

Embedded software

RoboTeam Twente documentation

Hans van der Heide – hansvdheide@live.nl

Introduction

The aim of this document is to get you started with the embedded software of the robots of RoboTeam Twente. Do not hesitate to send any questions you have to hansvdheide@live.nl. The embedded software consists of three programs: the motor controller in the FPGA, the general software in the microcontroller in the robot, and some software in the base station. This document will tell you how to get all the necessary tools working, the functionality of the code and the design decisions that led to that functionality, and lastly some recommendations for the team of next year to work on.

Microcontroller

Tooling: STM-Cube and ST workbench

The STM32F3 microcontrollers are used. For testing and the base station, we used the STM32F3Discovery. In the main PCB, the STM32F301R8T6 chips are used. Multiple sets of tooling can be used to program this chip. It is chosen to use STM-cube and ST workbench, because they give a graphical user interface, and should make initialization of interfaces easy. Cube is used for generating libraries for I/O devices and ST workbench is an eclipse extension which allows you to upload code.

Downloading and installing

STM cube can be downloaded using [this](#) link. Go to the bottom of the page and press the “get software” button.

The screenshot shows the STM32CubeMX website. At the top, there are tabs for 'QUICK VIEW', 'DESIGN', and 'GET SOFTWARE'. Below these, there is a table with columns 'Description', 'version', and 'Size'. A row is visible for 'SLA0047: Image V2 - SOFTWARE LICENSE AGREEMENT' with version '1.12' and size '99 KB'.

Below this, there is a section titled 'Tools and Software' with two tabs: 'DEVELOPMENT TOOLS' and 'EMBEDDED SOFTWARE'. Under 'DEVELOPMENT TOOLS', there is a table with columns 'Part Number', 'Manufacturer', and 'Description'. A row is visible for 'STSW-STM32095' by 'ST', described as 'STM32CubeMX Eclipse plug in for STM32 configuration and initialization C code generation'.

Below this, there is a section titled 'GET SOFTWARE' with a table. The table has columns: 'Part Number', 'Software Version', 'Marketing Status', 'Supplier', and 'Order from ST'. A row is visible for 'STM32CubeMX' with version '4.21.0', status 'Active', and supplier 'ST'. A yellow circle highlights the 'Get Software' button in the 'Order from ST' column.

At the bottom of the page, there is a footer with four columns of links: 'About STMicroelectronics', 'Media Center', 'Investor Relations', and 'Sustainability'. The 'About STMicroelectronics' column includes links like 'Who We Are', 'Management', 'ST Code of Conduct', and 'Blog'. The 'Media Center' column includes 'Newsroom', 'Backgrounders', 'Media Contacts', and 'Media Subscription'. The 'Investor Relations' column includes 'Investor Relations Home', 'Calendar & Presentations', 'Quarterly Results', 'Corporate Governance', and 'Contact Information'. The 'Sustainability' column includes 'ST Approach to Sustainability', 'Sustainability Reports', and 'ST Foundation'.

Below the footer, there is a line with 'All rights reserved © 2017 STMicroelectronics' and 'Terms of use | Sales Terms & Conditions | Privacy Policy | Contacts'. To the right of this is a newsletter subscription box with the text 'Enter email to subscribe to our newsletter' and a button with a right arrow.

You are requested to make an account. A zipfile will automatically be downloaded. Past the Zip in you program files folder (or wherever you want) and run the installer.

The next step is to download ST workbench from [here](#). Go to the download area. You should make an account (again).

Step 1: Explore

- Search for Content
- Visit Wiki Pages
 - You can start from [Wiki Home](#)
- Read Blogs and Forums
- Look at the FAQs
- Install System Workbench for STM32 - Bare Metal Edition
 - Instructions are provided [here](#)
 - You can download the System Workbench for STM32 installer from the [download area](#).


















Step 2: Ask a Question and Give Feedback

- Register to become part of the community
- Ask questions in the Forums
- Give feedback using Comments at the bottom of most pages

Choose the standard installer (provided you use windows and have a 64 bit machine).

Windows 7

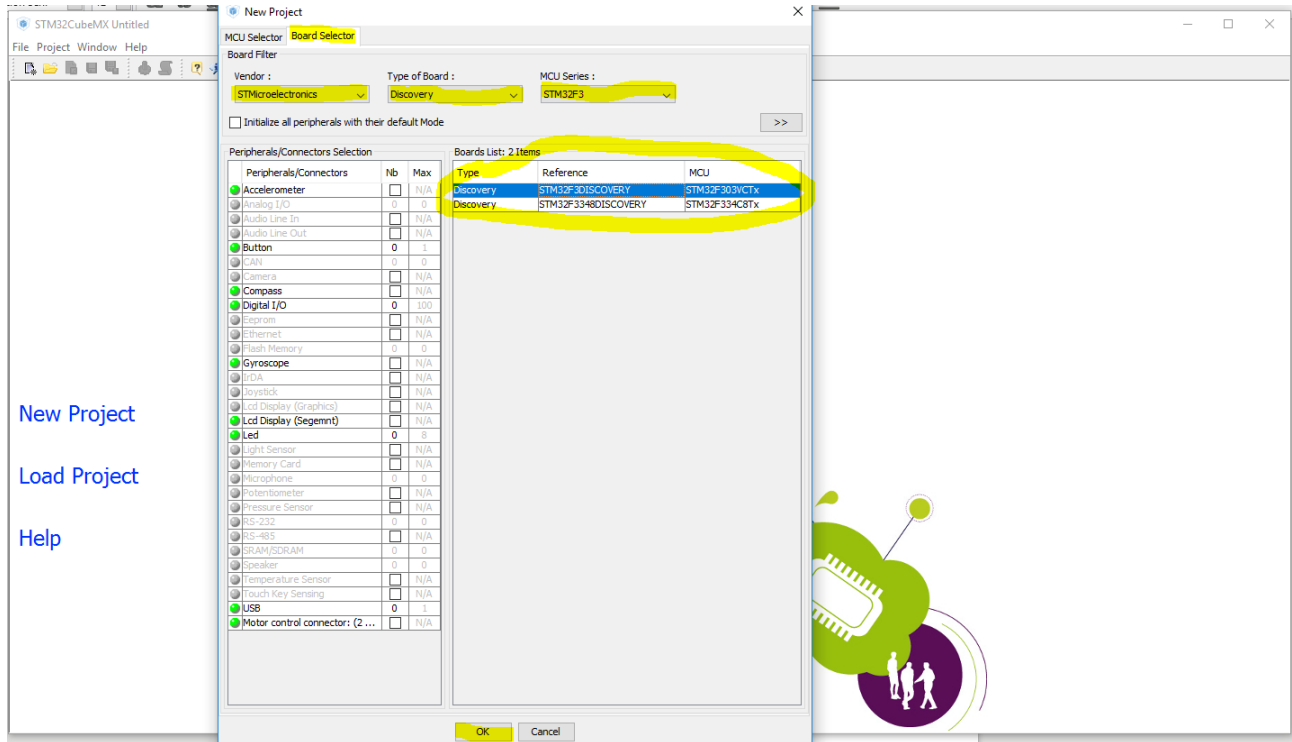
The Windows version is available for 32 and 64 bit systems. Note that we will need to install a device driver to communicate with the ST-Link debug probe, so you **must** select problems downloading an executable file (.exe), try downloading and extracting the ZIP file. In both cases you are advised to also download the MD5 or SHA256 checksum to v

- Latest Windows 7 64 bit installer Version v2.0, updated on Wednesday, May 17, 2017 at 18:01:54 CEST):
 - Installer: [install_sw4stm32_win_64bits-v2.0.exe](#) 
 - MD5 sum d106b3df34fe1f8e8b21e58e1f8b424e in [install_sw4stm32_win_64bits-v2.0.exe.md5](#) 
 - SHA256 sum 43a1596dd01742709decf0dc19ddb77a9c0691f12b077173afd2a4b50833d13 in [install_sw4stm32_win_64bits-v2.0.exe.sha256](#) 
 - Zipped installer: [install_sw4stm32_win_64bits-v2.0.zip](#) 
 - MD5 sum ZIP_MD5SUM in [install_sw4stm32_win_64bits-v2.0.zip.md5](#) 
 - SHA256 sum ZIP_SHA256SUM in [install_sw4stm32_win_64bits-v2.0.zip.sha256](#) 
 - The latest installer can always be retrieved from
 - Installer: [install_sw4stm32_win_64bits-latest.exe](#) 
 - MD5 sum of the installer: [install_sw4stm32_win_64bits-latest.exe.md5](#) 
 - SHA256 sum of the installer: [install_sw4stm32_win_64bits-latest.exe.sha256](#) 
 - Zipped installer: [install_sw4stm32_win_64bits-latest.zip](#) 
 - MD5 sum of the zip: [install_sw4stm32_win_64bits-latest.zip.md5](#) 
 - SHA256 sum of the zip: [install_sw4stm32_win_64bits-latest.zip.sha256](#) 
- Latest Windows 7 32 bit installer Version v2.0, updated on Wednesday, May 17, 2017 at 18:01:54 CEST):
 - Installer: [install_sw4stm32_win_32bits-v2.0.exe](#) 
 - MD5 sum d4a752b97284cd96a8be6557bae1f2aa in [install_sw4stm32_win_32bits-v2.0.exe.md5](#) 
 - SHA256 sum 03aed94bb3571bc910a8f3ff67bc303dbb97bcf9d449019c13226d5886944db2 in [install_sw4stm32_win_32bits-v2.0.exe.sha256](#) 
 - Zipped installer: [install_sw4stm32_win_32bits-v2.0.zip](#) 
 - MD5 sum ZIP_MD5SUM in [install_sw4stm32_win_32bits-v2.0.zip.md5](#) 
 - SHA256 sum ZIP_SHA256SUM in [install_sw4stm32_win_32bits-v2.0.zip.sha256](#) 
 - The latest installer can always be retrieved from
 - Installer: [install_sw4stm32_win_32bits-latest.exe](#) 
 - MD5 sum of the installer: [install_sw4stm32_win_32bits-latest.exe.md5](#) 
 - SHA256 sum of the installer: [install_sw4stm32_win_32bits-latest.exe.sha256](#) 
 - Zipped installer: [install_sw4stm32_win_32bits-latest.zip](#) 
 - MD5 sum of the zip: [install_sw4stm32_win_32bits-latest.zip.md5](#) 
 - SHA256 sum of the zip: [install_sw4stm32_win_32bits-latest.zip.sha256](#) 

