Doğukan Akça
19332652030

T.C.
BURSA TEKNİK ÜNİVERSİTESİ
MEKATRONİK MÜHENDİSLİĞİ BÖLÜMÜ
MEK0302_Mekatronik Mühendisliğinde Tasarım Raporu

İçindekiler

1. KARA KUTU

1.1 Tanıtım

Kara kutu; her hava taşıtında bulunması zorunlu bir olan kayıt, uluslararası kullanılan şekli ile bir uçuş kaydedicisi (flight recorder) ünitesidir. Uçuş ile ilgili bütün bilgiler kara kutunun içine belirli bir algoritma ile kaydedilir. Böylece ihtiyaç duyulan anlarda uçuş verilerini yetkililer için ulaşılabilir kılar.

1.2 Amaç

Bu tasarımda ise otomotiv sektöründe kullanılması planlanan ve hava taşıtlarındakine benzer bir algoritmaya sahip olan bir Kara Kutu tasarımı yapılmaktadır. Tasarımda kullanılan belirli cihazlar sayesinde Kara Kutu gerçekleştirilmektedir. Tasarımın amacı, araçların kaza öncesi ve kaza anındaki sıcaklık, basınç, saat, tarih gibi verilerinin bir SD karta yazılmasıdır. Fakat istenildiğinde tasarıma farklı sensörler entegre edilip verilerin çokluğu arttırılabilir.

1.3 Araştırma

Tasarıma başlamada önce çeşitli araştırmalar yapılmıştır. İlk olarak hangi mikrodenetleyicinin kullanılması gerektiği araştırılmıştır daha sonra hangi sensörlerin kullanılması ve en sonunda da verilerin nereye yazılması gerektiği araştırılmıştır. Kara Kuru tasarımı geliştirilmeye açık bir proje olduğundan seçilmesi istenilen mikrodenetleyicinin yüksek hızlı, birden fazla haberleşme protokollerine sahip, güç verimliliği yüksek, maliyeti düşük vb. sebeplerden dolayı piyasada da oldukça fazla bulunan STM32F407VGT6 DISCOVERY mikrodenetleyicisi seçilmiştir.

2. TANITIM VE MALZEMELER

2.1 STM32F407 DISCOVERY

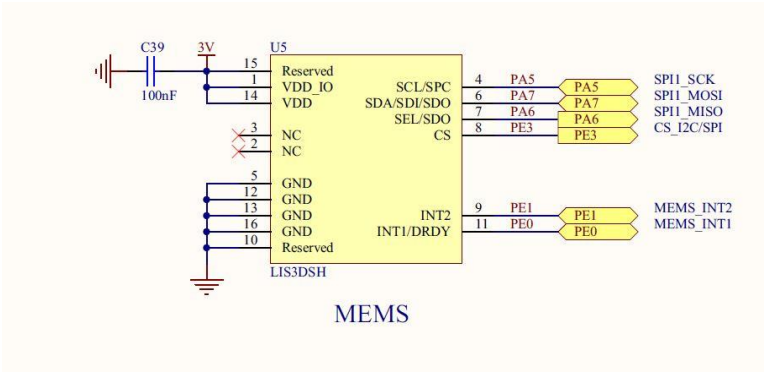
STM32F407 DISCOVERY ARM Cortex-M4 32-bit bir işlemciye ve RISC mimarisine sahip bir mikrodenetleyicidir. Bu mikrodenetleyici maksimum 168MHz CPU frekansında çalışabilmektedir ayrıca dahili 1 adet 32KHz RTC, 3 adet ADC, 14 adet Timer, 2 adet CAN, 3 adet I2C, 3 adet SPI, 2 adet UART, 4 adet USART vb. birimlerine sahiptir ve bu birimlerin çokluğundan dolayı bu tasarımın dahada çok geliştirilebilir olduğu görülmektedir. STM32F407 DISCOVERY mikrodenetleyicisi *Şekil 2.1*'de gösterilmiştir.

Şekil 2.1

Bu kartın seçilmesinin diğer bir sebebi ise dahili 3 eksen LIS3DSH lineer ivme sensörüne sahip olmasından kaynaklanmaktadır.

2.2 LIS3DSH

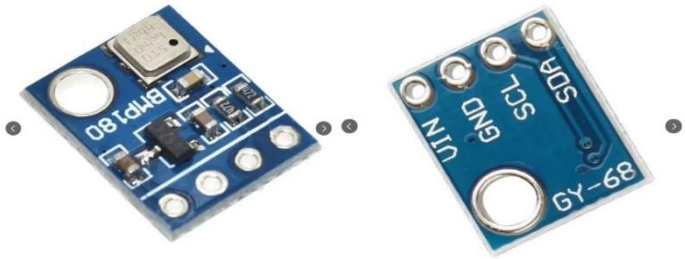
LIS3DSH sensörü ± 2g/± 8g dinamik tam tarama yapabilmektedir ve I2C/SPI dijital çıkış arayüzüne sahiptir. Fakat bu tasarımda en çok işimize yarayan özelliği ise kesme (interrupt) çıkışı üretebilmesidir. LIS sensörü birden fazla kesme üretebilmektedir bu kesmelerden işimize yarayacak olanlar Serbest Düşme (Free-Fall) ve (Wake-Up) kesmeleridir. FF kesmesi, 3 eksende de ani ivmelenme olduğu zaman kesme üretmektedir fakat WU kesmesinde X ve Y eksenlerindeki ani ivmelenmeden kaynaklı uyandırma kesmesi üretmektedir. LIS sensörünün ürettiği bu kesme sayesinde aracın kaza anını yakalayabilmektedir ve bu özelliği kullanarak kaza verilerini bir EEPROM'a veya bir SD karta yazdırılabilmektedir. Yazılım konfigürasyonunda bu sensör için SPI1 birimi kullanılmıştır. Dahili LIS sensörünün STM32F407 kartı üzerindeki bağlantıları *Şekil 2.2*'de verilmiştir.



