;
}

bool encodeData(unsigned short sizeofData, char data[]) {

      // IMPORTANT NOTICE: Before the calling internal functions,
      // Last stored data must be removed on memory. Otherwise, we can not sent
      // Last stored data to master device. And additional, data removing will refresh
      // the size of data in memory. This is most important thing ...
      for (char index = 0; index < sizeofGivenBuffer; index++)
            for (char subindex = 0; subindex < BUFFER_SIZE; subindex++)
                  givenBuffer[index][subindex] = '\0';

      sizeofGivenBuffer = 0;
      indexofGivenBuffer = 0;

      // -----

      if (sizeofData == 0 || data == NULL)
            return false;

      // Second, calculete size of data and modulus
      unsigned short modulusofGivenBuffer = (sizeofData % DIVISOR_NUMBER);
      sizeofGivenBuffer = (sizeofData / DIVISOR_NUMBER);

      // Add modulos of data, if possible
      if (modulusofGivenBuffer > 0)
            sizeofGivenBuffer++;

      // -----

      for (unsigned short index = 0; index < sizeofGivenBuffer; index++) {

            unsigned short subIndex;
            unsigned short upperBound = (index == sizeofGivenBuffer - 1 ?
                                         modulusofGivenBuffer : DIVISOR_NUMBER);

            for (subIndex = 0; subIndex < upperBound; subIndex++)
                  givenBuffer[index][subIndex + 1] = data[(index * DIVISOR_NUMBER) +
                                                          subIndex];

            if (sizeofGivenBuffer == 1)
                  givenBuffer[index][subIndex + 2] = (char)IDLE_SINGLE_START;
            else if (index == sizeofGivenBuffer - 1)
                  givenBuffer[index][subIndex + 2] = (char)IDLE_MULTI_END;
            else
                  givenBuffer[index][subIndex + 2] = (char)IDLE_MULTI_START;

            givenBuffer[index][0] = PROTOCOL_DELIMITERS[0];
            givenBuffer[index][subIndex + 1] = PROTOCOL_DELIMITERS[1];
```

```
            givenBuffer[index][subIndex + 3] = PROTOCOL_DELIMITERS[2];
            givenBuffer[index][subIndex + 4] = '\0';
      }

      // If everything goes well, we will arrive here and return true
      return true;
}

// -----

char *generateDelimiterBuffer(char *character, unsigned short sizeofData) {

      if (sizeofData == 0)
            return false;

      // Store count of found on this variable
      static char arrayofCharacter[BUFFER_SIZE * MINIMIZED_BUFFER_SIZE];

      // Set first bit as end-of-line biti this is for initialize
      arrayofCharacter[0] = '\0';

      for (unsigned short index = 0; index < sizeofData; index++)
            arrayofCharacter[index] = *character;

      // Set last bit as end-of-line bit
      arrayofCharacter[sizeofData] = '\0';

      return arrayofCharacter;
}

unsigned short howManyCharacter(char *character, unsigned short sizeofData, char data[]) {

      if (sizeofData == 0 || data == NULL)
            return 0;

      // Store count of found on this variable
      char countofCharacter = 0;

      for (unsigned short index = 0; index < sizeofData; index++)
            if (*character == data[index])
      countofCharacter++;

      return countofCharacter;
}

bool isNumeric(unsigned short sizeofData, char data[]) {

      if (sizeofData == 0 || data == NULL)
            return false;

      for (unsigned short index = 0; index < sizeofData; index++)
            if (isdigit(data[index]) == 0)
                  return false;

      return true;
}

bool isAlphanumeric(unsigned short sizeofData, char data[]) {

      if (sizeofData == 0 || data == NULL)
            return false;

      // Check function ID, type is alphanumeric
      for (unsigned short index = 0; index < sizeofData; index++)
```

```
            if (isalnum(data[index]) == 0)
                return false;

        return true;
    }
```

8.5 Slave.INO

```
#include <Wire.h>
#include <Serializer.h>
#include <MemoryFree.h>

// IMPORTANT NOTICE: These all constant is depending on your protocol
// As you can see, this protocol delimiter was declared in this scope
// That's mean, all function will use this delimiter constant on
// Communication between two or more devices
#define DEVICE_BRAND "intelliPWR"
#define DEVICE_MODEL "Slave Tester"
#define DEVICE_VERSION "1.0.0"

// IMPORTANT NOTICE: On I2C bus, You can send up to 32 bits on
// Each transmission. Therefore, if there is more data than 32 bits
// We should split it. Then we can send our data to master
#define DIVISOR_NUMBER 25

// Outside protocol delimiters
#define PROTOCOL_DELIMITERS ""
#define PROTOCOL_DELIMITERS_SIZE 3