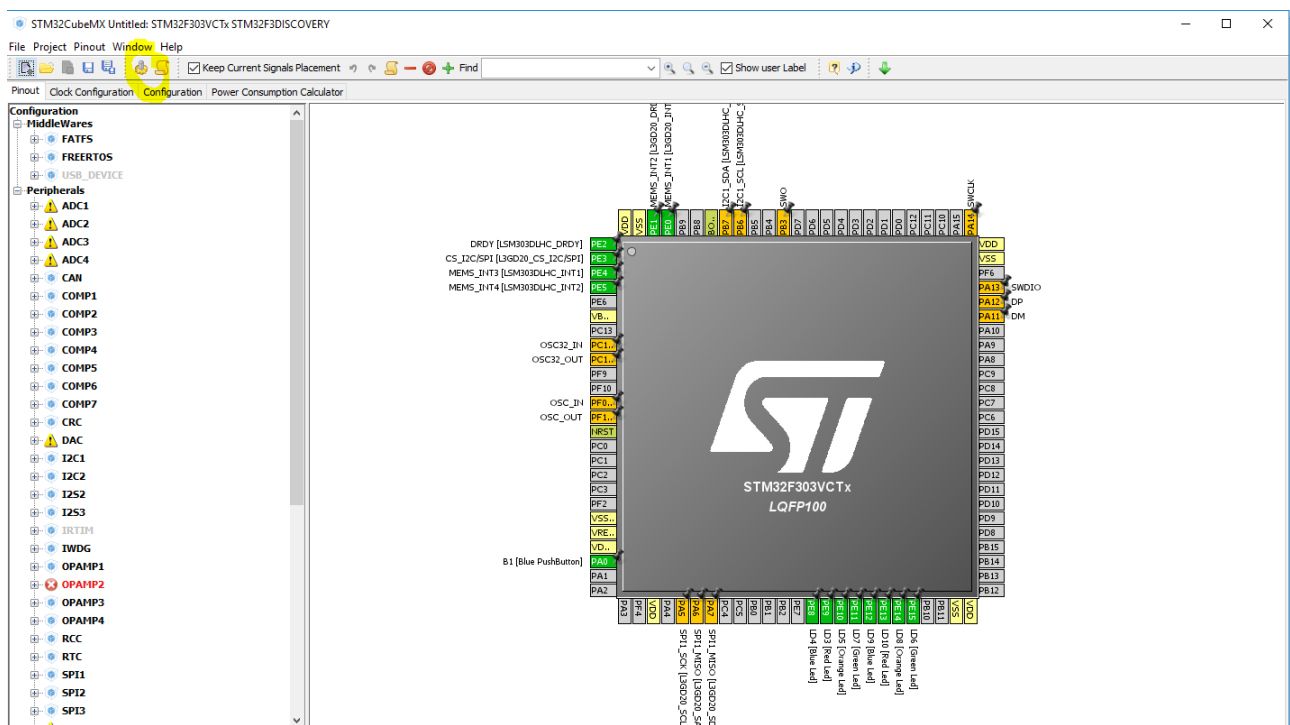
Installing should be self-explanatory.

Making a new project

In order to make a new project using the STM-cube / ST workbench tooling, first open Cube and choose “new project” After clicking this bottom, you get a screen that asks you to select a MCU. For this example, the STM32F3Discovery will be used. Choose in this order “board selector”, the “STMicroelectronics vendor” (should be default), the “Discovery” type of board, the “STM32F3” MCU series, and lastly the STM32F3Discovery and press ok.



You should get the following screen:



The picture in the middle of the screen is an overview of all the input and output pins of the microcontroller. These pins are configurable from this overview. At the left, all kind of I/O interfaces (e.g. USB, SPI, I2C) can be configured. For now, the standard configuration will be used. Press the “generate code” bottom (yellow circle in the picture above) to generate initialization code. Cube will ask for code generation setting. Give the project a name and choose the SW4ST32 IDE. By default the “generate under root” bottom will be pressed.

Project Settings

Project Code Generator Advanced Settings

Project Settings

Project Name
testProject

Project Location
C:\Users\gebruiker\workspace\ Browse

Toolchain Folder Location
C:\Users\gebruiker\workspace\testProject\

Toolchain / IDE
SW4STM32 ☒ Generate Under Root

Linker Settings

Minimum Heap Size 0x200

Minimum Stack Size 0x400

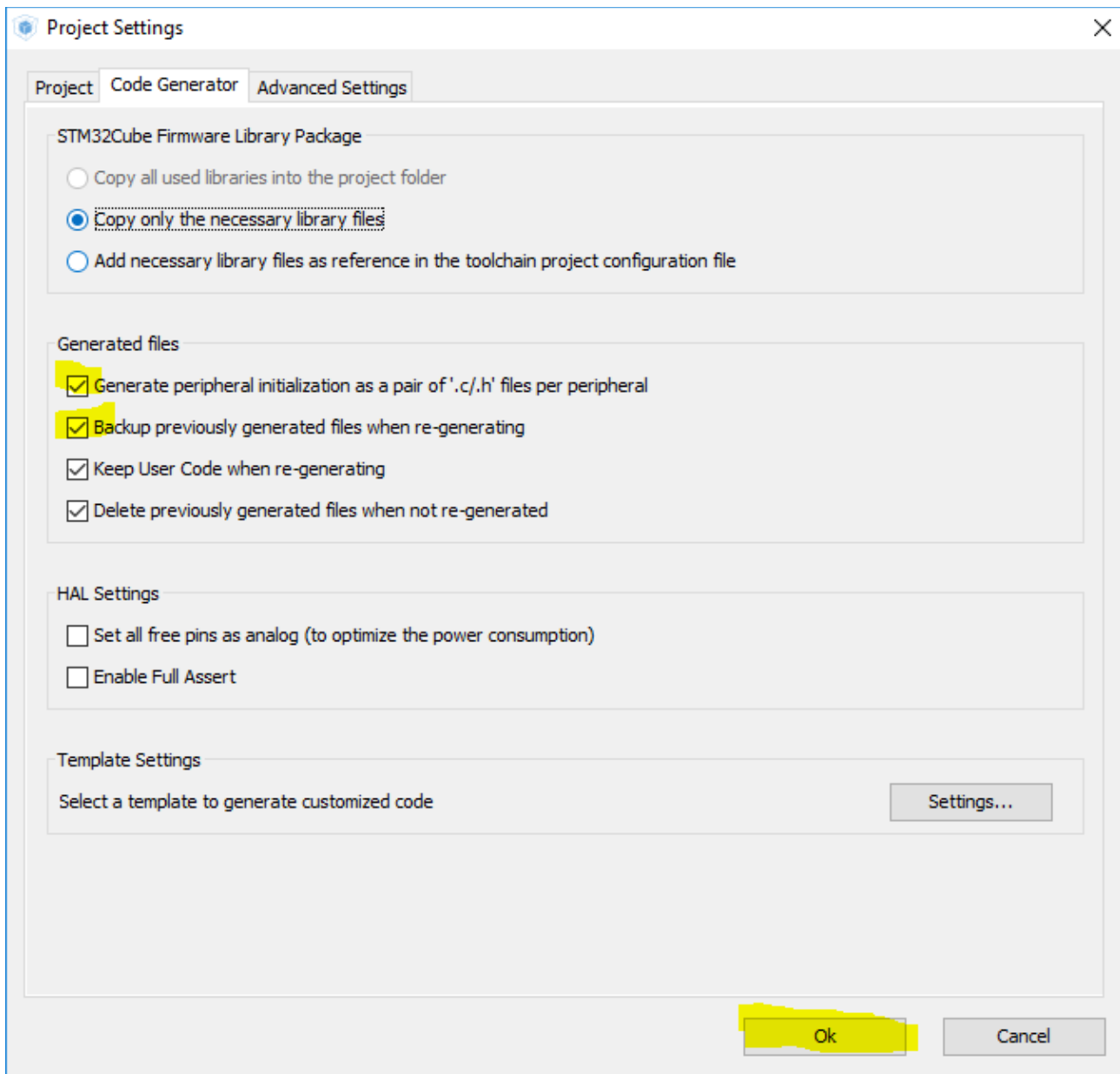
Mcu and Firmware Package

Mcu Reference
STM32F303VCTx

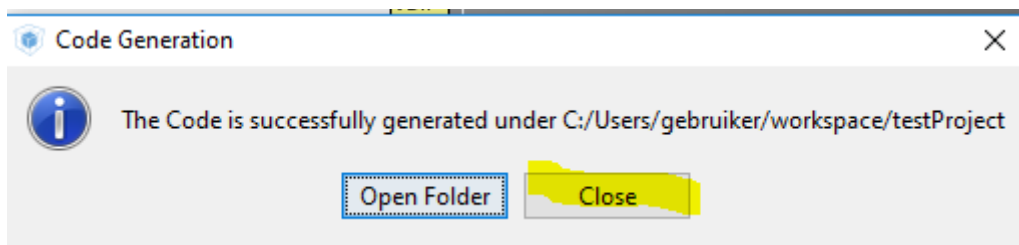
Firmware Package Name and Version
STM32Cube FW_F3 V1.6.0

Ok Cancel

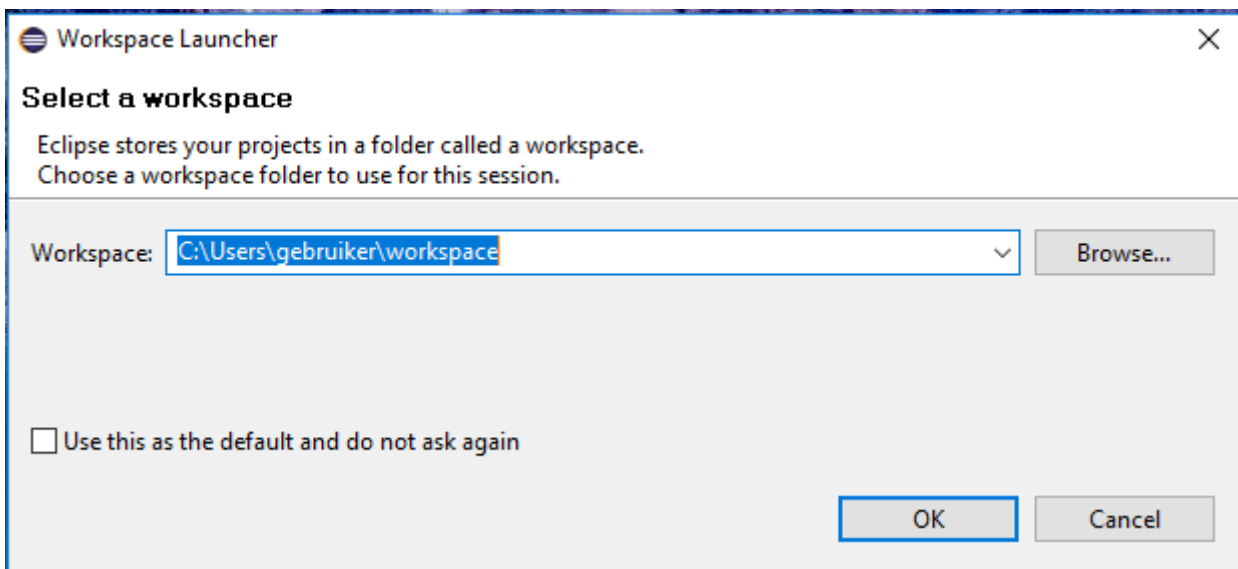
In the second tab, press the “generate pair of .c/.h files” (this will prevent the main file to become big) and “backup previous files” (for obvious convenience). After that, press ok.



