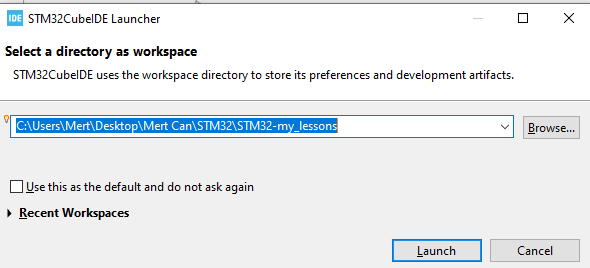
# STM32 DERSLERİ

## Başlangıç

STM32 programlama da HAL kütüphaneleri sayesinde kolayca programlama yapılabilmektedir. Register seviyesinden ziyade HAL kütüphaneleri Arduino tarzı fonksiyon çağırma gibi metotlarla kolayca ve bellekte çok yer kaplamadan yapılabilmektedir.

STM32 programlama için ben STM32F070RB mikroişlemcisini barındıran NUCLEO kartını kullanacağım. Programlama da kullanılacak IDE ise ST firmasının kendi ürünü olan STM32CubeIDE kullanacağım.

Programı ilk çalıştırdığımızda bizden bir çalışma alanı olarak “workspace” belirtmemizi isteyecektir. Program bu klasörün içerisine her bir proje için ayrı klasör oluşturup kendi özel dosyalarını eklemektedir. Bunun için çalışacağımız klasörü seçmemiz gerekir.

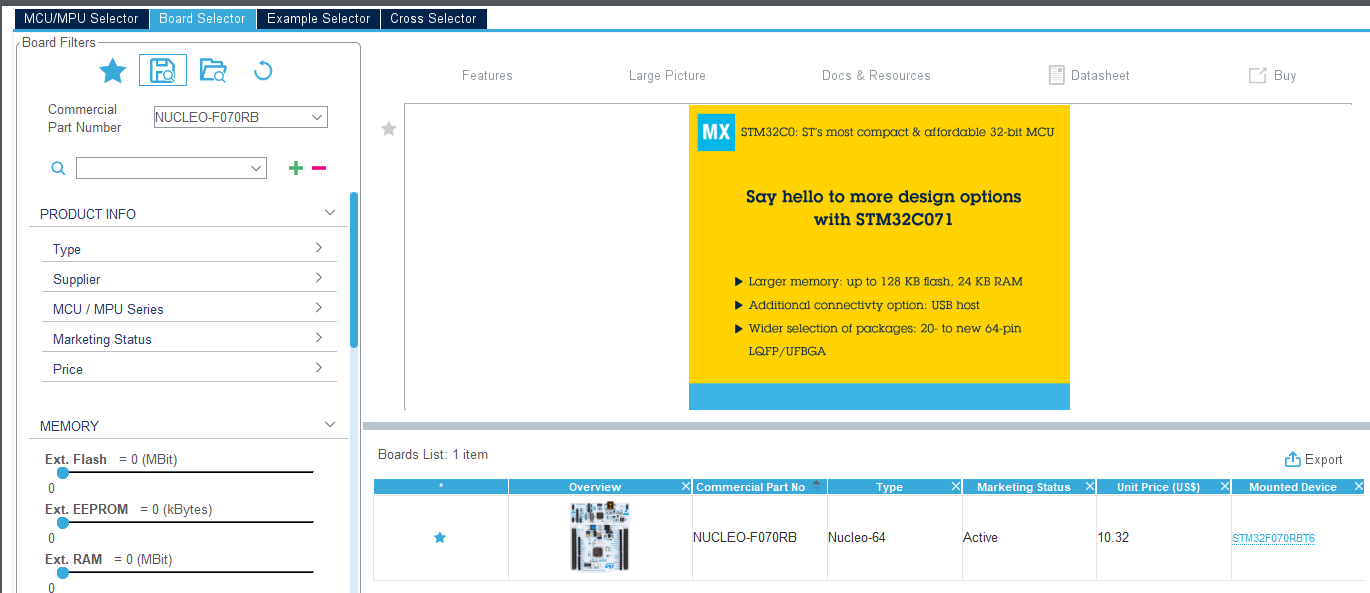


Daha sonra boş bir ekran üzerinde, solda projelerimiz olduğu ekran açılır. Buradan File>New >STM32 Project seçilerek sıfırdan bir proje açılır.

metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

Açılan ekranda “Board Selector” sekmesi altında “NUCLEO-F070RB” aratılarak geliştirme kartımız bulunur.



Next diyerek proje isimlendirilmesi yapılarak yeni projemiz açılır. Karşımıza. ioc uzantılı bir dosya açılır. Bu dosyada geliştirme kartımızda mikroişlemcimize bağlanmış olan bağlantılar gözükür. Ekstra olarak bağlamak istediğimiz giriş, çıkış veya başka birimlerin konfigürasyonu yapılarak projemize entegre edebilir ve otomatik kod oluşturmasını sağlayabiliriz.

