

HAKAN COŞKUN

Çalışılan Konular ve Özetleri

İçindekiler

[STM32CubeIDE 3](#_Toc94778014)

[Yeni Proje Oluşturma 3](#_Toc94778015)

[STM32F4XX DISCOVERY BOARD 4](#_Toc94778016)

[MEMORY BUS 4](#_Toc94778017)

[BUS (Veri Yolları) 4](#_Toc94778018)

[I-BUS 4](#_Toc94778019)

[D-BUS 4](#_Toc94778020)

[S-BUS 4](#_Toc94778021)

[MCU CLOCK SİSTEMİ 5](#_Toc94778022)

[Peripheral Clock Configuration(Çevresel saat konfigürasyonu) 5](#_Toc94778023)

[STM32FXX Discovery Board ve GPIO PORT 8](#_Toc94778024)

[GPIO Port MODE Register 9](#_Toc94778025)

[Stm32f4xx Header File(Başlık Dosyaso) ve Source File(Kaynak Dosyası) Oluşturma 10](#_Toc94778026)

[I2C 14](#_Toc94778027)

[GIT 17](#_Toc94778028)

[GIT NEDİR? 17](#_Toc94778029)

[GIT KURULUMU(WİNDOWS) 17](#_Toc94778030)

[GIT CMD 17](#_Toc94778031)

[GIT BASH 17](#_Toc94778032)

[GIT GUI 17](#_Toc94778033)

[Git Veri Akışı 18](#_Toc94778034)

[Git Komutları 18](#_Toc94778035)

[pwd 18](#_Toc94778036)

[ls 19](#_Toc94778037)

[cd 19](#_Toc94778038)

[touch DosyaAdı 19](#_Toc94778039)

[git rm 20](#_Toc94778040)

[git mv 20](#_Toc94778041)

[git init 20](#_Toc94778042)

[git status 21](#_Toc94778043)

[git add 21](#_Toc94778044)

[git commit –m “commit massage”: 22](#_Toc94778045)

[git log 22](#_Toc94778046)

[git diff 23](#_Toc94778047)

[git restore <file name> 23](#_Toc94778048)

[git checkout / git restore DosyaAdı 23](#_Toc94778049)

[git reset / git restore --staged <file>... 24](#_Toc94778050)

[git remote add BağlantıAdı GitHubLinki 25](#_Toc94778051)

[git push –u bağlantıadı master 25](#_Toc94778052)

[git pull 25](#_Toc94778053)

[cat >> .gitignore 25](#_Toc94778054)

[git branch DosyaAdı 25](#_Toc94778055)

[git checkout “DalAdı” 25](#_Toc94778056)

[git merge 26](#_Toc94778057)

[git reset 26](#_Toc94778058)

[git reset head “DosyaAdı” 26](#_Toc94778059)

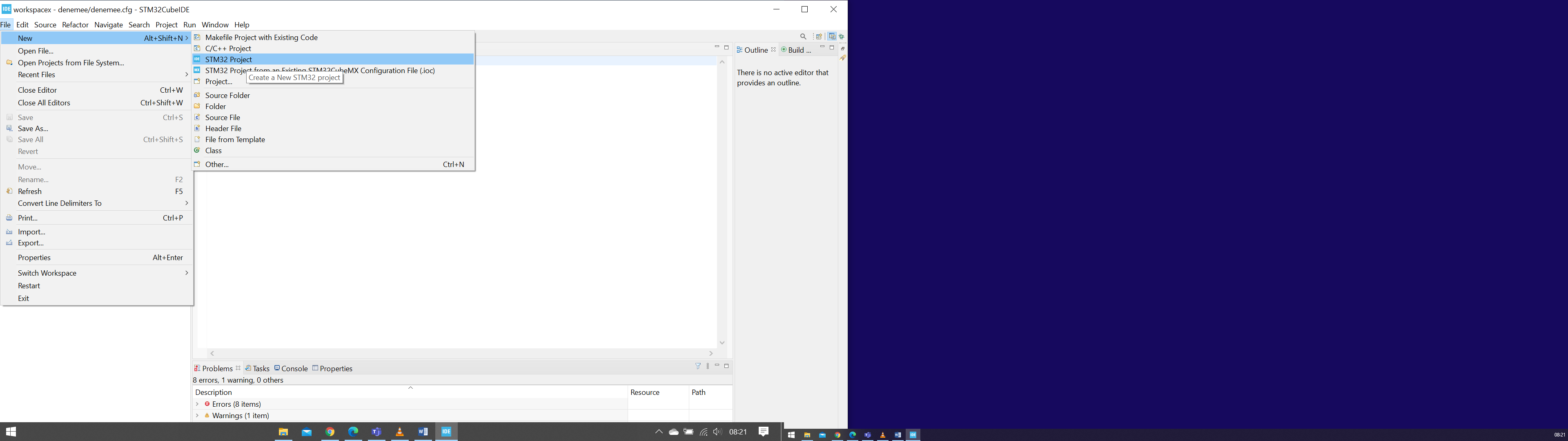
[git clean –f 26](#_Toc94778060)

[--dryrun-run 26](#_Toc94778061)

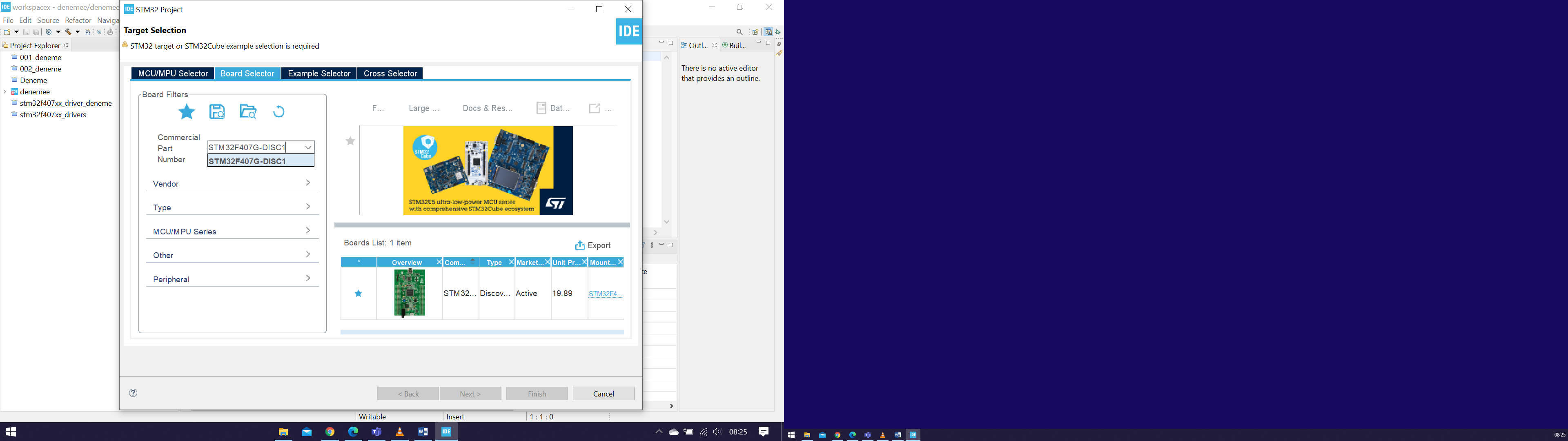
# STM32CubeIDE

STM32CubeIDE(Integrated Development Environment) STM32 mikro denetleyicileri ve mikroişlemciler için çevresel yapılandırma, kod oluşturma, kod derleme ve hata ayıklama özelliklerine sahip gelişmiş bir C/C ++ geliştirme platformudur. Eclipse(açık kaynaklı geliştirme programı) tabanlıdır.

## Yeni Proje Oluşturma



Şekil (Yeni Proje Başlatma)



Şekil

* Şekil 2’de “Targeted Project Type” kısmında STM32Cube seçilirse, proje STM32Cube ile başlar ve burada gerekli pin ayarları ve clock ayarları yapılabilir.

# STM32F4XX DISCOVERY BOARD

Burada kullanılan işlemci ayarları ve “Referance Manual” gibi dokümanlar farklı MCU’lar için farklı ayarlar olsa da temelde mantığı aynıdır. Yani burası rehber olarak kabul edilebilir.

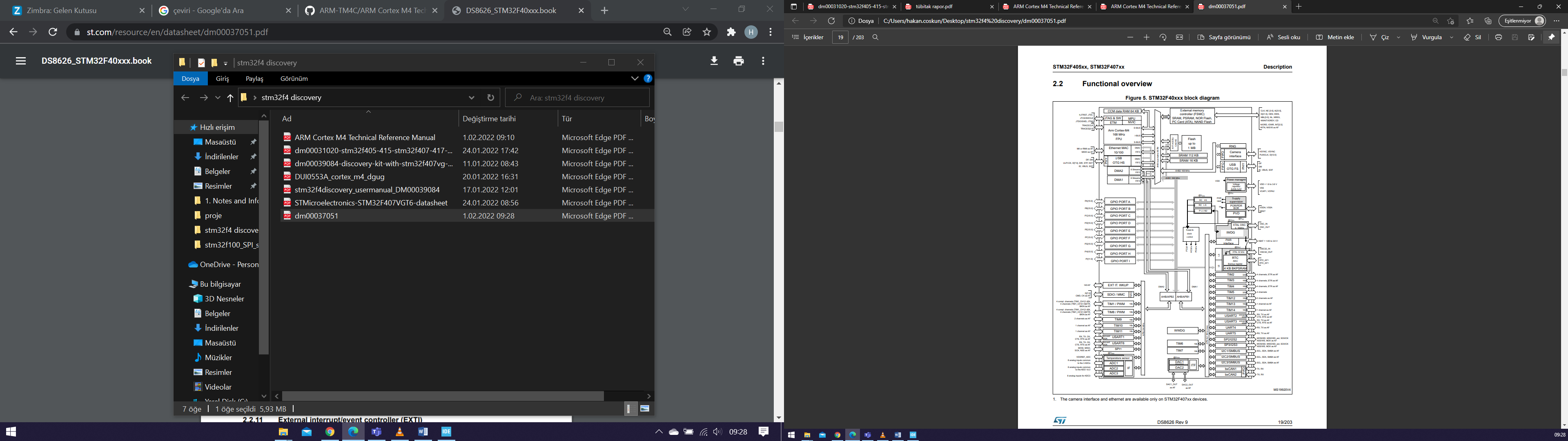
# MEMORY BUS

## BUS (Veri Yolları)

I-BUS: Bu veri yolu komutları getirmek için çekirdek tarafından(ARM M4kullanılır. Hedefi kod içerin bellektir(Internal Flash Memory, SRAM…). ARM’ın veri yolu üzerinden bellekteki kodları okur.