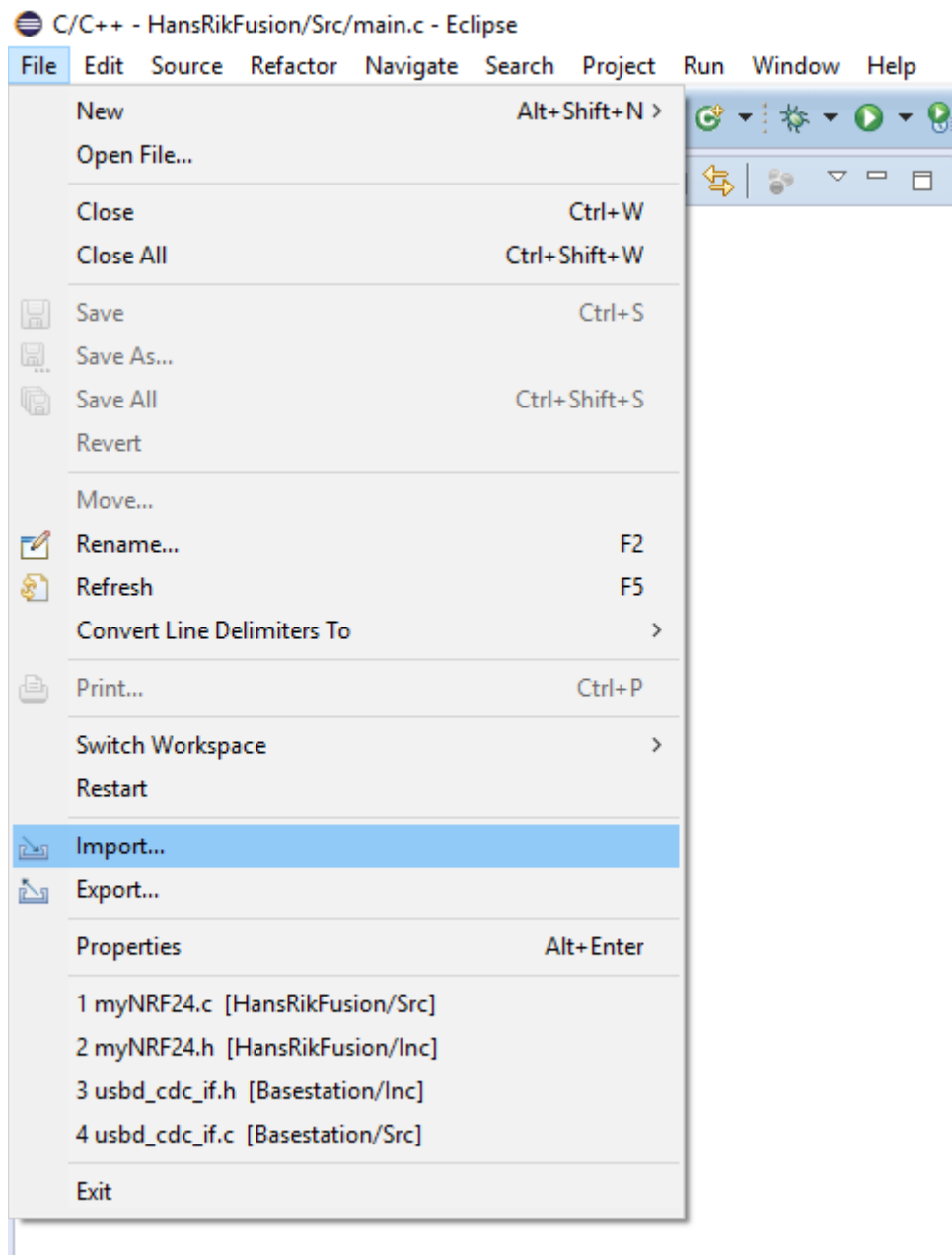
You can ignore the following message:



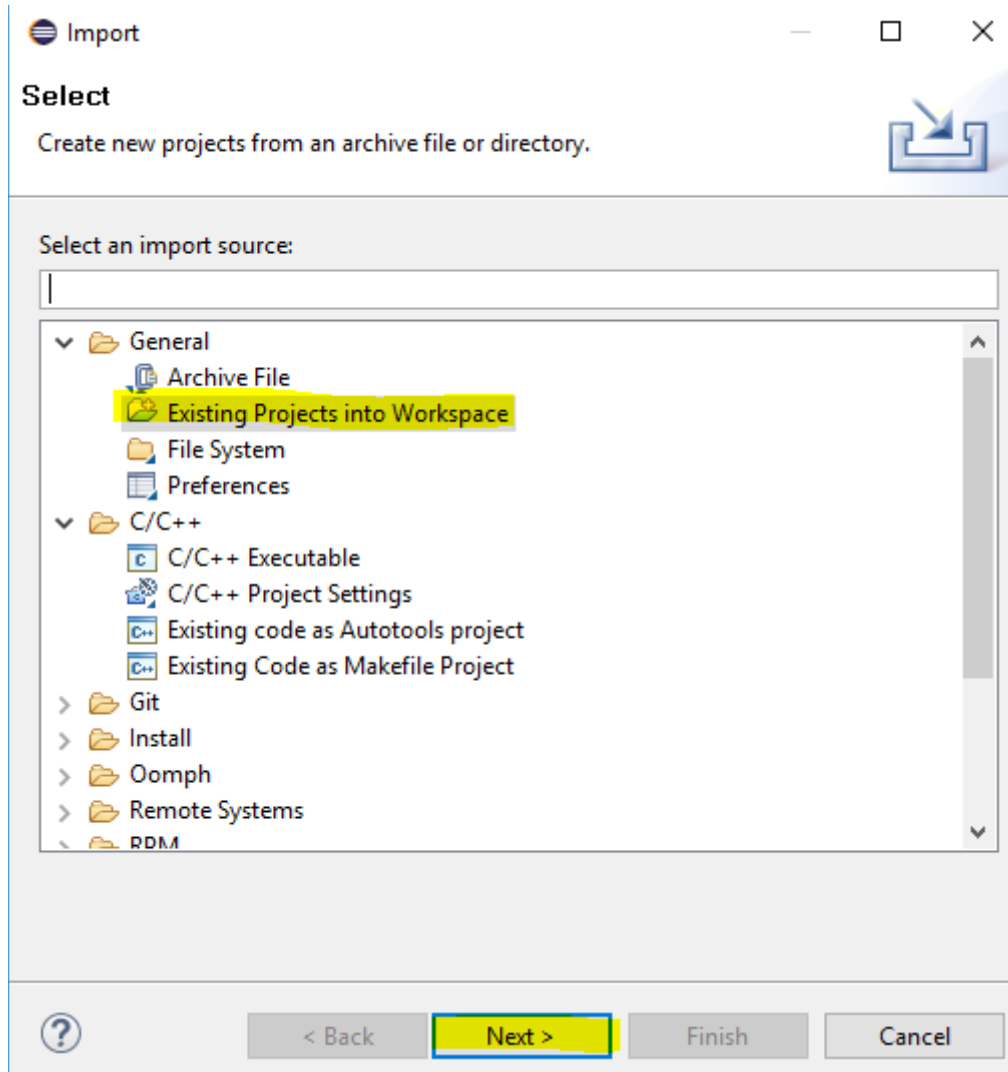
Congratulations! You have successfully generated libraries to use the STM32F3! The next step is to open it in ST workbench. Open ST workbench. It will ask you to choose a workspace. Make sure you use the same workspace as the cube generates its code in. By default, this will automatically work. Press ok.



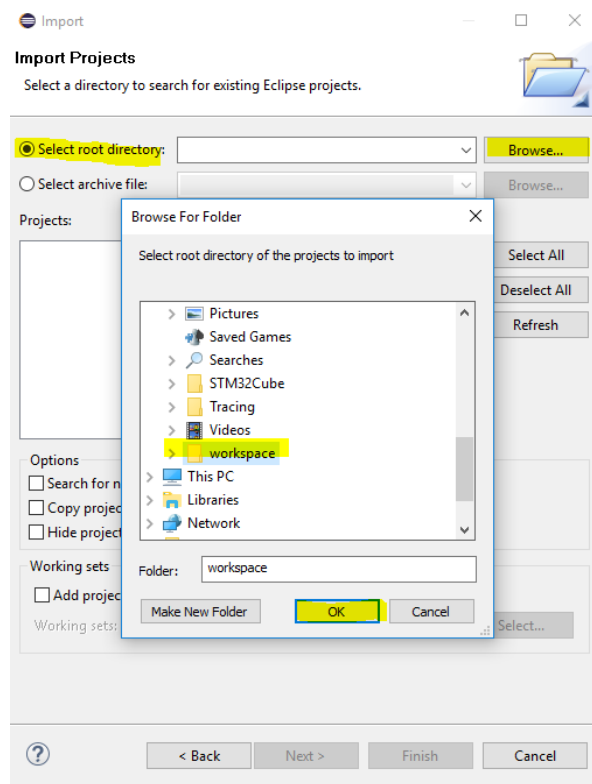
In ST workspace (= eclipse) press file->import



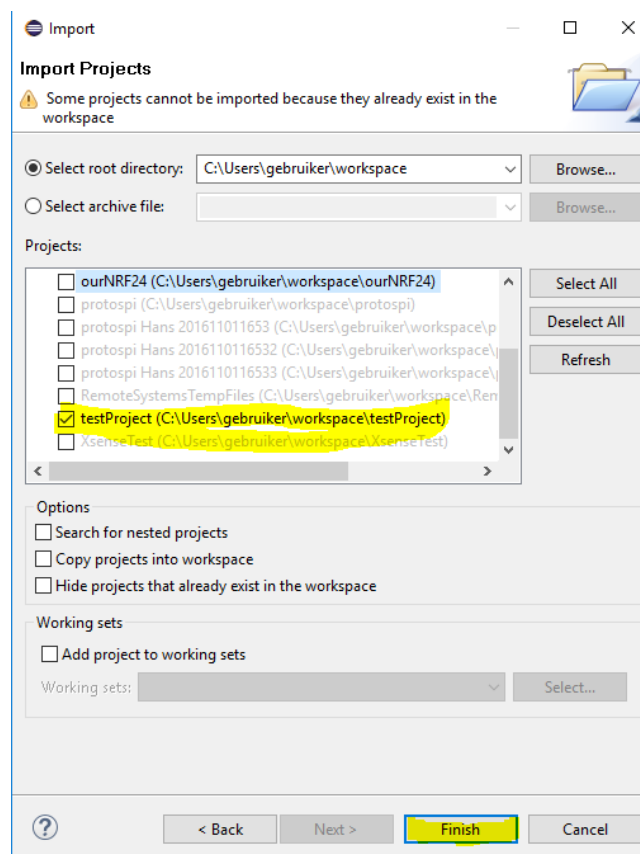
After that, choose “existing project into workspace” and press “next”.



