



Enchipsdatorer – Laboration 0 Hello Blinky!

Enchipsdatorer – Laboration 0 Hello Blinky!

Målsättning

- Bli bekant med utvecklingsmiljön STM32CubeIDE från ST Microelectronics.
- Använda Github för versionshantering.
- Titta på hur CubeMX-delen av CubeIDE fungerar.
- Lära sig skapa en seriell anslutning med Putty.
- Lära sig starta en debug-session i CubeIDE.
- Få dioder att blinka.

Förberedelse

Läst på det som föreläsning 1 tar upp och som finns i föreläsningsanteckningarna. Det rekommenderas att man läst kursboken enligt läsanvisningarna som finns i slutet på föreläsningsanteckningarna.

Laddat ner och installerat CubeIDE.

Laddat ner och installerat Github Desktop.

Laddat ner och installerat Putty.

Skrivit ut JU_STM32_HAL-beskrivning.pdf

Examination

Ingen examination för denna laboration.

Genomförande

Tanken är att du skall bli väl förtrogen med utvecklingsmiljön i CubeIDE och att man vet hur man kan använda felsöknings- (debug-) verktygen för att kontrollera sin kod. Du ska även börja använda ett versionshanteringssystem för att spara sina projekt på ett systematiskt sätt. Laborationen har 4 timmar handledd tid, men kan ta mer tid att genomföra, vilket då görs på egen hand.

Hårdvara

- 1x STM32 F411RE-kort
- 1x USB-kabel med USB-A kontakt
- 1x kopplingsdäck
- 1x lysdiod med matchande motstånd för 3.3V

Enchipsdatorer – Laboration 0 Hello Blinky!

1. Versionshantering med Github

1.1. Github setup

Börja med att skapa ett användarkonto på github.com (om du inte redan har ett) Som student kan du använda ditt konto på samma sätt som fullt betalande kunder.

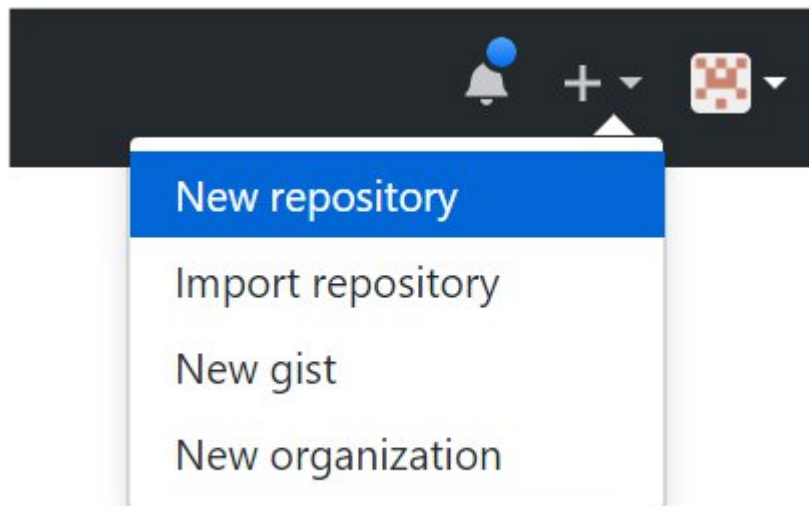
Gå till <https://education.github.github.com/pack> och ansök om "student pack" för github. För att få kontot måste du använda din JU-mail, för att visa att du är student.

1.2. Github Desktop

Ett grafiskt verktyg att använda för hantering av git är GitHub Desktop. Hämta och installera programvaran på din dator.

1.3. Github versionshantering

Börja med att skapa ett repository. Detta görs enklast inloggad i github.com. Klicka på "New Repository":



Enchipsdatorer – Laboration 0 Hello Blinky!

Ett nytt fönster dyker upp:

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository.](#)

Owner *

 axeand-ju ▾

Repository name *

/ Enchipsdatorer_VT2022 ✓

Great repository names are short and memorable. Need inspiration? How about [legendary-octo-pancake?](#)

Description (optional)

Labbar för kursen enchipsdatorer, våren 2022

☐



Public

Anyone on the internet can see this repository. You choose who can commit.

☒



Private

You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

☒ Add a README file

This is where you can write a long description for your project. [Learn more.](#)

☐ Add .gitignore

Choose which files not to track from a list of templates. [Learn more.](#)

☐ Choose a license

A license tells others what they can and can't do with your code. [Learn more.](#)

This will set  main as the default branch. Change the default name in your [settings](#).

Create repository

Fyll i repositorynamnet, som ska vara samma som kursen, tex.

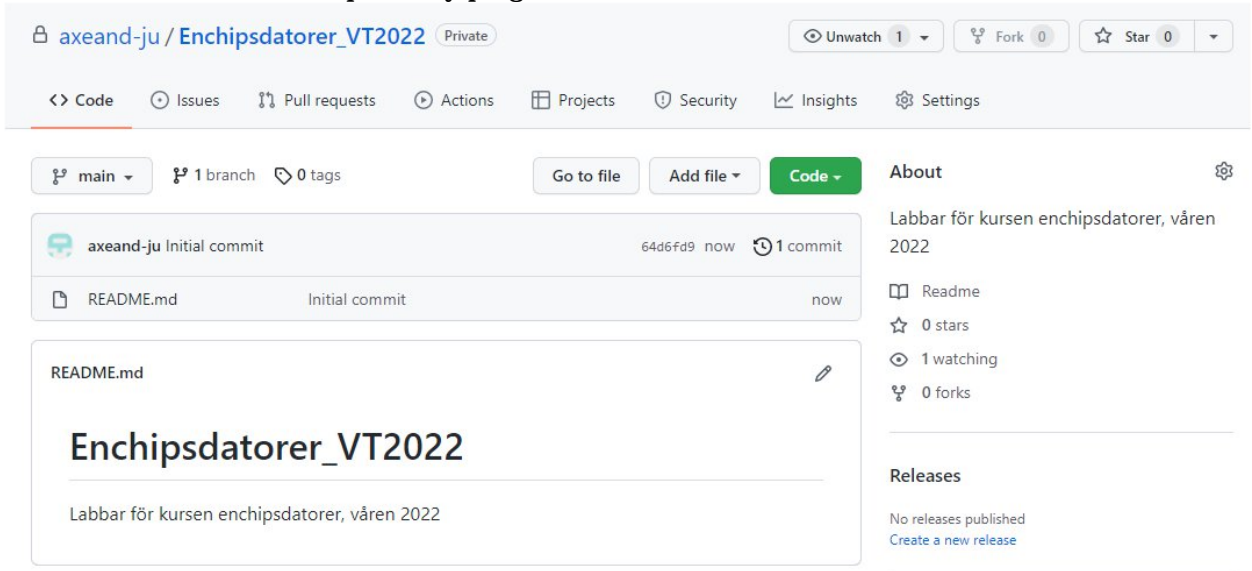
Enchipsdatorer_VT2022.

Fyll också i "description" och klicka i att projektet ska vara "private".

Klicka i att du vill skapa en Readme-fil. (Detta gör du för att katalogen ska innehålla minst en fil när du skapar den.) Klicka sen på "Create repository"

Enchipsdatorer – Laboration 0 Hello Blinky!

Du kan nu se ditt repository på github:



The screenshot shows a GitHub repository page for 'axeand-ju / Enchipsdatorer_VT2022'. The repository is private and has 1 branch (main) and 0 tags. It contains 1 commit (64d6fd9) and a README.md file. The README.md file contains the title 'Enchipsdatorer_VT2022' and the subtitle 'Labbar för kursen enchipsdatorer, våren 2022'. The right sidebar shows the repository's statistics: 0 stars, 1 watching, and 0 forks. The 'Releases' section shows 'No releases published' with a link to 'Create a new release'.

Nu har du skapat ett repository, med ett projekt och behöver kлона det så att det som nu finns i github också finns lokalt på din dator.

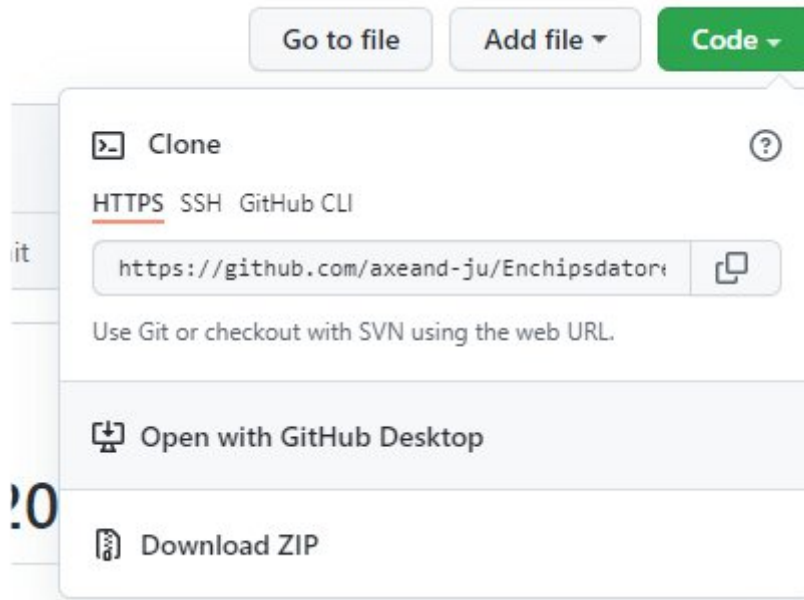
Se till att du har en katalog på datorn där du tänker spara dina filer från denna kurs, t.ex.

G:\Dokument\GitHub

OBS. Som alltid ska man undvika mellanrum, och åäö i namn och pather. Det gäller de flesta verktyg som du använder i de olika kurserna. Speciellt gäller detta för CubeIDE som brukar vägra att kompilera koden och ge väldigt kryptiska felmeddelanden när detta är problemet.