// Inside protocol delimiters, we called it data delimiters
#define DATA_DELIMITER ""
#define DATA_DELIMITER_SIZE 1

// Start and end type of protocol delimiters
#define IDLE_SINGLE_START 0x15
#define IDLE_MULTI_START 0x16
#define IDLE_MULTI_END 0x17

// -----

// Do not change default value of this variable
char *receivedBuffer = NULL;
unsigned short sizeofReceivedBuffer = 0;

// Do not change default value of this variable
char** givenBuffer = NULL;
unsigned short sizeofGivenBuffer = 0;
unsigned short indexofGivenBuffer = 0;

const char sizeofFunctionList = 8;
const char *functionList[] = {"getVendors",
                              "getFunctionList"
                             };

const char sizeofReturnList = sizeofFunctionList;
const char *returnList[] = {"0",
                            "0"
                           };

const char sizeofListenList = sizeofFunctionList;
const char *listenList[] = {"0",
                            "0"
```

```
                              };

// TEMPORARILY, will be delete on release
#define WIRE_BEGIN 0x60

void setup() {

     // Register events, onReceive and onRequest
     Wire.onReceive(receiveEvent);
     Wire.onRequest(requestEvent);

     // Initialize communication on serial protocol
     Serial.begin(9600);

     // Initialize communication on Wire protocol
     Wire.begin(WIRE_BEGIN);
}

void loop() {
}

void receiveEvent(unsigned short sizeofData) {

     // Declare variables about given size
     char *newReceivedBuffer = (char *) malloc(sizeof (char) * (sizeofData + 1));
     unsigned short indexofNewReceivedBuffer = 0;

     // Loop through all but the last
     while (Wire.available()) {

          char currentBuffer = Wire.read();
          newReceivedBuffer[indexofNewReceivedBuffer++] = (char)currentBuffer;

          if (currentBuffer == (char)PROTOCOL_DELIMITERS[2]) {
               newReceivedBuffer[indexofNewReceivedBuffer] = '\0';
               break;
          }
     }

     // -----

     if (!decodeData(indexofNewReceivedBuffer, newReceivedBuffer))
          unknownEvent(indexofNewReceivedBuffer, newReceivedBuffer);

     // Do not forget to free up local data
     free(newReceivedBuffer);
}

void requestEvent() {

     // IMPORTANT NOTICE: On the worst case, we are sending empty protocol
     // Data(s) this is the best way of worst case because the receiver device
     // Can always Decode this data and when we send "not filled" protocol
     // Data(s), this means we are sending empty data(s)
     if (indexofGivenBuffer >= sizeofGivenBuffer && sizeofGivenBuffer == 0) {
          Wire.write("UNKNOWN_REQUEST");

          // IMPORTANT NOTICE: This point can explaine as the least desirable
          // Situation. When program arrrives this point, that's mean master
          // Send more than calculated data or less than calculated data. But
          // Do not worry, I wrote all code blocks as far as stable. It will
          // Terminate itself automaticly when it was arrive there
          clearGivenBuffer();
     }
```

```
     else
          Wire.write(givenBuffer[indexofGivenBuffer++]);
}

// -----

void unknownEvent(unsigned short sizeofData, char data[]) {

     // Notify user about available memory
     Serial.print("[Available memory: ");
     Serial.print(freeMemory());
     Serial.print("] ");

     // Notify user
     Serial.print("Error! Unexpected <");
     Serial.print(data);
     Serial.print(">[");
     Serial.print(sizeofData);
     Serial.println("] data received.");

     // At the end, free up out-of-date buffer data
     clearGivenBuffer();
     clearReceivedBuffer();
}

void executeEvent(unsigned short sizeofData, char data[]) {

     // Store index of found function
     char foundIndex = 0;
     bool foundFlag = false;

     // Decode given data and store it
     char **insideData = Serialization.decode(DATA_DELIMITER_SIZE, DATA_DELIMITER,
                         sizeofData, data);

     for (char index = 0; index < sizeofFunctionList; index++) {

     // Null operator check
     if (insideData == NULL)
          break;

     // Null operator check
     if (insideData[0] == NULL || insideData[1] == NULL)
          break;

     // Calculate registered function length at the given index
     unsigned short sizeofInternalFunction = strlen(functionList[index]);
     unsigned short sizeofExternalFunction = strlen(insideData[0]);

     // Data size check
     if (sizeofInternalFunction != sizeofExternalFunction)
          continue;

     // -----

     // Found status flag, using for to find operate
     foundFlag = true;

     for (unsigned short subIndex = 0; subIndex < sizeofInternalFunction; subIndex++) {
          if (insideData[0][subIndex] != functionList[index][subIndex]) {
               foundFlag = false;
               break;
          }
```