Clock ayarları;

Yine. ioc uzantılı dosyada Clock Configuration sekmesi altında mikroişlemcinin çalışacağı clock ayarlarını yapabiliriz. Şimdilik bunlara dokunmayacağız. Ama ileride Timer gibi bazı özel işlerde kullanabileceğiz

## GPIO Kullanımı

İlk dersimizde her mikroişlemcide yaptığımız led yakma söndürme olan blink uygulamamızı yapacağız. Bunun için kartın üstünde buluna LD2 olarak tanımlanmış ledimizle yapabiliriz. Bu ledin hangi portun hangi pinine bağlı olduğunu anlamak için .ioc uzantılı dosyadan veya kartın ilgili dökümanlarına ST’nin kendi sitesinden bakabiliriz. Benim kullandığım kartta LD2 A portunun 5. Pinine bağlı bulunmakta.

İlk dersimiz için Lesson 0001 – GPIO\_Blink adlı dosya açalım. Bu dosyayı açtıktan sonra .ioc üzerinde bir şey yapmamız gerekmeyecek. Projeyi açtıktan sonra proje ağacımızın altında birçok dosya bulunmakta. Bunlara hiç dokunmadan Core dosyası altında Src ve Inc dosyaları bizleri ilgilendirmektedir. Inc altında C programlama ile yazılmış kütüphanelerin .h uzantılı header dosyaları, Src altında ise .c uzantılı dosyaları bulunmakta. Bizler kodlarımızı Src altında bulunan main.c’nin içine yazacağız. Bunların dışındaki dosyalar ST firmasının o işlemci özelinde oluşturduğu ve başlangıçta çalıştırılan veya HAL kütüphanesinin çalışması için gereken dosyalardır. Zamanla bu dosyalara da değineceğiz.   
Şimdi main.c’nin altında yine HAL kütüphanesini ilgilendiren veya işlemcinin çalışması için gereken bir çok kod göreceğiz. Bunları yine dokunmadan “USER CODE BEGIN” ile başlayan ve “USER CODE END” olarak biten yorum satırları arasına kodlarımızı uygun yere yazacağız. Bunların dışında bir yere yazılması halinde kodlarımız silinecek ve çalışmayacaktır.

\*\*\*Doğrudan led yakıp söndürmek için bu kısımları atlayabilirsiniz. Bu kısımda GPIO birimi için gerekli ayarlar nedir, neler yapmak gerekir onlar anlatılacaktır.\*\*\*

HAL kütüphanesinin hangi kodlarla hangi kodları çalıştırabileceğimize derinden öğrenmek için ST firmasının her seri için oluşturduğu “Description of STM32… HAL and low-layer drivers” dökümanına kendi sitesinden ulaşabilirsiniz. Bu dökümanda GPIO, TIMER, ADC, PWM gibi gibi her fonksiyon için kullanılan belirli fonksiyonları ve bunların açıklamalarını bulabilirsiniz. Benim kullandığım nucleo kartına uygun dökümanda GPIO için Generic Driver ve Extension Driver olarak iki bölümde açıklanmıştır. Şimdilik Generic driver için oluşturulmuş fonksiyonları inceleyeceğiz.

STM32 mikroişlemcilerinde diğer işlemciler de olduğu gibi bir pin üzerinde birden fazla birimler, yani GPIO, TIMER, ADC vb. gibi birimler barındırabilir. Hangi pinlerde hangi birimlerin hangi bacağı bulunduğunu datasheet ve referance manuel dökümanlarından ulaşabilirsiniz. Bir işlem veya uygulama için hangi birimi ve hangi pinleri kullanacağınızı belirledikten sonra bazı ayarlar yapmanız gerekmektedir. Bizim ilk uygulamamız olan led yakıp söndürme için GPIO biriminin gereken özelliklerini incelememiz gerekir. GPIO için yapılması gereken birkaç ayar vardır. Bunlar;

metin, yazı tipi, ekran görüntüsü, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bu ayarları uygulamanız için ne gerekiyorsa ona uygun ayarlayıp çıkış almamız gerekmektedir. Bütün bunları HAL kütüphanesi ile yapabileceğimiz gibi doğrudan register seviyesinde, ilgili registerların bitlerini değiştirerek yapabiliriz. GPIO biriminin bütün registerlarına ve ne işe yaradıklarına reference manuel dökümantasyonundan erişebilirsiniz. Bunun yerine HAL kütüphanesi bizlere kolaylık sağlamaktadır.

**Uygulama 1.1)** Led yak söndür;

İlk uygulamamızda led yakıp söndürmek için bu ayarları HAL kütüphanesinin her bir ayar için kullandığı struct yapısından ilgili moda göre karşılığını yazacağız.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Ledi yakıp 1 sn. aralıklarla söndürmek için ise;

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Uygulama 1.2)** Buton ile led kontrolü;

Yine GPIO ayarları yapılarak kart üzerindeki butonu giriş olarak ayarlayıp bununla ledimizi kontrol edeceğiz. Butonumuz işlemcimizin C portunun 13. Pinine bağlı.