Enchipsdatorer – Laboration 0 Hello Blinky!

Klicka sen på den gröna rutan "Code" och Open with GitHub Desktop.



Om du inte har GitHub Desktop installerat på din dator så kommer du omdirigeras till dess hemsida så att du kan ladda ner och installera det. GitHub Desktop är en grafisk klient för git som gör det enklare att visualisera olika repositories. Väl igång måste du logga in mot GitHub innifrån GitHub Desktop.



Enchipsdatorer – Laboration 0 Hello Blinky!

Du får upp en dialogbox där du ska ange var på din lokala dator du vill lägga projektet. Ange där G:\Dokument\Enchipsdatorer (eller det namn du gav ditt bibliotek ovan)

Det är viktigt att du här ger det katalognamn där du vill lägga dina projekt under kursen!!

Clone a repository

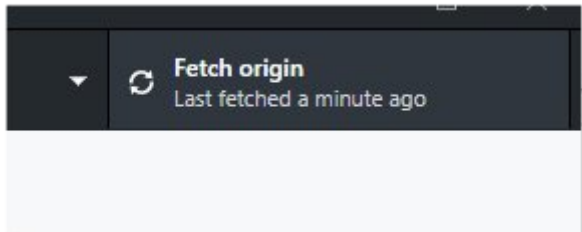
GitHub.com GitHub Enterprise **URL**

Repository URL or GitHub username and repository
(hubot/cool-repo)

Local path

Enchipsdatorer – Laboration 0 Hello Blinky!

Klicka på "Fetch origin" för att hämta senaste versionen av ditt "remote repository":



Du kan nu titta på filerna du fått i din katalog:

GitHub > Enchipsdatorer_VT2022 >		Search Enchipsdatorer_VT2022	
Name	Date modified	Type	Size
.git	2022-03-29 21:17	File folder	
README.md	2022-03-29 21:17	MD File	1 KB

Om du inte kan se .git-katalogen kan du välja under "view" att klicka i "View hidden items"

Som avslutning ska du också kopiera .gitignore-filen som finns i Canvas. Glöm inte att det är filen som ligger inne i zip-filen som ska läggas i projektmappen, inte zip-filen själv.

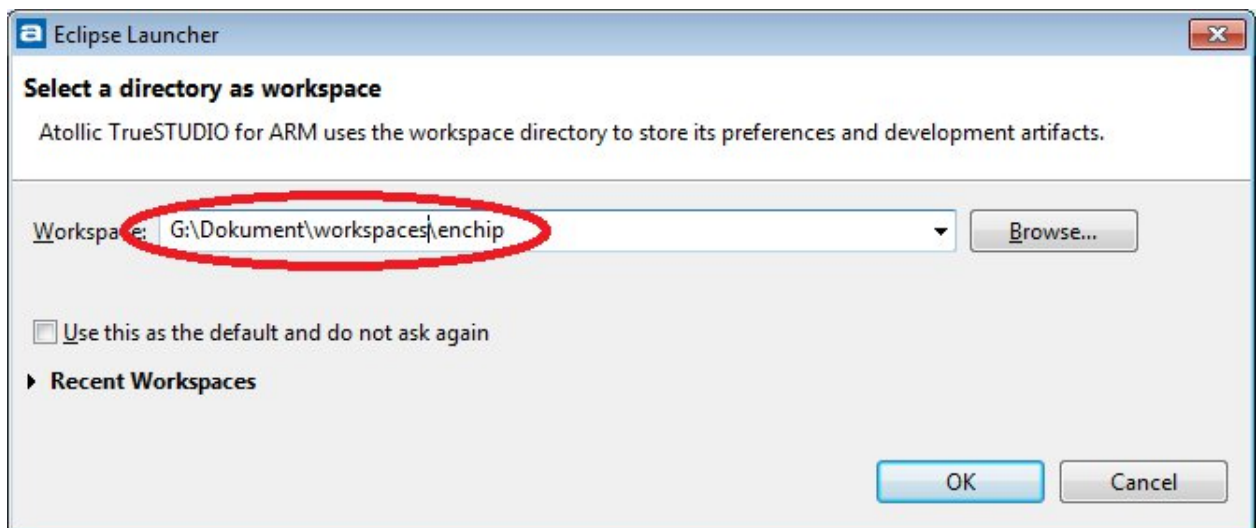
Nu är vi redo att skapa vårt STM32-projekt!

Enchipsdatorer – Laboration 0 Hello Blinky!

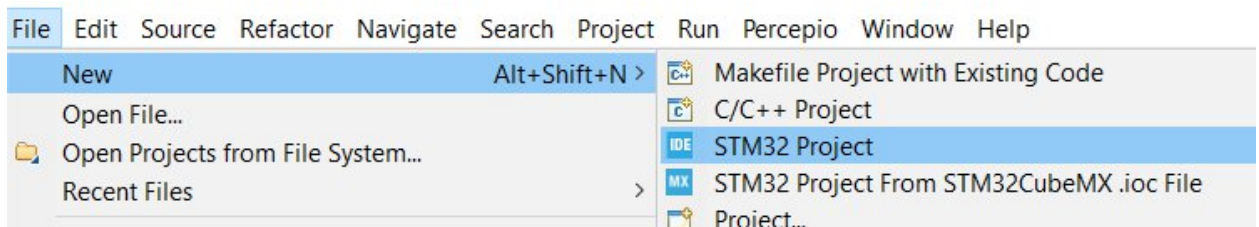
2. STM32CubeIDE

2.1. Starta ett projekt

Starta upp STM32CubeIDE. Det första som kommer upp är att du ska välja "workspace". Ett workspace motsvarar ungefär en solution i Visual Studio. Det rekommenderas att du har ett workspace per kurs. Så skapa ett workspace för den här kursen, så att det matchar var du satte upp ditt git-repo. Och som tidigare nämnt så är det **mycket viktigt** att projektets sökväg inte innehåller några mellanslag eller icke-engelska tecken.



När STM32CubeIDE är rätt inställt efter sin första start, så starta ett nytt projekt. Välj menyalternativet enligt bilden nedan.



Enchipsdatorer – Laboration 0 Hello Blinky!

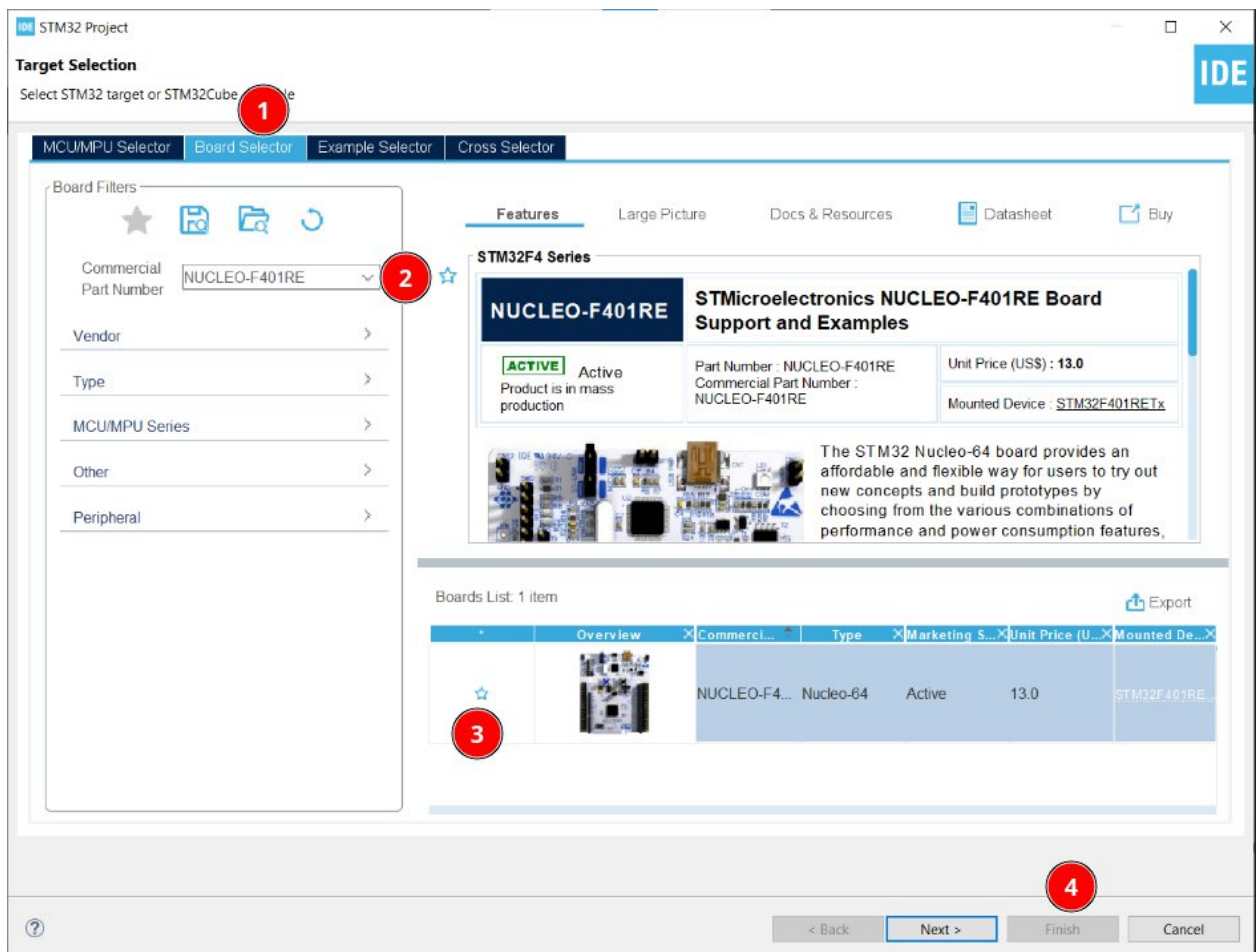
Klicka/fyll i bildens nummerordning.