```
        }

        // If a function found in function list,
        // Store index of this function and break current loop
        if (foundFlag) {
            foundIndex = index;
            break;
        }
    }

        // In the worst case, we can not find a function and found index is -1
        if (foundFlag)
            callReceivedFunction(foundIndex, strlen(insideData[1]), insideData[1]);
        else
            unknownEvent(sizeofData, data);

        // At the end, free up out-of-date buffer data
        clearReceivedBuffer();
    }

        // -----

        bool decodeData(unsigned short sizeofData, char data[]) {

        if (sizeofData == 0 || data == NULL)
            return false;

        // Decode given data, Calculate up-of-date and needed buffer size
        char **newReceivedBuffer = Serialization.decode(PROTOCOL_DELIMITERS_SIZE,
                                   PROTOCOL_DELIMITERS, sizeofData, data);
        unsigned short sizeofNewReceivedBuffer = strlen(newReceivedBuffer[0]);

        // Null operator check
        if (newReceivedBuffer == NULL)
            return false;

        // Null operator check
        if (newReceivedBuffer[0] == NULL || newReceivedBuffer[1] == NULL)
            return false;

        // -----

        switch (newReceivedBuffer[1][0]) {
        case IDLE_SINGLE_START:
        case IDLE_MULTI_START:
        case IDLE_MULTI_END:
            break;
        default:
            return false;
    }

        // Malloc and realloc a sentence, a list of words
        if (receivedBuffer == NULL)
            receivedBuffer = (char *) malloc(sizeof (char) * (sizeofReceivedBuffer +
                             sizeofNewReceivedBuffer + 1));
        else
            receivedBuffer = (char *) realloc(receivedBuffer, sizeof (char) *
                             (sizeofReceivedBuffer + sizeofNewReceivedBuffer + 1));

        for (unsigned short index = 0; index < sizeofNewReceivedBuffer; index++)
            receivedBuffer[sizeofReceivedBuffer + index] = newReceivedBuffer[0][index];

        // Add an endofline character to tail
        receivedBuffer[sizeofReceivedBuffer + sizeofNewReceivedBuffer] = '\0';
```

```
        sizeofReceivedBuffer += sizeofNewReceivedBuffer;

        // -----

        // Go to next step, decoding inside data
        if (newReceivedBuffer[1][0] == (char)IDLE_SINGLE_START || newReceivedBuffer[1][0] ==
                                       (char)IDLE_MULTI_END)
            executeEvent(sizeofReceivedBuffer, receivedBuffer);

        // If everything goes well, we will arrive here and return true
        return true;
    }

    bool encodeData(unsigned short sizeofData, char data[]) {

        if (sizeofData == 0 || data == NULL)
            return false;

        // Second, calculete size of data and modulus
        unsigned short modulusofGivenBuffer = (sizeofData % DIVISOR_NUMBER);
        sizeofGivenBuffer = (sizeofData / DIVISOR_NUMBER);

        // Add modulos of data, if possible
        if (modulusofGivenBuffer > 0)
            sizeofGivenBuffer++;

        // We do not need realloc code because of the top of encoding function
        givenBuffer = (char **) malloc(sizeof (char *) * (sizeofGivenBuffer + 1));

        // -----

        for (unsigned short index = 0; index < sizeofGivenBuffer; index++) {

            givenBuffer[index] = (char *) malloc(sizeof (char) * (DIVISOR_NUMBER +
                                 PROTOCOL_DELIMITERS_SIZE + 1));

            unsigned short subIndex;
            unsigned short upperBound = (index == sizeofGivenBuffer - 1 ?
                                         modulusofGivenBuffer : DIVISOR_NUMBER);

            for (subIndex = 0; subIndex < upperBound; subIndex++)
                givenBuffer[index][subIndex + 1] = data[(index * DIVISOR_NUMBER) +
                subIndex];

            // IMPORTANT NOTICE: At the here, We have two status for encoding data(s)
            // But if you set this var as multiEND, this means encoding is over
            // We are making this for receiver side. singleSTART means that data can encode
            // As one packet, do not need any more encoding
            if (sizeofGivenBuffer == 1)
                givenBuffer[index][subIndex + 2] = (char)IDLE_SINGLE_START;
            else if (index == sizeofGivenBuffer - 1)
                givenBuffer[index][subIndex + 2] = (char)IDLE_MULTI_END;
            else
                givenBuffer[index][subIndex + 2] = (char)IDLE_MULTI_START;

            givenBuffer[index][0] = PROTOCOL_DELIMITERS[0];
            givenBuffer[index][subIndex + 1] = PROTOCOL_DELIMITERS[1];
            givenBuffer[index][subIndex + 3] = PROTOCOL_DELIMITERS[2];
            givenBuffer[index][subIndex + 4] = '\0';
        }