D-BUS**:** ARM flash’da saklanan verileri okumak için bu veri yolunu kullanır. Ayrıca debug erişimi içinde kullanılır.

S-BUS**:** Bu veri yolu, bir çevre biriminde veya SRAM'de bulunan verilere erişmek için kullanılır. Bu veri yolunun hedefleri dâhili SRAM1, SRAM2 ve SRAM3, APB çevre birimleri dâhil AHB1 çevre birimleri, AHB2 çevre birimleri ve FSMC/FMC aracılığıyla harici belleklerdir. Cortex-M4 işlemcisi 3 adet AHB(Advanced High Performance) ve 1 adet APB(Advanced Peripheral Bus) içerir.

.

Şekil STM32F40xxx block diagram

Şekil 3’de Veri yolları görülüyor.

*[Daha detaylı bilgi için Referance Manual’da “Memory and bus architecture” bölümüne ve Cortex-M4 Technical Reference Manual’da “Bus interfaces” bölümleri incelenebilir.]*

## MCU CLOCK SİSTEMİ

3 farklı clock kayağı bulunur. Bunlar:

* HSI osilatör clock
* HSE osilatör clock
* Main PLL (PLL) clock

**HSI ocsillator**: HSI clock sinyali dâhili 16 MHz’lik RC osilatöründen üretilir, ve doğrudan sistem clock’unu beslemek için ya da PLL girişi olarak kullanabiliriz. HSI RC osilatörünü kullanmanın önemli bir avantajı düşük maliyetle clock kaynağını elde edebilmemiz, çünkü başka bir komponent kullanmamıza ihtiyaç yok. Ayrıca HSE kristal osilatörüne göre sistem başlangıç hızı daha fazladır, ancak kalibrasyon yapsanız bile üretilen osilatör frekansı harici kristalden yada seramik rezonatörden üretilen frekanstan daha kötüdür.

HSE osilatör clock: Yüksek hızlı harici clock sinyali (HSE) 2 clock kaynağından üretilebilir:

* HSE harici kristal
* HSE harici kullanıcı clock’u

**Harici kristal**: Bu modda, harici clock kaynağı sağlanmalı. RCC clock kontrol kayıtçısında (RCC\_CR) bulunan HSEBYP ve HSEON bitlerini set ederek bu moda geçiş yapabilirsiniz.

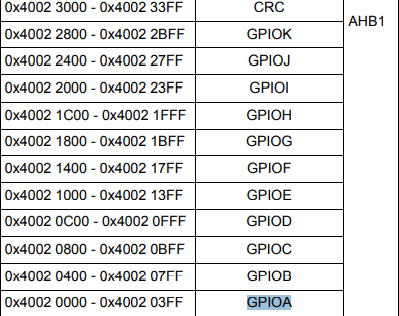
**HSE harici kullanıcı clock’u:** HSE kullanmak doğru bir clock sinyalinin üretimi için avantajlıdır.

**Main PLL (PLL) clock:** STM32F4xx ürünleri 2 adet PLL bulundurur. Temel PLL (PLL) HSE yada HSI osilatörü tarafından ayarlanır, ve 2 farklı clock çıkışı sağlar:

– Birinci çıkış yüksek hızlı sistem clock’u üretmek amacı ile kullanılır (168 MHz e kadar). – İkinci çıkış USB OTG FS clock’u ( 48MHz ), rastlantısal analog üreteci ( ≤ 48 MHz ) ve SDIO ( ≤ 48 MHz ) üretmek amacı ile kullanılır.

## Peripheral Clock Configuration(Çevresel saat konfigürasyonu)

* Bu bölümde belirli bir çevre birimi(ADC, GPIO vb.) için saatin nasıl etkinleştirileceğini açıklayacağım. Öncelikle belirli bir çevre birimini kullanmadan önce bu çevre birimi için gerekli olan saati etkinleştirmemiz gerekir çünkü default olarak kapalı durumdadır.
* STM32 mikro denetleyicilerinde, çevresel saatler RCC kayıtları(register) aracılığıyla yönetilir.
* Örneğin GPIOA çevre birimini etkinleştirmek istediğimizi ve 10. Portu set olarak ayarlamak istediğimizi varsayalım. İlk olarak Referance manuel dokümanında Memory Map bölümüne gidiyoruz ve GPIOA biriminin arıyoruz.

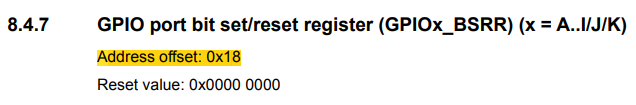


Bağlı olduğu BUS

Base Address

Şekil

* Şekil 4’te görüldüğü gibi 0x4002 000 GPIOA biriminin temel adresidir. Sonrasında GPIO port bit set/reset register bölümüne gidip Address offset’i alıyoruz.



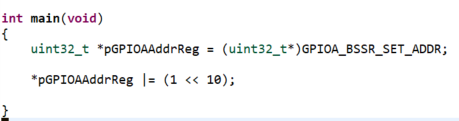
Şekil

* Yani GPIOA pinini set olarak ayarlamak istediğimizde base address’e address offset’i(şekil 5) eklememiz gerekiyor. STM32CubeIDE’deki C kodları aşağıdaki şekil 6’daki gibidir.



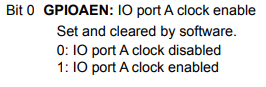
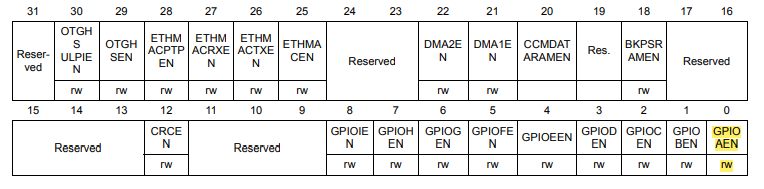
Şekil

* Şimdi GPIOA’nın 10. Pinini set etmek için bu adresi kullanalım. Şekil 7’de görüldüğü gibi kodlama işlemi yapılır



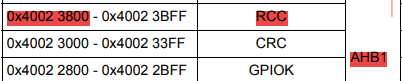
Şekil

* Kodu bu şekilde bırakırsak istediğimiz bit set olmaz çünkü daha öncede bahsettiğimiz gibi öncelikle gerekli olan saati etkinleştirmeliydik. O da şekil xx’de görüldüğü gibi AHB1’dir. Ayrıca Data Sheet dokümanından STM32F40xxx block diagram şemasını açarsak, hangi çevre biriminin hangi saate bağlı olduğunu oradan da görebiliriz.
* RCC AHB1 peripheral clock enable register (RCC\_AHB1ENR) bölümüne gidiyoruz ve orada GPIOA için Bit 0’ı 1 yapmamamız gerektiğini anlıyoruz. Bunu o bölümden alınan aşağıdaki şekil 8’e bakarak da anlayabiliriz.



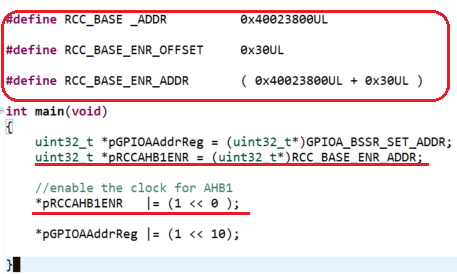
Şekil

* Öncelikle RCC base adresi bulmamız gereklidir. Bunu da referance manual dokümanının memory map bölümünden aşağıdaki şekil 9’daki gibi bulabiliriz. Ayrıca RCC\_AHB1ENR adres offset değerini de almalıyız(0x30).



Şekil

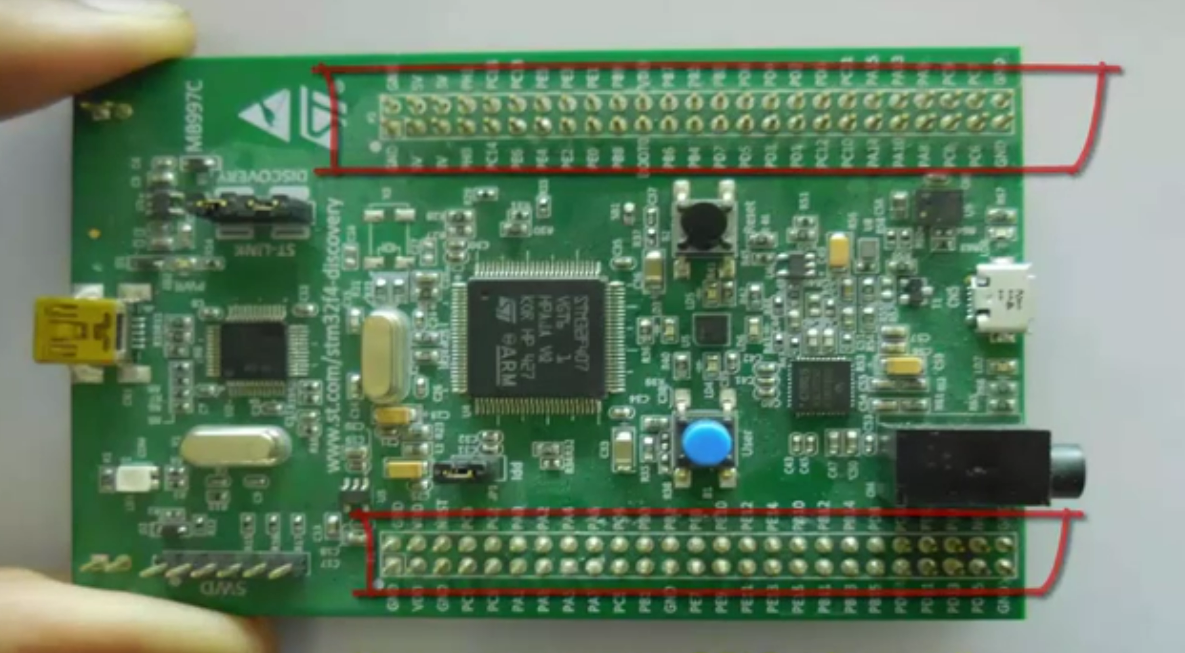
* Sonuç olarak C kodları şekil 10’daki gibi olur.