Vi väljer "Board Selector" (1) eftersom du har ett utvecklingskort utgivet av ST själva. Hade vi tillverkat vårt eget kort så hade vi använt standardfliken.

Skriv in namnet på ditt kort i Commercial Part Number (2). Ditt kort heter "F411RE" (bilden är från 2022 då F401RE användes).

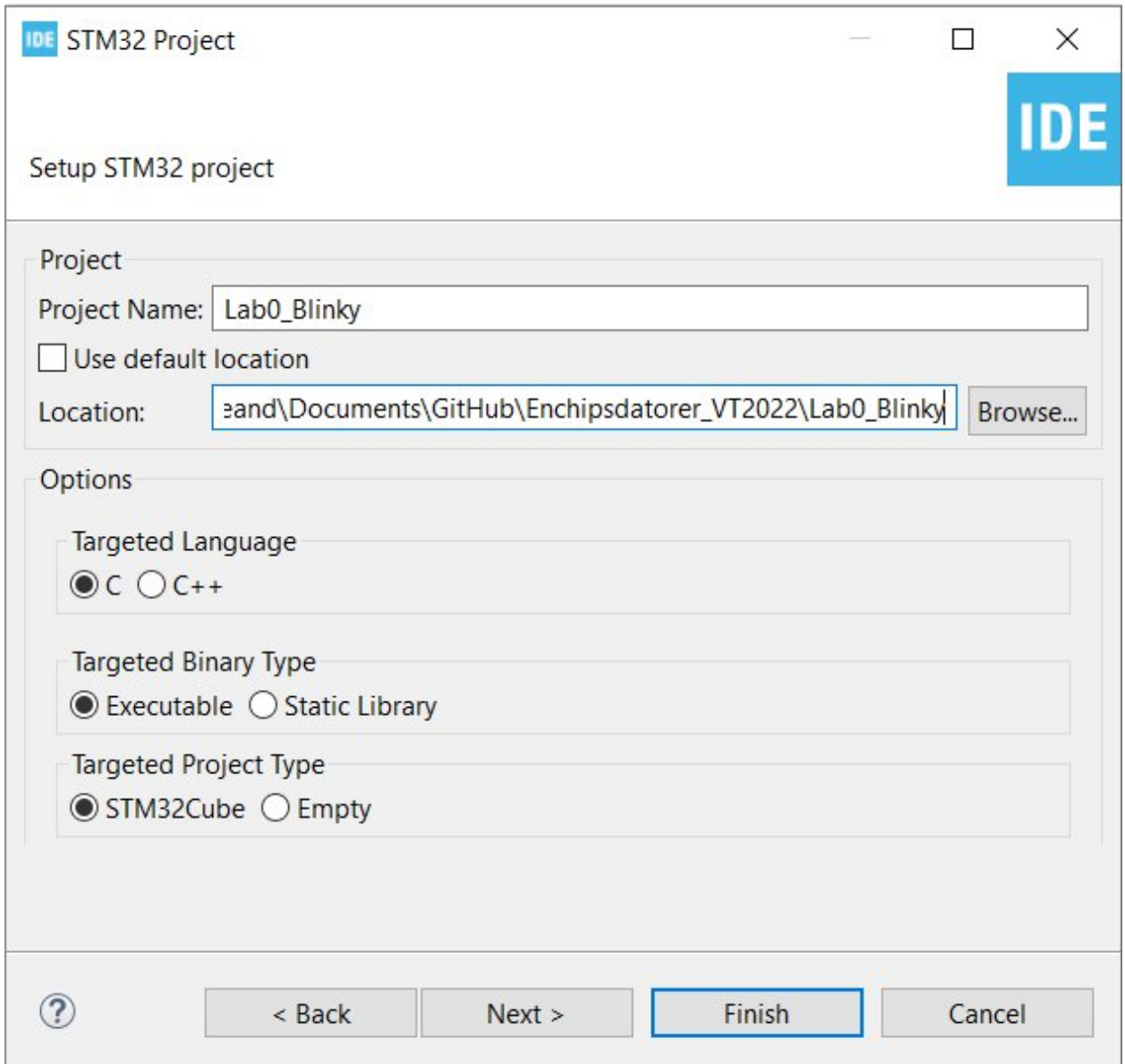
Klickar du i den lilla stjärnan (3) så kommer du snabbare åt ditt kort framöver.

Klicka sedan Next.



Enchipsdatorer – Laboration 0 Hello Blinky!

När denna ruta kommer så tryck på Finish. All programmering i CubeIDE kommer att vara i C, inte C++.



IDE STM32 Project

Setup STM32 project

IDE

Project

Project Name: Lab0_Blinky

☐ Use default location

Location: eand\Documents\GitHub\Enchipsdatorer_VT2022\Lab0_Blinky Browse...

Options

Targeted Language

☒ C ☐ C++

Targeted Binary Type

☒ Executable ☐ Static Library

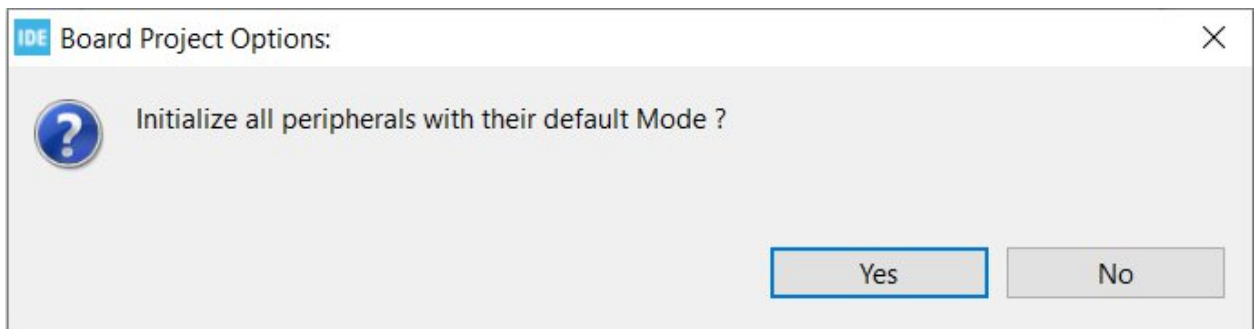
Targeted Project Type

☒ STM32Cube ☐ Empty

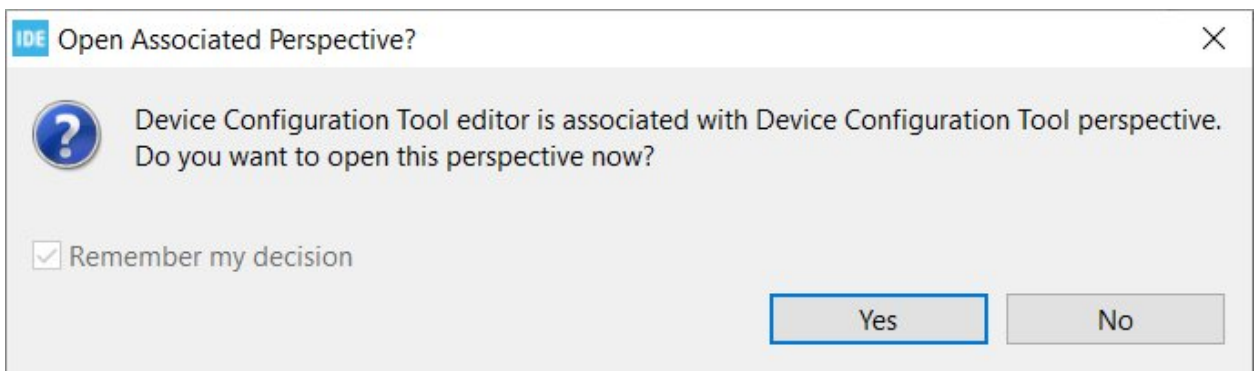
? < Back Next > Finish Cancel

Enchipsdatorer – Laboration 0 Hello Blinky!

Kommer en dialog med texten "Initialize all peripherals with their default Mode ?" upp så tycker ni Yes.



Eventuellt ser ni också dialogen nedan. **Välj Yes.**



Notera att första gången man skapar ett projekt för en ny mikrokontroller behöver miljön ladda ner ett mjukvarupaket för just den familjen processorer. Det kan ta 5-10 minuter om det är många som kör samtidigt.











När projektet är skapat så ser man ett fönster där mikrokontrollernas pinnar och funktioner visas. Vi kommer gå igenom detta mer senare i kursen. Det är ett hjälpmedel som kallas CubeMX som används för att kunna konfigurera alla settings för olika peripherals enkelt.



Enchipsdatorer – Laboration 0

Hello Blinky!

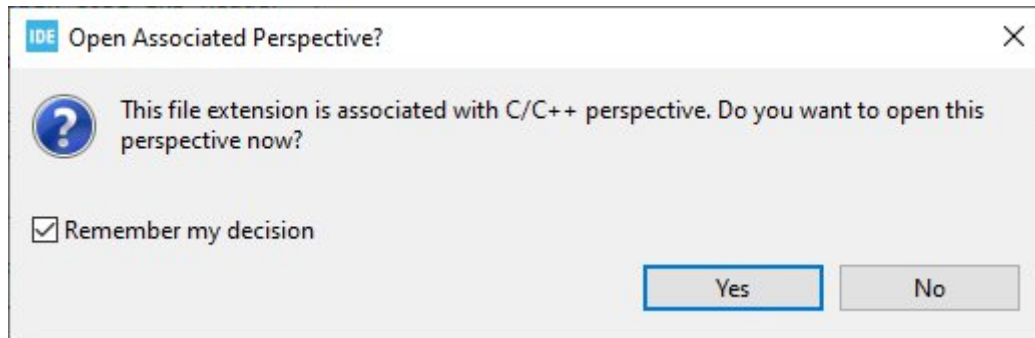
Kollar man i utforskaren så ser ni att det genererats filer i mappen där ni klonade ert github-projekt.