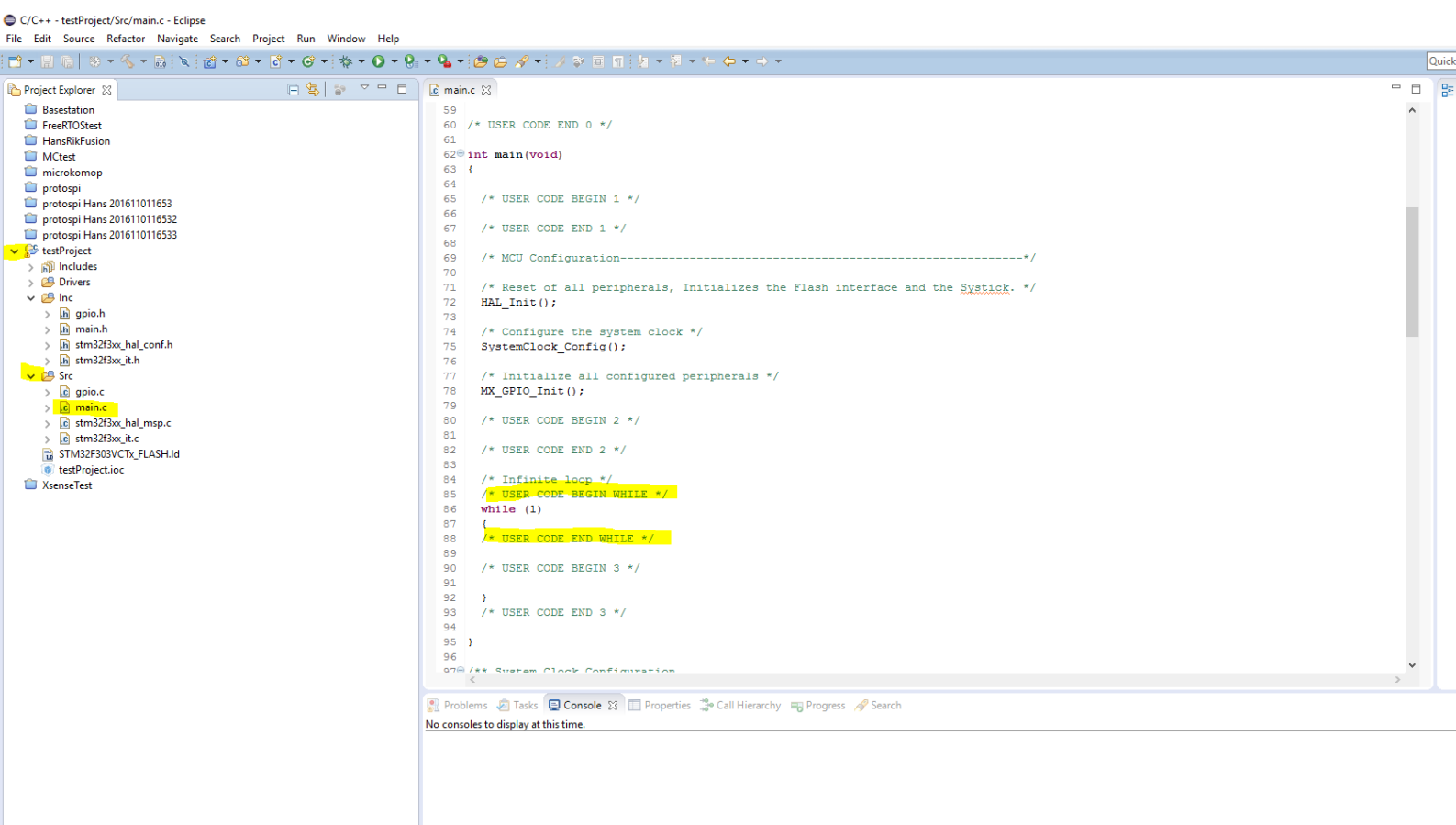
After that, select “select root directory” and press “browse next to it”. Select “workspace” and press “ok”.



Select the project you generated using cube, and press “finish”.



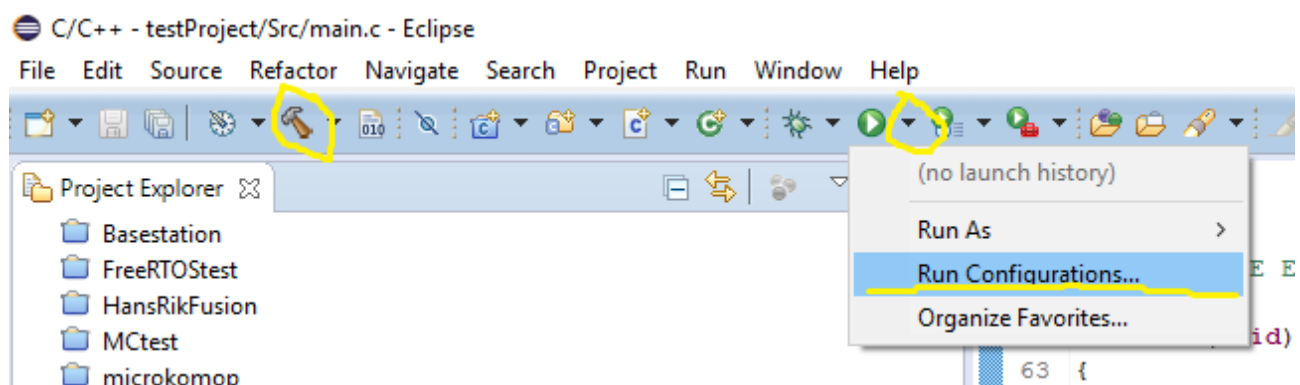
You just opened the project! Cube generated a couple of files for you. You can check out the main.c file by clicking testProject (or your project name) → src → main.c.



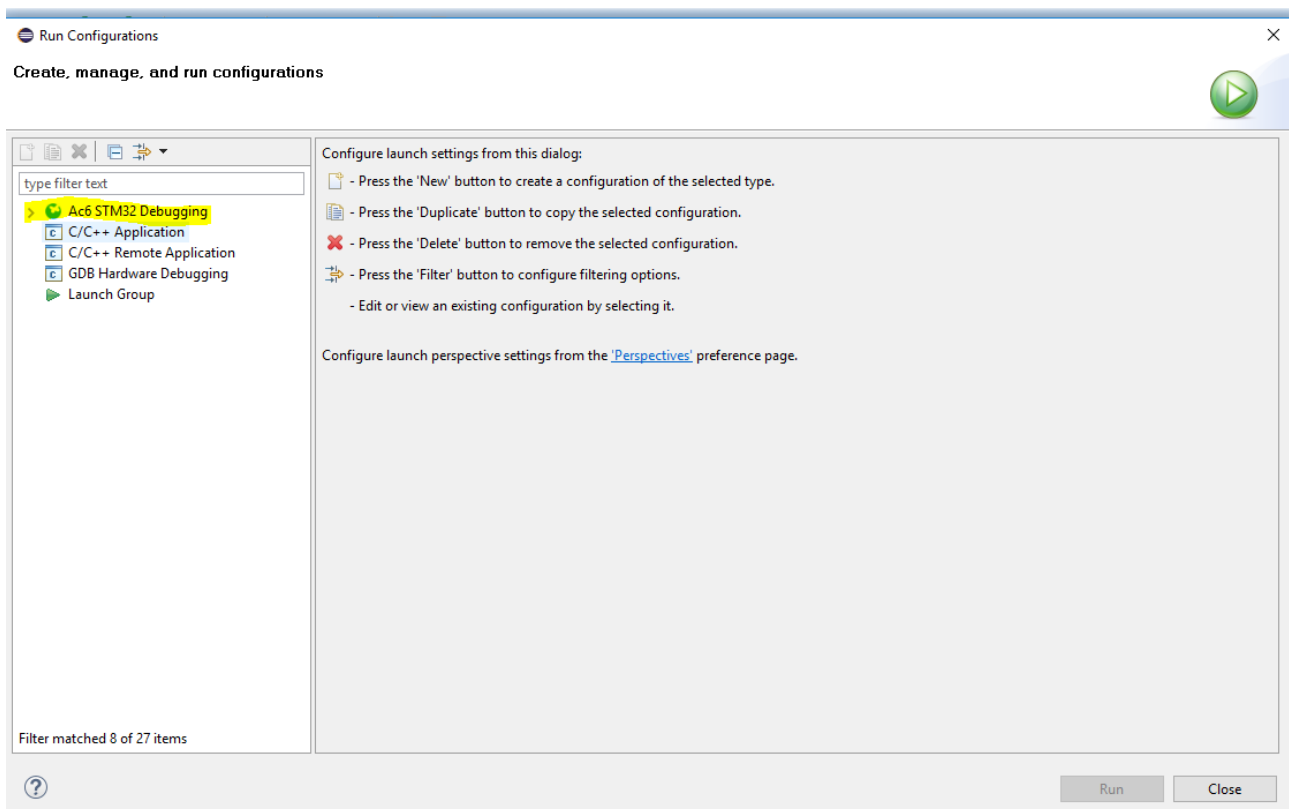
If you look at the main file, you see that cube already wrote a lot of code for initializing the STM32F3. The user may write code between “user code begin” and “user code end”. If code is written in other places, **it will be deleted** if cube generates new code. (And the user will probably use cube to generate new code, e.g. for initializing more pins.)