Şekil

## STM32FXX Discovery Board ve GPIO PORT

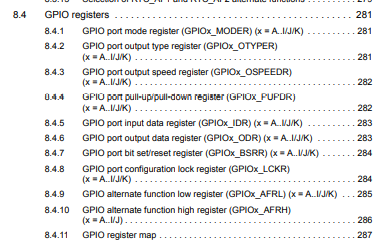
STM32F407VG mikro denetleyicisinde, GPIOA’dan GPIOI’ya kadar olan toplam 9 adede kadar GPIO bağlantı noktasını destekler. Ve her grupta 16 adet(P0-P15) iğneli grup bulunur. Her bağlantı noktasının kendi yapılandırma kayıt kümesi(configuration registers) vardır. Fakat Discovery Bord’unda GPIOA’dan GPIOE’ye kadar grup vardır. Şekil 11’de Board üzerindeki pinler görülebilir.



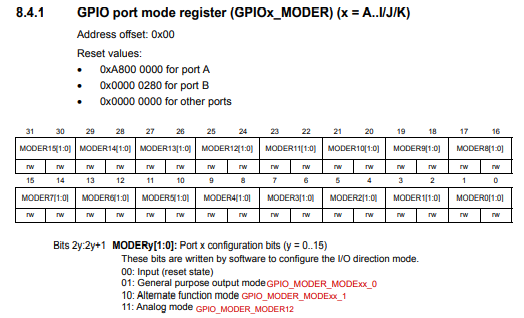
Şekil

## GPIO Port MODE Register

GPIO çevre biriminin kayıt kümeleri için Referance Manuel dokümanından yaralanmalıyız. Örnek olarak GPIO portları için şekil 12’ye bakınız.



Şekil



1

Şekil

* GPIO portunun 4 adet MODER modu vardır(şekil 13 1 numara). Bunlar: Input, : General purpose output mode, Alternate function mode, Analog mode. Burada her iki bit bir portu temsil etmektedir. Örneğin GPIOA portunun P0 pinini analog mode olarak ayarlamak istiyoruz. O zaman Moder0’ı “11” olarak ayarlamalıyız.



Şekil

* Yukarıdaki şekil 14’de görüldüğü gibi GPIO Pinler farklı amaçlarda kullanılabilir. Ama bütün pinler bu görevler için kullanılamaz. Örneğin bütün pinleri I2C’nin SDA hattı olarak kullanamayız. Bunun için belirli pinler özelleştirilmiştir.

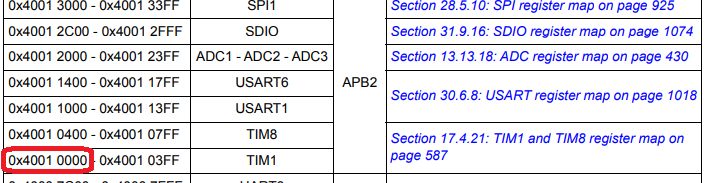
## Stm32f4xx Header File(Başlık Dosyası) ve Source File(Kaynak Dosyası) Oluşturma

* Üstteki örneklerdeki gibi her bir program için base adresleri tekrar tekrar oluşturmak yerine bir header file ve gerekli ise buna bağlı source file dosyaları oluşturmak daha mantıklı olur. STM32 mikro denetleyicilerinde kullanılan “HAL” kütüphaneleri buna bir örnektir. Şimdi bu kütüphaneler nasıl oluşturulur onunla ilgili örnek kodları aşağıda göstereceğim.
* Şimdi base adreslerin nasıl oluşturduğum ile ilgili örneği aşağıda ele alacağım.
* Öncelikle AHBX ve APBX için Base adresleri aşağıdaki şekil 15’deki gibi tanımlayalım.



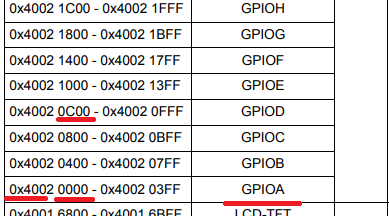
Şekil

* Adresler ise o BUS’a ait başlangıç adresidir. Örnek olarak aşağıdaki şekil 16’da APB2 BUS’ının başlangıç konumu belirtilmiştir.



Şekil

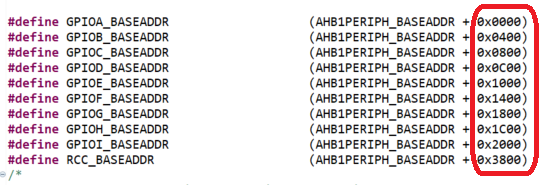
* Şekilde görüldüğü gibi APB2’nin başlangıç adresi 0x4001 0000 olarak görülmektedir. Aynı tabloda eğer AHB1’in başlangıç adresine bakarsak, adresin 0x4002 0000 olduğunu görürüz. Şekil 16’daki tablo Referance Manual Dokümanının Memory Map bölümünde bulunmaktadır.
* Bu BUS’ların içindeki adreslere gelecek olursak onları da şu şekilde buluruz.
* Örneğin AHB1’in başlangıç adresi 0x4002 0000’dir ve bu da GPIOA + OFFSET’e denktir veya GPIOD’nin adresi AHB1 + OFFSET’dir. Bu ofsetleri aşağıdaki şekil 17’de gösterildiği gibi adresin ikinci kısmıdır ve kodlamada bunu kullanacağız.



OFFSET

Şekil

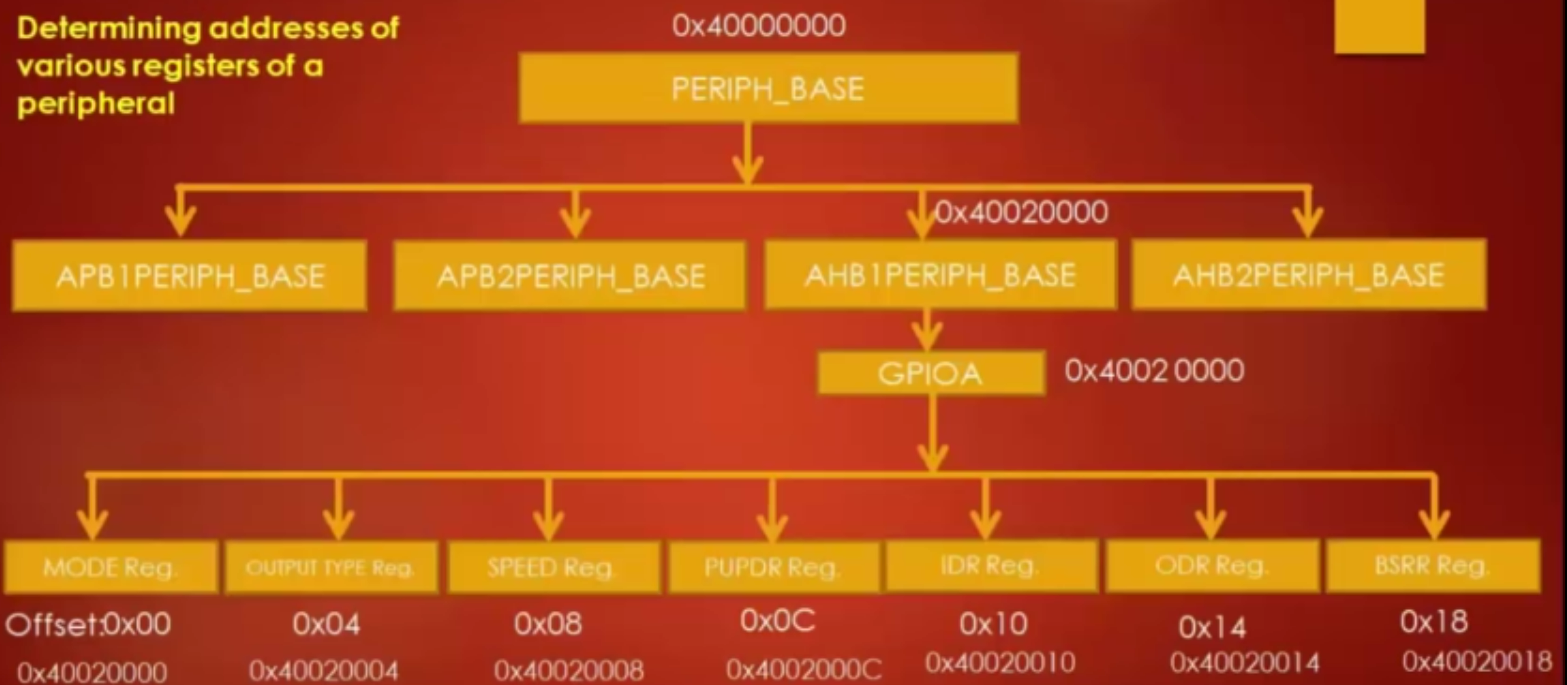
* Kodlaması aşağıdaki şekil 18’deki gibi Memory Map bölümünde ki “Table 1. STM32F4xx register boundary addresses” den referansla alınan başlangıç adresleri ve ofsetlere dayanarak yapılmıştır.