Şekil 2.2

2.3 BMP180

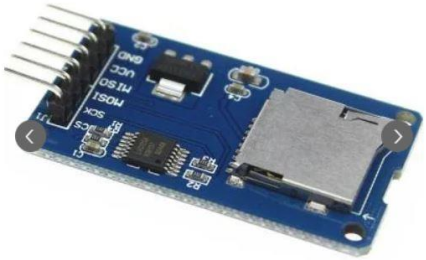
Sıcaklık ve basınç değerlerini gözlemlemek için BMP180 sıcaklık ve basınç sensörü seçilmiştir. BMP180 sensörü I2C haberleşme protokolüne sahiptir ve STM32F407 mikrodenetleyicisine harici olarak bağlanmıştır. Yazılım konfigürasyonunda bu sensör için I2C1 birimi kullanılmıştır. Sensör 3V ile beslenmektedir bu nedenle sensörün beslemesi mikrodenetleyicinin 3V çıkış pinine, SCL pini mikrodenetleyicinin PB6 pinine, SDA pini mikrodenetleyicinin PB7 pinine bağlanmıştır. BMP180 sensörü *Şekil 2.3*'te gösterilmiştir.



Şekil 2.3

2.4 SD KART

Verilerin depolanması için bir SD kart kullanılmıştır bu seçimin sebebi hali hazırda 8gb bir mikro SD karta sahip olmamdan dolayıdır. Mikrodenetleyiciden SD karta veri yazabilmek için bir mikro SD kart modülü kullanılmıştır. Bu SD kart modülü SPI haberleşme protokolünü kullanmaktadır ve 3.3-5V beslemeye sahiptir. SD kart modülünün beslemesi mikrodenetleyicinin 5V çıkış, MOSI pini mikrodenetleyicinin PB15, MISO pini mikrodenetleyicinin PB14, SCK pini mikrodenetleyicinin PB13 ve çip seçim (chip select) pini mikrodenetleyicinin PB12 pinine bağlanmıştır. SD kart modülü *Şekil 2.4*'te gösterilmiştir.



Şekil 2.4

2.5 RTC

STM32F407 kartında zaman, tarih, alarm vb. kurulabilen dahili 32768Hz osilatör ile çalışan Gerçek Zamanlı Saat (RTC) bulunmaktadır. Bu tasarımda SD karta yazılan verilerden biri tarih ve zamandır bu yüzden dahili RTC kullanılmıştır. Dahili RTC'nin dezavantajı kart güçten çekildiğinde zaman ve tarih bilgisinin durmasıdır bunun için her seferinde tarih ve zaman manuel olarak BCD formatta girilmelidir fakat bu tasarım geliştirilmeye açık olduğundan harici bir RTC kullanılarak otomatikleştirilebilir.

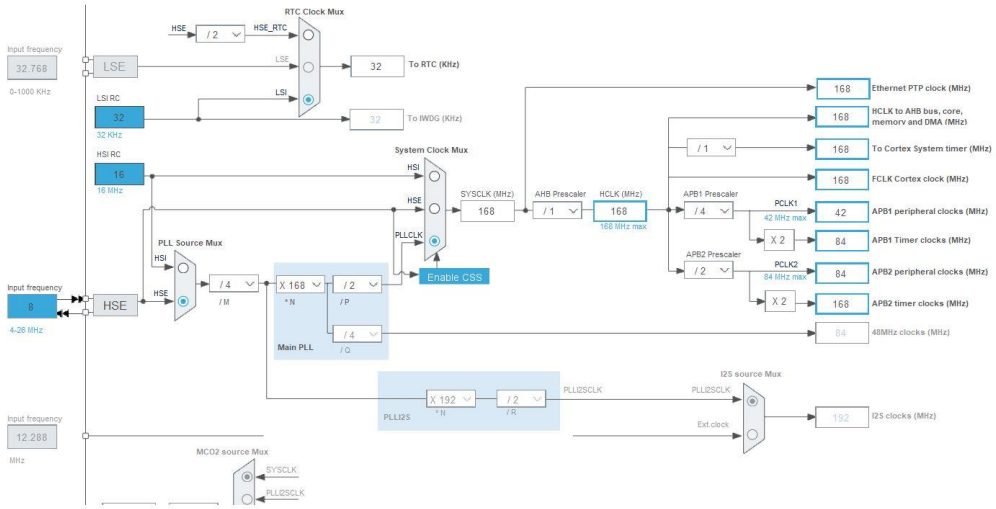
3. YAZILIM VE KONFİGÜRASYON

3.1 STM32CubeIDE

Tasarımın yazılımı STM32CubeIDE programı kullanılarak yazılmıştır. Yazılımın nasıl yazıldığı sırasıyla anlatılmıştır.

3.2 Saat Ayarı

İlk olarak CubeIDE'de bir proje oluşturulmuştur ardından clock konfigürasyonu yapılmıştır kartın maksimum çalışma frekansı olan 168MHz'e ayarlanmıştır bunu yapmak için Yüksek Hızlı Saat (HSE) Kristal/Seramik Rezonatör olarak seçilmiştir ardından giriş frekansı 8MHz, HSE ve PLLCLK seçilmiştir en sonunda da HCLK yerine 168MHz yazılarak CLK konfigürasyonu tamamlanmıştır. Ayrıca RTC için Dahili Düşük Hızlı (LSI) saat seçilmiştir. Saat ayarları *Şekil 3.2'*de gösterilmiştir.