Uploading code

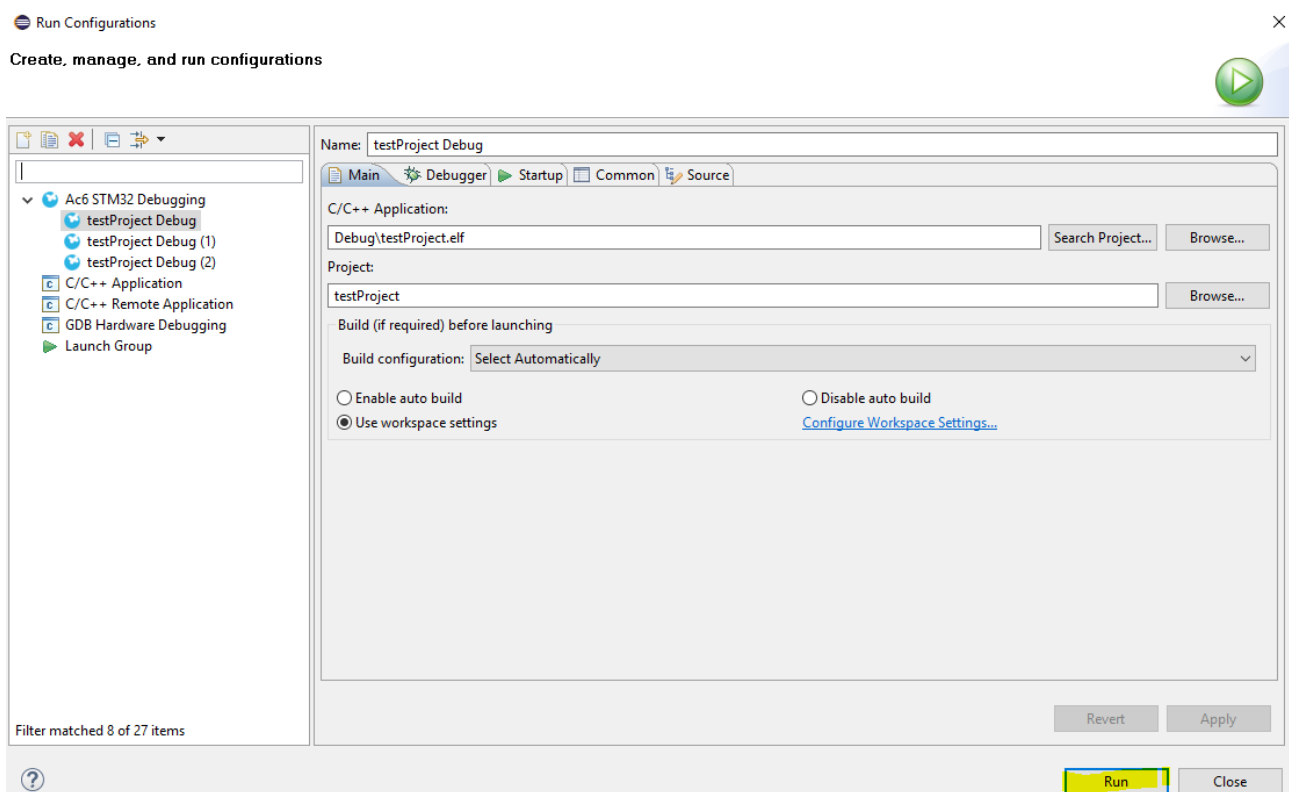
To build your project, press the hammer. After that, connect your discovery board to your computer using a mini-USB cable. Use the USB port in the middle (see picture below). The port at the side can be used for user-applications (we will cover this later). Press the arrow next to the “run” button and choose “run configuration”.

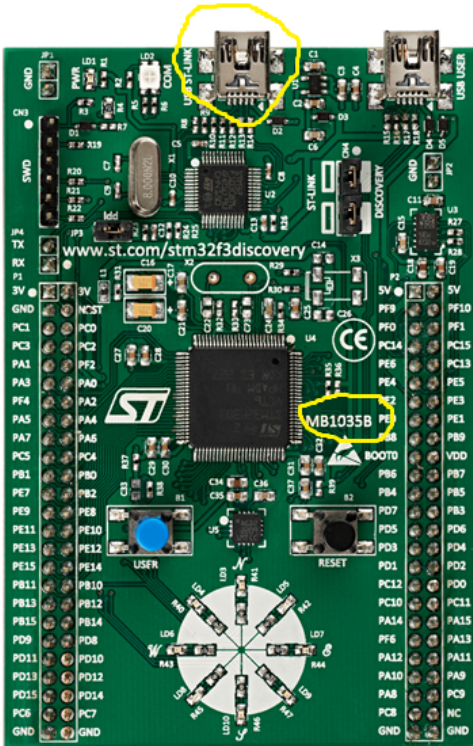


Double-click Ac6STM32 Debugging

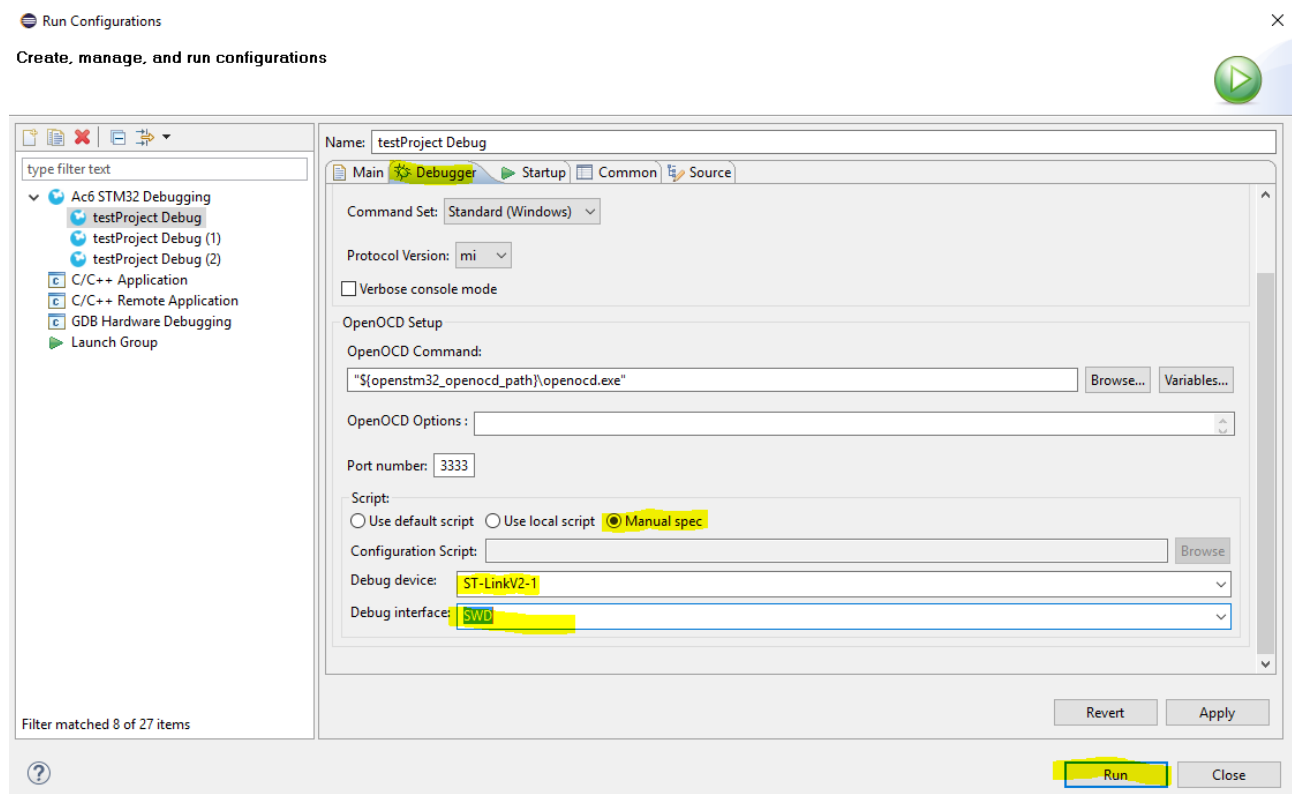


We have three different types of STM-discovery boards. They are almost the same, however, the procedure of uploading code is different. We have a MB1035B, MB1035C and MB1035D. Version B uses the ST Link V2, while version C and D uses ST Link V2-1. The version numbers are on the board (see picture below). Also version B and C/D are distinguishable by them having USB-ports in different shades of gray. For version B, simply press “run”.

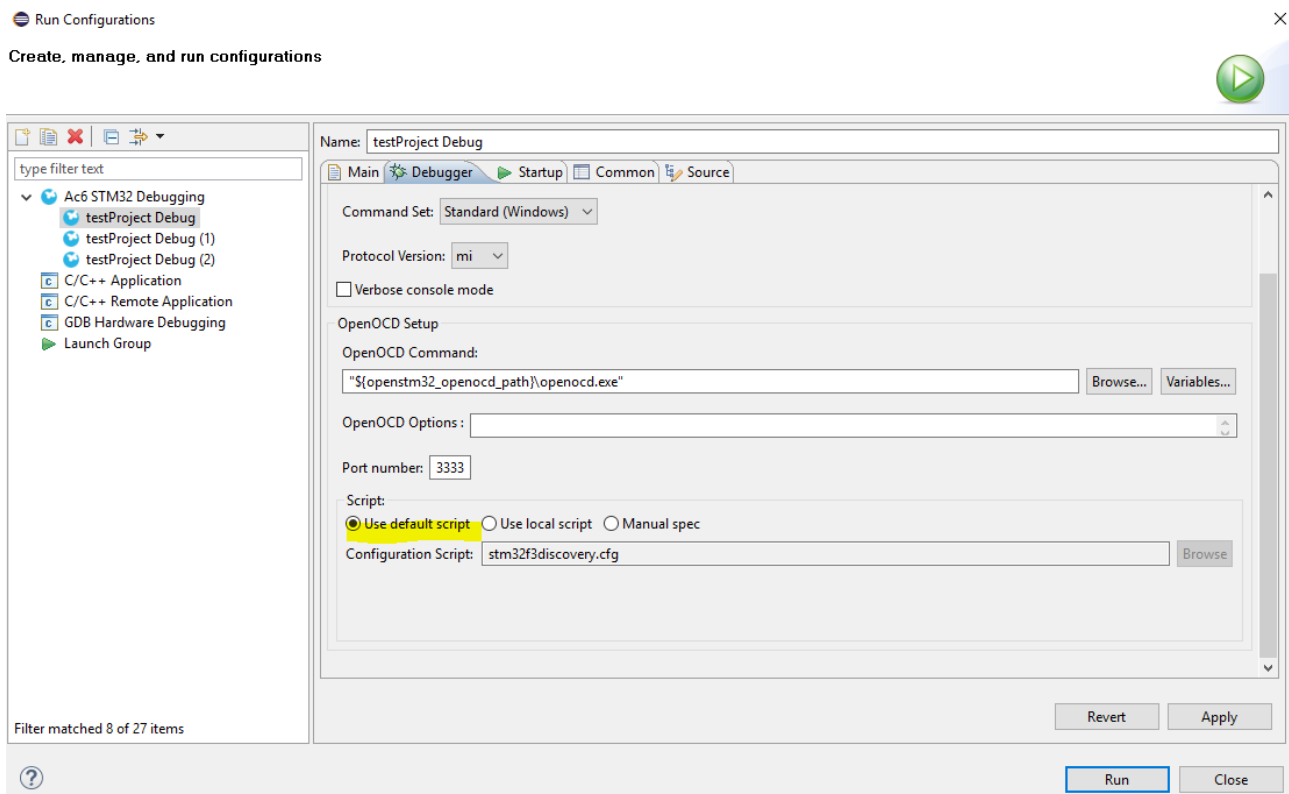




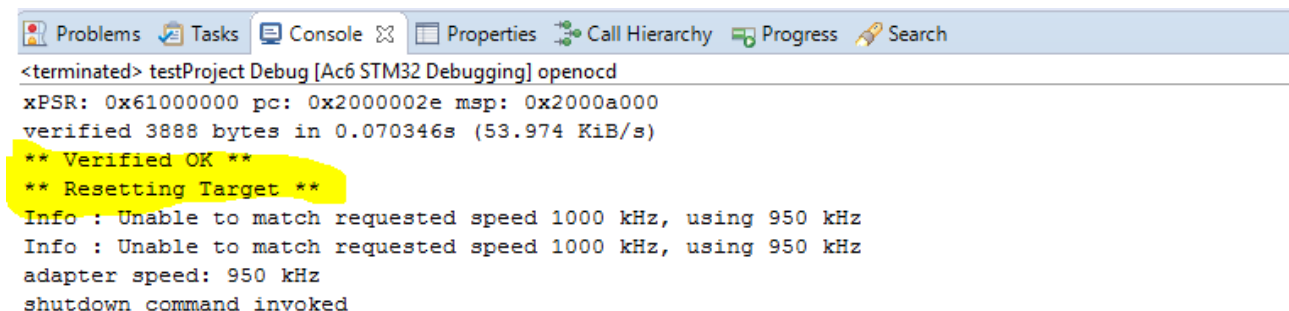
For version C/D go to the “debugger” tab. Than, select “manual spec” and choose “ST-Link v2-1” and “SWD”. Than, press run.