OFFSET

Şekil

* Diğer adresleri yukarıdaki örneklerde faydalanarak yapabiliriz.

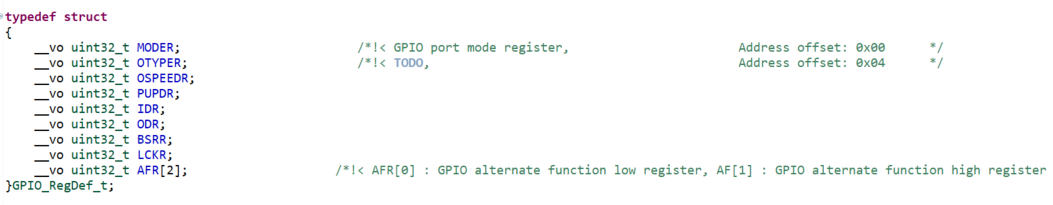


2

1

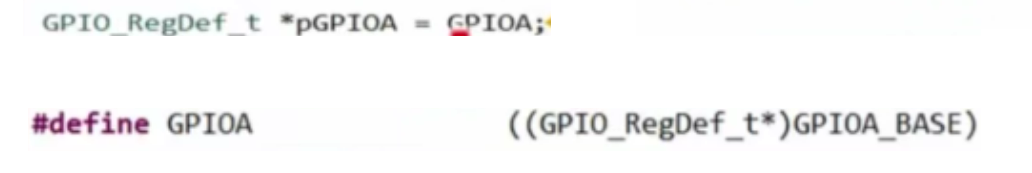
Şekil

* Yukarıdaki şekil 19’da 1 numaralı kısma kadar olan yerde örnek olarak öncesinde yaptığımız işlemleri anlayabiliriz. Yani AHB1 adresinin 0x4002 0000 olduğu ve GPIOA’nın adresinin 0x4002 0000 olduğunu görebiliriz. Fakat hala 2 numaralı kısım için teker teker Makrolar gerekli. Bu çok sıkıcı ve uzun bir süreç olur. Bunun için yapmamız gerekenleri aşağıdaki şekilde ve maddelerde belirteceğim.



Şekil

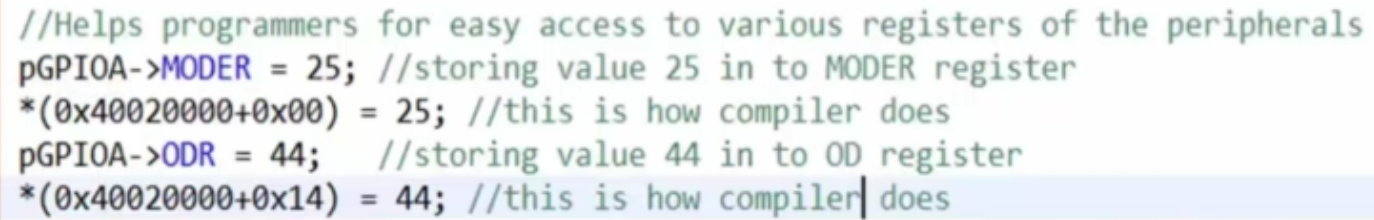
* Bu değerleri belirlemek ve saklamak için en verimli yöntem yukarıdaki şekil 20’deki gibi bir STRUCT yapısı oluşturmaktır. Buradaki ilk üye bir değişkendir. “uint32\_t” veri tipinde bir değişken ve GPIOx için verileri t tutan bir değişken. I2C, SPI, UART vb. içinde oluşturmamız gereken bezer yapılar vardır.



1

Şekil

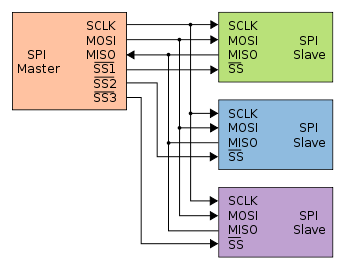
* Bu yapıyı oluşturduktan sonra, bunun için bir işaretçi değişkeni oluşturmalıyız(şekil 21 1 numara). Örneğin GPIOA kayıtlarını kullanmak ve yapılandırmak istiyorum. Yukarıdaki şekildeki gibi GPIOA için bir adres tanımlayıp bu adresi bu yapıda kullanabiliriz.



Şekil

* Yukarıdaki şekilde yaptığımız şey, o işaretçiye atıfta bulunup, Moder kaydını GPIOA moder için gerekli olan adresi belirtmektir.
* Geriye kalan birimler için makroları da bu şekilde yazabiliriz.

# SPI(SERIAL PERIPHERAL INTERFACE)

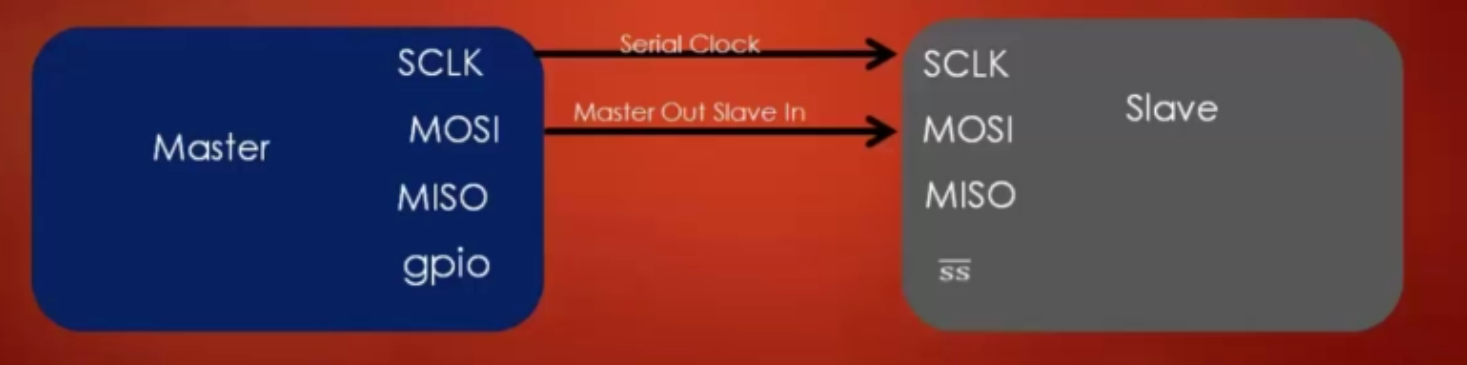


Şekil

* SPI bir seri haberleşme protokolüdür. SPI master ve slave cihazları arasında iletiş kurmak için kullanılır ve sadece bir adet master olabilir. Haberleşme için en az 4 hatta ihtiyaç duyar(şekil 23). Bunlar:
* SLCK
* MOSI
* MISO
* SS1-SS2…
* SLCK: Seri haberleşme için saat(clock) sinyali üretir. Bu sinyal sadece master tarafından üretilir.
* MOSI: Anlamı “MASTER OUTPUT SLAVE INPUT”. Master’ın mesaj gönderdiği ve slave’in bu mesajı aldığı hattır.
* MISO: Anlamı “MASTER INPUT SLAVE OUTPUT”. Slave’in mesaj gönderdiği ve Master’ın bu mesajı aldığı hattır.
* SS: Slave seçiminin yapıldığı hattır. Seçilen Slave için o hat LOW konuma getirilir.

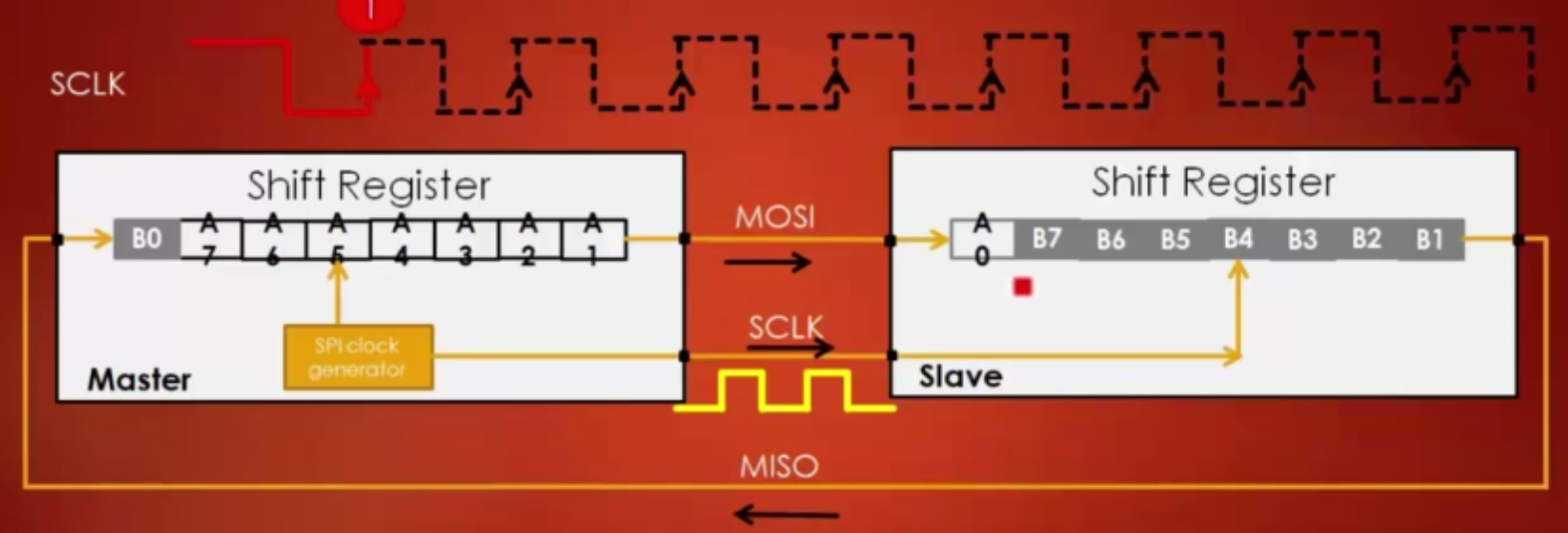
## Minimal SPI BUS

SPI haberleşmede bir master cihaz ile birden fazla slave cihaz arasında iletişim olabilir. Fakat bir master ve bir slave kullanılan durumlarda sadece iki hat aşağıdaki şekil 24’deki gibi yeterli olabilir.