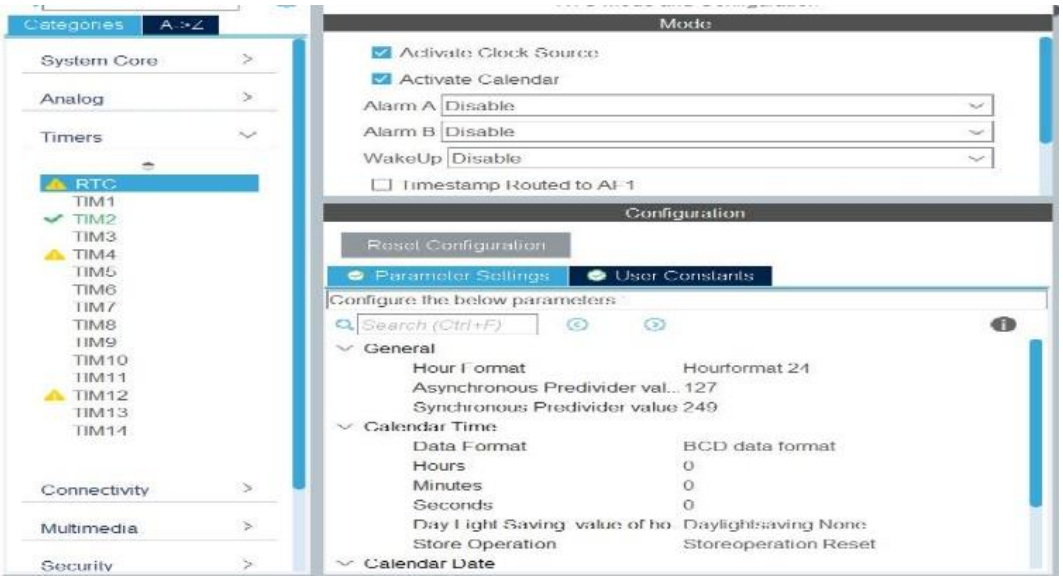


Şekil 3.2

3.3 RTC Ayarı

RTC konfigürasyonu için saat kaynağı ve takvim aktif edilmiştir ayrıca zaman ayarı için asenkron ön bölücü değerine 127+1 ve senkron ön bölücü değerine 255+1 girilmiştir bu değerler 1Hz clock ayarı için girilmiştir. RTC ayarları *Şekil 3.3'*te gösterilmiştir.

$$f_{CK\_SPRE} = \frac{f_{RTCCLK}}{(PREDIV\_S+1)*(PREDIV\_A+1)} = \frac{32768Hz}{(127+1)*(255+1)} = 1\text{Hz} = 1\text{saniye}$$



Şekil 3.3

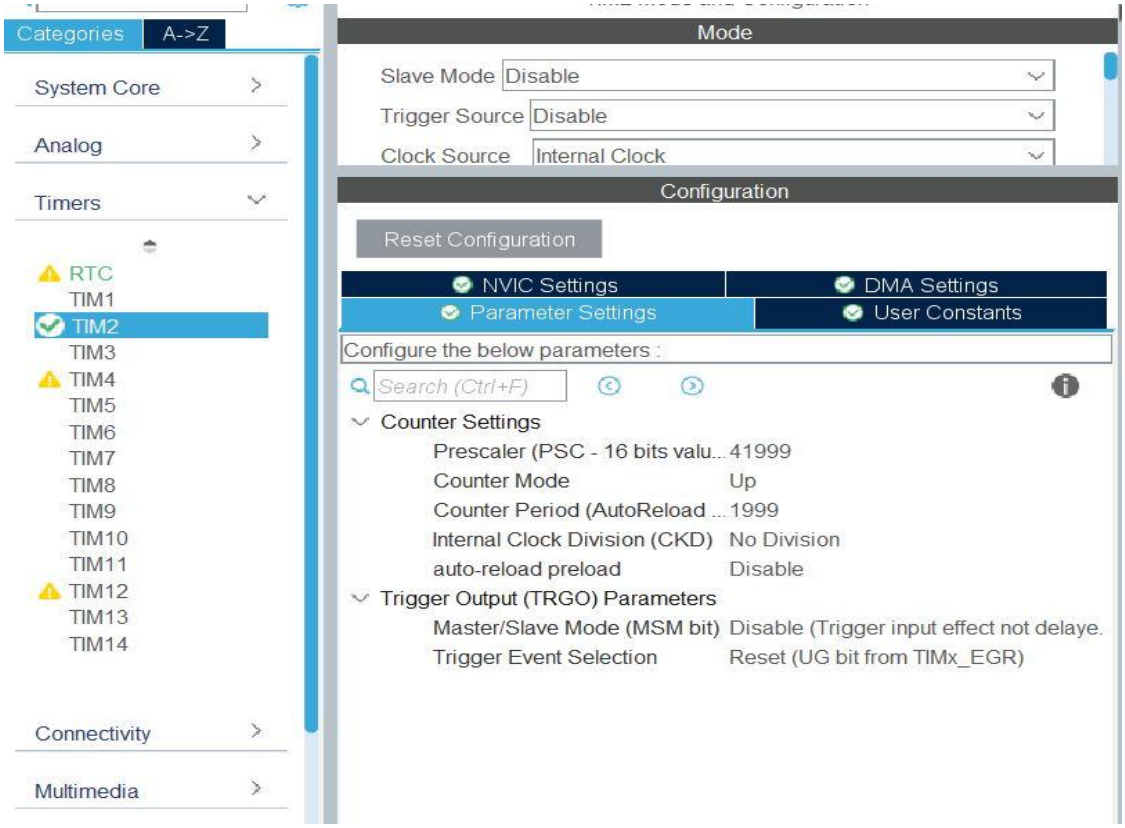
3.4 SD Kart Ayarı

SD kartta 2 adet dosya bulunmaktadır bu dosyalardan biri DATA isimli dosyadır ve içinde kaza öncesi varsayılan veriler bulunmaktadır bu veriler 2 saniyede bir yenilenmektedir. Diğer bir dosya ise CRASH isimli kaza dosyasıdır ve içinde sadece kaza anındaki veriler tutulmaktadır. SD karta veri yazmak için FatFs dosya işlemleri (User Defined) aktif edilmiştir.

