Bluetooth ile verileri iletmek adına elimde bulunan esp32-cam modülünü ana geliştirme kartım olarak kullandım. Geliştirme boyunca pek çok avantajı sebebiyle esp32 kartı için ide olarak PlatformIO (VS code eklentisi) tercih ettim. İlk aşamada espressif idf ile geliştirmeyi planlıyordum ancak bazı BLE kütüphanelerini workspace üzerine eklemede problem yaşadım, zamandan tasarruf edebilmek amacıyla da sürece PIO ile devam ettim.

Elimde i2c sensörü olmadığından dolayı Arduino uno kullanarak sensör simüle etmeyi düşündüm ancak i2c voltaj seviyesi esp32 ile uyumlu değildi. (Arduino 5V, esp-cam 3.3). Arduino i2c voltaj seviyesini düşürmek için logic converter entegresine sahip olmadığımdan dolayı sensör simülasyon sürecinde elimde bulunan bir diğer geliştirme kartı olan raspberry pi pico (i2c 3.3V) kartını tercih ettim.

Elimde bulunan dht11 sensorü ile elde ettiğim verileri pico aracılığıyla i2c haberleşmesi ile esp32 cam modülüne aktarmayı planladım buna ek olarak dht sensöründen elde edemeyeceğim basınç ve direnç verilerini sabit değerler şeklinde göndereceğim. Bu noktada birden fazla sensörü simule etmek adına 2 farklı i2c pini kullanacağım. Yönergede 3 adet istenmesine rağmen Pico kartının 2 adet i2c modülüne sahip olması sebebiyle 3 adet sensörü simüle edemedim. Pico kartının bir i2c modülünden sıcaklık ve nem iletirken diğerinden sabit basınç ve direnç verileri ileteceğim.

Simüle edeceğim sensörler:

ilk sensörümüz bme280 sıcaklık ve nem verileri için kullanacağımız sensör. 0X76 adresine sahip ve hem spi hem i2c haberleşmesini desteklemekte. Aşağıdaki şematik ürünün datasheet dökümanından elde edilmiş olup cihaz ile haberleşme örneği gösterilmiştir.

6.2.1 I²C write

Writing is done by sending the slave address in write mode (RW = '0'), resulting in slave address 111011X0 ('X' is determined by state of SDO pin. Then the master sends pairs of register addresses and register data. The transaction is ended by a stop condition. This is depicted in Figure 9.

Figure 9: I²C multiple byte write (not auto-incremented)

6.2.2 I²C read

To be able to read registers, first the register address must be sent in write mode (slave address 111011X0). Then either a stop or a repeated start condition must be generated. After this the slave is addressed in read mode (RW = '1') at address 111011X1, after which the slave sends out data from auto-incremented register addresses until a NOACKM and stop condition occurs. This is depicted in Figure 10, where register 0xF6 and 0xF7 are read.

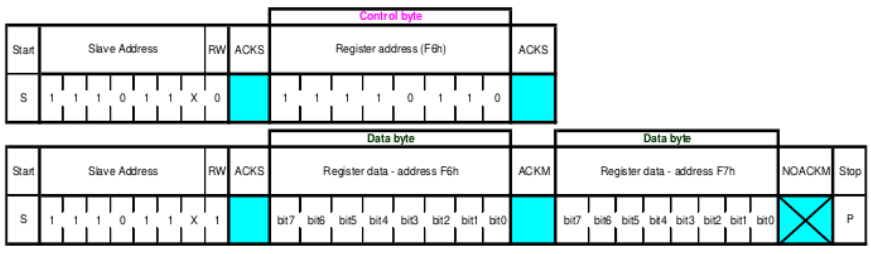


Figure 10: I²C multiple byte read

Görüldüğü üzere sensöre yazma işleminde ilk önce adres daha sonra control verisi ve en son aşamada ise yazılacak değer gönderilmekte.

Okuma işleminde ise cihazın adresi gönderildikten sonra hangi değerin okunacağını bildiren control verisi gönderilmekte.

Bu aşamada cihaz 8 er bitlik verileri ardışık sırayla önce düşük adresteki veriyi daha sonra da yüksek adresteki veri olacak şekilde göndererek işlemi tamamlamakta.
Burdaki bilgiler ışığında Raspberry Pico cihazımızı kodladık.

Sağ tarafta ise bme280 sensörü için verilerin bulunduğu register adresleri belirtilmiştir. Bu adreslerden ihtiyacımız olan değerler sıcaklık için 0xfa ve nem için 0xfd, bu değerler ile i2c veri yolundan gelen istekleri kontrol edeceğiz.

*Ürünlerin datasheet dökümanlarına source klasöründen ulaşabilirsiniz.

5.4.8 Register 0xFA...0xFC *"temp" (_msb, _lsb, _xlsb)*

The *"temp"* register contains the raw temperature measurement output data ut[19:0]. For details on how to read out the pressure and temperature information from the device, please consult chapter 4.

Table 30: Register 0xFA ... 0xFC *"temp"*

Register 0xFA...0xFC *"temp"*	Name	Description
0xFA	temp_msb[7:0]	Contains the MSB part ut[19:12] of the raw temperature measurement output data.
0xFB	temp_lsb[7:0]	Contains the LSB part ut[11:4] of the raw temperature measurement output data.
0xFC (bit 7, 6, 5, 4)	temp_xlsb[3:0]	Contains the XLSB part ut[3:0] of the raw temperature measurement output data. Contents depend on pressure resolution.

5.4.9 Register 0xFD...0xFE *"hum" (_msb, _lsb)*

The *"temp"* register contains the raw temperature measurement output data ut[19:0]. For details on how to read out the pressure and temperature information from the device, please consult chapter 4.

Table 31: Register 0xFD ... 0xFE *"hum"*

Register 0xFD...0xFE *"hum"*	Name	Description
0xFD	hum_msb[7:0]	Contains the MSB part uh[15:8] of the raw humidity measurement output data.
0xFE	temp_lsb[7:0]	Contains the LSB part uh[7:0] of the raw humidity measurement output data.