Şekil

* Bu hatlardan bir tanesi SCLK bir diğeri ise MOSI veya MISO olabilir. SS hattı kullanmamızın nedeni hali hazırda birden fazla seçim şansımız olmaması ve SLAVE SS pininin doğrudan LOW’a çekilip iletişime hazır hale getirilebilmesidir.

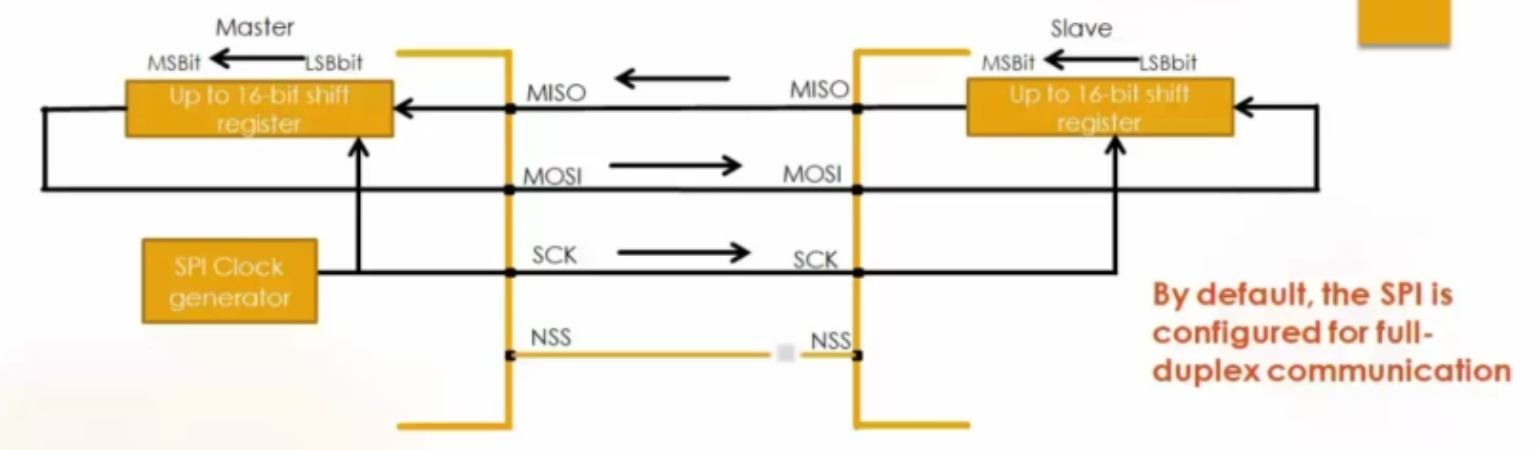


Şekil

* Yukarıdaki şekil 25’de anlatılmak istenen, master ve slave 8 bitlik shift registera sahiptir. Bu 8 bit gönderilen ve alınan mesajdır. Her bir saat çevriminde bir bit transfer edilir. Şekilde görüldüğü gibi ilk saat çevriminde A0 biti slalve kaydırılmış ve B0 biti ise master’a kaydırılmıştır. Yani eğer MISO hattı bağlıysa her şekilde veri alamaya devam eder.

## SPI FULL-DUPLEX, HALF DUPLEX & SIMPLEX

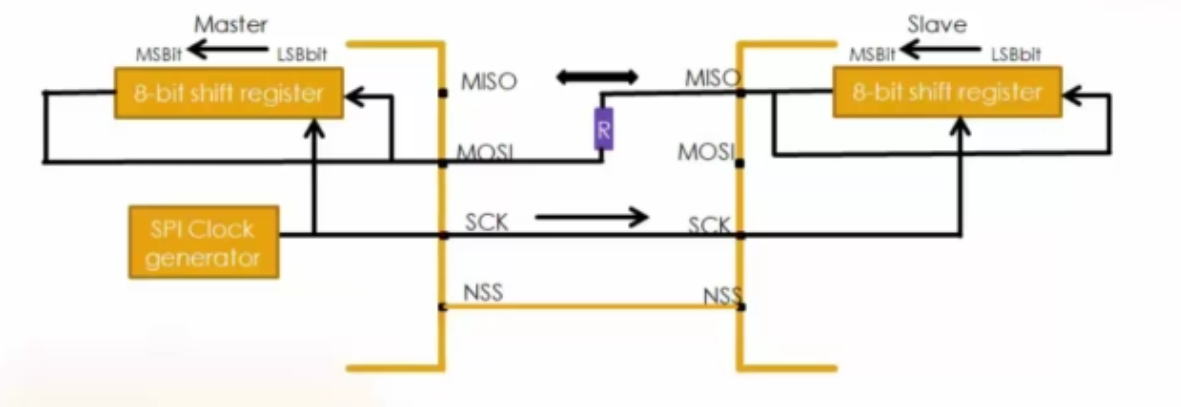
FULL-DUBLEX



Şekil

* Yukarıdaki şekil 26’da görüldüğü gibi Full-Duplex iletişim iki yönlüdür ve iki ayrı hat kullanılır yani hem veri alınır hem de veri gönderili ve bütün hatlar kullanılır. Mikro denetleyicilerimizde SPI haberleşme Full-Duplex olarak ayarlanmıştır.

HALF-DUPLEX



Şekil

* Half-Duplex konfigürasyonda veri iletimi tek bir çapraz bağlantı şekil 27’deki gibi kullanılır. Yani Half-Dublex iletişimde veri alıp göndermek için tek bir hat kullanılır. Genelde çoğu mikro denetleyici Half-Duplex iletişimi destekler, fakat bazı mikro denetleyiciler bunu desteklemeyebilir. Bunu öğrenmenin en iyi yolu ilgili referance manual dokümanını kontrol etmektir.

## SIMPLEX



Şekil

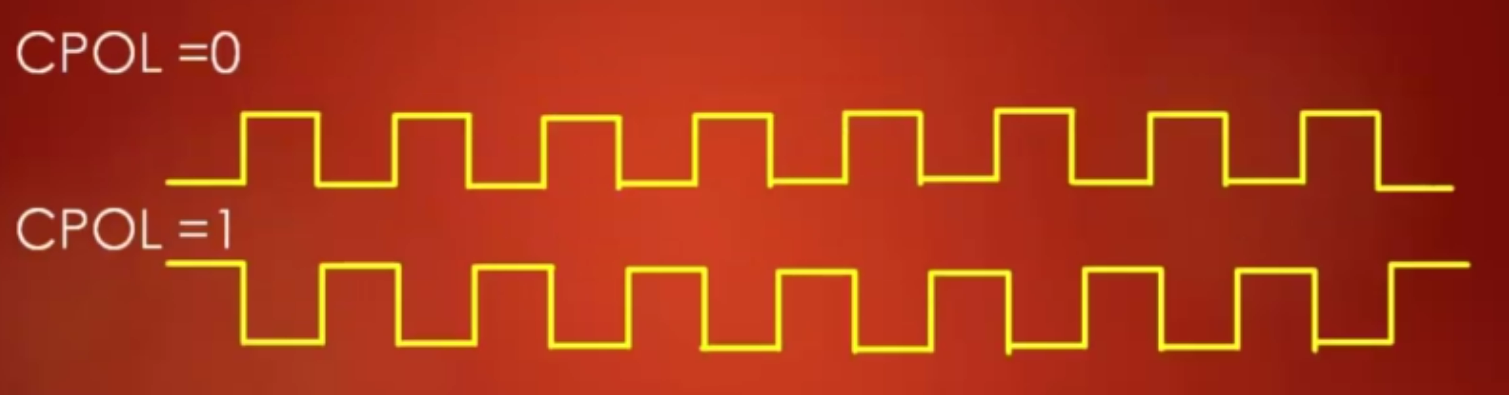
* Simplex iletişim tek yönlüdür ve diğer türe göre daha basit yapısı vardır(şekil 28). Simplex iletişimdi iki tür vardır. Bunlar, transmit-only(sadece veri iletimi) ve receive-only(sadece veri alma).

## SPI İletişim Formatları(SPI Communication Formats)

* SPI için 3 adet iletişim formatı vardır. Bunlar, SCLK PHASE(CPHA), SCLK POLARITY(CPOL) ve Data Frame Format(DFF).

### CPOL

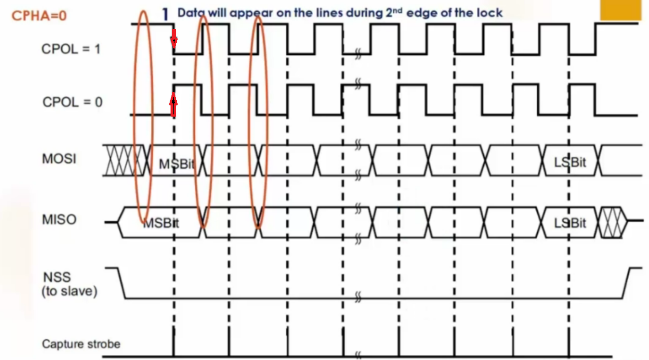
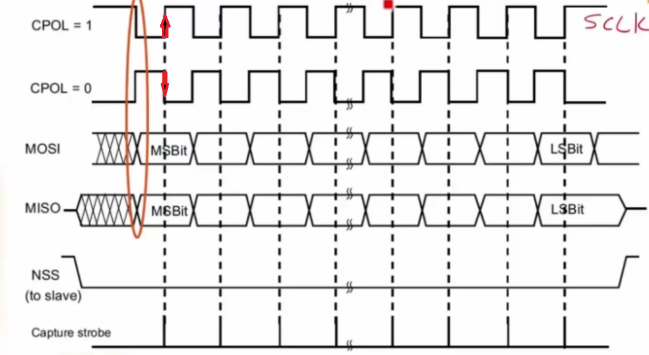
CPOL RESET durumdayken, SLCK hattı iletişim yok iken LOW durumdadır. CPOL SET durumdayken, SCLK hattı iletişim yok iken HIGH durumları. Aşağıda şekil 29’da CPOL SET ve RESET durumları gösterilmiştir.