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu  
Butona basıldığında lek yakması için if-else yapısı;

metin, ekran görüntüsü, yazı tipi içeren bir resim

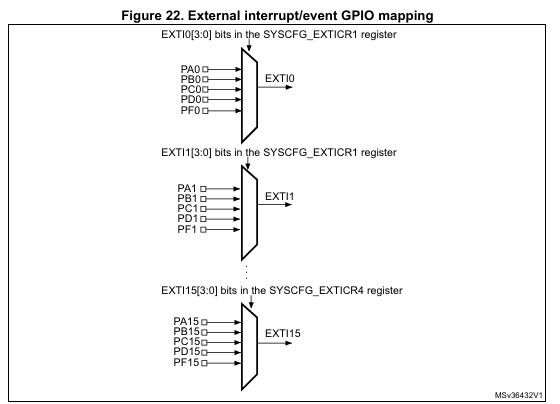
Açıklama otomatik olarak oluşturuldu

**Uygulama 1.3)** Interrupt (kesme) kullanarak buton kontrolü;

Interruptlar yani kesmeler, program akışında bir olayın gerçekleşmesi anında programın akışını durdurup istenen bloğu çalıştırmak için kullanılır. Gömülü yazılım uygulamaları genellikle bir main fonksiyonunun içerisinde sonsuz bir döngüde gerçekleştirilir. Bu sonsuz döngü içerisinde beklenen olaylar bir if-else gibi yapılarla kontrol edilir ve şartlar gerçekleştiğinde istenen durumlar yapılır. Bu sonsuz döngüde işlemler yukarıdan aşağı bir şekilde tamamlanır. Eğer programınız çok fazla işlem içeriyor ve işlemler uzun sürüyorsa bir şartın kontrolünü yapmakta geç kalabilir ve anlık olayları kaçırabilir. Kesmeler bu sonsuz döngüden bağımsız bir şekilde, beklenen şart gerçekleştiğinde sonsuz döngüyü durdurur ve kesme için belirtilen fonksiyona dallanıp işlemleri yapar. Sonrasında döngüde kaldığı yerden devam eder.

Mikroişlemcilerde kesmeler iç kesme (internal) ve dış kesme (external) olarak ikiye ayrılır. İç kesmeler mikroişlemcinin içerisinde bulunan bir birimde gerçekleşen olay için gerçekleşir. Örnek olarak timer biriminde istenen zamana ulaşınca oluşan kesme veya uart ile seri haberleşmede istenen bir bilgi geldiğinde bunlar iç kesmedir. Dış kesme ise dışarıdan yani buton, sensör vb. gibi bir olayın gerçekleştiği zaman oluşan kesmelerdir. Bu uygulamada dış kesmeyi kullanacağız.

STM32f070’de dış kesmelerle ilgili ayarların yapılacağı birim EXTI birimidir. Burada Dış kesmenin kaynağı belirtilmelidir. EXTI birimden 0’dan başlayıp 32’ye kadar kanal bulunmaktadır. Dışarıdan bağlı olan kanallar 15’e kadar olup sonrasında başka birimler bağlıdır. Her bir portun bir pini bir EXTI kanalına bağlıdır. Yani A, B, C, D, E ve F portlarının 0. Bitleri EXTI0’a, 1. Bitleri EXTI1’e şeklinde bağlıdır. Her bir kanala birden fazla portun pinleri bağlı olduğundan dolayı burada bir seçim yapılması zorunludur.



GPIO ayarlarını ve kesme ayarlarını yapıyoruz;

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Kesme olmadan yani butona basılmaz iken çalışacak kod;

metin, ekran görüntüsü, yazı tipi, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Kesme fonksiyonu;

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Burada for döngüsünün açılmasının sebebi, kesme içerisinde delay kullanamayız. Delay kullanmaz isek kesme fonksiyonuna girebildiğinden emin olamayız. Bunun için biraz süre alacak bir for döngüsü oluşturuyoruz.

## Modüler Yapılar Oluşturmak

Şimdiye kadar yazılan kodlarda bütün programı main.c’nin içine yazdık. Bu tür basit programlarda bu sıkıntı değil. Lakin uzun programlarda bütün işlerin derli toplu olması, bir değişiklik yapmak için çok uzun kodların arasından değişecek kodu bulup değiştirmenin zor olması gibi durumlarda modüler yapılar kolaylık sağlar. Modüler yapı kullanmak için kullandığımız her birim için farklı kendi kütüphanemiz tarzı dosyalar oluşturup bunu main içerisinde çağırmamız gerekir. Basit bir blink kodu için;

**Uygulama 2.1)** Modüler yapıda Blink uygulaması;

Blink uygulamasında led yakıp söndürme işlemini yapıyorduk. Kullanacağımız led yine kartın üzerinde bulunan A5’e bağlı LD2 ledi. Modüler yapı için bütün dosyalarımızın olacağı bir klasör açmamız gerekecek. Bunu >Drivers klasörünün içine açıyoruz. Ardından bu klasörün içine aynı isimli .h uzantılı ve .c uzantılı dosya açıyoruz.

yazı tipi, ekran görüntüsü, metin, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu 

.h uzantılı dosyanın kodu;

metin, yazı tipi, ekran görüntüsü, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

.c uzantılı dosyadaki kod;

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Sonrasında main.c’nin içine en yukarıya **#include** "myDrivers\_Led.h" ile kütüphanemizi ekliyoruz. Daha sonra ledin yanması için gerekli ayarların yapılması için userLed\_init() fonksiyonunu çağırıyoruz.

metin, yazı tipi, ekran görüntüsü, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

Şimdi ledin yanıp sönmesi için while içerisinde ledi yakıp bekletip söndürmemiz gerekiyor;

metin, yazı tipi, ekran görüntüsü, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

İşlem bu kadar. Burda bir sıkıntı olması durumunda projenin üstüne sağ tıklayıp properties sekmesinde C/C++ Build>Setting altında Include Paths sekmesinden myDrivers\_Led klasörünü seçin.