İkinci sensörümüz ise bme680 isimli yine bme280 ile aynı aileden olan bir sensör. Sensör seçimi aşamasında adres çakışması yaşamamak adına özellikle sensör adresleri kontrol edilmeli aksi halde i2c çoklayıcı gibi ekstra komponentlere gereksinim duyulabilir. 0X77 adresini sahip olan sensörümüzden basınç ve hava direnci verilerini alacağız. Bu sensör de diğer sensörümüz gibi hem i2c hem de spi haberleşmesini desteklemekte.

5.3.4.1 *Pressure data*

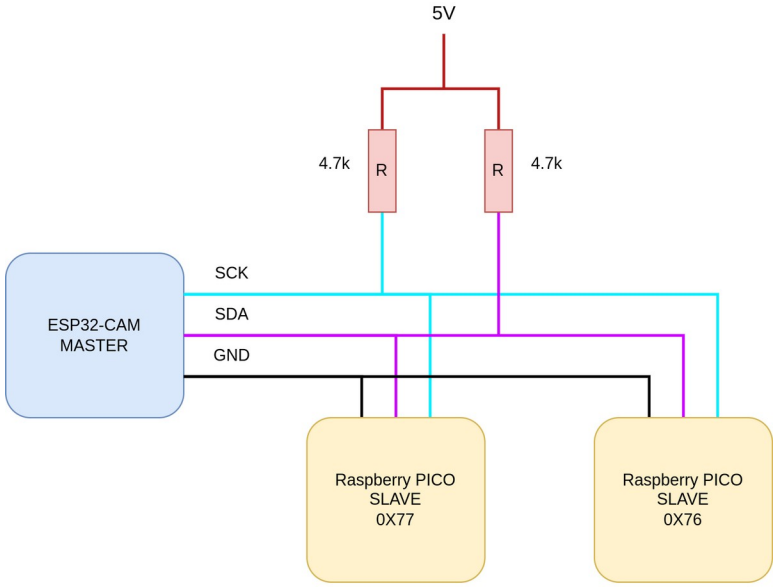
Register Name	Address	Content<bit position>	Description
press_msb	0x1F	press_msb<7:0>	Contains the MSB part [19:12] of the raw pressure measurement output data.
press_lsb	0x20	press_lsb<7:0>	Contains the LSB part [11:4] of the raw pressure measurement output data
press_xlsb	0x21	press_xlsb<7:4>	Contains the XLSB part [3:0] of the raw pressure measurement output data. Contents depend on pressure resolution controlled by oversampling setting.

5.3.4.4 *Gas resistance data*

Register Name	Address	Content<bit position>	Description
gas_r_msb	0x2A	gas_r<7:0>	Contains the MSB part gas resistance [9:2] of the raw gas resistance.
gas_r_lsb	0x2B	gas_r<7:6>	Contains the LSB part gas resistance [1:0] of the raw gas resistance.

Ürün datasheet belgesinden elde ettiğimiz register değerleri yandaki gibidir. Yine aynı şekilde yüksek anlamlı verinin saklandığı adresler bize haberleşme esnasında yardımcı olacak. Basınç için 0x1f, hava direnci için 0x2a değerlerini kullanıcaz.

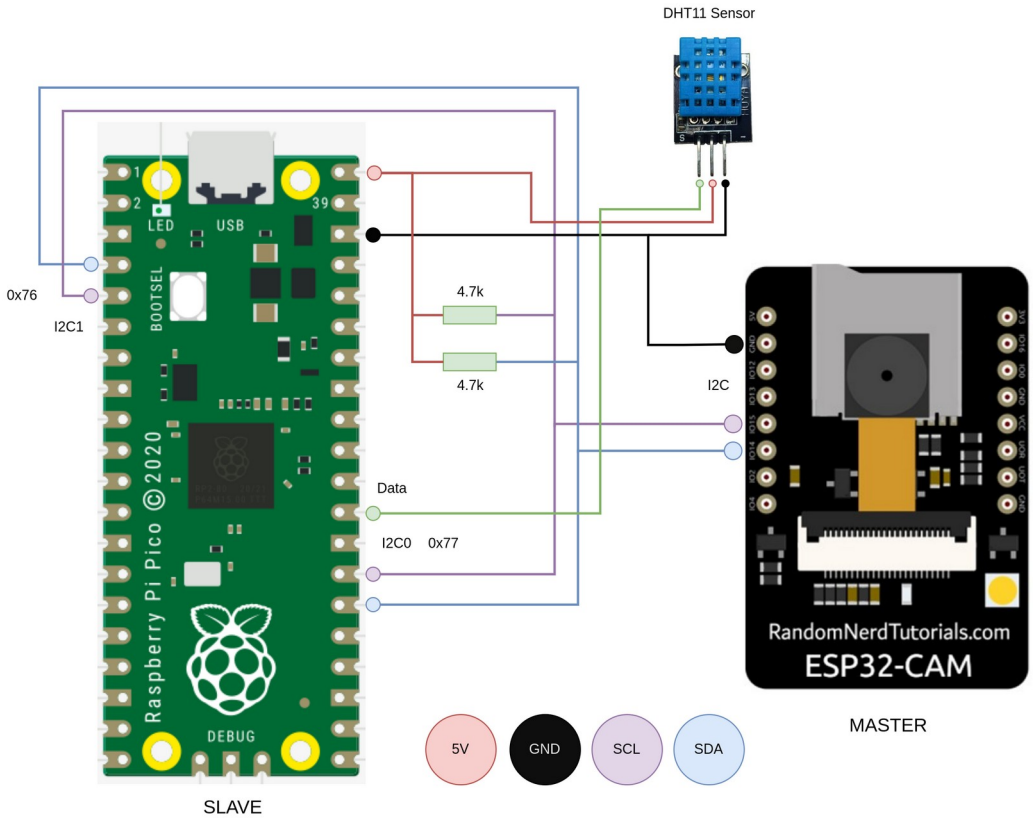
Bu uygulamayı gerçekleştirirken pull up direnci kullanacağım. I2C haberleşmesinde birimler open drain veya open collector özelliği göstermektedir. Bu birimler tarafından, SDA ve SCL üzerindeki sinyaller lojik 0 seviyesine çekilebilir, lojik 1 yapılamaz. Bu sebeple SDA ve SCL hatları birer pull-up dirençiyle lojik 1 seviyesine çekilmelidir. Pull-up direnci olarak genellikle 1K ile 10K aralığında dirençler kullanılmaktadır. 5V gerilim değeri için 4.7k ohm direnç, 3.3V gerilim içinse 2.4k ohm direç kullanılmaktadır. Biz devrede piconun 5v çıkışı ile 4.7kohm luk dirençler kullandık.