        // Do not forget to end-of-line char to tail of 2D array
        givenBuffer[sizeofGivenBuffer] = '\0';
```

```
     // If everything goes well, we will arrive here and return true
     return true;
}

// -----

void clearGivenBuffer() {

     // IMPORTANT NOTICE: Before the calling internal functions,
     // Last stored data must be removed on memory. Otherwise, we can not sent
     // Last stored data to master device. And additional, data removing will refresh
     // the size of data in memory. This is most important thing ...
     for (char index = 0; index < sizeofGivenBuffer; index++)
          free(givenBuffer[index]);

     // After free up top level pointer
     free(givenBuffer);
     givenBuffer = NULL;
     sizeofGivenBuffer = 0;
     indexofGivenBuffer = 0;
}

void clearReceivedBuffer() {

     // IMPORTANT NOTICE: Before the calling internal functions,
     // Last stored data must be removed on memory. Otherwise, we can not sent
     // Last stored data to master device. And additional, data removing will refresh
     // the size of data in memory. This is most important thing ...
     free(receivedBuffer);
     receivedBuffer = NULL;
     sizeofReceivedBuffer = 0;
}

bool isNumeric(unsigned short sizeofData, char data[]) {

     if (sizeofData == 0 || data == NULL)
          return false;

     for (unsigned short index = 0; index < sizeofData; index++)
          if (isdigit(data[index]) == 0)
               return false;

     return true;
}

bool isAlphanumeric(unsigned short sizeofData, char data[]) {

     if (sizeofData == 0 || data == NULL)
          return false;

     // Check function ID, type is alphanumeric
     for (unsigned short index = 0; index < sizeofData; index++)
          if (isalnum(data[index]) == 0)
               return false;

     return true;
}

void callReceivedFunction(unsigned short indexofFunction, unsigned short sizeofData, char
                          data[]) {

     // At the first, free up out-of-date buffer data
     clearGivenBuffer();
```

```
        clearReceivedBuffer();

        // -----

        switch (indexofFunction) {
        case 0:
            getVendors(sizeofData, data);
            break;
        case 1:
            getFunctionList(sizeofData, data);
            break;
        default:
            break;
        }
}

// ------------------------------------------------------------
// ----- INTERNAL FUNC(S) -------------------------------------
// ------------------------------------------------------------

void getVendors(unsigned short sizeofData, char data[]) {

        // Notify user about available memory
        Serial.print("[Available memory: ");
        Serial.print(freeMemory());
        Serial.print("] ");

        // Notify users
        Serial.print("Done! ");
        Serial.print(__func__);
        Serial.println(" function triggered.");

        char *delimiterBuffer = "";
        char *resultBuffer[] = {DEVICE_BRAND, DEVICE_MODEL, DEVICE_VERSION};
        char *result = Serialization.encode(2, delimiterBuffer, 3, resultBuffer);
        encodeData(strlen(result), result);
}

void getFunctionList(unsigned short sizeofData, char data[]) {

        // Notify user about available memory
        Serial.print("[Available memory: ");
        Serial.print(freeMemory());
        Serial.print("] ");

        // Notify users
        Serial.print("Done! ");
        Serial.print(__func__);
        Serial.println(" function triggered.");

        char *delimiterBuffer = "";
        char *resultBuffer[] = {"External", "Core"};
        char *result = Serialization.encode(1, delimiterBuffer, 2, resultBuffer);
        encodeData(strlen(result), result);
}
```

9. ÖZGEÇMİŞ

Ad–Soyad:	Veysel Berk Altun
Doğum Tarihi ve Yeri:	03.12.1994, Muş
B.Sc:	Pamukkale Üniversitesi
E–posta:	contact@vberkaltun.com

Deneyim:

Haziran 2016	Kıdemli Yazılım Geliştirici, 40idea Laboratory Engineering Lab. & Consultancy Co. Ltd.
Haziran 2012	Servis Teknisyeni, Unipa A.S.

Onur ve Ödüller:

Mart 2013	ITURO Olimpiyatları Çeyrek Final Ödülü İstanbul Teknik Üniversitesi, İstanbul
Mayıs 2013	"Eğitimlence: eğlenerek öğrenmektir." Birincilik Ödülü Yaşar Üniversitesi, İzmir
Haziran 2013	Anadolu Teknik Dalında Birincilik Ödülü Mazhar Zorlu Anadolu Teknik Lisesi, İzmir
Mayıs 2018	"Genç Girişimciler Yeni Fikirler" Birincilik Ödülü Ege Üniversitesi, İzmir