İlerleyen bölümlerde bu modüler yapıyı kullanmayacağız. Ama bilinmesi önemlidir.

## Timers (Zamanlayıcılar)

Timer birimi yani zamanlayıcılar mikroişlemcinin çalıştığı frekansa bağlı olarak zaman sayımı ya da doğrudan sayım yapabilirler. Mikroişlemcinin her clock darbelerinin yükselen ya da düşen kenarı ile birlikte sayım yaparlar. Bu sayım bir register üzerinde kaydedilir ve istenildiğinde kullanılır. En basit kullanımı, program içerinde kullandığımız delay fonksiyonlarıdır. Bir başka uygulama da zaman ölçümüdür. Timer birimini aktif ettiğimizde iki farklı olayın gerçekleşmesi arasındaki zamanı ölçebiliriz.

Zaman sayması yapabilmek için timer birimine uygulanan frekansı bilmemiz gerek. Çünkü clock sinyalimiz bir kare dalgadır. Ve her bir kare dalgada bir sayı arttırırız. Örnek olarak 1 Hz’lik bir frekansta her 1 saniyede 1 adım sayarız. 1kHz’lik bir frekansta her 1 ms’de bir adım ya da 1 saniyede 1000 adım sayarız.

Yani; Adım = 1/f ‘dir.

Bazı timer terimleri;

Taşma (Overflow): Sayıcı registerlarının dolma durumudur. Örnek olarak 8 bitlik bir sayıcıda 0b11111111 yani 255 değerinden sonra taşma oluşacaktır. Taşma durumunda sonra register 0’a dönecektir.

Taşma Kesmesi: Her bir taşma olduğunda kesme üretebilir ve sayım işlemi istenildiği sayıya ulaştığında bir işlem yaptırabiliriz.

Yakalama Modu (Capture Mode): Mikroişlemciye dışarıdan bir sinyal geldiği anda timer’ın içinde bulunan değeri kopyalayıp başka bir değere atayabiliriz.

Karşılaştırma Modu (Compare Mode): Karşılaştırmak için kullandığımız bir değeri bir değere atayarak timerın o değere ulaştığında bir olay oluşturabiliriz. Genellikle delay fonksiyonlarında kullanılır.

PWM (Pulse Width Modulation): Çıkışın 1-0 olma arasındaki süreleri değiştirerek çıkışın ortalama gerilimini ayarlamak için kullanılır. Doluluk oranı (duty cycle) ve periyodun değiştirilerek çıkışı değiştirebiliriz.

STM32’de Timer birimi;

Benim kullanıdğım F070 kartında 11 adet timer bulunmakta. Bunlar;

metin, ekran görüntüsü, yazı tipi, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

Biz burada bait amaçlı olan TIM6 ve TIM7’yi kullanacağız. TIM6 16 bit değerinde bir registera sahiptir. Aynı zamanda auto-reload registerına sahiptir. Yani bizim ulaşmasını istediğimiz değeri bu registera yükleyerek 16 bitin hepsinin dolmasını beklemeden istediğimiz olayı gerçekleştirebiliriz. Timer biriminin genel yapısı şu şekildedir;

metin, ekran görüntüsü, diyagram, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Timerın çalışması için clock frekansına ihtiyaç duyduğumuzu söylemiştik. Ama mikroişlemcinin kullandığı clock frekansı ile timerın kullanacağı frekans aynı değerde olmayabilir. Daha alçak bir frekans kullanmak istersek prescaler ile bu değeri bölebiliriz.

**Uygulama 3.1)** Timer ile Interrupt oluşturup led yakıp söndürme;

Bu uygulamada gerekli konfigürasyonları .ioc üzerinden yapıp otomatik oluşturacağız. 070rb için TIM6 kullanacağız. Timers sekmesi altında aktif edip gerekli değerleri girmemiz gerekiyor. NVIC üzerinden de kesmeleri aktif etmemiz gerekiyor.

metin, ekran görüntüsü, yazılım, web sayfası içeren bir resim

Açıklama otomatik olarak oluşturuldu

Mikroişlemcimiz 48Mhz’de çalışmaktadır. Her 1 sn’de ledi yakıp söndürmemiz için 1 hz’lik sinyale gerek var fakat 1Hz gibi bir değer giremeyiz. Bunun için sinyali 10MHz’e düşürüp auto load registerina ise 10000 değerini girerek her 1 saniyede taşma olacak, kesme devreye girip ledimizi yakacak ve söndürecektir. Buradaki sayıların 1 eksiğini giriyoruz çünkü mikroişlemcilerde sayma 0’dan başlar.

Timer Clk = APB Clk/Prescaler, Burada APB CLK, APB Peripheralına uygulanan clock değeri

Prescalar ise timer biriminin kendi prescalarıdır.

Freq = Timer Clk/ARR, Burada ARR auto-load registerımızdır. Yani buraya kadar sayıp sonrasında 0’a dönecektir. Her taşmada bir interrupt sinyali alacağız.