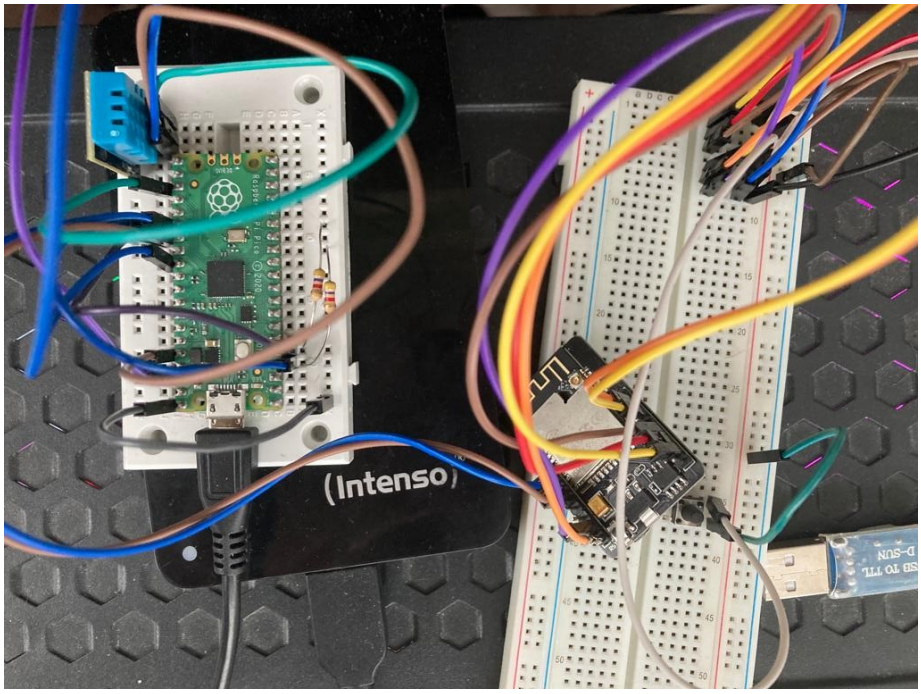
I2C Connection schematic

*Draw.io üzeriden çizmiş olduğum diagramalara diagrams klasöründen ulaşabilirsiniz. Ayrıca Pullup resistor konusu hakkında Texas Instrument firmasının yazmış olduğu detaylı bir kaynağı da yine source klasörüne ekledim.

Son aşamada gerçekleştirdiğimiz bağlantılar görseldeki gibidir:



Geliştirme sürecinde “Brownout detector was triggered” problemi ile çok fazla karşılaştım, bu problem genellikle wifi/bluetooth bağlantıları sırasında ortaya çıkmakta. Araştırmalarım sonucunda usb-ttl (rs232) bağlantısını değiştirerek ve bu bağlantı noktalarının durumunu kontrol ederek hatanın sıklığını azalttım ancak kesin çözüm için bir kapasitör lehimlemem gerekmekte.



Geliştirme kartlarındaki kodların açıklamaları:
Slave kart olan pico geliştirme kartını kullanım aşamasında Esp8266 kartından da yararlandım, pico kartının i2c haberleşmesini çift taraflı olarak kontrol ettim ve geliştirmeyi sağladım. Slave cihazı kodlarken arduino ide kullandım. Ek olarak yalnızca wire ve dht kütüphanelerini projeye ekledim.

```
#include <Arduino.h>
#include <DHT.h>
#include <DHT_U.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>
```

```
#define DHTPIN 22
#define DHTTYPE    DHT11
DHT_Unified dht(DHTPIN, DHTTYPE);
uint8_t SLAVE_ADDR2 = 0x76;
uint8_t SLAVE_ADDR1 = 0x77;
```

```
void setup(){
  Serial.begin(115200);
  Wire.setSDA(20);
  Wire.setSCL(21);
  Wire.begin(SLAVE_ADDR1);
  Wire.onReceive(Receive_first);
  Wire.onRequest(Request_first);

  Wire1.setSDA(2);
  Wire1.setSCL(3);
  Wire1.begin(SLAVE_ADDR2);
  Wire1.onReceive(Receive_second);
  Wire1.onRequest(Request_second);

  dht.begin();
}
```

Setup aşamasında öncelikle seri portun hızını ayarladım daha sonra 2 adet i2c modülü kullanmak için ilgili adresleri ve bulundukları pinleri modüllere tanımladım. Böylece hangi adres hangi pine ait belirlemiş oldum. Son aşamada ise tanımladığım hdt cihazını başlattım böylece sensör ile haberleşme sağlandı.

*Bu aşamaları detaylı anlattığım ve PlatformIO idesinden bahsettiğim videoya şu link üzerinden ulaşabilirsiniz:
https://www.youtube.com/watch?v=jw_fCzNrvNs

```
void loop(){
  // Get temperature event and print its value.
  sensors_event_t event;
  dht.temperature().getEvent(&event);
  temp = event.temperature;
  bitwise((int)temp, 0);

  // Get humidity event and print its value.
  dht.humidity().getEvent(&event);
  hum = event.relative_humidity;
  bitwise((int)hum, 1);
  Serial.println(hum);
  Serial.println(temp);

  bitwise((int)pres,2);
  bitwise((int)resis,3);
}
```

Loop aşamasında sensörden sıcaklık ve nem verisini alarak bunların bitwise operatörü ile 2 ayrı byte verisine dönüşmesini sağladık.

```
volatile float temp, hum, resis = 28, pres = 83;
```

Bunun dışında basınç ve hava direnci verileri de constant data olarak tanımlandı ve yine bitwise işlemi ile bölündü.