Şekil

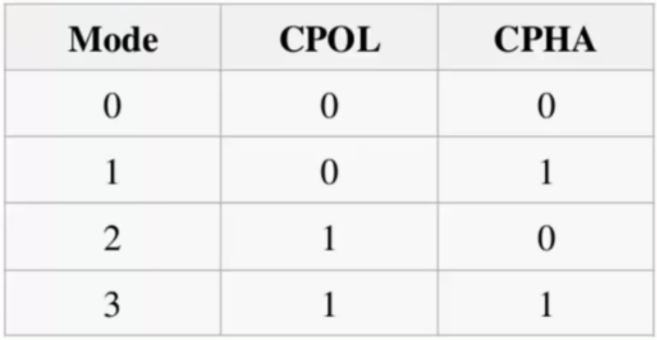
### CPHA

CPHA SCLK’nın hangi kenarında verini okunacağına kara verir. CPHA’yı ve CPOL’ü anlamak için aşağıdaki şeklileri(şekil 30-31) inceleyebiliriz.



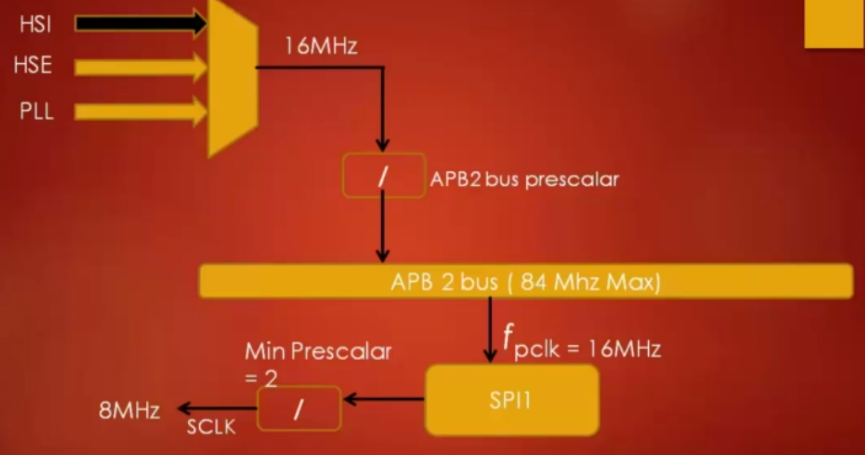
Şekil Şekil

* CPHA ve CPOL kombinasyonlarına dayanarak 4 farklı SPI moduna sahibiz. Bunlar aşağıda şekil 32’de gösterilmiştir.



Şekil

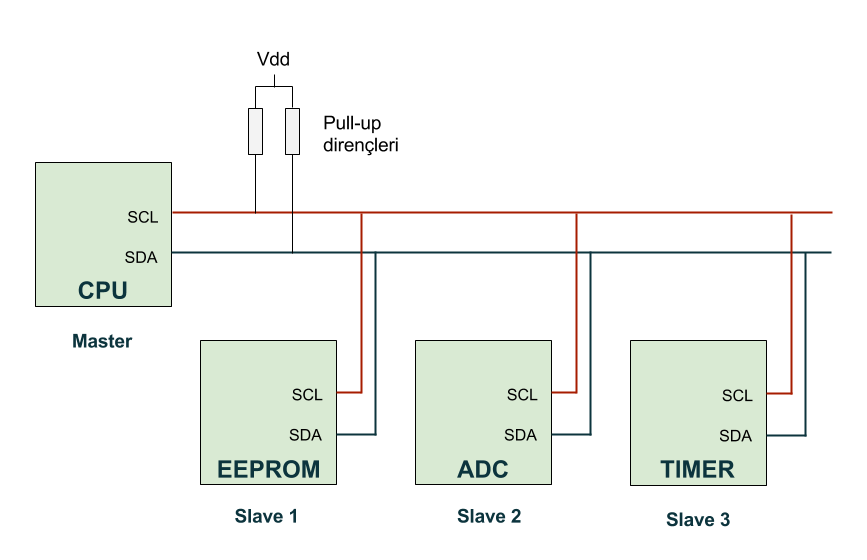
* Eğer internal RC osilatör kullanıyorak SP1, SPI2, SPI3 için maksimum clock 8MHZ olur çünkü HSI’nın maksimum hızı 16MHZ’dir. Bu hızın hesaplanışı aşağıda şekil 33’te verilmiştir.



Şekil

* APB2 veri yolunun maksimum ulaşabileceği hız STM32F4XX için 84MHZ’dir yani yukarıdaki hesaplamayı dikkate alırsak SPI 42 MHZ’ye kadar saat(clock) sinyali üretebilir.

# I2C

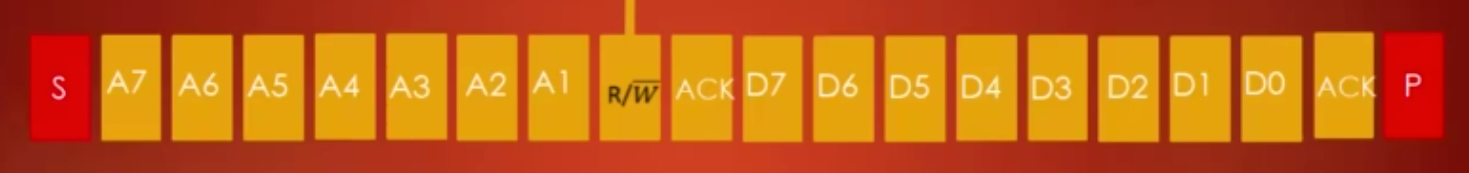


Şekil

I2C bir seri haberleşme protokolüdür. Haberleşme için yalnızca 2 adet hat kullanılır. Veri iletimi için SDA ve saat(clock) için SCL hattı kullanılır. I2C protokolünün bağlantısı temel olarak yukarıdaki şekilde ki gibidir. Şimdi madde madde I2C haberleşme protokolünün detaylarını sıralayacağım. Şekil 34’de I2C ile ilgili örnek şema görülmekte.

* SDA ve SCL hatları boş durumda iken HIGH konumdadır.
* Veri aktarımı her zaman SDA hattından master tarafından başlatılır(start condition).
* Start condition’dan sonra adres aşaması başlar. Aşağıda şekil 35’de görüldüğü gibi toplam 8 bittir.

Şekil



P(stop)

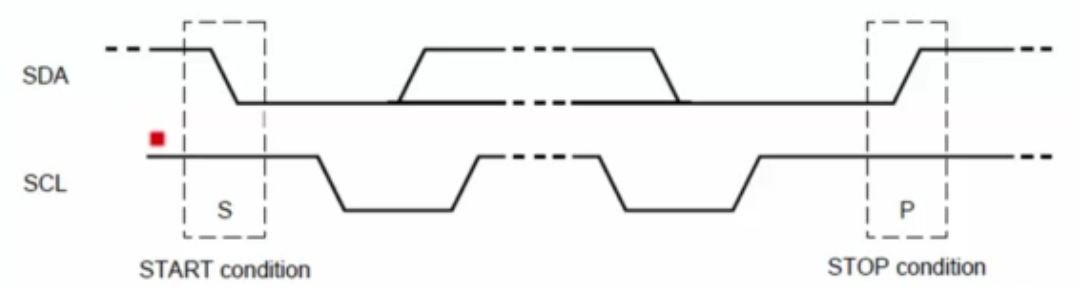
Veri baytı D7…D0

Slave Adresi A7…A1

Okuma-yazma işlemlerine karar verir

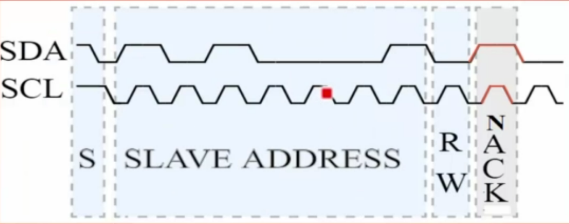
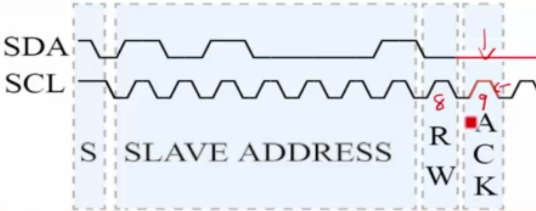
Start Conditions

* Her baytın ardından onaylama biti gelmektedir.(ACK-NACK)
* ACK bitinden sonra veri baytları gönderilir ve birden fazla bayt gönderilebilir. P(stop) biti ile master tarafından durdurulur.



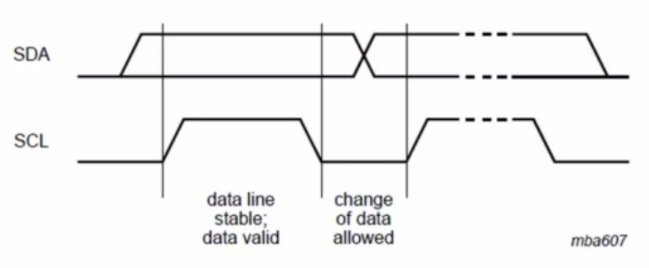
Şekil

* SCL ve SDA “HIGH” durumdayken, SDA “LOW” duruma gelirse “START condition” durumu oluşur(şekil 36).
* SCL “HIGH” durumdayken, SDA “LOW” konumdan “HIGH” duruma geçerse “STOP condition” durumu oluşur(şekil 36).