Bu uygulamada; Prescaler’ı 4800, ARR’yi de 10.000 seçiyoruz. Yani;

Timer Clk = 48 MHz/4800 = 10.000 Hz=10 kHz, Yani her 0.1ms’de 1 adım sayacak,

10.000/10.000 = 1 = Freq olmuş oluyor. 10.000 adım sayması için 1 sn gerekli

.ioc’den çıkarak otomatik kod oluşturuyoruz. Timer için oluşturulan kodlar şu şekilde olmalıdır;

metin, ekran görüntüsü, yazılım, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Şimdi kesmeyi aktif etmemiz gerekiyor;

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Şimdi ise kesme fonksiyonunun içine ledimizin durumunu değiştiriyoruz;

metin, yazı tipi, ekran görüntüsü, çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

İşlem bu kadar. Ledimiz 1sn aralıklarla yanıp sönecektir.

**Uygulama 3.2)** PWM çıkışı üretmek;

PWM’den daha önce bahsetmiştik. Çıkışın belirli zamanında 1 diğer zamanda 0 olması ve bu sürelerin ayarlarını değiştirerek çıkış gücünü değiştirebiliriz. Duty Cycle yani doluluk oranı bir sinyalin bir periyodunda aktif olduğu yani 1 olduğu sürenin bir periyoda oranıdır. Yani basitçe 1 Hz’lik sinyalde sinyal periyodu 1 saniyedir. %20 duty cyle oranı verilirse bu periyotta 0.2 s. yüksek yani 1, 0.8 s. alçak yani 0 olmuş olur.

Yukarıda hesaplanan değerlerden devam edeceğiz. Yani timerdan çıkış alacağımız frekans 1Hz olacaktır. PWM duty cyle oranımız ise CCR/ARR olarak bulabiliriz. Yani ARR değeri 10.000 iken CCR 10.000 olursa duty cycle %100 olarak bulunur.

Duty Cycle (%) = CCRx/ARR \* 100;

CCR burada Capture-Compare registerımızdır. Sayma işlemi 0’dan başlayarak ilerler. Burada yazdığımız değere ulaşınca ARR’ye yazdığımız değere kadar çıkışı 0’lar. Sayma işlemi son sayıma gelince 0’lanır ve çıkış tekrar aktif edilip aynı döngü tekrarlanır.

metin, diyagram, çizgi, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Burada led olarak kart üzerindeki LD2 ledini değil dışarıdan harici led bağlayacağız. Çünkü LD2 ledi herhangi bir timer’a bağlı bir pin değildir. TIM14 birimini kullanacağımız için PA4’ü ya da kart üzerindeki A2 pinini kullanacağız. Ledin + bacağını buraya – bacağını GND’ye bağlıyoruz.

Yeni proje açıldıktan sonra .ioc’den TIM14 aktif edilip Channel 1’den PWM Generation CH1 seçilir. TIM14’e bağlı pin PA4’dür. Dosya kaydedilip gerekli kod oluşturulur. Oluşturulan kod şu şekildedir;

metin, elektronik donanım, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Şimdi PWM’i başlatmamız gerekiyor;

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

PWM başladı fakat bunu görebilmel için duty cycle oranını belirli aralıklarla arttırarak çıkışta görmemiz gerekecek;

metin, ekran görüntüsü, yazılım, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bütün işlem bu kadar. Ledimiz 500ms aralıklarla %20 parlaklıklarla artacaktır.

**Uygulama 3.3)** PWM ile Servo Motor Kontrolü;

Servo motorlar açılarını ayarlayarak istediğimiz konuma getirebildiğimiz motorlardır. Servo motorlar PWM sinyalinin duty cycle oranlarına göre açılandırılırlar. Kullanacağımız servo motor SG90 tipi bir servo motordur ve datasheet bilgilerine göre;

metin, ekran görüntüsü, diyagram, paralel içeren bir resim

Açıklama otomatik olarak oluşturuldu

Yani 50 Hz’lik bir frekansta ya da bir periyodu 20 ms süren bir sinyalde -90 açısı için 1ms sürelik yüksek, 0 derece 1.5 ms süreli yüksek, +90 açısı için ise 2 ms süreli yüksek sinyal gitmelidir. Yani sırayla %5, %7.5 ve %10’luk duty cycle oranı gerekir.

50 Hz’lik sinyali üretebilmek için 48MHz’lik ana frekansımızı 9600-1 prescalar değerine bölersek 5kHz’lik sinyal alırız. Bu 5kHz’i de 100-1 ARR değerine bölerek 50 Hz elde ederiz. ARR değerimiz 100 olduğu için %5 için 5, %7.5 için yaklaşık 8, %10 için ise 10 değerinde bir karşılaştırma bizim için yeterli olur.

SG90 servo motorumuzun + pinini +5V’a, GND’yi GND’ye ve Sinyal pinini ise kart üzerindeki A2’ye yani mikroişlemcinin PA4’üne bağlıyoruz.