3.5 TIMER2 Ayarı

Kaza verilerini SD kart'a yazdırmadan önce düzenli olarak varsayılan veriler TIMER2 kullanılarak 2 saniyede bir yazdırılmıştır TIMER2 konfigürasyonu için Dahili Saat, sayaç modu yukarı mod olarak seçilmiştir. TIMER2 kesmesi aktif edilerek sayaç periyoduna (counter period = 1999 + 1) ve ön ölçekleyici değerine (prescaler = 41999 + 1) yazılmıştır. TIMER2 APB1 clock hattına bağlı olduğu için 42MHz dir. TIMER2 ayarları *Şekil 3.5'*te gösterilmiştir.

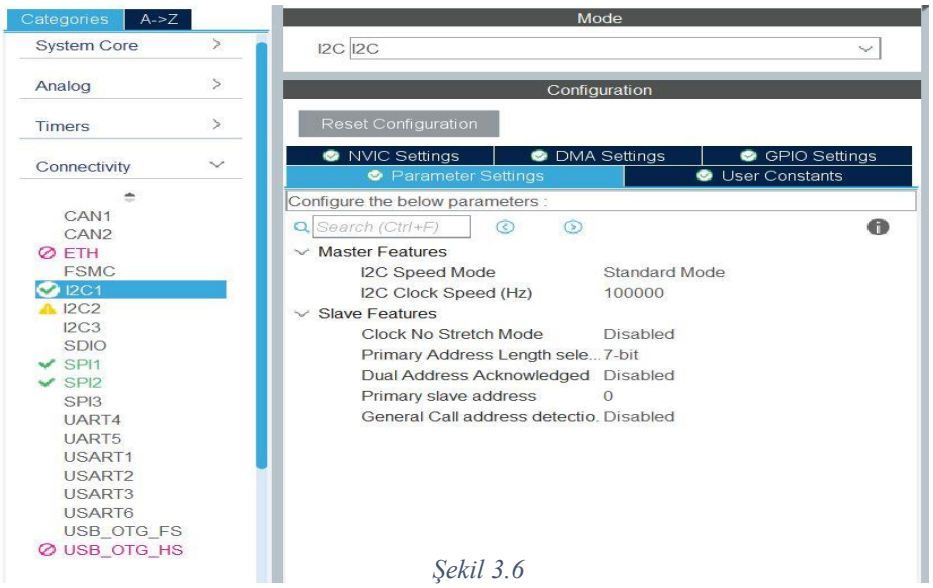
$$f_{TIM} = \frac{f_{APB1}}{(PERIOD+1)*(PRESCALER+1)} = \frac{42\ 000\ 000Hz}{(1999+1)*(41999+1)} = 0.5\text{Hz} = 2\text{saniye}$$



Şekil 3.5

3.6 BMP180 Ayarı

BMP180 sensörü için I2C1 konfigüre edilmiştir ve varsayılan değerler kullanılmıştır. I2C1 ayarları *Şekil 3.6'*da gösterilmiştir.

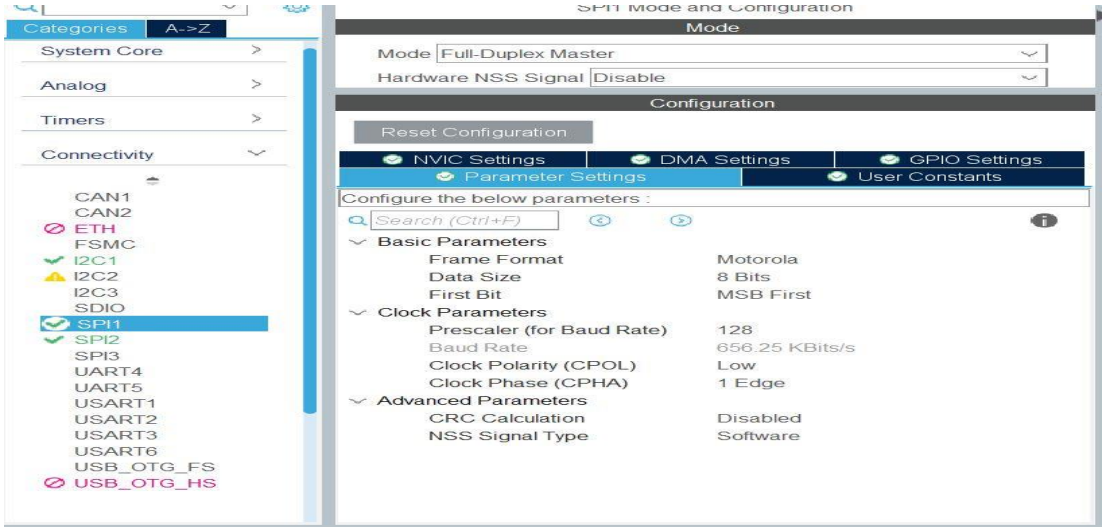


Şekil 3.6

3.7 LIS3DSH SPI1 Ayarı

LIS3DSH sensörü dahili bir sensör olduğu için SPI1 kullanılmalıdır. SPI1 konfigürasyonu için mod olarak Full-Duplex master mod seçilmiştir, Slave Select (Hardware NSS) aktif edilmemiştir bunun nedeni LIS sensörünün CS bacağı SPI1 ile bağlantılı değildir bu yüzden CS bacağı yazılımsal olarak PE3 pininden kontrol edilmiştir. Gönderilecek veriler 8bit olduğundan veri büyüklüğü 8 bit seçilmiştir ve LIS sensörünün çalışması için CPOL ve CPHA '0' seçilmiştir. SPI1 ayarları *Şekil 3.7'*de gösterilmiştir. LIS sensörünün kesme üretme bacağı INT1 kullanılmıştır bu bacak PE0 portuna bağlıdır bu yüzden Line0 NVIC'ten aktif edilmiştir.