Şekil

* 9. Bitte SDA low ve SCL High ise ACK durumu oluşur(şekil 37).
* 9. Bitte SDA High ve SCL High ise NACK durum oluşur(şekil 37).



Şekil

* Data Validity: I2C’de SCL High durumdayken veriler değişmemelidir(veriler stabil olmalıdır). Örnek olarak şekil 38. Örnek şekil X. Start ve Stop condition için geçerli değildir.
* I2C\_CR2 ve I2C\_CCR registerları I2C serial clock ayarlarını kontrol etmek ve diğer zamanlama ayaları için kullanılır.
* Clock Stretching: Slave veri almaya veya göndermeye hazır olmadığında, master clock sinyalini low tutar(ACK olana kadar). Bu özelliği kullanmak için aktif edilmesi gereklidir.

# GIT

## **GIT NEDİR?**

Git temelde bir versiyon kontrol sistemidir. Yani proje üzerinde yaptığımız değişiklikleri adım adım kayıt eder ve gerekirse bunu internet üzerinde depoda (repository(GitHub, GitLab gibi)) saklamamızı ve yönetmemizi sağlar. Açık kaynak kodludur. Değişikliklerin sadece son halini değil yapılan bütün değişiklikleri saklar bu sayede projenin herhangi bir kısmında bir hata yapılırsa istenilen aşamaya geri dönülebilir. Kaydedilen her klon(örneğin farklı bilgisayarlarda ve GitHub gibi repostory’lerde) tüm verilerin yedeğidir yani herhangi bir yedekte sorun çıkarsa proje geri yüklenebilir ve kurtarılabilir. Ayrıca bir proje üstünde birden fazla kişi çalışabilir ve bu kişiler proje üzerindeki değişiklikleri sorun çıkmadan kontrol edebilir. Proje farklı dallara(branch) ayrılabilir ve daha sonradan birleştirilebilir. Bu sayede birden fazla kişi aynı projenin farklı aşamaları yapabilir ve sonunda sorun çıkmadan birleştirebilir. Git’in bence en önemli özelliklerinden biri budur.

## GIT KURULUMU(WİNDOWS)

<https://git-scm.com/downloads> adresinden “windows” için olan seçenekten kurulum dosyası indirilir. Kurulum ayarlarında varsayılan seçenekler kullanılarak kurulum tamamlanır. Kurulum sonucu 3 program görülür. **Git CMD, Git BASH** ve **Git GUI.**

## GIT CMD

Git CMD, standart Windows komut istemi gibidir. Git özelliklerini komut satırı üzerinden kullanmamızı sağlar. Git CMD Windows tabanlıdır.

## GIT BASH

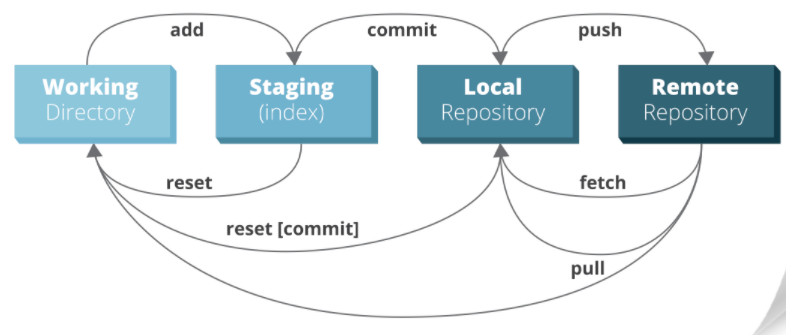
Git BASH tıpkı Git CMD gibi komut satırı üzerinden Git’i kullanmamızı sağlar. Kullanılan komutlar LUNIX tabanlıdır.

## GIT GUI

Git GUI(Graphical user interface) bize grafiksel bir ara yüzden çalışma imkânı sağlar.

* Git CMD, Git BASH ve Git GUI temelde aynı şeyi yapar yani Git’i kullanmamızı sağlar. Hangisini kullanılacağı kullanıcın tercihine bağlıdır. Ben bundan sonra Git BASH üzerinden devam edeceğim.

## Git Veri Akışı



Şekil

Git projelerimiz kayıt edilirken yukarıdaki şekil 39’da verilen veri akışını kullanır.

Working Directory: Çalışmakta olduğumuz klasördür. Yapılan değişiklikler yerelde tutulur.

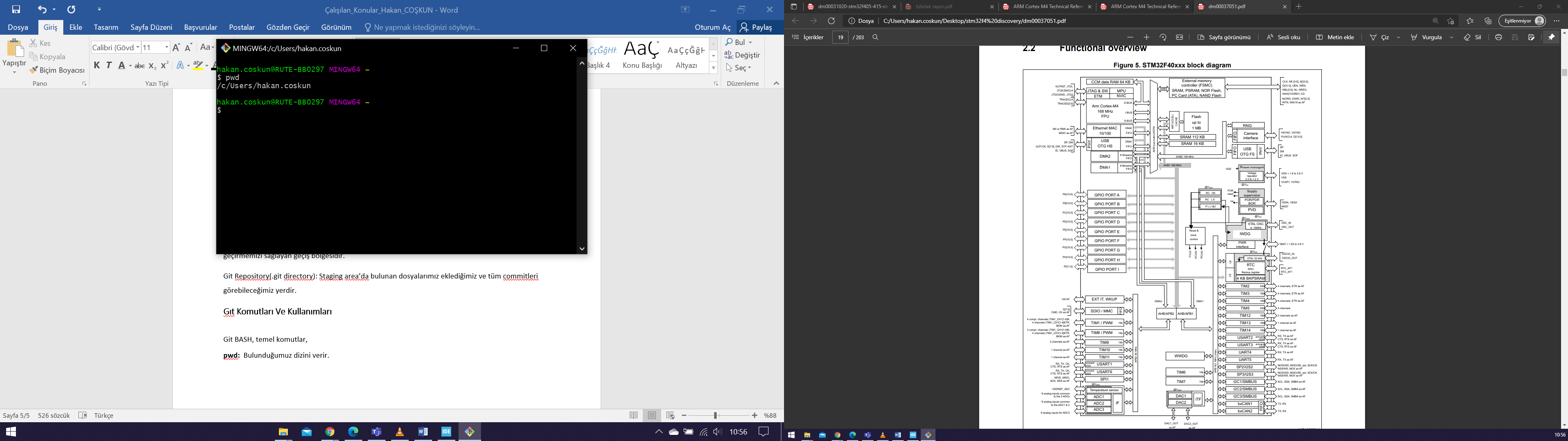
Staging Area: Projedeki dosyalarımızın working directoryden sonraki adresidir. Projedeki değişiklikleri Repository’ye göndermeden önce kayıt altında tutuğumuz ve yaptığımız değişiklikleri gözden geçirmemizi sağlayan geçiş bölgesidir.

Git Repository(.git directory): Staging area’da bulunan dosyalarımız eklediğimiz ve tüm commitleri görebileceğimiz yerdir. Gıt Komutları Ve Kullanımları

## Git Komutları

Git BASH, temel komutlar

pwd**:** Bulunduğumuz dizini verir.(şekil 40)



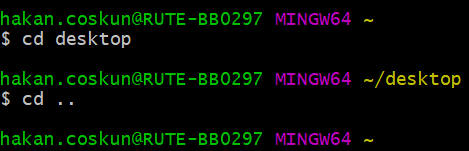
Şekil

ls**:** Bulunduğumuz klasördeki dizinleri listeler.(şekil 41)



Şekil

cd**:** Bulunduğumuz dizini değiştirir. Ayrıca “cd ..” komutu ile bir önceki klasöre dönülebilir. (şekil 42)

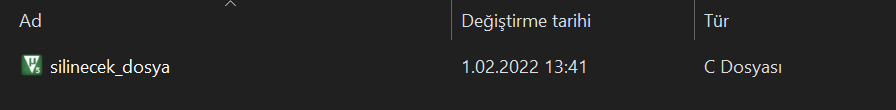


Şekil

touch DosyaAdı: Bulunan dizine dosya oluşturur.(şekil 43-44)



Şekil



Şekil

git rm: Belirtilen dosyayı siler. Sonuna “-r” konulursa belirtilen klasörü ve içindekileri siler. .(şekil 45 )



Şekil

git mv: Dosyanın yerini ve ismini değiştirmeyi sağlar. İsim değiştirmek için “*git rm dosyaadı yenidosya adı*” şeklinde komut yazılır. Dosyayı taşımak içinde benzer olarak *“git rm dosyaadı taşınacakyer/”* şeklinde yazılır.(şekil 46-47 )



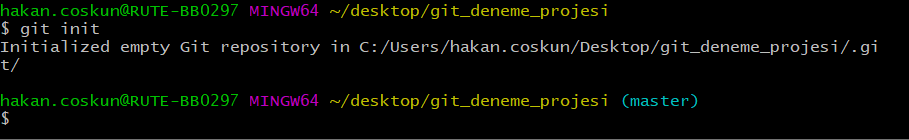
Şekil



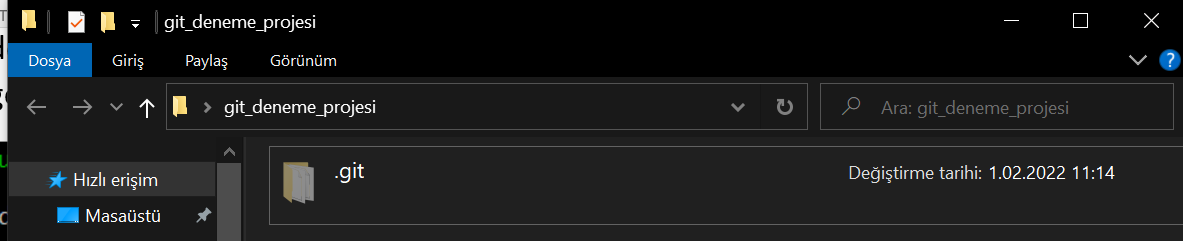
Şekil

Git BASH repository oluşturma,

git init: klasörün git projesi olduğunu belirtir. Klasörde .git adlı bir dosya oluşturur ve değişiklikler bunun içinde tutulur. Bu klasörü görmek için görünümden gizli dosyaları gösterme seçeneğini seçmemiz gerekir.(şekil 48 )