Proje kodlarımızı bir önceki uygulamadaki gibi .ioc uzantılı dosyadan aktif edip gerekli değerleri giriyoruz;

metin, ekran görüntüsü, yazılım, web sayfası içeren bir resim

Açıklama otomatik olarak oluşturuldu

Main.c içerisinde PWM’i başlatıyoruz;

metin, yazı tipi, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

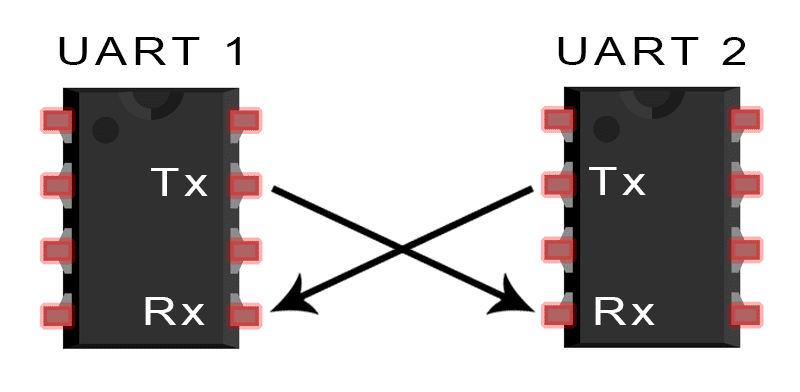
While içerisinde 500ms aralıklarla konumu -90, 0, +90 olarak değiştirebiliriz. Ayrıca for döngüsü ile de açıyı -90’dan +90’a sürekli gidip gelmesini sağlayabiliriz;

metin, ekran görüntüsü, yazılım, işletim sistemi içeren bir resim

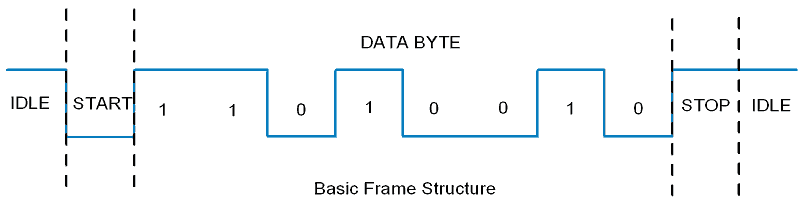
Açıklama otomatik olarak oluşturuldu

## UART Haberleşme

UART haberleşme protokolü mikroişlemcilerin başka mikroişlemciler veya pc benzeri uart haberleşmeyi destekleyen cihazlar arasında veri aktarımı yapmak için kullanılır. UART protokolünde 2 adet bağlantı vardır. Her cihaz için TX ve RX bağlantısı. TX transmitter yani gönderici ve RX receiver yani alıcı bağlantısıdır. Her bir cihazın TX bağlantısı diğer cihazın RX bağlantısına ve RX bağlantısı diğer cihazın TX bağlantısına bağlanmalıdır. Çünkü birinin gönderdiği veriyi diğeri almalıdır. Bu bağlantıya ek olarak iki mikroişlemcinin veya cihazın GND’leri birbirine bağlı olmak zorundadır.



Uart haberleşmede veriler veri paketleri denilen bir paket ile gönderilir. Bu veri paketi 11 bittir. 0. Bir başlatma, 1 ile 8. bit veriyi, 9. Bit parity yani eşleşme, 10. Bit ise durdurma ya da bitiş bitidir. 0. Bitin düşen kenarı ile haberleşme başlatılır. Ardından gelen 8 bit ile veri aktarılır. Eşleşme biti ile verinin gidip gitmediği kontrolü yapılır ve bitiş biti ile sonlandırılır.



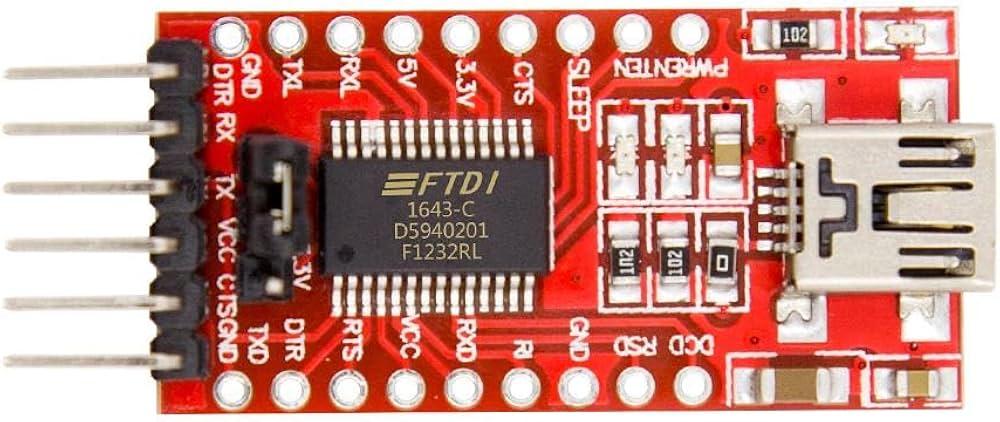
Bu veri paketlerinin gönderilme hızı ya da frekansını baud rate denilen kavramla belirtiriz. Baud rate 1 saniye içerisinde ne kadar veri taşıyacağını belirtir (bps). Haberleşecek olan iki cihazında aynı baud rate hızına sahip olması gerekir çünkü bir taraf veriyi gönderirken diğer taraf o veriyi çözümler. Örnek olarak 1 bps baud rate sahip iki cihaz haberleşirken 1 saniye boyunca yüksek (1) sinyal gönderdiğinde karşı taraf bunu “1” olarak algılar. Fakat gönderen taraf 2 bps, alıcı taraf 1 bps hızında olsaydı alıcı bu veriyi ”11” olarak algılayacak ve veri tamamen yanlış gönderilecekti.