Değişkenleri özellikle volatile olarak tanımladım daha sonra değerleri değişebileceğinden dolayı değişkenin bellekte ekstra yer kaplamasını ve riskli durumlara neden olmasını istemedim.

Bitwise işlemi için oluşturduğum fonksiyon yan taraftaki gibidir, genel amacı i2c haberleşmesini kolaylaştırmak olan bu fonksiyonda 8bit veriler halinde gerçekleşen i2c haberleşmesi için anlamlı verilerimizi msb ve lsb olarak bölmekteyiz.

```
void bitwise(int value, int type){
  switch (type){
    case 0:
      //temp shifting
      temp_lsb = (unsigned)value & 0xff;
      temp_msb = (unsigned)value >> 8;
    break;
    case 1:
      //hum shifting
      hum_lsb = (unsigned)value & 0xff;
      hum_msb = (unsigned)value >> 8;
    break;
    case 2:
      //pres shifting
      pres_lsb = (unsigned)value & 0xff;
      pres_msb = (unsigned)value >> 8;
    break;
    case 3:
      //resis shifting
      resis_lsb = (unsigned)value & 0xff;
      resis_msb = (unsigned)value >> 8;
    break;
    default:
    break;
  }
}
```

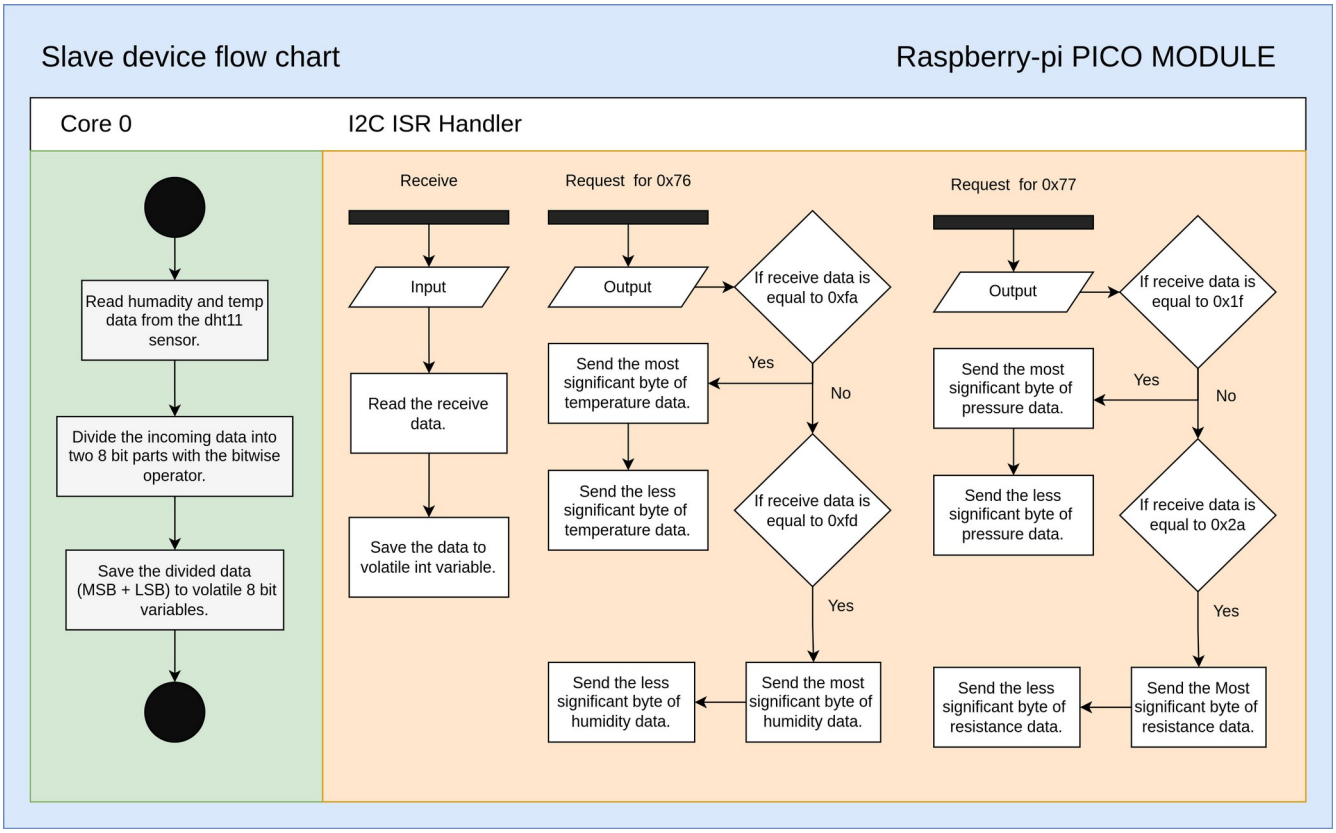
I2C haberleşmede interrupt amacıyla kullanılan request ve receive fonksiyonlarına göz atacak olursak; receive fonksiyonumuzun veri almayı, request fonksiyonumuzun ise veri göndermeyi sağladığını görmekteyiz.

```
void Receive_first(int a){//0x77
  receive_data1 = Wire.read();
}
```

0x77 adresine gelen veriler bu değişkene kopyalanmakta ve gönderme işleminde kontrol olarak kullanılmakta.

```
void Request_first(){
  if(receive_data1 == 0x1f){//press
      Wire.write(byte(pres_msb));
      Wire.write(byte(pres_lsb));
  }else if(receive_data1 == 0x2a){//resis
      Wire.write(byte(resis_msb));
      Wire.write(byte(resis_lsb));
  }
}
```

Soldaki fonksiyonda görüleceği üzere önce msb daha sonra da lsb değerleri master cihaza gönderilmekte.
Aynı işlem farklı fonksiyonlar ile 0x76 adresi için de yapılmakta.