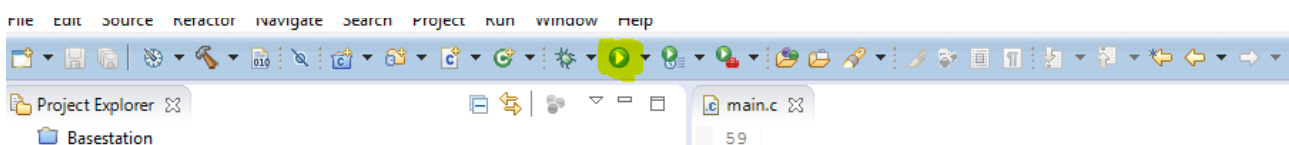
If after doing this one wants to upload to version B again, simply go back to this tab and press “use default script”.



The program should not do anything, but you can see whether or not you successfully uploaded the code in the terminal.



The next time code needs to be uploaded, the green “run” button can simply be pressed.



Opening already created projects

To open an already created project, copy the project into your workspace, and follow the steps starting from “opening st Workbench”.

Basic functionality: GPIO and Delay

The next step is to make a basic program. The program in this example waits until the user presses a button. After that, it turns on and off LEDs in a circle.

In main.c, find “/* USER CODE BEGIN WHILE */” and type the following code:

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */

while(!HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0)){
|
while (1)
{
    HAL_GPIO_WritePin(GPIOE, LD3_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOE, LD4_Pin, GPIO_PIN_SET);
    HAL_Delay(100);
    HAL_GPIO_WritePin(GPIOE, LD4_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOE, LD6_Pin, GPIO_PIN_SET);
    HAL_Delay(100);
    HAL_GPIO_WritePin(GPIOE, LD6_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOE, LD8_Pin, GPIO_PIN_SET);
    HAL_Delay(100);
    HAL_GPIO_WritePin(GPIOE, LD8_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOE, LD10_Pin, GPIO_PIN_SET);
    HAL_Delay(100);
    HAL_GPIO_WritePin(GPIOE, LD10_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOE, LD9_Pin, GPIO_PIN_SET);
    HAL_Delay(100);
    HAL_GPIO_WritePin(GPIOE, LD9_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOE, LD7_Pin, GPIO_PIN_SET);
    HAL_Delay(100);
    HAL_GPIO_WritePin(GPIOE, LD7_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOE, LD5_Pin, GPIO_PIN_SET);
    HAL_Delay(100);
    HAL_GPIO_WritePin(GPIOE, LD5_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOE, LD3_Pin, GPIO_PIN_SET);
    HAL_Delay(100);

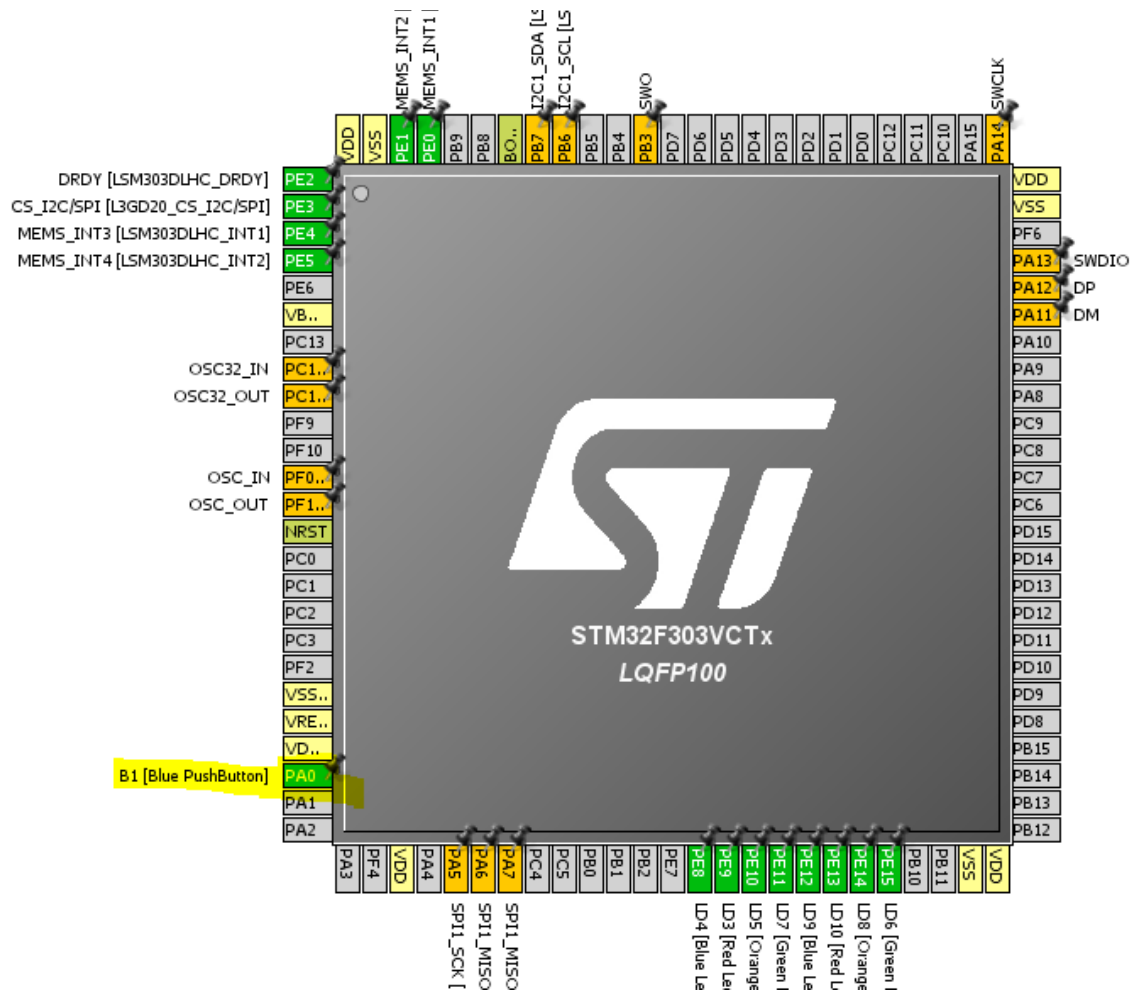
/* USER CODE END WHILE */
```

The command lines and the while(1) are generated by cube. Cube will not delete any code in between “/* USER CODE BEGIN xxx */” and “/*USER CODE END xxx*/”. Cube also generated the while(1) to make an infinite loop the user can use.

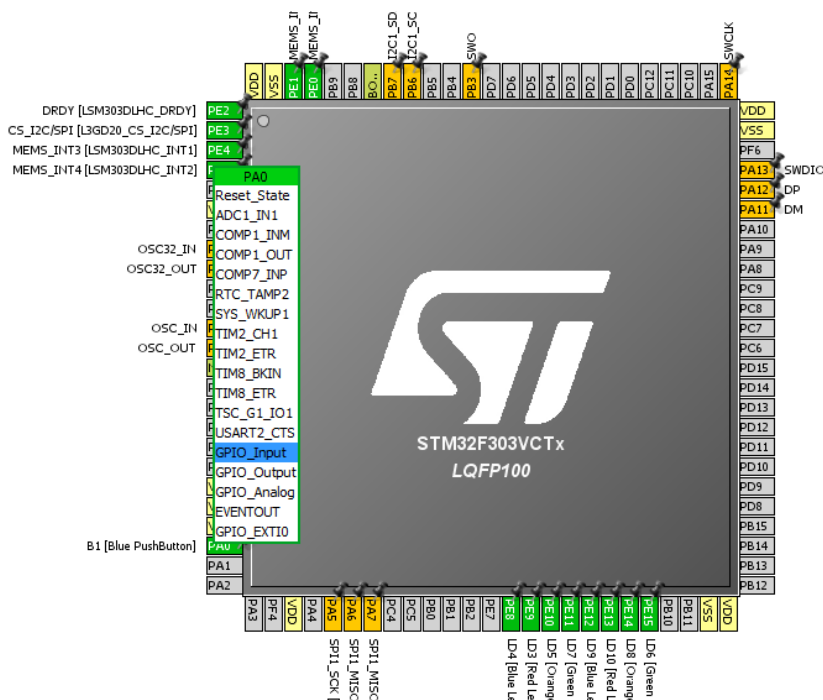
Above the infinite loop, the following line is placed:

```
while(!HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_0)){}
```

This means the program gets stuck in a while-loop, which does nothing, until GPIO pin A0 is high. As made visible in the picture below, when cube generated the project, it initialized the gpio-pin that connects to the blue push-button.



If the green PA0 bottom is clicked, it is visible that it is an input port. Note that the function `HAL_GPIO_Readpin` has two arguments: the first one is the pinbank name, the second one is the pinname.



HAL_Delay(int);
provides a delay in milliseconds.

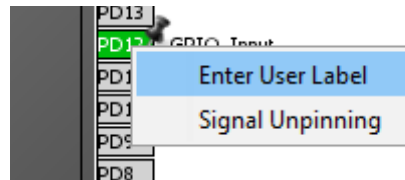
provides a delay in milliseconds.