Uart haberleşme protokollerinde baud rate’leri belirli hızlarda yapılır. En çok kullanılan baud rate’ler; 4800, 9600, 19200, 28800, 38400, 57600, 76800, 115200’dür.

**Uygulama 4.1)** Uart ile PC’ye veri gönderme;

Bu uygulamada STM32 kartımızdan PC’ye istediğimiz veriyi göndereceğiz. Bu verileri PC’ye aktarabilmek için USB-TTL dönüştürücü ya da FTDI dönüştürücüye ihtiyacımız var. Birde bu verileri görüntüleyebilmek için ayrı bir programa ihtiyacımız var. Bu program olarak da Hercules’i kullanacağız. Kısa bir araştırma ile indirebilirsiniz.

Kullandığım FTDI çevirici, üzerinde CH340 entegresi bulunduran ve USB üzerinden haberleşmemizi sağlayan bir karttır. Kart üzerindeki TX ve RX pinleri ile haberleşmeyi sağlayacağız. FTDI kartının üzerindeki TX pini STM32F070 kartının RX pinine, FTDI çeviricinin RX pini STM32F070 kartının TX pinine bağlanacaktır. Bu uygulamada program atabilmek için USART2 birimini kullanıyor. Yani diğer USART birimleri boşta. Biz USART1 birimini kullanacağız. USART1 biriminin RX pini PA10, TX pini PA9’dur.



Yeni proje açıyoruz ve .ioc uzantılı dosyadan Connectivity sekmesi altından USART1’e geliyoruz. Buradan Mode->Asynchronous’u seçiyoruz. Baud Rate’i 9600 seçip otomatik olarak kodu oluşturuyoruz.

**metin, ekran görüntüsü, yazılım, bilgisayar simgesi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

Otomatik kod oluşturma tamamlandı. Oluşan kod;

metin, ekran görüntüsü, yazılım, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Şimdi main.c üzerine kodlarımızı yazabiliriz.

Göndereceğimiz veriyi bir değişkene atıyoruz.

metin, yazı tipi, ekran görüntüsü, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Buradaki \n ASCI kodlamada bir sonraki satıra geçme anlamına gelir. Değişkenimizi 8 bitlik bir int değişkene yazıyoruz.

Daha sonra bunu sürekli göndermesi için while döngüsü içerisinde hal kütüphanesinin fonksiyonu ile veriyi iletiyoruz;  
metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Buradaki data\_tx oluşturduğumuz değişken, 6 gönderilecek bit sayısı, 1000 ise timeout yani zaman aşımı süresidir.

Program bu kadar. Şimdi hercules programı üzerinden gelen verilere bakıyoruz. Önce FTDI çeviricimizi bilgisayara bağlıyoruz. Sanrasında aygıt yöneticisinden hangi COM porta bağlı olduğunu bulup hercules üzerinden Serial sekmesi altında COM portumuzu seçiyoruz. Open dedikten sonra gönderdiğimiz veri ekrana yansıyacaktır.

metin, ekran görüntüsü, ekran, görüntüleme, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Uygulama 4.2)** Uart ile PC’den veri almak;

Bir önceki uygulamada uart ile bilgisayara veri gönderdik. Peki bilgisayardan ya da başka cihazdan nasıl veri alacağız? Bir önceki projedeki bağlantılar ve başlangıç ayarları aynı olacak şekilde devam ediyoruz. Bu sefer gönderilecek veri için değil alınacak veri için değişken oluşturuyoruz.

metin, yazı tipi, ekran görüntüsü, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

While içerisinde ise verinin gelip gelmediğini kontrol eden HAL fonksiyonunu çağırıyoruz;

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Gelen veriyi görüntülemek için bu sefer Debug modunu kullanacağız. Yukarıdan debug seçeneğini işaretleyerek yazılım debug modunda çalışacaktır. Yükleme tamamlandıktan sonra sağ tarafta yeni sekmeler açılacaktır. Buradan Live Expression seçeneği altında istediğimiz değişkenleri görüntüleyebiliriz. Şimdi Debug modunda iken yine yukarıdan resume seçeneği tıklıyoruz. Ardından live expression sekmesinde add new expressin seçeneğine tıklayarak data\_rx değişkenimizi ekliyoruz. Burada hata verebilir. Eğer hata verirsen uygulamadan çıkıp yeniden girin.

Bütün bunlardan sonra FTDI çeviricimiz bilgisayara bağlı iken “Send” kısmına göndermek istediğimiz veriyi yazıyoruz. Ben burada “mrtcnblgc” yazdım. Ardından send butonuna tıkladıktan sonra stm32CubeIde üzerinde leive expression kısmında gönderdiğimiz veri gözükecektir. Ben “mrtcnblgc” verisi almak istediğim için data\_rx değişkenini 10 elemanlı bir dizi olarak oluşturdum.

ekran görüntüsü, yazılım, multimedya yazılımı, grafik yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Uygulama 4.3)** Uart ile Kesme Kullanarak Veri Alma;