*Pico için yazdığım arduino koduna code klasöründen ulaşabilirsiniz.

Esp32-cam geliştirme kartı, picodan verileri i2c yoluyla alarak bluetooth üzerinden akıllı telefona iletecektir. Esp32 çift çekirdekli bir işlemciye sahip olduğundan dolayı doğrudan 2 task ile bu 2 çekirdeği kontrol edeceğiz bu noktada freertos kütüphanelerini kullancağımızı da belirtmem gerek. Çift çekirdek avantajı sayesinde bağımsız çalışan görevler ile herhangi bir gecikme kullanmadan projeyi gerçekleşitirebiliriz. Daha önce belirttiğim üzere pek çok avantajı sebebiyle PlatformIO kullancağım.

```
#include <Arduino.h>
#include <stdio.h>
#include <stdint.h>
#include "FreeRTOSConfig.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "Wire.h"
#include "sdkconfig.h"
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
#include "soc/rtc_wdt.h"
#include "freertos/event_groups.h"
#include "esp_system.h"
#include "esp_log.h"
#include "nvs_flash.h"
#include "esp_bt.h"
#include "esp_gap_ble_api.h"
#include "esp_gatts_api.h"
#include "esp_bt_defs.h"
#include "esp_bt_main.h"
#include "esp_gatt_common_api.h"
#include <CircularBuffer.h>
```

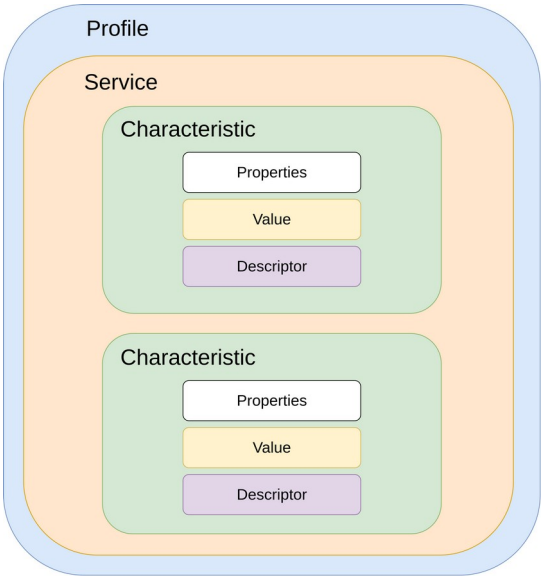
Kullandığım bütün kütüphaneler yan taraftaki görselde belirtilmiştir. Harici olarak eklediğim bir adet kütüphane bulunmakta, projede ring buffer yapısı kullandığımdan dolayı bu kütüphaneyi ekledim. Kalan diğer kütüphaneler sıkça karşılaştığımız freertos, ble ve arduino kütüphaneleridir.

```
void setup() {
  Serial.begin(115200);
  Wire1.setPins(14,15);
  Wire1.begin();
```

Setup aşamasında öncelikle seri haberleşeme hızı ayarladım ve daha sonra i2c pinlerini belirterek master cihazda i2c haberleşmesini başlattım.

Daha sonraki aşamada ble için gerekli konfigürasuyonları ayarladım ancak ondan önce ble yapısına kısaca göz atalım. Gatt (generic attributes) mantığında çalışan ble server kullanacağız. Bu yapı önceden tanımlanmış öznitelikler ile veri alış verişi sağlanmaktadır. Bu nitelikler, cihaz adı, hizmet UUID'leri (Evrensel Olarak Benzersiz Tanımlayıcılar), karakteristik UUID'ler ve bir BLE cihazının veya hizmetinin işlevselliğini belirlemeye ve açıklamaya yardımcı olan diğer bilgileri içerir.

Örneğin, bizim projemizde BLE özellikli bir sıcaklık monitörünün UUID'si "2A1F" (Sıcaklık için standart UUID'dir) değerine sahiptir. Bu UUID'ler, BLE genel özniteliklerinin bir parçasıdır ve evrenseldir bu sayde farklı BLE cihazları alınan bu UUID değerlerine göre veriyi anlamlandırabilmektedir.



Yan taraftaki şemada görüldüğü üzere 1 profil 1 servis ve birden fazla karakteristik kullancağız. Gönderdiğimiz her bir veri için ayrı bir karakteristik tanımlayacağız ve içerisine ona ait verileri göndereceğiz. Örneğin sıcaklık ve nem ayrı birer karakteristik olacak ve bunların değerleri value içersinde belirtilecek. Bu noktada şunu eklemem gerekirse geliştirme aşamasında verileri client cihazdan okuyabilmek için verileri UTF-8 tipine parse ettim. Bu seçenek nRF Connector uygulamasından kolayca gerçekleştirilebilmekte.

Bütün sensör verilerinin karakteristikleri aşağıdaki gibidir.

```
// Temperature
  BLECharacteristic bmeTemperatureCelsiusCharacteristics("cba1d466-344c-4be3-ab3f-189f80dd7518", BLECharacteristic::PROPERTY_NOTIFY);
  BLEDescriptor bmeTemperatureCelsiusDescriptor(BLEUUID((uint16_t)0x2A1F));

// Humidity
  BLECharacteristic bmeHumidityCharacteristics("ca73b3ba-39f6-4ab3-91ae-186dc9577d99", BLECharacteristic::PROPERTY_NOTIFY);
  BLEDescriptor bmeHumidityDescriptor(BLEUUID((uint16_t)0x2A6F));

// Pressure
  BLECharacteristic bmePressureCharacteristics("0167d753-8af0-46f0-963d-3da7fc2c0ad7", BLECharacteristic::PROPERTY_NOTIFY);
  BLEDescriptor bmePressureDescriptor(BLEUUID((uint16_t)0x2724));

// Resistance
  BLECharacteristic bmeResistanceCharacteristics("05c3676f-d165-4b7b-857b-b0b2c8534ccf", BLECharacteristic::PROPERTY_NOTIFY);
  BLEDescriptor bmeResistanceDescriptor(BLEUUID((uint16_t)0x2902));
```

Bu aşamada https://www.bluetooth.com/specifications/assigned-numbers/ sitesinden elde ettiğim BLUETOOTH BLEUUID değerlerini ilgili sensör verilerini isimlendirdim ancak gönderim aşamasında id lerin 2902 olarak sabit olduğunu fark ettim bu durum şimdilik bu şekilde hatalı.