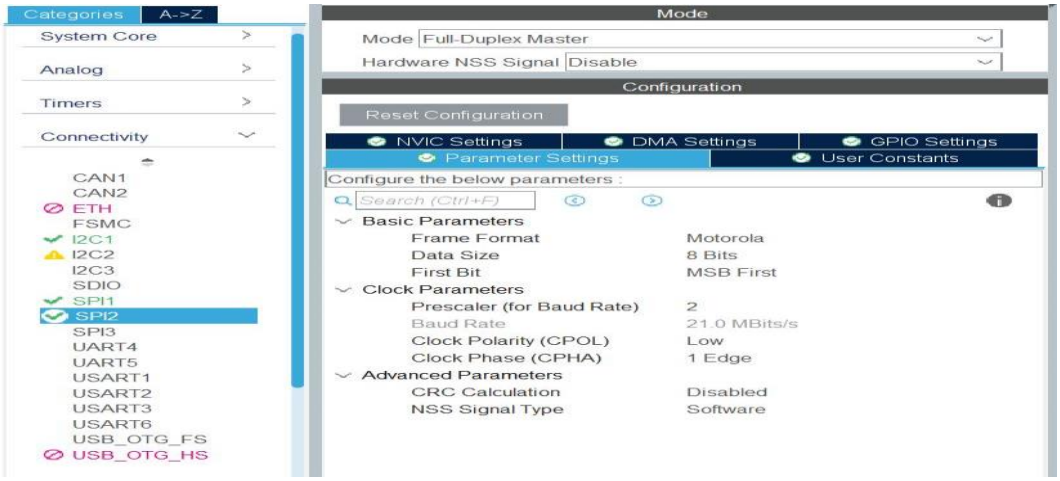
*Şekil 3.9'*da kullanılan pinler ve NVIC ayarları gösterilmiştir.



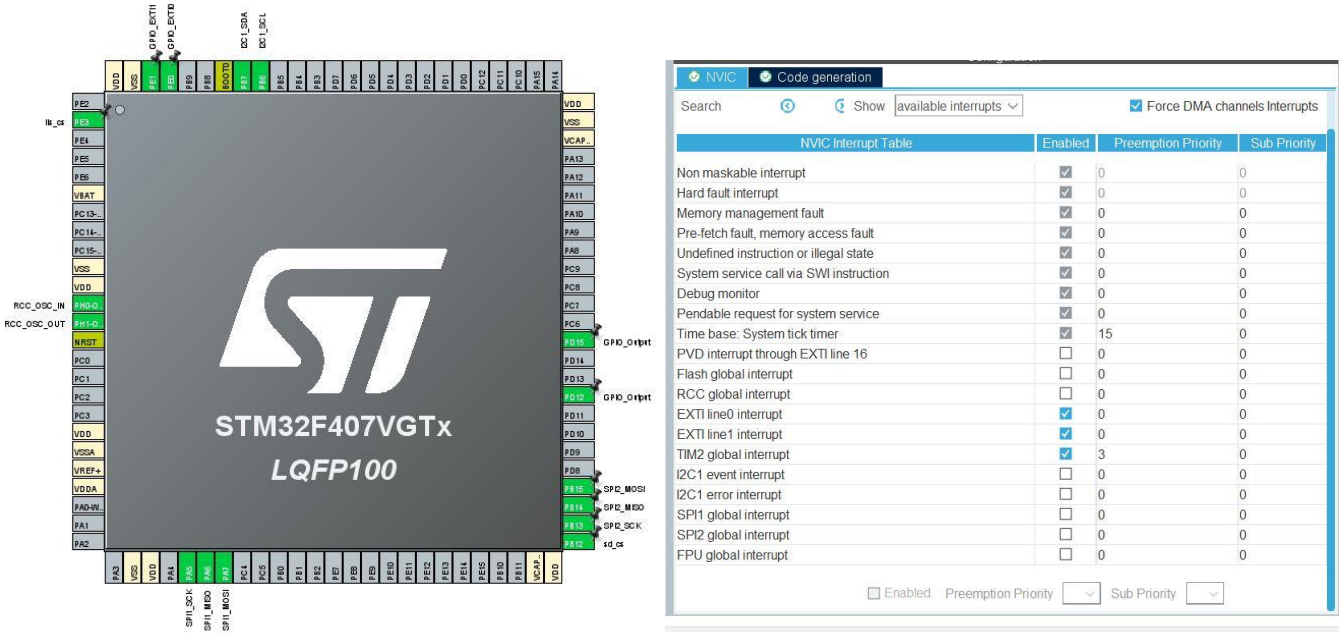
Şekil 3.7

3.8 SD Kart SPI2 Ayarı

SD kart modülü için SPI2 kullanılmıştır. SPI2 konfigürasyonu için mod olarak Full-Duplex master mod seçilmiştir, Slave Select (Hardware NSS) aktif edilmemiştir bunun nedeni SD kart modülü harici bağlandığı için CS bacağı yazılımsal olarak kontrol edilmiştir (PB12). Gönderilecek veriler 8bit olduğundan veri büyüklüğü 8 bit seçilmiştir. SPI2 ayarları *Şekil 3.8'*de gösterilmiştir.