Yukarıdaki işlemlerde veri alma veya gönderme işlemleri program akışı içerisinde yapıldı. Bu durum bir projede sıkıntılar oluşturabilir. Çünkü veri alma veya gönderme işlemleri sırasında diğer işlemler aksayabilir ve istenen çıktılar oluşmaz. Bunu denemek için 6 bitlik bir veri değilde 1000-2000 bit sayısı içeren bir veri göndermeyi deneyebilirsiniz.   
Bu sıkıntının önüne geçmek için kesme birimini kullanabiliriz. Daha önce gördüğümüz kesme işlemi GPIO için geçerliydi. Aynı şekilde kesme Timer, UART, ADC gibi farklı birimlerde de bulunabilir. UART için kesme birimi veri gönderilmek istendiğinde ya da almak istendiğinde gerçekleşir. Veri almak istendiğinde veri geldiği anda kesme fonksiyonuna dallanır ve bu fonksiyonda gelen veriler bir tampona kaydedilebilir.

.ioc dosyasından USART1’i aktif ettikten sonra USART1 sekmesinin altında NVIC Settings altından USART1 global interrupt seçeneğini aktif ediyoruz. Otomatik kod oluşturduktan sonra ilgili kesmelerin aktif edilmesi için gerekli fonksiyonu tanımlıyoruz;

metin, ekran görüntüsü, yazı tipi, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Daha sonra gelecek veriler için data\_rx, gönderilecek veriler için data\_rx adlı iki adet dizi değişkeni oluşturuyoruz.

metin, yazı tipi, ekran görüntüsü, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

Data\_tx dizisinin her elemanını aynı olacak şekilde for döngüsü ile dolduruyoruz. Daha sonra ilgili kesme fonksiyonlarını çağırıyoruz;

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Programın işeyişinin değişip değişmediğini görmek için while içerisinde ledimizi 250ms aralıklarla yakıp söndürüyoruz;

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Verilerin gelip gelmediğini görmek için Hercules programına bakabiliriz. Aynı zamanda veri gönderip debug seçeneği ile bakabiliriz. Burada önemli olan veri alışverişinde ledin yanmasının aynı süreler içerisinde olduğunu görmektir.

# ADC – Analog-Dijital Dönüştürücü

Mikroişlemcinin dış dünyasındaki her veri analogdur. Yani bir ışığın şiddeti, bir gerilimin değeri, ses vb. gibi birçok nicelik analog bir veridir. Mikroişlemci bu analog verileri okuyup yorumlama yapabilmesi için kendi anlayacağı dile yani ikilik tabanda bir sayıya çevirmesi gerekir. Analog bir sinyalin herhangi bir iki noktası arasında sonsuz sayı vardır. Ama dijital bir verinin durumu ya 0 yani düşük, ya da 1 yani yüksektir.

ADC’nin çalışması analog bir sinyalin belirli bir örnekleme zamanı içerisinde, çözünürlük sayısının maksimum değeri ile 0 arasında belirtir.

metin, diyagram, çizgi, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Yani örnek olarak 10 bit çözünürlüğe sahip, 0-3.3V arasında referans gerilimi olan bir ADC’de 2^10= 1024 değeri maksimum gerilim olan 3.3v’a denk gelecektir. 0V ise 0 sayısına denk gelecektir. ADC’de her bir adım değeri ise Vref/(2^n-1) ile bulunur. Bu örnekte 3.3V\*1024-1=0.0322’dir. Yani ADC’de okunan her bir adım artışı 0.0322V’a karşılık gelecektir.

STM32’de ADC Birimi;

STM32F070’de ADC birimi 6, 8, 10, 12 bit şeklinde çözünürlük bulunur. 16 dış, 2 iç analog kanalı vardır. 2.4 ile 3.6V arasında referans gerilimi verebiliriz. Ya da mikroişlemcinin kendi güç seviyesini vref olarak kullanabiliriz.

**Uygulama 5.1)** ADC ile Pot Okuma

Yeni proje açıyoruz ve .ioc uzantılı dosyadan Analog sekmesi altında ADC seçeneğini açıyoruz. Buradan IN0 yani kanal 1’i aktif ediyoruz. IN0 kanalı mikroişlemcinin üzerinde PA0, kart üzerinde ise A0 bacağını temsil eder. Otomatik kod oluşturduktan sonra main.c ana dosyamızda gelen verileri yazacağımız ve bu verileri uart ile göndermek için iki adet değişken oluşturuyoruz;

metin, yazı tipi, ekran görüntüsü, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Daha sonra while içerisinde ADC’yi başlatıp Polling metodu ile ADC okuması yapıp değişkenimize aktarıyoruz. ADC’yi durdurup veriyi UART ile görüntülemek için gönderiyoruz;

metin, ekran görüntüsü, yazı tipi, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bütün işlem bu kadar. İster veriyi bir seri iletişim ekranlarında ya da debug ile live expression altında görebiliriz.

**Uygulama 5.2)** ADC ile Interrupt ile Okuma;

Bir önceki uygulamaya ek olarak bir ADC interrupt’ı ve bu interruptı başlatmamız gerekiyor. .ioc’den adc’yi aktif edip NVIC settings altında interruptı aktif ediyoruz. Daha sonra interruptı tanımlıyoruz;

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Daha sonra while içerisinde adc interruptını çalıştırıyoruz;

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Burda da işlem bu kadar. Veriyi debug ile görüntüleyebilirsiniz.