Enchipsdatorer_VT2022 > Lab0_Blinky			Search Lab0_Blinky	
Name	Date modified	Type	Size	
 .settings	2022-03-29 21:27	File folder		
 Core	2022-03-29 21:27	File folder		
 Debug	2022-03-29 22:02	File folder		
 Drivers	2022-03-29 21:27	File folder		
 .cproject	2022-03-29 21:27	CPROJECT File	25 KB	
 .mxproject	2022-03-29 21:27	MXPROJECT File	8 KB	
 .project	2022-03-29 21:27	PROJECT File	2 KB	
 Lab0_Blinky.ioc	2022-03-29 21:27	IOC File	6 KB	
 STM32F401RETX_FLASH.Id	2022-03-29 21:27	LD File	5 KB	
 STM32F401RETX_RAM.Id	2022-03-29 21:27	LD File	5 KB	

Enchipsdatorer – Laboration 0

Hello Blinky!

Du bör nu få en fråga om att byta perspektiv. Kryssa i Remember-rutan innan du väljer Yes.



Enchipsdatorer – Laboration 0 Hello Blinky!

2.3. Skriv kod

All kod du ser i main.c är kod som CubeMX genererat åt dig. Leta reda funktionen `main()`. Notera alla parvis kommentarer som ser ut som denna bild:

```
/* USER CODE BEGIN 1 */  
  
/* USER CODE END 1 */
```

Varje gång som CubeMX genererar kod så är det *ENDAST* kod mellan ett matchande par "BEGIN"/"END" som sparas. All annan kod tas bort när ny genereras. Så var noggrann med var du skriver din kod.

Lokalisera nu din main-loop inne i `main()`. Den är i slutet av funktionen och har kommentaren

```
/* Infinite loop */
```

precis ovanför sig. Inuti while-loopen (mellan matchande kommentarer) så ska vi skriva kod för att blinka lampan på kortet, i vår egen takt.

Kolla upp funktionen `HAL_GPIO_WritePin()` i dokumentet `JU_STM32_HAL-beskrivning.pdf`, som du bör ha ett utskrivet exemplar av.

Från vyn i CubeMX så minns vi att benet som dioden sitter på heter LD2. Så raden för att skriva en etta på det benet blir denna:

```
HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_SET);
```

De två fördefinierade värdena som kan anges som argument 3 är `PIN_SET` och `PIN_RESET` (det enda viktiga är att `RESET` är en nolla).

Efter den raden så lägg till följande rad:

```
HAL_Delay(200);
```

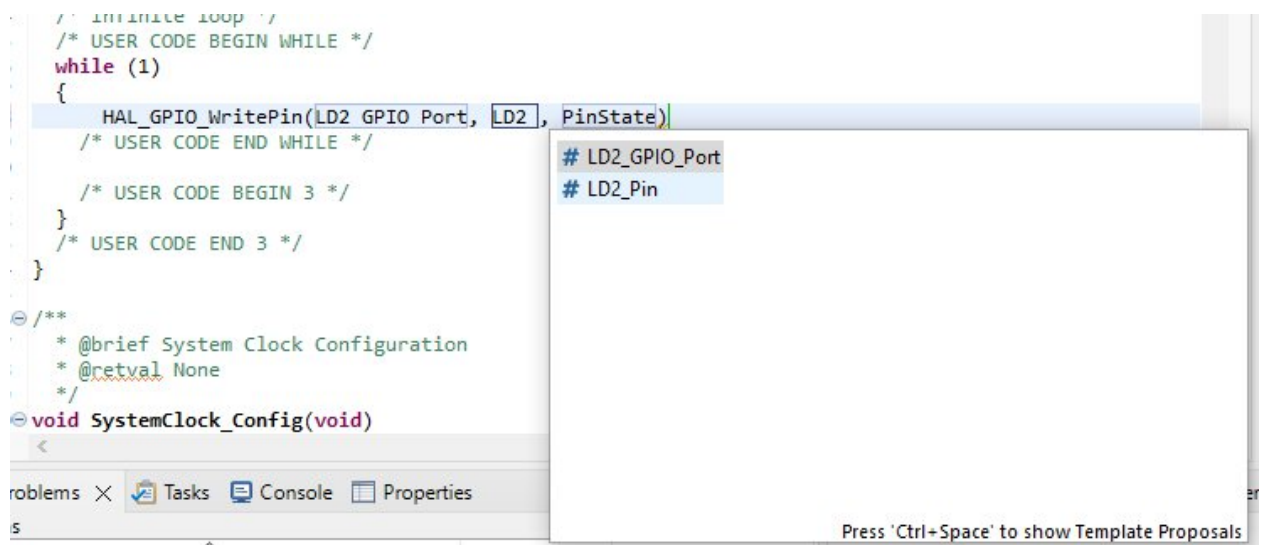
Detta är en paus i 200 "ticks". Ett "tick" är nästan alltid en millisekund långt. Så det betyder att det på det går 1000 tick på en sekund. Och 1000 är den absolut vanligaste tickrate:n ni kommer att stöta på i inbyggda system.

Enchipsdatorer – Laboration 0 Hello Blinky!

Blinka kan man bara göra om man slår på och av någonting. Så låt oss lägga till två till rader:

```
HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);  
HAL_Delay(800);
```

Vill du se hur auto-kompletteringen i CubeIDE fungerar så sudda ut innehållet i en `WritePin()`, skriv LD2 och tryck sedan Ctrl+mellanslag. Då bör det se ut som bilden nedan:



Denna kod räcker för att få vår diod att blinka – Något du i tre år kommer att tränas i, för att kunna göra med stor bravur!

Enchipsdatorer – Laboration 0 Hello Blinky!

2.4.Första bygget, körningen och uppdatering

Hitta knapparna i bilden:

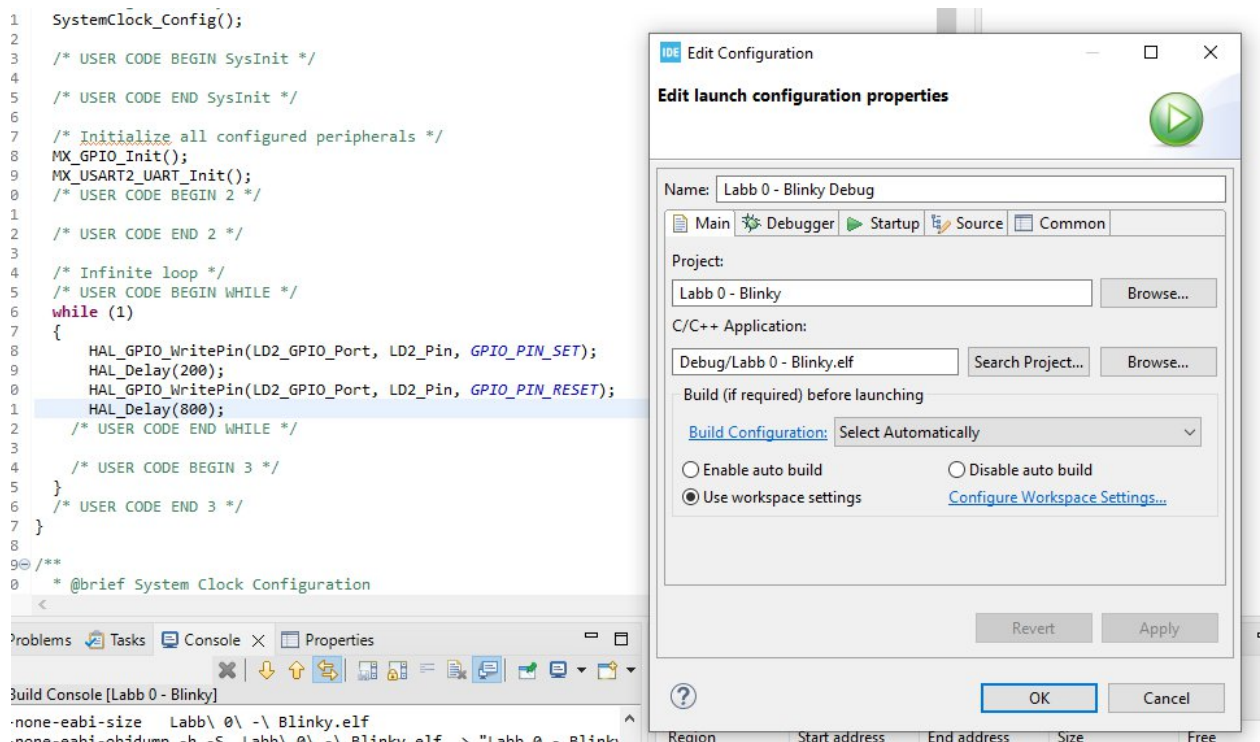


Knapparna är

1. Bygg
2. Debug
3. Kör

För att bygga ditt program, tryck Ctrl+B eller klicka på hammar-ikonen. Det bör snurra förbi text i konsolen längst ner i CubeIDE, men det bör vara utan klagomål.