Şekil 3.8



Şekil 3.9

Yukarda yapılan konfigürasyonları generate ettikten sonra ana kodda BMP180.h, fatfs.h, stdio.h ve string.h kütüphane tanımlamaları yapılmıştır. BMP180 sensörü için hazır bir kütüphane kullanılmıştır bu kütüphane src ve inc dosyalarının altına eklenmiştir.

4. KODLAMA

Kodlama kısmına geçildiğinde ilk olarak LIS3DSH sensörünün kaydedici tanımlamaları (register define) yapılmıştır daha sonra SPI haberleşmesi için WriteSpi ve ReadSpi adında iki fonksiyon tanımlanmıştır. WriteSpi adreslere veri yazması gerektiği için adres ve veri adında 8 bitlik iki değişken almaktadır, ReadSpi ise sadece okuma yapacağından 8 bitlik adres bilgisi almaktadır.

4.1 WriteSpi Fonksiyonu

WriteSpi fonksiyonunda ilk olarak LIS3DSH ile haberleşmeyi başlatmak için sensörün CS bacağı (PE3) düşük (low) yapılmıştır daha sonra HAL_SPI_Transmit fonksiyonu ile öncelik olarak yazılacak adres bilgisi ardından yazılacak veri bilgisi gönderilmiştir en sonunda da haberleşmeyi durdurmak için sensörün CS bacağı (PE3) yüksek (high) yapılmıştır. WriteSpi fonksiyonu Şekil 4.1.1'de verilmiştir.

```
644 }
645
646 void WriteSpi(uint8_t addres, uint8_t data)
647 {
648
649     HAL_GPIO_WritePin(lis_cs_GPIO_Port, lis_cs_Pin, GPIO_PIN_RESET);
650
651     HAL_SPI_Transmit(&hspi1, &addres, 1, 100);
652
653     HAL_SPI_Transmit(&hspi1, &data, 1, 100);
654
655     HAL_GPIO_WritePin(lis_cs_GPIO_Port, lis_cs_Pin, GPIO_PIN_SET);
656
657 }
658
```

Şekil 4.1.1

4.2 ReadSpi Fonksiyonu

ReadSpi fonksiyonunda okunacak verinin saklanması için bir adet işaretsiz 8 bitlik buffer adında bir değişken tanımlanmıştır. LIS3DSH sensörünün kullanım kılavuzunda yazan bilgiye göre okuma yapabilmek için okunacak adresin 8. Biti 1 yapılmalıdır bu bilgiden dolayı okunacak adres verisi 0x80 ile OR'lanmıştır (adres = adres | 0x80). Daha sonra LIS3DSH ile haberleşmeyi başlatmak için sensörün CS bacağı (PE3) düşük (low) yapılmıştır, HAL_SPI_Receive fonksiyonu ile okunacak adres bilgisi ardından okunacak veri bilgisi okunmuştur, haberleşmeyi durdurmak için sensörün CS bacağı (PE3) yüksek (high) yapılmıştır. Son olarak okunan veri buffer değişkeni ile geri döndürülmüştür (return buffer). ReadSpi fonksiyonu Şekil 4.2.1'de verilmiştir.

```
uint8_t ReadSpi(uint8_t addres)
{

    uint8_t buffer;

    if(HAL_SPI_GetState(&hspi1) == HAL_SPI_STATE_READY)
    {
        if(flag_control == 1)
        {
            addres = addres;
        }
        else if (flag_control == 0)
        {
            addres = addres | 0x80;     // addresin basina 1 yazarak okuma yapcagimizi soyluyoruz
        }

        HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_RESET);

        HAL_SPI_Transmit(&hspi1, &addres, 1, 100);

        HAL_SPI_Receive(&hspi1, &buffer, 1, 100);

        HAL_GPIO_WritePin(GPIOE, GPIO_PIN_3, GPIO_PIN_SET);
    }

    return buffer;

}
```

Şekil 4.2.1

4.3 Lis_init Fonksiyonu

LIS3DSH sensörünün konfigürasyonu için lis_init adında bir fonksiyon oluşturulmuştur. Bu konfigürasyonda sensörün Free-Fall kesme ayarları yapılmıştır.

Lis_init fonksiyonunda ilk olarak ReadSpi fonksiyonunu kullanarak WHO_AM_I registeri okunmuştur. Eğer LIS3DSH sensörü mikrodenetleyici ile sağlıklı bir şekilde haberleşme yapabiliyor ise WHO_AM_I registeri okunduğunda sensör tarafından geri 0x3F verisi gönderilmektedir bu yüzden ilk önce bu register okunarak doğru veri geldiğinde konfigürasyon ayarları yapılmıştır. Sensörün tüm registerlarına veri yazma işlemi WriteSpi fonksiyonu ile yapılmıştır.

CTRL_REG4 registırına 0x77 yazılarak veri aralığı 400Hz yapılmıştır ve X, Y, Z eksenleri aktif edilmiştir.

CTRL_REG3 registırına 0x78 yazılarak sensörün interrupt'ı ve INT1 ve INT2 bacağı aktif edilmiştir.

CTRL_REG5 registırına 0x08 yazılarak tam tarama (Full-scale selection) +/- 4G kuvvet ve SPI haberleşmesinin full duplex olacağı seçilmiştir.

CTRL_REG1 registırına 0x01 yazılarak Durum Makinesi 1 (SM1) aktif edilmiştir.

CTRL_REG2 registırına 0x09 yazılarak Durum Makinesi 2 (SM2) ve INT2 bacağı aktif edilmiştir.

Burdan aşağısı Free-Fall içindir.

TIM1_1L registırına lis_convert_time(miliSeconds) (bu fonksiyon aşağıda açıklanmıştır) yazılarak geri sayma işlemi için istenilen Timer1 değeri yazılmıştır.

THRS2_1 registırına lis_convert_threshold(miliG) (bu fonksiyon aşağıda açıklanmıştır) yazılarak interrupt oluşması için kaç G kuvveti gerekli olduğu yazılmıştır.