Kodumuzdaki setup aşamasına tekrar geri dönecek olursak, Bu noktada server adı ve servis UUID si vererek profil ve servisi tanımladık. Bu isim ve UUID değişkenleri kodumuzun ilk satırlarında define edilmiştir.

```
BLEDevice::init(bleServerName);

BLEServer *pServer = BLEDevice::createServer();
pServer->setCallbacks(new MyServerCallbacks());

BLEService *bmeService = pServer->createService(SERVICE_UUID);
```

```
// Temperature
  bmeService->addCharacteristic(&bmeTemperatureCelsiusCharacteristics);
  bmeTemperatureCelsiusDescriptor.setValue("BME temperature");
  bmeTemperatureCelsiusCharacteristics.addDescriptor(&bmeTemperatureCelsiusDescriptor);
// Humidity
  bmeService->addCharacteristic(&bmeHumidityCharacteristics);
  bmeHumidityDescriptor.setValue("BME humidity");
  bmeHumidityCharacteristics.addDescriptor(&bmeHumidityDescriptor);

// Pressure
  bmeService->addCharacteristic(&bmePressureCharacteristics);
  bmePressureDescriptor.setValue("BME pressure");
  bmePressureCharacteristics.addDescriptor(&bmePressureDescriptor);

// Reistance
  bmeService->addCharacteristic(&bmeResistanceCharacteristics);
  bmeResistanceDescriptor.setValue("BME resistance");
  bmeResistanceCharacteristics.addDescriptor(&bmeResistanceDescriptor);

// Start the service
bmeService->start();
```

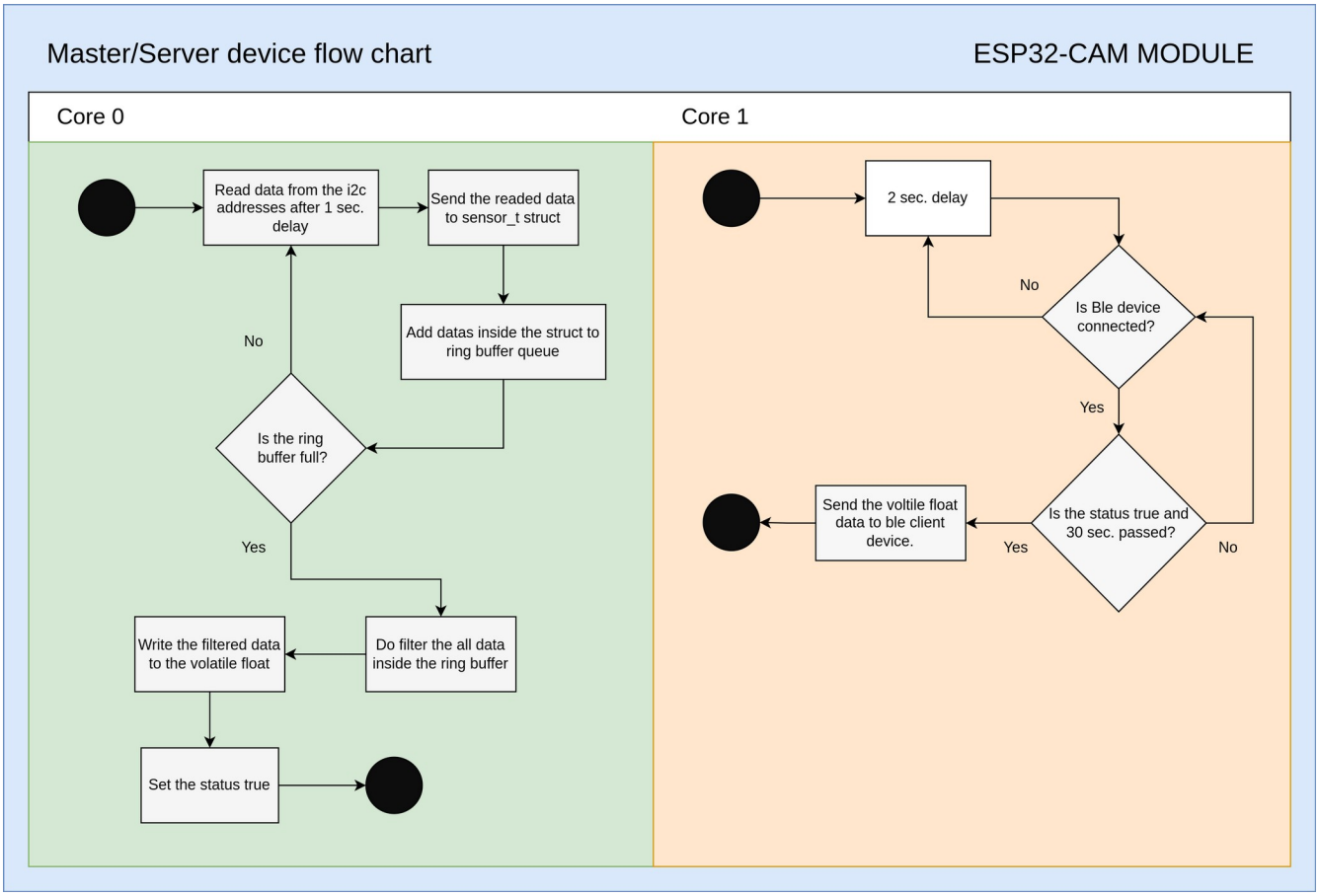
Sonraki aşamda 4 farklı karakteristiği oluşturduğumuz servis içerisine tanımlıyoruz ve servisimizi başlatıyoruz. Son olarak ble için gerekli olan akışı tanımlıyoruz ve profilden client cihazlara akışı başlatıyoruz, bu noktada bluetooth için gerekli olan tüm implementasyonu tamamladık ve diğer cihazlar için görünürlüğü sağladık.

```
// Start advertising
BLEAdvertising *pAdvertising = BLEDevice::getAdvertising();
pAdvertising->addServiceUUID(SERVICE_UUID);
pServer->getAdvertising()->start();
Serial.println("Waiting a client connection to notify...");
```