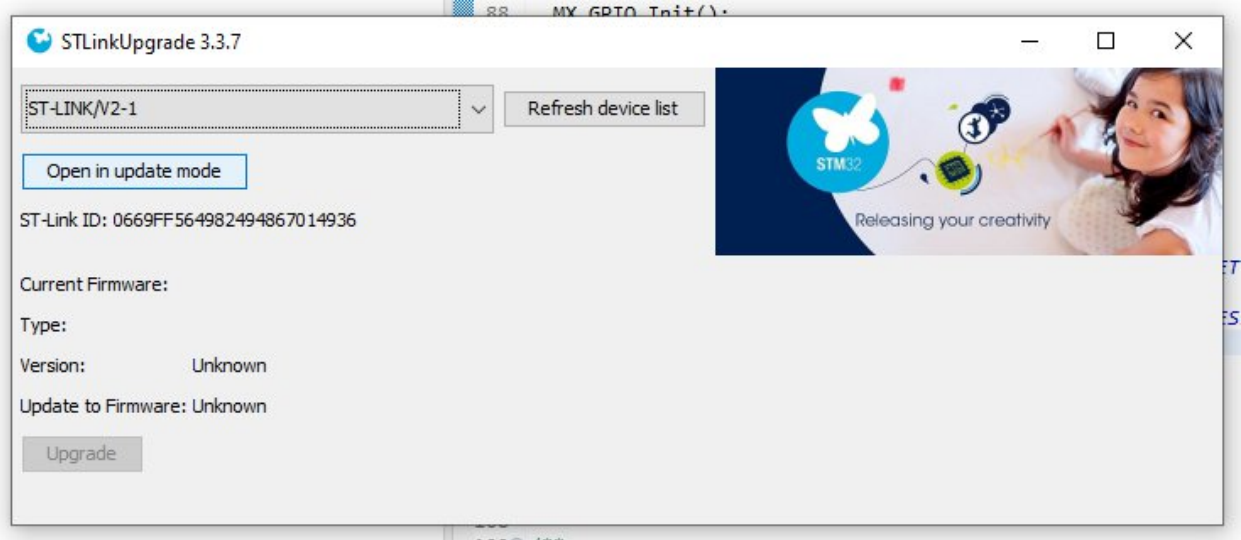
Klicka nu på (3), Kör-knappen. Då ska detta fönster dyka upp:



När du tryckt "OK" så ska ditt program köras på kortet.

Enchipsdatorer – Laboration 0 Hello Blinky!

Inte? Nej, såklart är det dags att köra uppdateringar på ditt kort nu när CubeIDE förstått att du faktiskt har ett anslutet kort.



Klicka på "Open in update mode" och sedan på "Upgrade".

Om du gjorde uppgradering så lär du behöva klicka på kör-knappen en gång till. Men NU så ska det blinka en gång i sekunden, och vara tätt en femtedels sekund åt gången.

Grattis! Du har lyckats med Blinky! Ta en liten paus innan du fortsätter.

Enchipsdatorer – Laboration 0 Hello Blinky!

3. Putty och utskrifter över UART

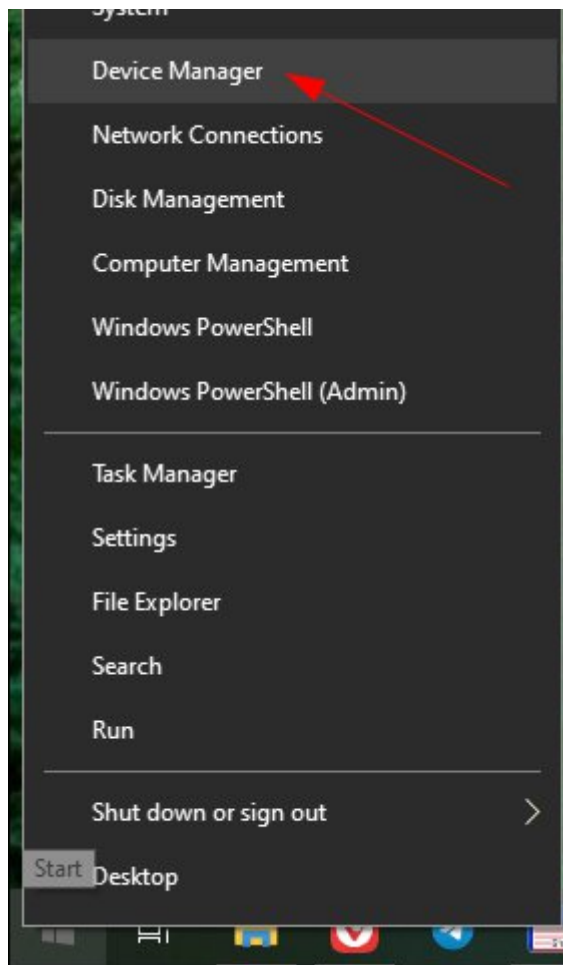
3.1. Hitta din COM

Om du tittar i CubeMX (fliken med bilden på chipet) så kommer du längst ner till vänster på chipet hitta USART_TX och USART_RX. Dessa betyder

Universal Synchronous and Asynchronous Receiver-Transmitter

TX är kontakten för Transmit och RX den för Receive. Din MCU har inbyggd hårdvara för skicka data över dessa vilket du strax ska göra. För att kunna skapa en anslutning så måste du ta reda på vilket COM-enhet som ditt kort blivit tilldelat, såsom COM4 eller COM7.

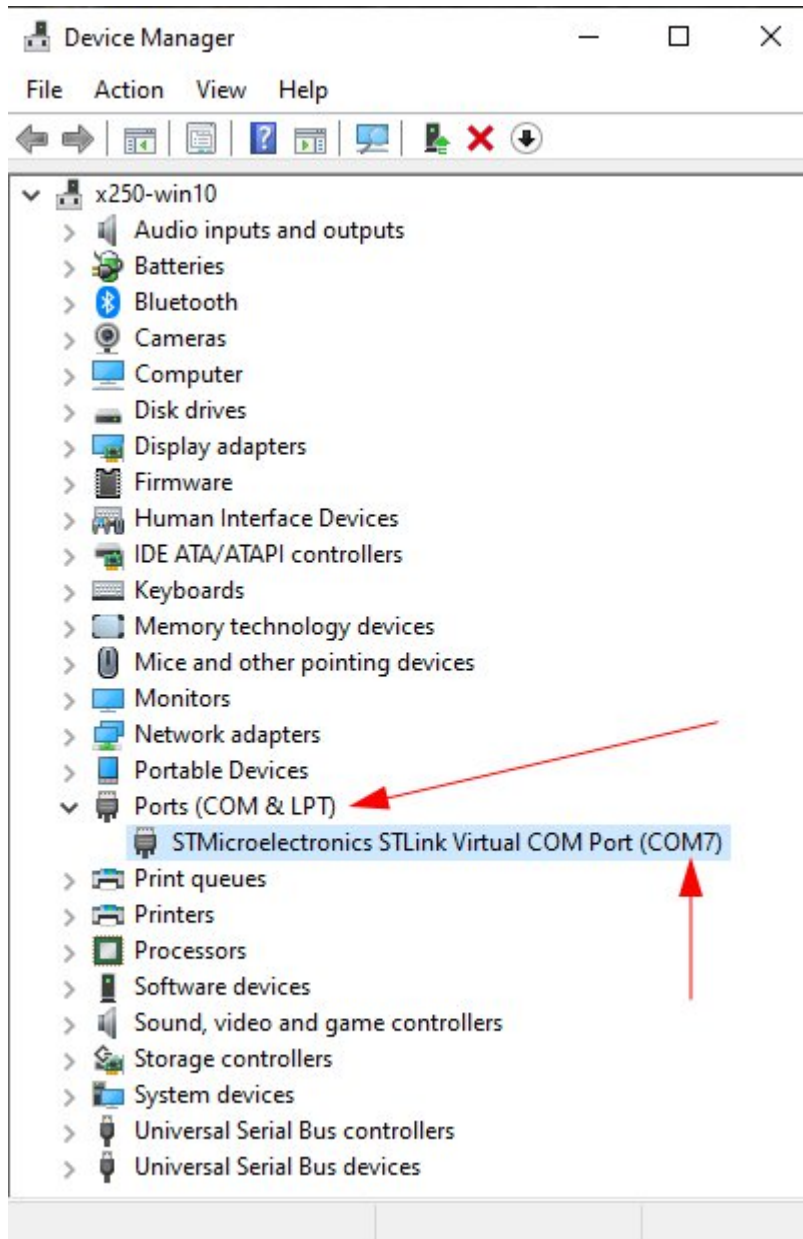
Denna bild är från ett engelskt Windows 10. Det vi vill komma åt är "Enhetshanteraren" eller "Device Manager". Den går att komma åt på fler sätt än att högerklicka på Start-knappen (kontrollpanelen lär ha den någonstans).



Enchipsdatorer – Laboration 0 Hello Blinky!

När du har Enhetshanteraren öppen så kolla upp vilket COM-port-nummer ditt kort blev tilldelat och kom ihåg det.

Om du vill experimentera så ryck ut enheten och stoppa in den igen. Det är inte helt osannolikt att kortet får ett nytt COM-nummer.



Som sagt, kom ihåg denna siffra.

Enchipsdatorer – Laboration 0 Hello Blinky!

3.2. Lägg till kod för utskrift

Gå tillbaka till koden i CubeIDE. Någonstans nära början av filen ska du ha raden

```
#include "main.h"
```

Strax under den finns en kommentar om

```
/* USER CODE BEGIN Includes */
```

I det blocket ska du lägga raden

```
#include <stdio.h>
```

Plocka fram JU_STM32_HAL-beskrivning.pdf och titta igenom båda beskrivningarna av HAL_UART_Transmit(). Koden här inunder liknar den i slutet av filen.