MSK1_B registırına 0xA8 yazılarak SM1 (state machine) için eksenlerin pozitif kısımları aktif edilmiştir.

MSK1_A registırına 0xA8 yazılarak SM1 (varsayılan state machine) için eksenlerin pozitif kısımları aktif edilmiştir.

SETT1 registırına 0xA3 yazılarak SM1 için tepe algılama, işaretli eşik değeri ve interrupt oluştuğunda dur ve devam et komutları aktif edilmiştir. (State machine çalışma prensibi LIS3DSH sensörünün datasheetinde anlatılmıştır).

ST1_1 registırına 0x0A (NOP | LLTH2) yazılarak ilk durumda (state1) tüm eksenler THRS2'den küçük ya da eşitse bir sonraki duruma geç.

ST1_2 registırına 0x61 (GNTH2 | TI1) yazılarak ikinci durumda (state2) Timer1'in içindeki değerden aşağıya say ve sayma esnasında herhangi bir eksen THRS'den büyükse kesme oluştur.

ST1_3 registırına 0x11 yazılarak üçüncü durumda (state3) reset noktasından çalışmaya devam et komutu yazdırılmıştır.

Burdan aşağısı Wake-Up içindir.

THRS1_2 registırına lis_convert_threshold(miliG) (bu fonksiyon aşağıda açıklanmıştır) yazılarak interrupt oluşması için kaç G kuvveti gerekli olduğu yazılmıştır.

MSK2_B registırına 0xA8 yazılarak SM1 (state machine) için eksenlerin pozitif kısımları aktif edilmiştir.

MSK2_A registırına 0xA8 yazılarak SM1 (varsayılan state machine) için eksenlerin pozitif kısımları aktif edilmiştir.

SETT2 registırına 0xA3 yazılarak SM2 için tepe algılama, işaretli eşik değeri ve interrupt oluştuğunda dur ve devam et komutları aktif edilmiştir. (State machine çalışma prensibi LIS3DSH sensörünün datasheetinde anlatılmıştır).

ST2_1 registırına 0x05 (NOP | GNTH1) yazılarak ilk durumda (state1) herhangi bir eksen THRS1'den büyük ya da eşitse bir sonraki duruma geç.

ST2_2 registırına 0x11 yazılarak ikinci durumda (state2) reset noktasından çalışmaya devam et komutu yazdırılmıştır.

Sonda koruma amaçlı olarak 0x5F adresi okunarak önceki interrupt bayrağı sıfırlanmıştır.

Bu sensörün ürettiği interruptlar sayesinde EXTI0_IRQHandler ve EXTI1_IRQHandler fonksiyonları altında flag_control adında bir değişken 1'e eşitlenerek ana kodda kaza verileri SD Kart'a yazdırılmıştır.

Lis_init fonksiyonu Şekil 4.3'te verilmiştir.

```
if(ReadSpi(WHO_AM_I) == 0x3F)                       // sensorle haberlesme olup olmadigi kontrol ediliyor
{
    WriteSpi(CTRL_REG1, 0x01);                      // Durum Makinesi 1 aktif
    WriteSpi(CTRL_REG2, 0x09);                      // Durum Makinesi 2 aktif, INT2 bacagi aktif
    WriteSpi(CTRL_REG4, 0x77);                      // Veri cikis araliği 400Hz, X,Y ve Z ekseni aktif
    WriteSpi(CTRL_REG3, 0x78);                      // Interrupt 1 ve Interrupt 2 aktif
    WriteSpi(CTRL_REG5, 0x08);                      // Tam tarama secimi +/- 4G, full duplex


    //FREE-FALL CONFIG
    WriteSpi(TIM1_1L, lis_convert_time(200));       // Sifirlama icin Timer1 degeri (16bit --> TIM_1H & TIM_1L)
    WriteSpi(TIM1_1H, lis_convert_time(0));
    WriteSpi(THRS2_1, lis_convert_threshold(100));  // SM1 için Esik degeri
    WriteSpi(MASK1_B, 0xA8);                        // X,Y ve Z eksenleri pozitif maskeleme aktif
    WriteSpi(MASK1_A, 0xA8);                        // X,Y ve Z eksenleri pozitif maskeleme aktif
    WriteSpi(SETT1, 0xA3);                          // SM1 tepe algilama, stop ve cont komutlari kullanimi aktif
    WriteSpi(ST1_1, 0x0A);                          // Tum eksenler THRS2'den kucuk yada esitse bir sonraki duruma gec
    WriteSpi(ST1_2, 0x61);                          // Timer1 in icindeki degerden assagi say ve sayma esnasinda herhangi bir
    WriteSpi(ST1_3, 0x11);                          // Reset durumdan calismaya devam et


    //WAKE-UP CONFIG
    WriteSpi(THRS1_2, lis_convert_threshold(3500)); // SM2 için Esik degeri
    WriteSpi(MASK2_B, 0xA8);                        // X,Y ve Z eksenleri pozitif maskeleme aktif
    WriteSpi(MASK2_A, 0xA8);                        // X,Y ve Z eksenleri pozitif maskeleme aktif
    WriteSpi(SETT2, 0xA3);                          // SM2 tepe algilama, stop ve cont komutlari kullanimi aktif
    WriteSpi(ST2_1, 0x05);                          // Tum eksenler THRS1'den kucuk yada esitse bir sonraki duruma gec
    WriteSpi(ST2_2, 0x11);                          // Reset durumdan calismaya devam et


    ReadSpi(0x5F);                                  // kesme bayragini temizliyoruz
}
```

Şekil 4.3

4.4 Lis_convert_time Fonksiyonu

Lis_convert_time fonksiyonunda interrupt oluşması için gerekli G kuvvetine maruz kalma süresi hesaplanmıştır. LIS3DSH sensörünün kullanım kılavuzunda izin verilen maruz kalma süresi 2.5ms – 637.5ms arasındadır hesaplamalar bu aralığa göre yapılmalıdır. Bu fonksiyon ms tipinde miliseconds adında bir değer almaktadır. Yukarda sensörün veri aralığını 400Hz seçmiştik bu değer 2.5 milisaniye etmektedir. Miliseconds değeri 2.5 milisaniyeye bölündüğünde bu değer işaretsiz integer tipine dönüştürülüp return edilmiştir ve TIM1_1L registırına yazılmıştır.