Setup aşamasındaki son configürasyonumuz freertos için olacak ve yazının başında da belirttiğimiz üzere esp32 geliştirme kartının sahip olduğu 2 çekirdeği de kullanacağız. Bu çekirdeklere eşit öncelik verdim.

```
xTaskCreatePinnedToCore(
  TaskFirst
  ,  "TaskFirst"   // Name
  ,  4096  // This stack size can be checked & adjusted by reading the Stack Highwater
  ,  NULL
  ,  2  // Priority, with 2 (configMAX_PRIORITIES - 1) being the highest, and 0 being the lowest.
  ,  NULL
  ,  RUNNING_CORE_0);

xTaskCreatePinnedToCore(
  TaskSecond
  ,  "TaskSecond"
  ,  4096  // Stack size
  ,  NULL
  ,  2  // Priority
  ,  NULL
  ,  RUNNING_CORE_1);
```

İlk task kısmında; saniye bir veri okuma, bu veriyi circular buffer içerisine yazma ve buffer dolduktan sonra içerisindeki tüm verileri filtreleme görevlerini gerçekleştireceğiz. İkinci task kısmında ise; eğer ble bağlantısı sağlandıysa veriler 30 saniyelik periyotta gönderilecek. Bu görevleri aşağıdaki akış diyagramında daha detaylı inceleyebilirsiniz.



İlk task koduna detaylı göz atacak olursak her 1 saniyede sensörden veri okuma görevini 1000ms gecikme ile sağladık daha sonra bu değerler circular buffer içerisine gönderilmekte. Her bir sensör verisi için ayrı ring buffer tanımladım böylece 30 index uzunluğunda 4 adet ring buffer ile verileri saklayabilmekteyiz. Bu noktada oluşturduğum struct yapısını incelemek gerekirse. Sensör bilgisi için 3 adet değişken tanımladım. İlki id diğer ikisi ise sensörden okunan değerler için.

```
typedef struct {
  uint32_t sensor_id;
  float first_value;
  float second_value;
} sensor_t;

// bme680 0x77
// bme280 0x76
sensor_t bme680,bme280;
```

```
void TaskFirst(void *pvParameters)  // This is first task for read raw sensor data and filtering it.
{
  (void) pvParameters;
  for (;;)
  {
    delay(1000);
    // Read data 1hz

    i2c_sensor_read(0x77,&bme680);
    buffer1.push(bme680.first_value);
    buffer2.push(bme680.second_value);

    i2c_sensor_read(0x76,&bme280);
    buffer3.push(bme280.first_value);
    buffer4.push(bme280.second_value);

   if (buffer4.isFull()) {
    // Do filtering with moving median filter into the ring buffer data.
    filter_sensor_value();
    status = true;
   }
  }
}
```

Örneğin bme280 sensörü için id 0x76, first value sıcaklık, second value nem değerlerini taşımakta. Bu değerleri i2c den okuyarak struct içerisine yazabilmek içinse oluşturduğum i2c_sensor_read fonksiyonunu açıklamam gerekmekte. Bu fonksiyon 2 farklı argüman almaktadır. Birincisi i2c haberleşmesi için gerekli olan i2c adresi ikincisi ise verilerin kopyalanacağı sensör struct adresi. Daha sonra fonksiyon içerisinde i2c haberleşmesi ile pico geliştirme kartından değerler sırayla okunmakta ve bitwise operatörü ile birleştirilmekte.

Sonraki aşamada ise 4. buffer dolduğu durumda bufferlar içerisindeki veriler ayrı ayrı işlenmek için filter_sensor_value fonksiyonu içerisine alınır. Burada filtrelenen son değerler sıcaklık, nem, basınç, direnç değişkenlerine kopyalanmakta ve eğer ki bir ble bağlantısı varsa ikinci task içerinde client cihaza gönderilmekte.

```
void i2c_sensor_read(uint8_t device_address, sensor_t *sensor_type){
  int read_pres,read_resis,read_temp,read_hum;
  if(device_address == 0x77){
    sensor_type->sensor_id = device_address;
    Wire1.beginTransmission(device_address);
    Wire1.write(byte(0x1f));              // sends data code
    Wire1.endTransmission();    // stop transmitting
    Wire1.requestFrom(device_address, (uint8_t) 2);
    if(2 <= Wire1.available()){
      read_pres = Wire1.read();
      read_pres = read_pres << 8;
      read_pres |= Wire1.read();
    }
```

filter_sensor_value fonksiyonu aşağıdaki görselde gösterilmektedir. Tüm bufferlarda bulunun tüm değerleri tek tek çekilerek ortalaması alınmakta ve ilgili değişkene bu filtrelenmiş nihai değerler yazılmakta.