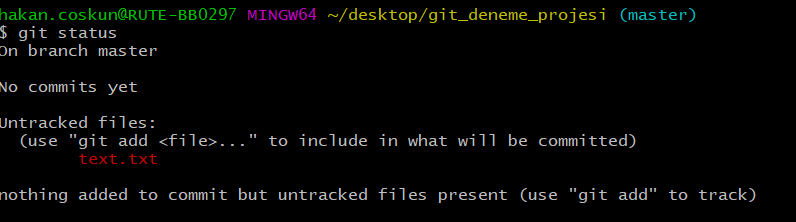
Şekil



Şekil

Git BASH dosyalarda değişiklik yapma ve değişiklikleri izleme,

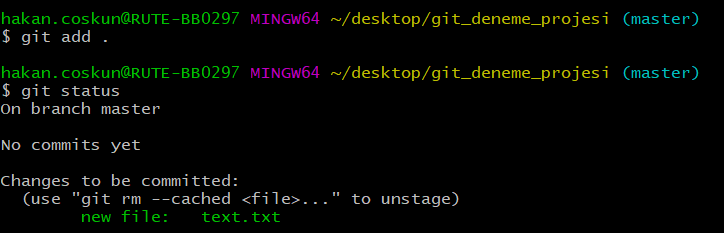
git status: dizindeki farklılıkları gösterir. (şekil 50)



Şekil

Yukarıdaki şekilde görüldüğü gibi yeni eklediğim text.txt dosyasının commit edilmediğini ve izlenmediğini söylüyor. Aşağıda bahsedilen komutlar ile bunu düzelteceğiz.

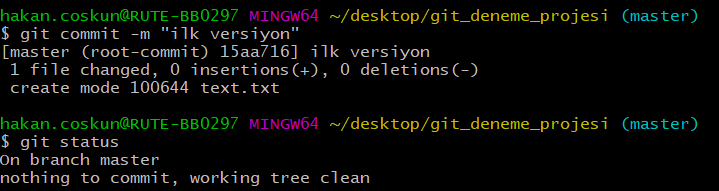
git add **. :** Dizindeki dosyaların hepsini git deposuna ekler. Komutun sonundaki nokta “.” dizindeki dosyaların hepsini kasteder. Eğer dosyaların hepsini değil de belirli bir tanesini eklenmesini istiyorsak nokta yerine dosya adı yazılmalı .(şekil 51)



Şekil

Yukarıdaki şekilde görüldüğü gibi text.txt dosyamız new file olarak tanımlandı. Fakat hala commit edilmemiş gözüküyor.

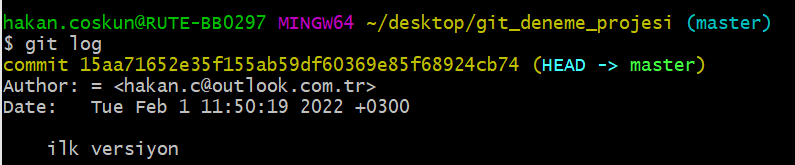
git commit –m “commit massage”: Geçiş bölgesindeki dosyaları(text.txt gibi) git deposuna ekler yani commit işlemini yapar. Ayrıca commit mesajı eklenir. Bu sayede yaptığımız değişiklikleri belirleyebiliriz. (şekil 52)



Şekil

Yukarıdaki şekilde görüldüğü gibi git status komutunu yazdıktan sonra herhangi bir değişiklik veya commit edilecek dosya göstermiyor.

git log**:** Commit –m “commit massage” komutu ile aldığımız versiyonları gösterir.(şekil 53)

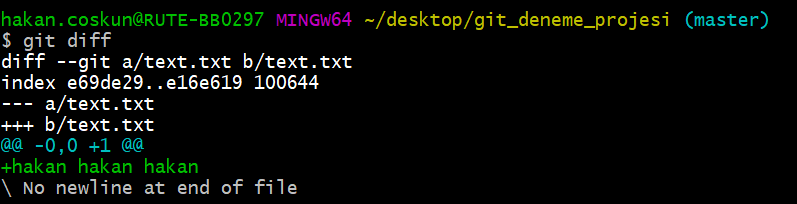


Versiyonun ID

Versiyonun adı

Şekil

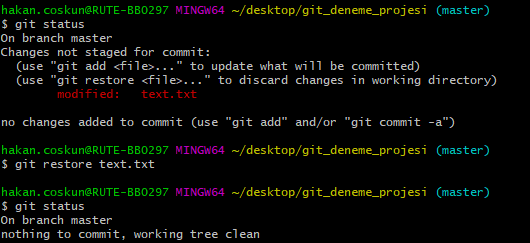
git diff**:** Yapılan değişiklikleri gösterir. Git diff commit1 commit2 komutu ise iki commit arasındaki farkı gösterir.(şekil 54)



Yapılan değişiklik

Şekil

git restore <file name> : yapılan değişikliği commit etmeden önce geri alır. (şekil 55)

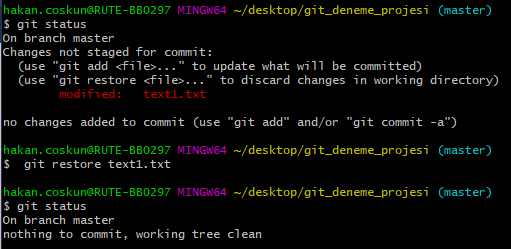


Değişiklik yapılan dosya

Değişikliğin geri alınması

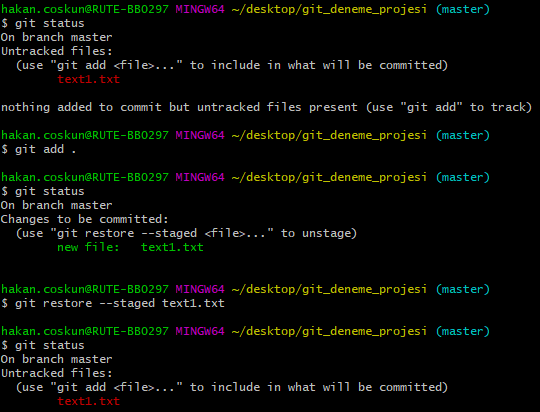
Şekil

git checkout / git restore DosyaAdı**:** Yapılan değişikliği çalışma dizinindeyken geri alır. **git checkout *ID***şeklinde yazıldığı zaman, “**git log**” komutu ile gördüğümüz değişikliklerden ID numarasını yazdığımıza dönebiliriz. (şekil 56)



Şekil

git reset / git restore --staged <file>...**:** “git add . ” komutu ile geçiş bölgesine aktarılmış değişikliği aşağıdaki şekildeki gibi geri alır. (şekil 57)



Şekil

GitHub ve GitLab Komutları,

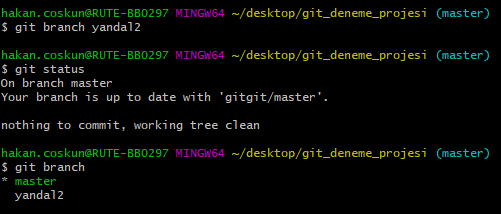
git remote add BağlantıAdı GitHubLinki: GitHub bağlantısı ekleme komutu. Yalnız “git remote” komutu yazılırsa bize bağlantı adı verir. Bağlantı adı kullanıcı tarafından belirlenir.

git push –u bağlantıadı master: Yerel git dosyalarını uzak repository gönderir. “-u” anlamı bütün doyalar demek.

git pull : GitHub’daki dosyaları çeker(yerel dosylara).

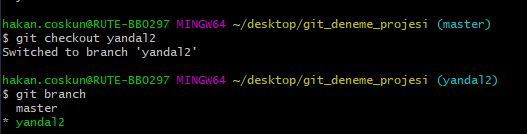
cat >> .gitignore: Proje dizinine .gitignore adında gizli bir dosya yaratır. Bu dosyanın içine yazılan dosyalar git tarafından takip edilmez. Git tarafından takip edilmesini istemediğimiz dosyalar için kullanılır.

git branch DosyaAdı: Normalde projemizin ana dalı master dalıdır. Masterdan kopya aldığımız zaman yeni bir dal oluşturmuş oluruz. Bunun için “git branch DosyaAdı” komutunu kullanırız. Sadece git branch komutunu yazarsak şekildeki gibi bize hangi dalların olduğunu ve hangisinde olduğumuzu belirtir. (şekil58)



Şekil

git checkout “DalAdı” *:* Bu komut ile sahip olduğumuz dallar arasında geçiş yapabiliriz. Örnek aşağıdaki şekilde görüldüğü gibi önce git checkout “DalADı” komutunu kullanıyorum sondasında ise git branch komutu ile bulunduğumuz dalın değiştiğini görebiliriz. (şekil 59 )



Şekil

git merge: dalları birleştirmek için kullanılır. Eğer master dalındaysak ve örneğin yandal2 ile birleştirmek istiyorsak yandal2 şeklinde yazmamız yeterli.

git reset: İzlenen dosyaları izlemeden çıkartır

git reset head “DosyaAdı”**:** Belirli bir dosyayı çıkartmak için kullanılır.

git clean –f**:** Bu komut untracked dosyaları silmemizi sağlar.

--dryrun-run: komutu çalıştırmadan önce komutu simüle eder. Örneğin aşağıdaki gibi “git clean –dry-run” komutunu yazarsak bize silinecek dosyaları gösterir. –dry-run yerine sadece –n’ de yazmamız yeterlidir.