Lägg till tre variabler precis innan while-loopen: En sträng (buffert) att skicka som argument för utskrifter, en variabel för dess längd och en siffra så att utskriften inte blir för tråkig.

```
char    str[81] = { '\0' };  
uint16_t str_len = 0;  
int     nblink  = 0; // number of blinks
```

Sedan, efter anropet till HAL_Delay(800), lägg till dessa rader:

```
nblink++;  
str_len = sprintf(str, "Blinky has succeeded %d times!\r\n", nblink);  
HAL_UART_Transmit(&huart2, (uint8_t*) str, str_len, HAL_MAX_DELAY);
```

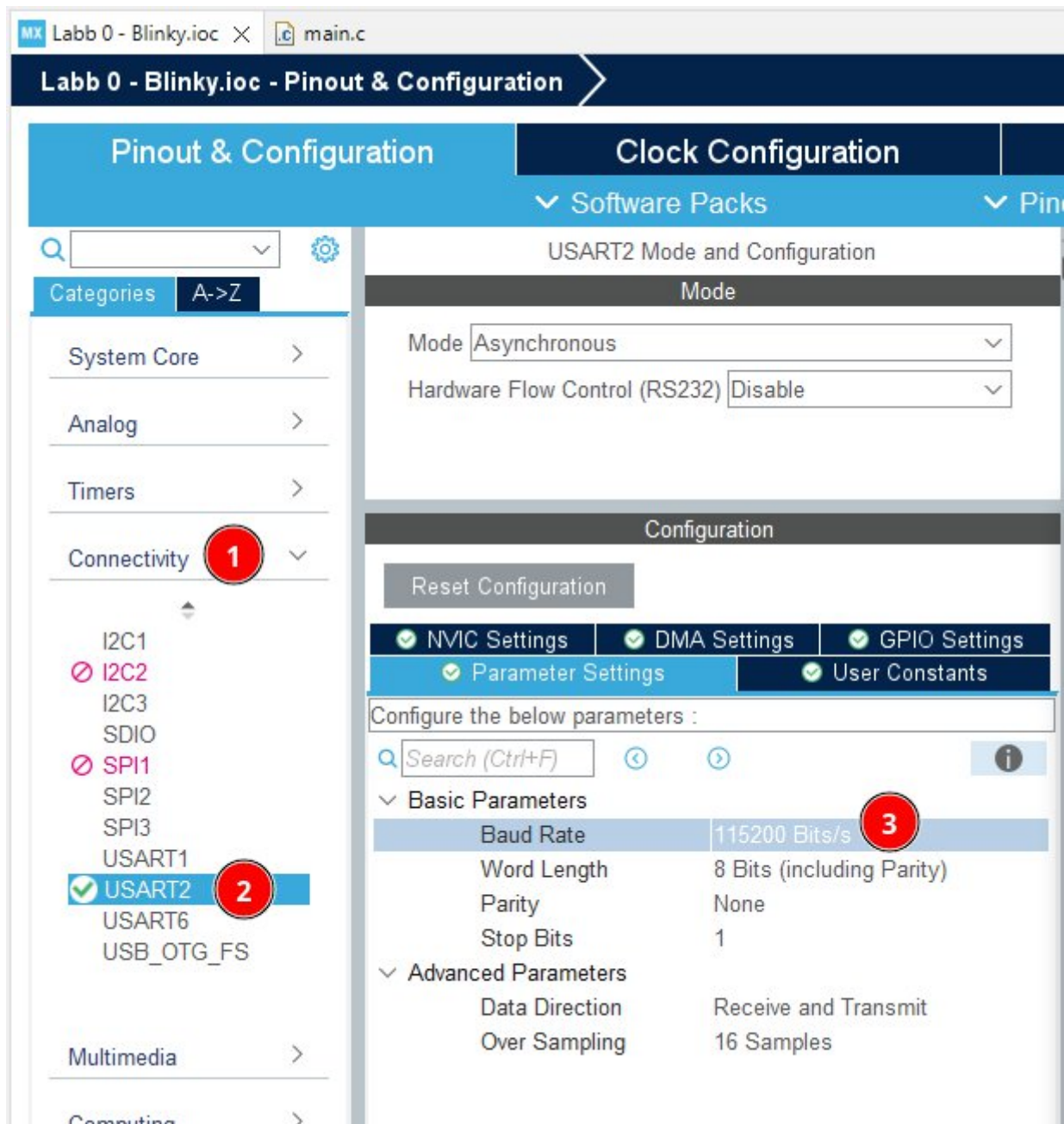
Då kommer ditt kort att skriva ut över enheten UART2, vars "handle" är huart2. Funktionen HAL_UART_Transmit() ska dock ha en pekare till en handle, så det är vad vi skickar med &huart2.

Funktionen printf() (utan "s") är hårdkodad att spotta ut sig resultatet på stdout (som kallas cout i C++). Vi har ingen sådan sak på ett STM32-kort. Istället så använder vi sprintf() som inte skriver ut, utan sparar "utskrifts"-resultatet i en buffert (första argumentet). Innehållet i denna buffert (kallad str i koden ovan) är sedan vad som skickas över UART-anslutningen.

Enchipsdatorer – Laboration 0 Hello Blinky!

3.3. Anslut med Putty

Öppna CubeMX, välj sedan Connectivity (1), klicka på USART2 (2) och ta reda på din aktuella Baud Rate (3):



"Baud Rate" betyder överföringshastighet.

Enchipsdatorer – Laboration 0 Hello Blinky!

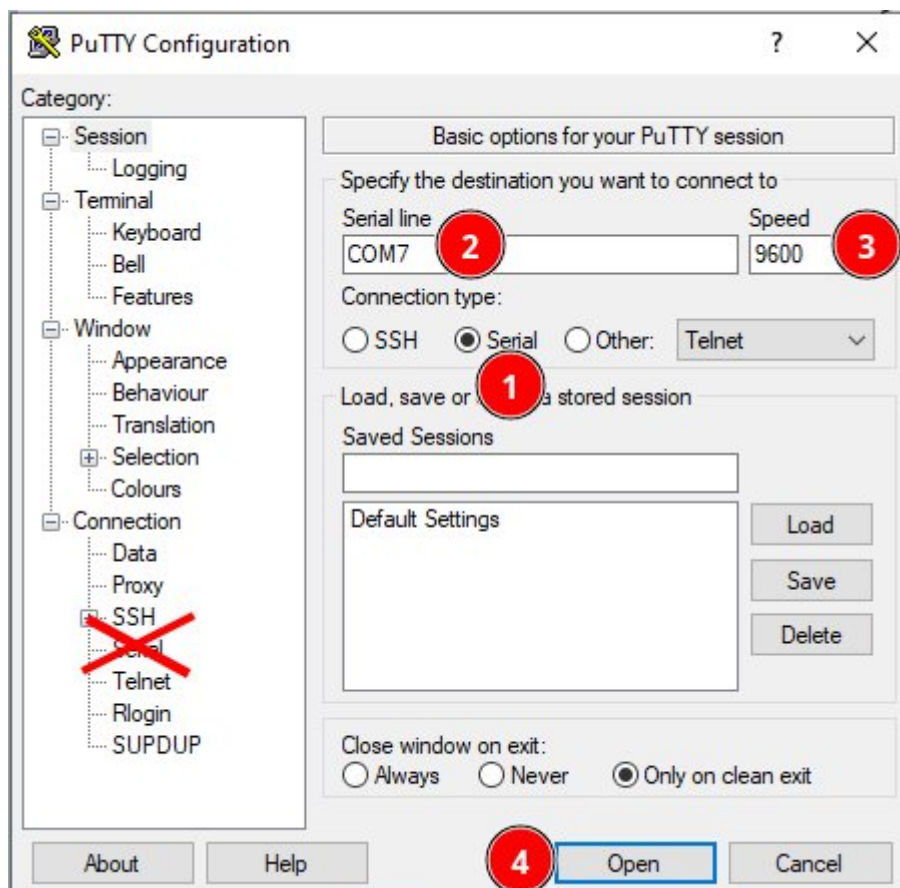


-användare? Den här länken bör täcka dina behov.
Förmodligen klarar du dig utan drivrutin (du har ingen DB-9-adapter utan UART till USB görs på ditt kort).
Använd helst programmet *screen* då det finns på alla UNIX-system.

<https://pbxbook.com/other/mac-tty.html>

Öppna Putty och klicka på "Serial" (1) som din "Connection type". Det "Serial" som är överkryssat i bilden, är till för om du vill matcha ändringar du gjort i CubeMX, tex. "Word Length" eller "Parity".

Putty bör nu se ut så här:

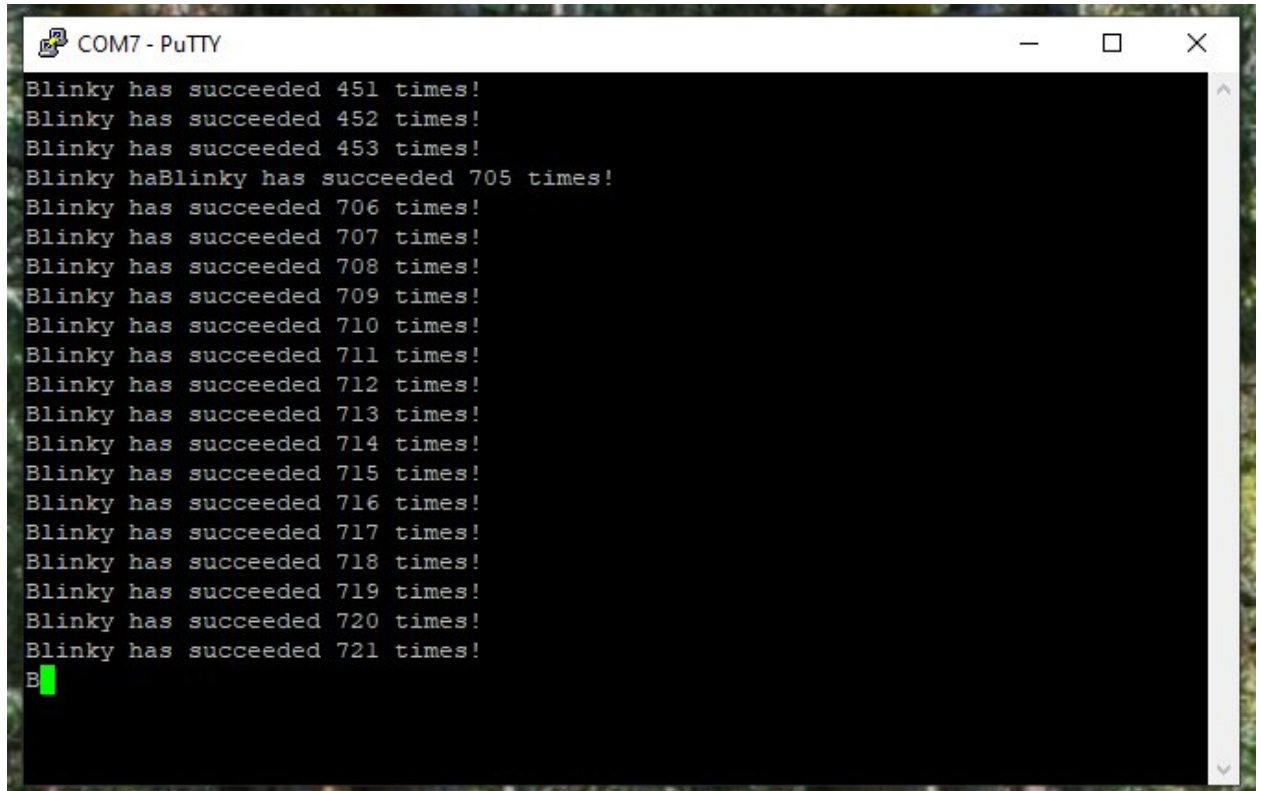


Kommer du ihåg ditt COM-nummer? Bra! Skriv in det i (2).

Skriv in hastigheten du kollade upp i CubeMX i (3) och öppna sedan en anslutning (4).

Enchipsdatorer – Laboration 0 Hello Blinky!

Detta bör vara vad du nu ser:



(ignorera siffrornas oregelbundenhet i bilden, det kom av scrollande)

Nu har du knåpat ihop både Blinky och en lyxig Hello World! Inte illa pinkat!

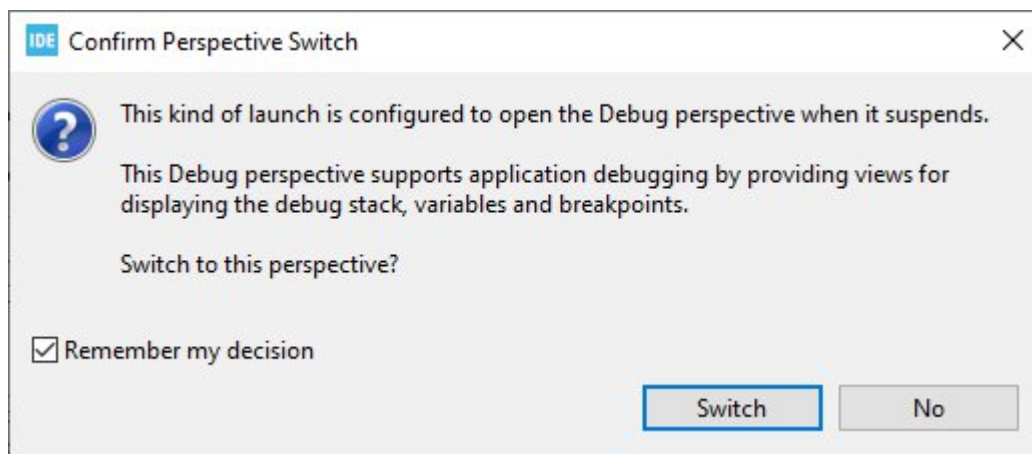
Ta en liten bensträckare innan de sista momenten.

Enchipsdatorer – Laboration 0 Hello Blinky!

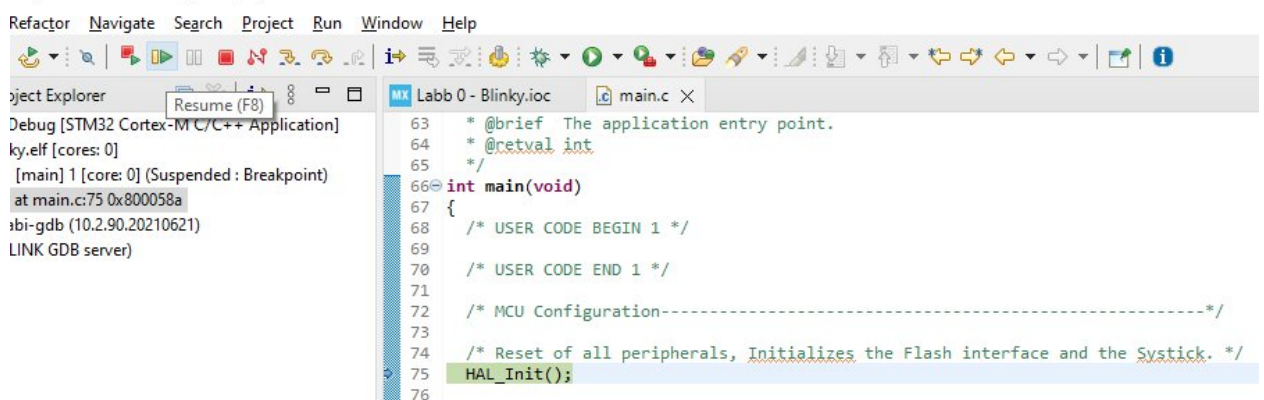
4. Debugging

4.1. Starta en debug-session

Klicka nu på lilla insektsknappen i CubeIDE. Rutan nedanför ska dyka upp. Klicka i Remember och välj Switch.



När debug-vyn är laddad så ska programmet vara pausat på raden för `HAL_Init()`. Dvs. ingen kod alls har körts, eftersom programmet pausats innan första kodraden i `main()`:

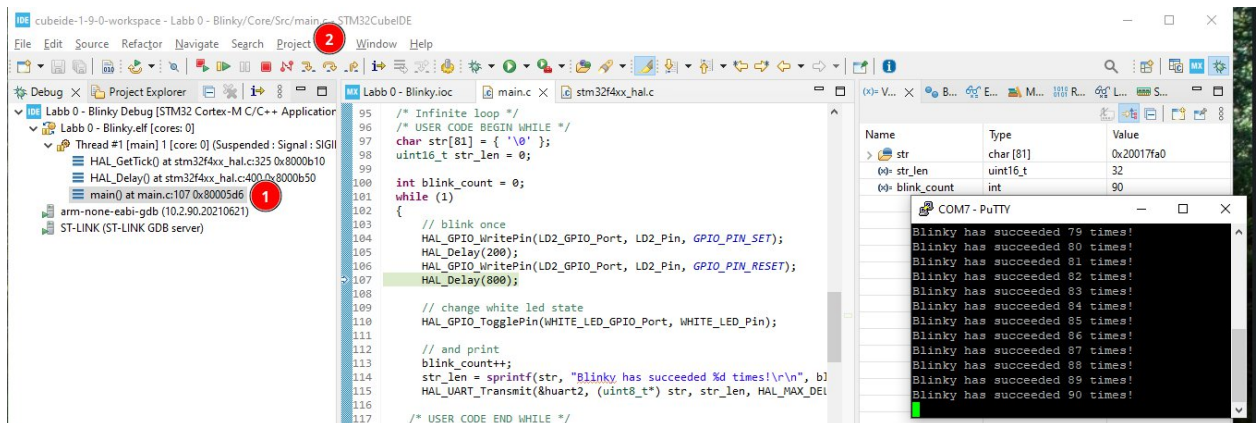


Du kan nu klicka på Resume-knappen (eller trycka F8). Dioden ska blinka och Putty ska visa upp vad som skrivs ut av ditt program.

Enchipsdatorer – Laboration 0

Hello Blinky!

Klickar du på Pause-knappen så lär CubeIDE öppna någon fil och visa dig kod som absolut inte var din. Din kod tillbringar den absoluta majoriteten av sin tid i funktionen `HAL_Delay()`, så du lär få se den funktionens källkod. Klicka då på `main()`, som markerat med (1) i bilden och sedan på "Step Return" (se nedan):



Knapparna som är vid (2) är "Step Into", "Step Over" och "Step Return" (out of). Detta är hur långt du vill att koden ska gå när körningen pausats av användaren eller av breakpoints (högerklicka på ett radnummer för att lägga till breakpoint).

Experimentera med knapparna och se vad som händer. Ett bra sätt är att göra en "Step Into" på `HAL_UART_Transmit()` först. Du behöver absolut inte förstå vad koden till den funktionen gör. Enbart hur de olika Step-knapparna relaterar till funktionsanrop.

Lägg även märke till listan med variabler till höger, ovanför Putty i bilden. Det är vad som är på stacken för funktionen i anropskedjan (call-stack) du valt till vänster (1). Om du klickar på funktionsnamnen till vänster så kan du se hur variablerna till höger byts ut.

Enchipsdatorer – Laboration 0 Hello Blinky!