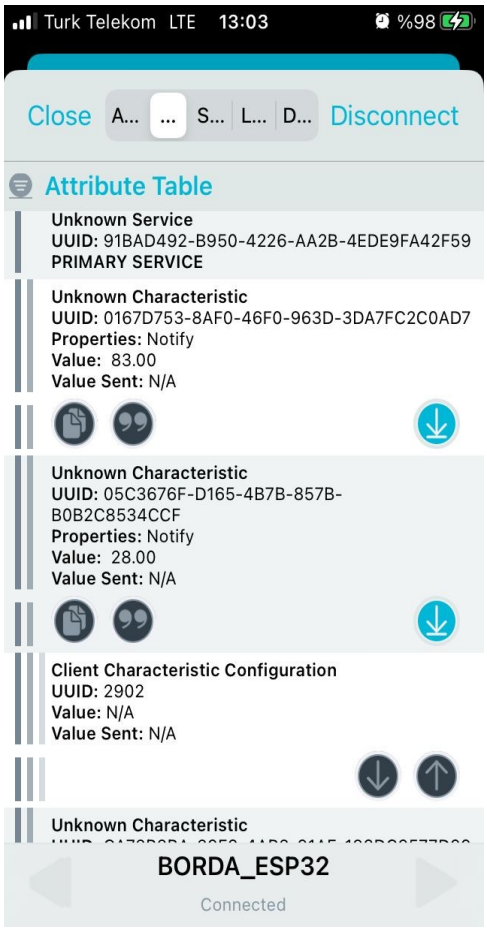
```
int i = 0, j = 0, k = 0, l = 0;
while(!buffer1.isEmpty()){
  pres += buffer1.pop();
  i++;
}
pres = pres/i;

while(!buffer2.isEmpty()){
  resis += buffer2.pop();
  j++;
}
resis = resis/j;

while(!buffer3.isEmpty()){
  temp += buffer3.pop();
  k++;
}
temp = temp/k;

while(!buffer4.isEmpty()){
  hum += buffer4.pop();
  l++;
}
hum = hum/l;
```

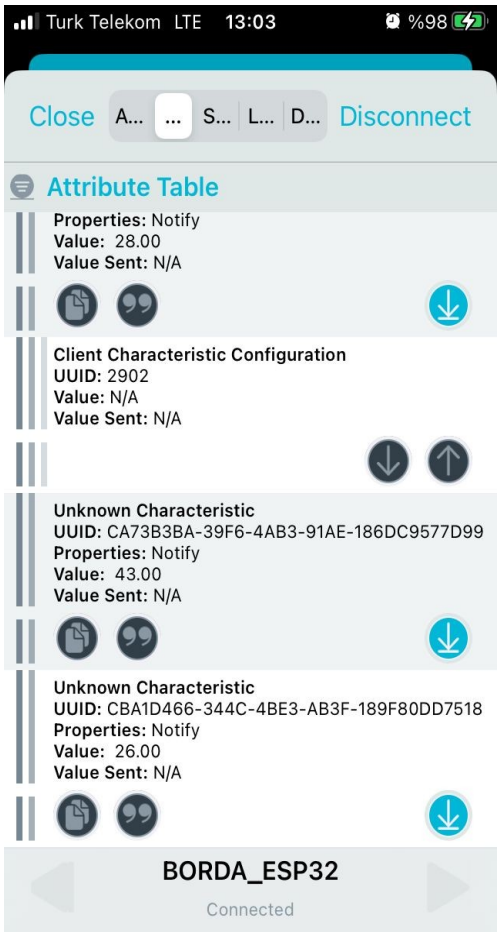
İkinci task aşamasında ise başından beri bahsettiğimiz verilerin gönderme işlemini gerçekleştirmekteyiz. Bu noktada bir bağlantı sağlannırsa 30 saniyelik periyotlar ile bütün karakteristik verileri gönderilmekte.

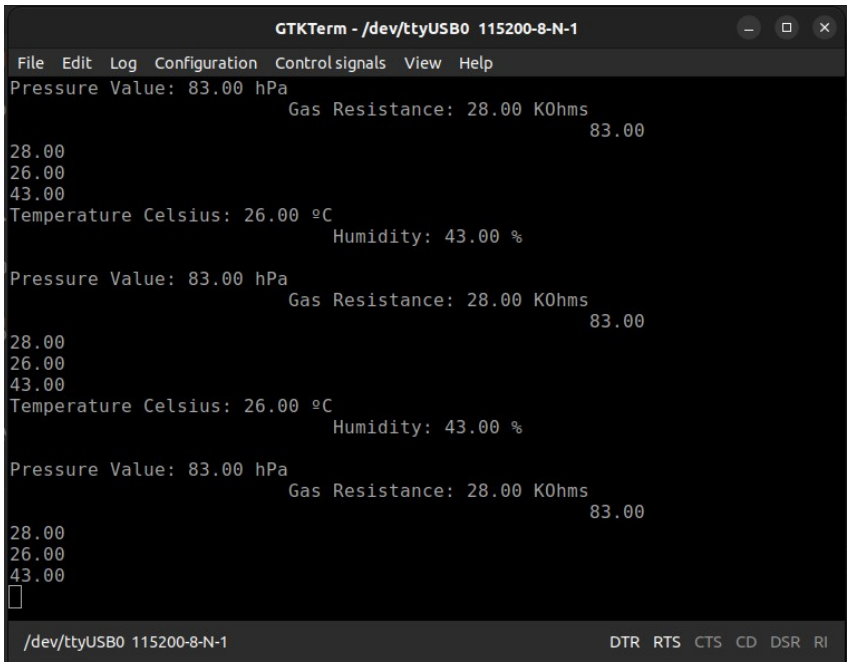
```
void TaskSecond(void *pvParameters)   // This is second task for send fi
{
  (void) pvParameters;
  for (;;)
  {
    Serial.println("Searching ble connection.\n");
    delay(2000);
    while (deviceConnected) {
      if ((millis() - lastTime) > timerDelay && status) {
        static char temperatureCTemp[6];
        dtostrf(temp, 6, 2, temperatureCTemp);
        bmeTemperatureCelsiusCharacteristics.setValue(temperatureCTemp);
        bmeTemperatureCelsiusCharacteristics.notify();
        Serial.print("Temperature Celsius: ");
        Serial.print(temp);
        Serial.print(" ºC\n");
```

*Daha detaylı incelenebilmesi adına master kodlarını da yine code klasörü içerisine koydum.

Nihai sonuçların Port üzerinden ve bluetooth verisi üzerinden karşılaştırılması:







Yararlandığım kaynaklar:

- https://randomnerdtutorials.com/esp32-bme680-sensor-arduino/
- https://randomnerdtutorials.com/esp32-ble-server-client/
- https://randomnerdtutorials.com/esp32-i2c-communication-arduino-ide/
- https://www.youtube.com/watch?v=0Yvd_k0hbVs
- https://randomnerdtutorials.com/esp32-i2c-communication-arduino-ide/
- https://docs.arduino.cc/learn/communication/wire