4.5 Lis_convert_threshold Fonksiyonu

Lis_convert_threshold fonksiyonunda interrupt oluşması için gerekli olan G kuvveti hesaplanmıştır. LIS3DSH sensörünün kullanım kılavuzunda izin verilen G kuvveti 15.625mg – 3984mg arasındadır hesaplamalar bu aralığa göre yapılmalıdır. Bu fonksiyon mg tipinde miliG adında bir değer almaktadır. Yukarda sensörün tam taramasını (Full-scale) 2G seçmiştik bu değer 2^7 değerine bölünmelidir ((2*9.81… / 128) = 15.625). İstenilen G değeri bu değer (15.625) ile çarpılıp elde edilen değer işaretsiz integer tipine dönüştürülüp return edilmiştir ve THRS2_1 registırına yazılmıştır.

Bu eşik değeri ve zaman hesabı fonksiyonları Şekil 4.4-5'te verilmiştir.

```
uint8_t lis_convert_time(float miliseconds)
{
    // 1 LSB = 1/ODR = 1/400 Hz
    // The minimum duration (in milliseconds) of subthreshold accelerations for recognize a free-fall condition.
    // Allowed range [2,5 - 637,5]ms.

    float var = miliseconds / (2.5);
    int byte = (int)var;
    if(byte < 0)
        return 0;
    else if(byte > 255)
        return 255;
    else
        return (uint8_t)byte;
}

uint8_t lis_convert_threshold(float miliG)
{
    // 1 LSB = 2g/(2^7)
    // The maximum acceleration (in milli-g) that is recognized as free-fall condition.
    // The lower is the threshold, more accurate is the recognition.
    // Allowed range [15,625 - 3984]mg.
    // we use 4g//(2^7) --> 31.25
    float var = miliG / (31.25);
    int byte = (int)var;
    if(byte < 0)
        return 0;
    else if(byte > 255)
        return 255;
    else
        return (uint8_t)byte;
}
```

Şekil 4.4-5

4.6 TIM2_IRQHandler Fonksiyonu

TIM2_IRQHandler interrupt fonksiyonunda count_interrupt değişkeni her TIM2 kesmesi geldiğinde 1 arttırılmıştır bu count_interrupt değişkeni ana kodda değiştirilebilmektedir, bu değişken sayesinde kaç saniyede bir SD karta düzenli veri yazılması gerektiği kontrol edilmiştir.

4.7 FatFs Dosya İşlemleri

FatFs dosya işlemleri için fatfs, result ve file_pointer adında FatFS değişkenleri tanımlanmıştır. FatFs_Init adında bir fonksiyon tanımlayarak sağlıklı bir şekilde dosya oluşturma işlemi yapılmıştır. Bu fonksiyonun içinde result değişkeni ile SD kart haberleşmesi olup olmadığı kontrol edilmiştir. Daha sonra DATA.txt ve CRASH.txt adında iki dosya mikro SD kart içinde oluşturulmuştur.

Write_SD adında bir fonksiyon oluşturularak içerisinde oluşturulan dosyalara veri yazmak için FA_OPEN_APPEND komutu ile veriler yazdırılmıştır.

4.8 RTC İşlemleri

HAL_RTC_GetDate ve HAL_RTC_GetTime fonksiyonları kullanılarak time_second, time_minute, time_hour, date_week, date_month, date_year değişkenlerine zaman ve tarih verileri atılmıştır.

Temperature ve Pressure adında 8 bitlik işaretsiz integer tipinde iki değişken tanımlanıp bu değişkenlere BMP180_GetTemp, BMP180_GetPress fonksiyonları ile sıcaklık ve basınç değerleri atılmıştır.

4.9 Read_all fonksiyonu

Read_all fonksiyonu tüm verileri okuyup SD kart'a yazmak için oluşturulmuştur. Read_all fonksiyonu Şekil 4.9.1'de verilmiştir.

```
689
690 void Read_all(void)
691 {
692       x = ReadSpi(x_address);
693
694       y = ReadSpi(y_address);
695
696       z = ReadSpi(z_address);
697
698       Temperature = BMP180_GetTemp();
699
700       Pressure = BMP180_GetPress (0);
701
702       Altitude = BMP180_GetAlt(0);
703
704       Read_RTC();
705
706       SD_Write();
707
708 }
709
```

Şekil 4.9.1

5. REFERANSLAR

- https://www.st.com/resource/en/reference_manual/rm0090-stm32f405415-stm32f407417-stm32f427437-and-stm32f429439-advanced-armbased-32bit-mcus-stmicroelectronics.pdf
- https://www.st.com/resource/en/datasheet/dm00037051.pdf
- https://www.st.com/resource/en/user_manual/dm00039084-discovery-kit-with-stm32f407vg-mcu-stmicroelectronics.pdf
- https://cdn.sos.sk/productdata/57/cf/6b4eb30d/lis3dsh-1.pdf
- http://www.emcu.it/MEMS/LIS3DSH/LIS3DSH_State_Machine.pdf
- https://www.scribd.com/document/247001234/LIS3DH-App-Note-pdf

Kod Dogukan1412 github sayfamda da bulunmaktadır.