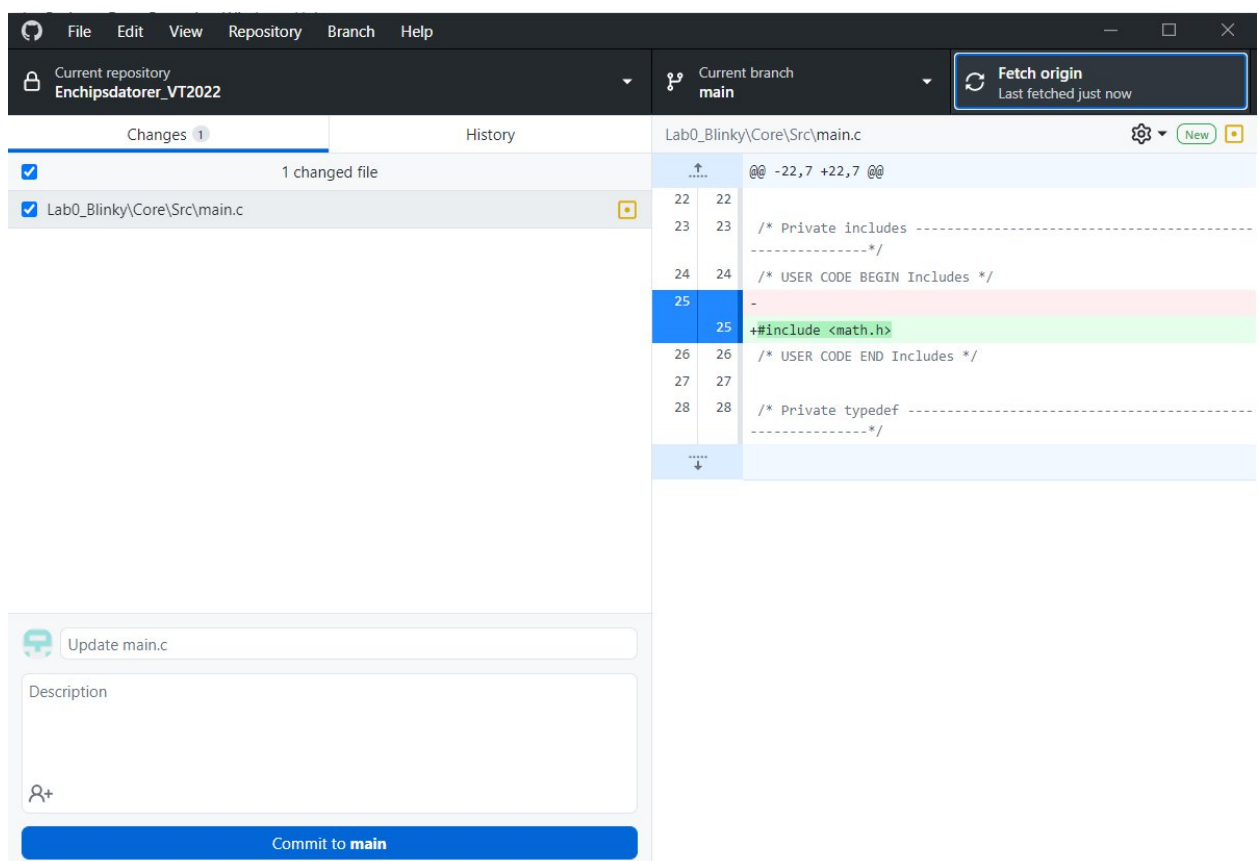
5. Spara arbetet på Github

5.1. Git commit

När man lagt till ny kod bör man regelbundet "checka in" eller commit som det heter på Git-språk sin kod. Det innebär att man tar en snapshot på koden och lagrar undan som en ny version. På så vis kan man senare gå tillbaka och se vilka ändringar som är gjorda och när i tiden det skedde. Det är extra viktigt att göra en commit innan en större ändring i koden. T.ex. om man ska lägga till en helt ny funktionalitet som innebär att man kommer ändra mycket.

När flera personer jobbar med samma projekt är det också mycket kraftfullt att kunna jobba med sin egna version som man sedan kan 'merg:a' (sammanfoga) med de andras kod vid lämpliga tillfällen.

I figuren nedan visas hur det ser ut i Github Desktop när man har ändrat en fil.



Man ser vilka filer som har ändringar och till höger kan man se vad som skiljer filen mot den tidigare versionen.

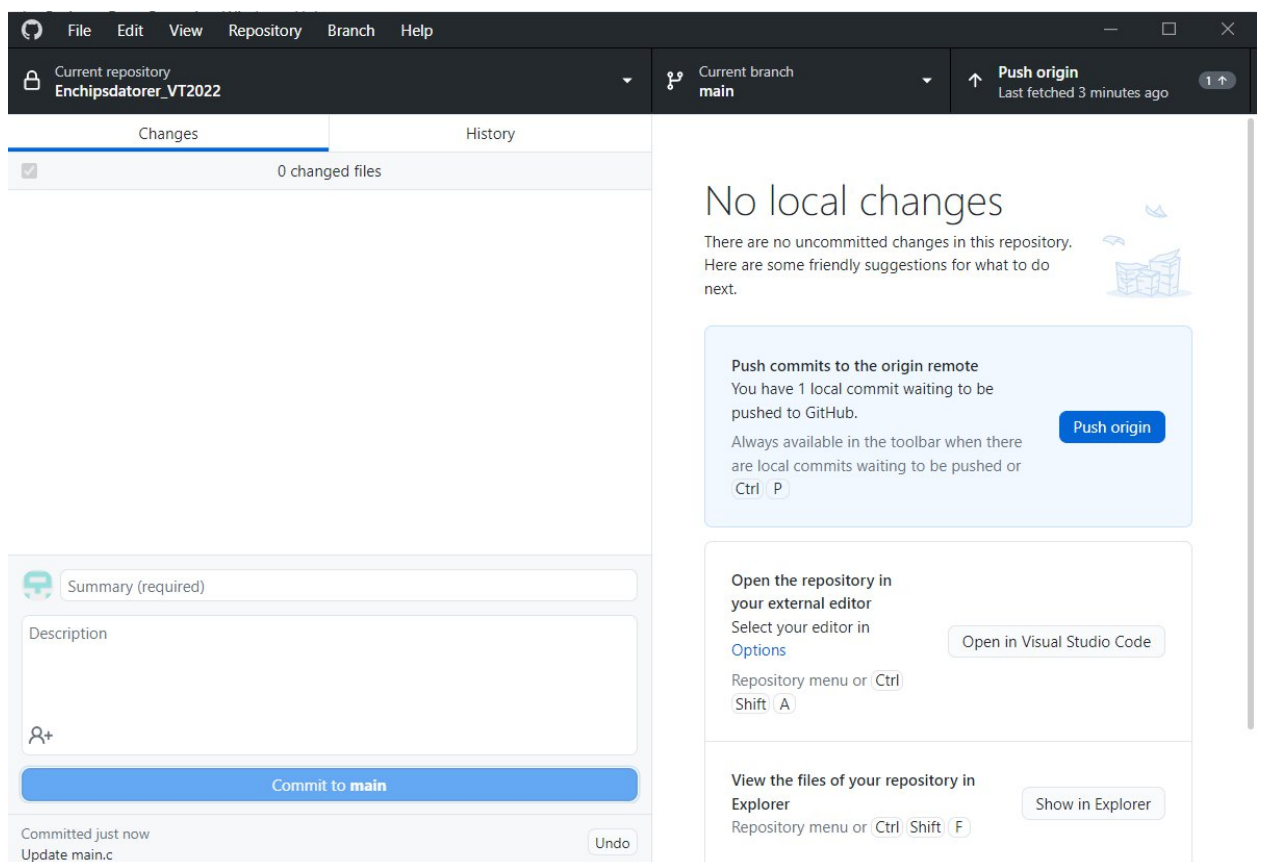
Genom att skriva något i summary och description och sedan trycka 'commit to main' så sparas den nya versionen.

Enchipsdatorer – Laboration 0

Hello Blinky!

5.2. Git push

Notera att när man gör commit så sparas den nya versionen bara lokalt först. Vill man flytta upp den nya versionen till molnet på github.com så måste man göra 'Push origin', se figuren nedan. Man behöver inte göra push efter varje commit, men det kan vara bra att göra det ganska regelbundet då det blir en säkerhetskopiering i molnet ifall din dator kraschar. Skulle man vilja fortsätta utvecklingen på en annan maskin kan man bara kлона ner projektet från github.com och fortsätta. All versionshistorik finns då med om man pushat upp sina ändringar.



Vi kommer senare i kursen gå igenom fler features i Git, t.ex. användning av branches (grenar eller sidospår) och hur man kan använda Git från ett terminal-fönster, men med det ni lärt er idag klarar ni er långt med.



Enchipsdatorer – Laboration 0 Hello Blinky!

6. Bonusuppgift

Om du känner att du hinner så kan du testa att koppla på lite sladdar till ditt utvecklingskort. Som vi såg i CubeMX så finns dioden med aliaset LD2 på ben PA5.

Ta fram papperslappen som kom med ditt kort och leta reda på vilket av benen som är PA5. Bygg en enkel krets från det benet med en diod och ett motstånd, som går till jord någonstans på kortet. Lyckas du få dioderna att blinka samtidigt?

Benet PA6 ligger alldeles intill PA5, både på kortet och i CubeMX. Använd dig av det. Ställ in CubeMX så att det är samma GPIO-inställningar för PA6 som för PA5. Börja med att klicka på benet och sätta det till "GPIO_Output".

Om du har uppe inställningarna för GPIO (under System Core, till vänster) så kan du även skriva in ett alias för benet där. Vad de gjort med hakparenteserna för LD2 är lite överdrivet och görs ofta inte. Men ge ett passande alias till PA6, tex. "LED_YLW". Spara sedan och generera kod.

Koppla om din krets så att den externa dioden sitter på benet för PA6 istället för PA5.

I din main-loop så lägg till ett anrop till `HAL_GPIO_TogglePin()` som ska få din diod att blinka. Är du osäker på hur den fungerar så kolla i `JU_STM32_HAL-beskrivning.pdf`. Ett anrop per loop blir bra.

Glöm inte att spara och uppdatera ditt repo efter att denna uppgift är klar.