```
HAL_GPIO_WritePin(GPIOE, LD3_Pin, GPIO_PIN_RESET);
```

```
HAL_GPIO_WritePin(GPIOE, LD3_Pin, GPIO_PIN_RESET);
```

To upload and test the code, use the uploading guide above.

In order to change the user-label, right-click the pin, choose “enter user label” and type something.



The next step is saving the project by clicking the blue disk. **NOTE:** if the “save as” option is chosen, and the project is saved in the same place, it will delete all files in the folder, including all the code typed in ST workbench



After saving the project, press the generate code button.



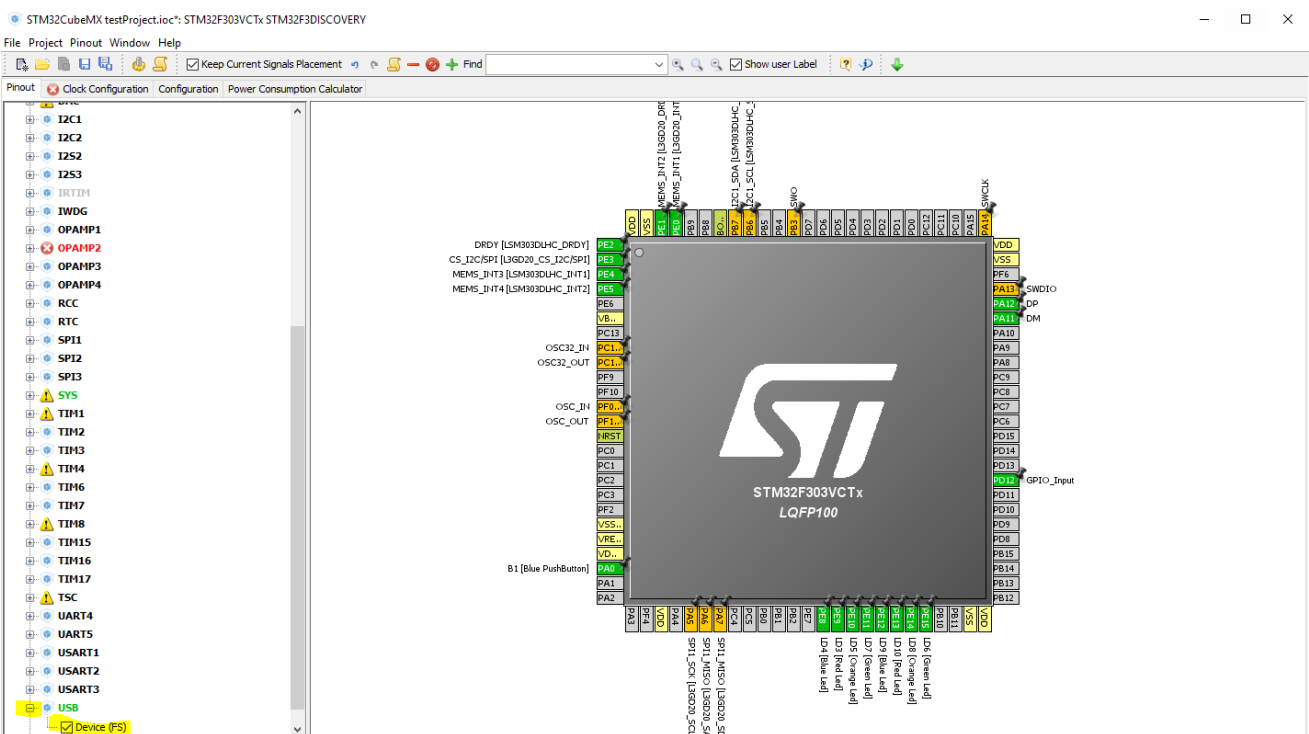
Once the code is generated, use the read and write functions in ST workbench to read or write from the initialized pins.

USB and print to screen

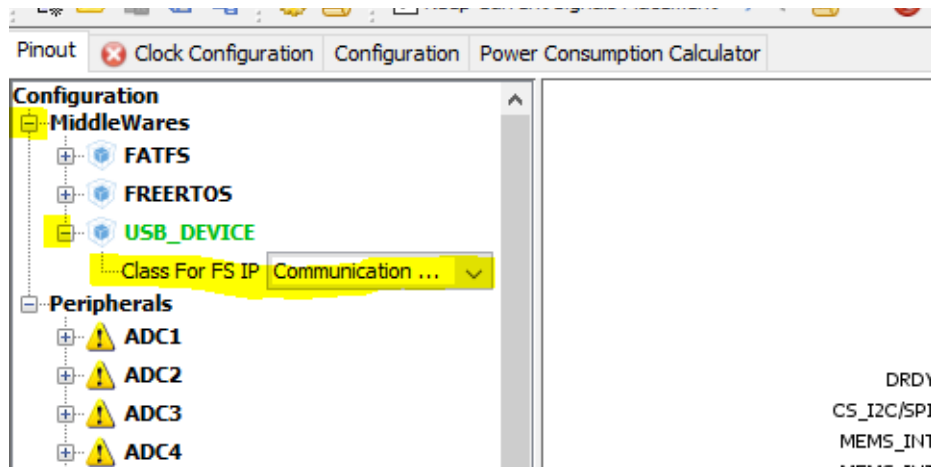
USB is used for communication between the base station and the computer. Furthermore, it is really useful for debugging on the development board. Note that this is not available for the main PCB.

Cube

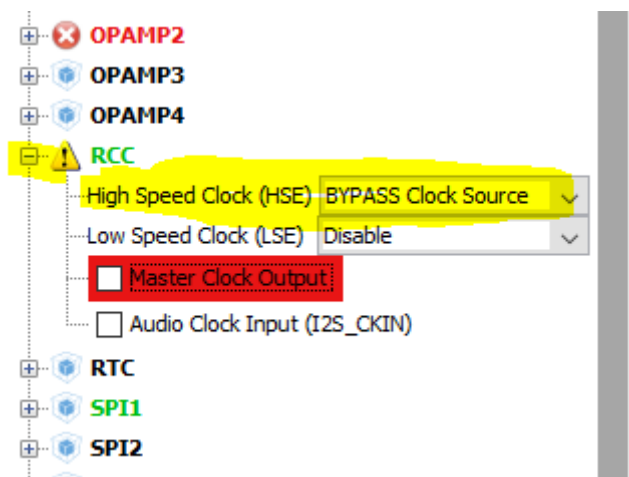
The first step is to use cube for setting up the USB-connection. In cube, go to “USB” and hit the “Device (FS)” button at the left menu.



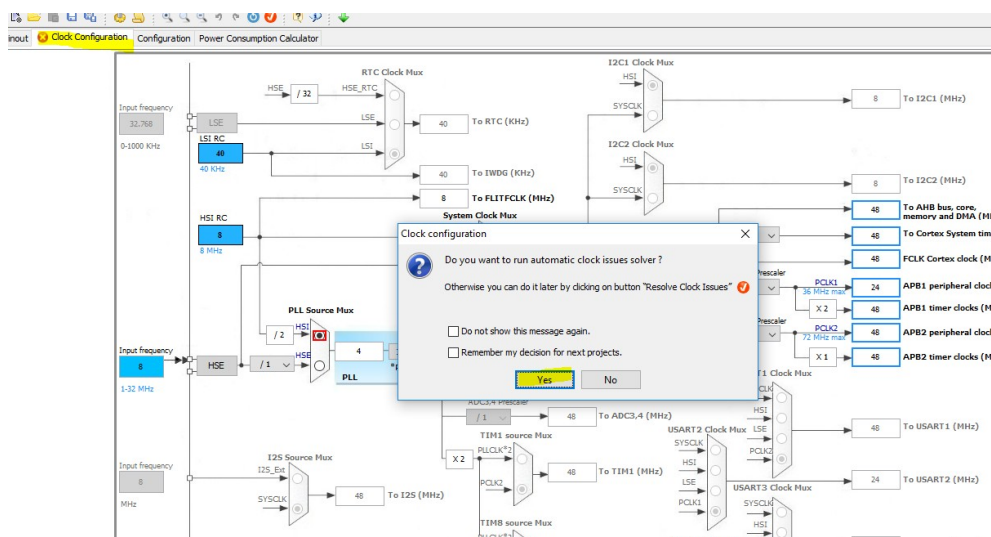
Next, go to the top of this menu, middleware → USB_DEVICE → Class for FS profile and select “communication Device Class (virtual Port Com)”



This action broke the clock configuration. To fix this, go to RCC → High Speed Clock (HSC) and choose “BYPASS clock source” in the dropdown menu.



Next, go to the “clock configuration tab”. Cube will show a window asking you to fix clock issues automatically. Press yes.



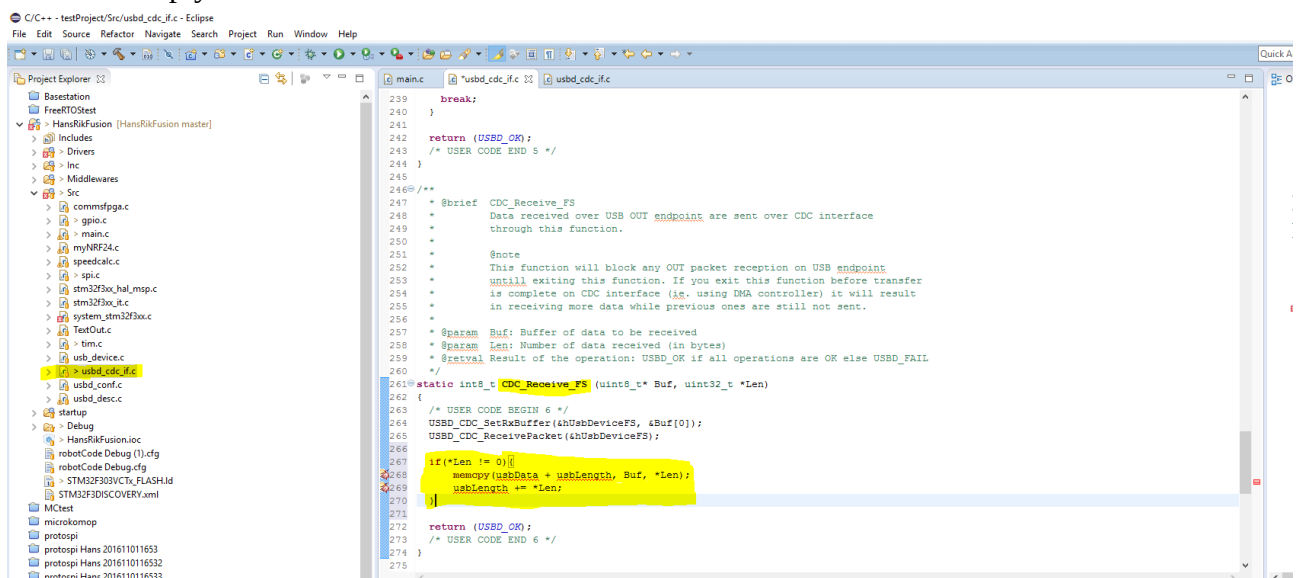
Generate the project as described above, and go to the project in eclips (ST workbench).

Receiving

Cube generated a couple of files, among which “usb_cdc_if.c” in the src map, and “usb_cdc_if.h” in the inc map. If these files are not visible, refresh the project. These files are libraries to use USB. However, the user the handling of input and output should be handled by the user. Open “usb_cdc_if.c” and find the function “CDC_Receive_FS” and add the following lines:

```
if(*Len != 0){
    memcpy(usbData + usbLength, Buf, *Len);
    usbLength += *Len;
}
```

This will empty the internal buffer of the USB to our own external buffer “usbData”



“usbData” and usbLength are variables that are not generated by cube, so they need to be declared. It is desired to use them outside “usb_cdc_if.c”, so it should be extern variable. Remember that code can only be typed in the “user code” blocks. In “usb_cdc_if.c”, find “/* USER CODE BEGIN PRIVATE_VARIABLES */” and add:

```
int usbLength = 0;

uint8_t usbData[64];
```

```
/* USER CODE BEGIN PRIVATE_VARIABLES */
int usbLength = 0;
uint8_t usbData[64];
/* USER CODE END PRIVATE_VARIABLES */
```

In “usb_cdc_if.h” find “/* USER CODE BEGIN EXPORTED_VARIABLES */” and add:

```
extern int usbLength;  
  
extern uint8_t usbData[64];
```

```
99  
100 /* USER CODE BEGIN EXPORTED_VARIABLES */  
101 extern int usbLength;  
102 extern uint8_t usbData[64];  
103 /* USER CODE END EXPORTED_VARIABLES */  
104
```

Lastly, go to main, and find “USER CODE BEGIN INCLUDES” and include “usb_cdc_if.h”

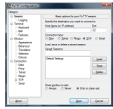
```
- -----  
43 /* Includes -----  
44 #include "main.h"  
45 #include "stm32f3xx_hal.h"  
46 #include "usb_device.h"  
47 #include "gpio.h"  
48  
49 /* USER CODE BEGIN Includes */  
50 #include "usbd_cdc_if.h"
```

Now, the stm should be able to read from USB. To test this, go to the infinite loop in the main file and type the following snippet:

```
/* Infinite loop */  
/* USER CODE BEGIN WHILE */  
while (1)  
{  
    HAL_Delay(1);  
    if(usbLength != 0){  
        if(usbData[0] == 't'){  
            HAL_GPIO_TogglePin(GPIOE, GPIO_PIN_9);  
        }  
        usbLength = 0;  
    }  
}  
/* USER CODE END WHILE */
```

Note that the 1 microsecond delay is necessary if the loop is otherwise empty. It is not found why this is the case. Probably it has something to do with emptying buffers.

Now, download Putty, if you didn't already have it. <http://www.putty.org/>



Download PuTTY

PuTTY is an SSH and telnet client, developed originally by Simon Tatham for the Windows platform. PuTTY is open source software that is available with source code and is developed and supported by a group of volunteers.

You can download PuTTY [here](#).

Below suggestions are independent of the authors of PuTTY. They are *not* to be seen as endorsements by the PuTTY project.

Bitvise SSH Client

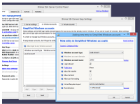


Bitvise SSH Client is an SSH and SFTP client for Windows. It is developed and supported professionally by Bitvise. The SSH Client is robust, easy to install, easy to use, and supports all features supported by PuTTY, as well as the following:

- graphical SFTP file transfer;
- single-click Remote Desktop tunneling;
- auto-reconnecting capability;
- dynamic port forwarding through an integrated proxy;
- an FTP-to-SFTP protocol bridge.

Bitvise SSH Client is **free to use**. You can [download it here](#).

Bitvise SSH Server



Bitvise SSH Server is an SSH, SFTP and SCP server for Windows. It is robust, easy to install, easy to use, and works well with a variety of SSH clients, including Bitvise SSH Client, OpenSSH, and PuTTY. The SSH Server is developed and supported professionally by Bitvise.

You can [download Bitvise SSH Server here](#).

Using the standard installer is the easiest, and you probably need the 64-bit version

Package files

You probably want one of these. They include all the PuTTY utilities.

(Not sure whether you want the 32-bit or the 64-bit version? Read the [FAQ entry](#).)

MSI ('Windows Installer')

32-bit:	putty-0.69-installer.msi	(or by FTP)	(signature)
64-bit:	putty-64bit-0.69-installer.msi	(or by FTP)	(signature)

Unix source archive

.tar.gz:	putty-0.69.tar.gz	(or by FTP)	(signature)
----------	-----------------------------------	-----------------------------	-----------------------------

Alternative binary files

The installer packages above will provide all of these (except PuTTYtel), but you can download them one by one if you prefer.

(Not sure whether you want the 32-bit or the 64-bit version? Read the [FAQ entry](#).)

putty.exe (the SSH and Telnet client itself)

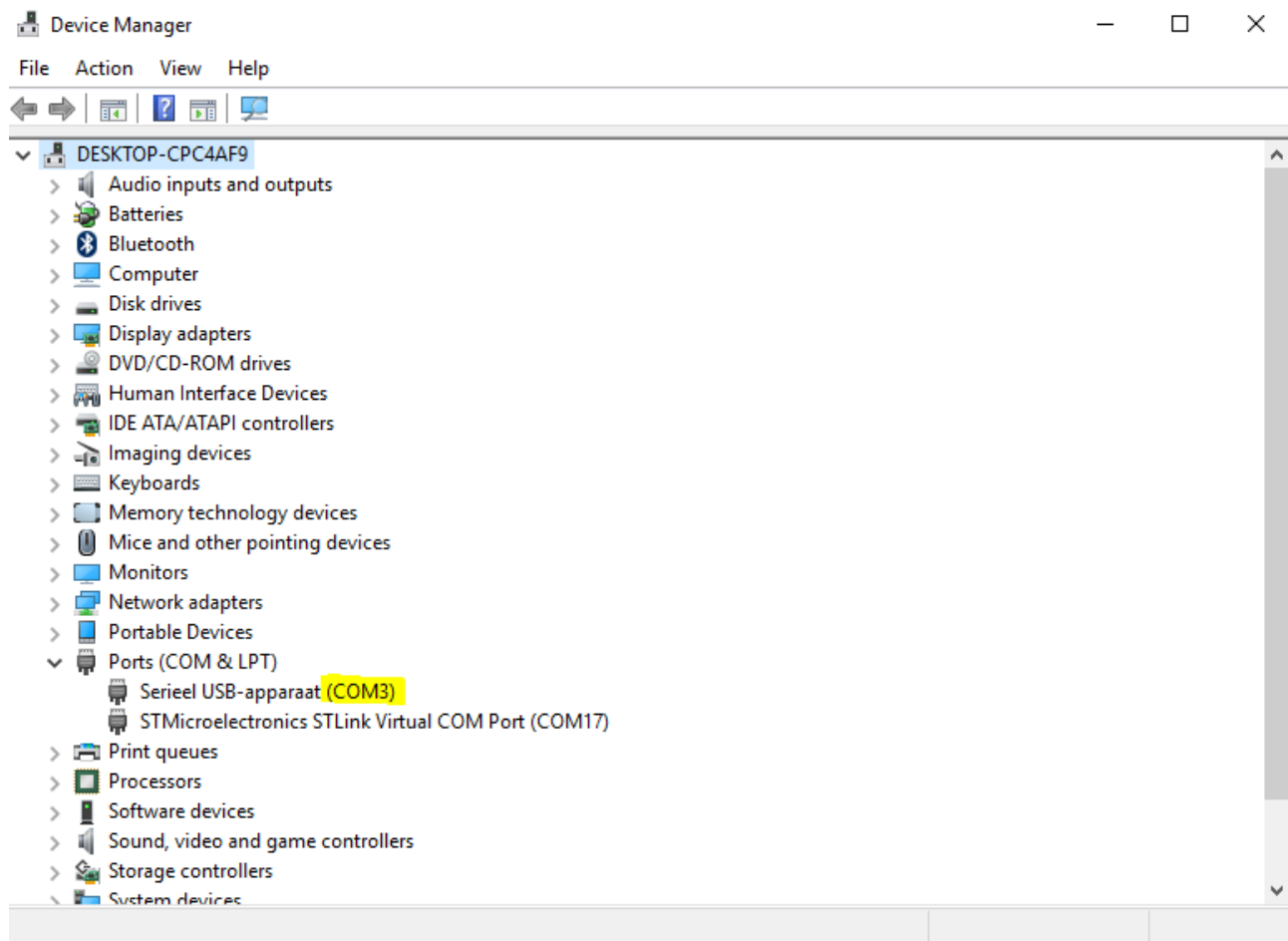
32-bit:	putty.exe	(or by FTP)	(signature)
64-bit:	putty.exe	(or by FTP)	(signature)

pscp.exe (an SCP client, i.e. command-line secure file copy)

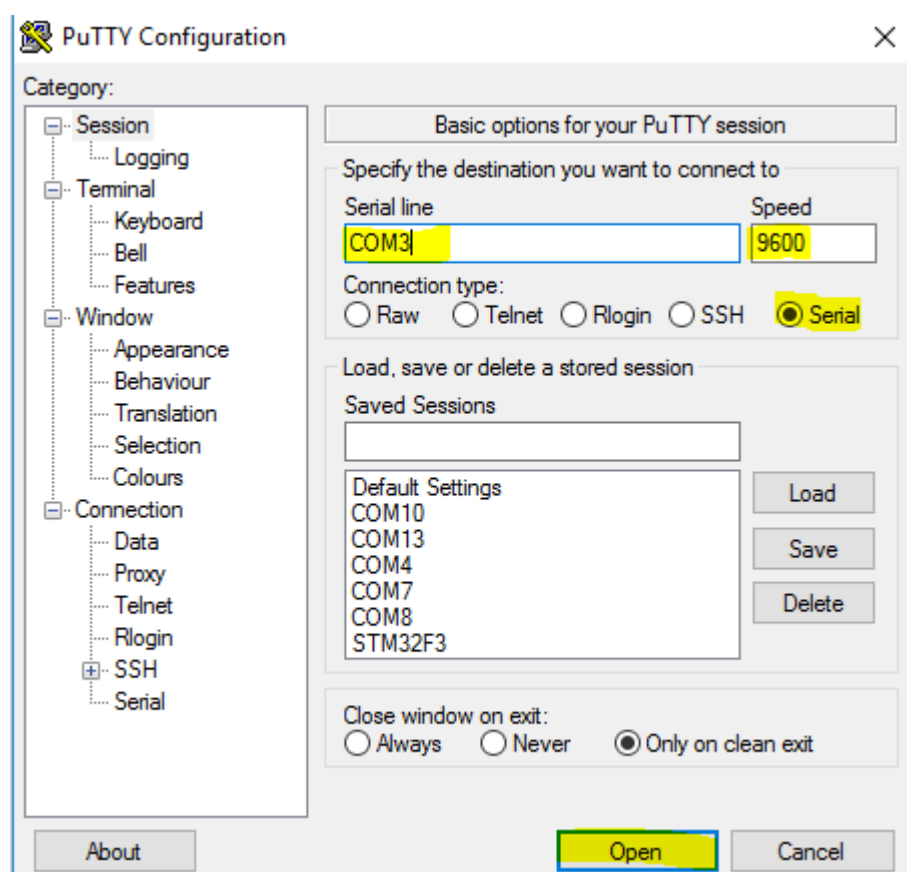
32-bit:	pscp.exe	(or by FTP)	(signature)
---------	--------------------------	-----------------------------	-----------------------------

The next step is to connect the user-USB of the Discovery. The user-USB is the USB-port that is not used for uploading code. If version C or D of the Discovery ("uploading code") is used, the board needs to be powered, either via the USB-port for uploading code, or the 3 or 5 volt input, before the user-USB is connected. Also, code should already be present, and the line "MX_USB_DEVICE_Init();" should be executed before the user-USB is connected. This means that if new code is uploaded to the discovery, the user-USB needs to be plugged out and in again.

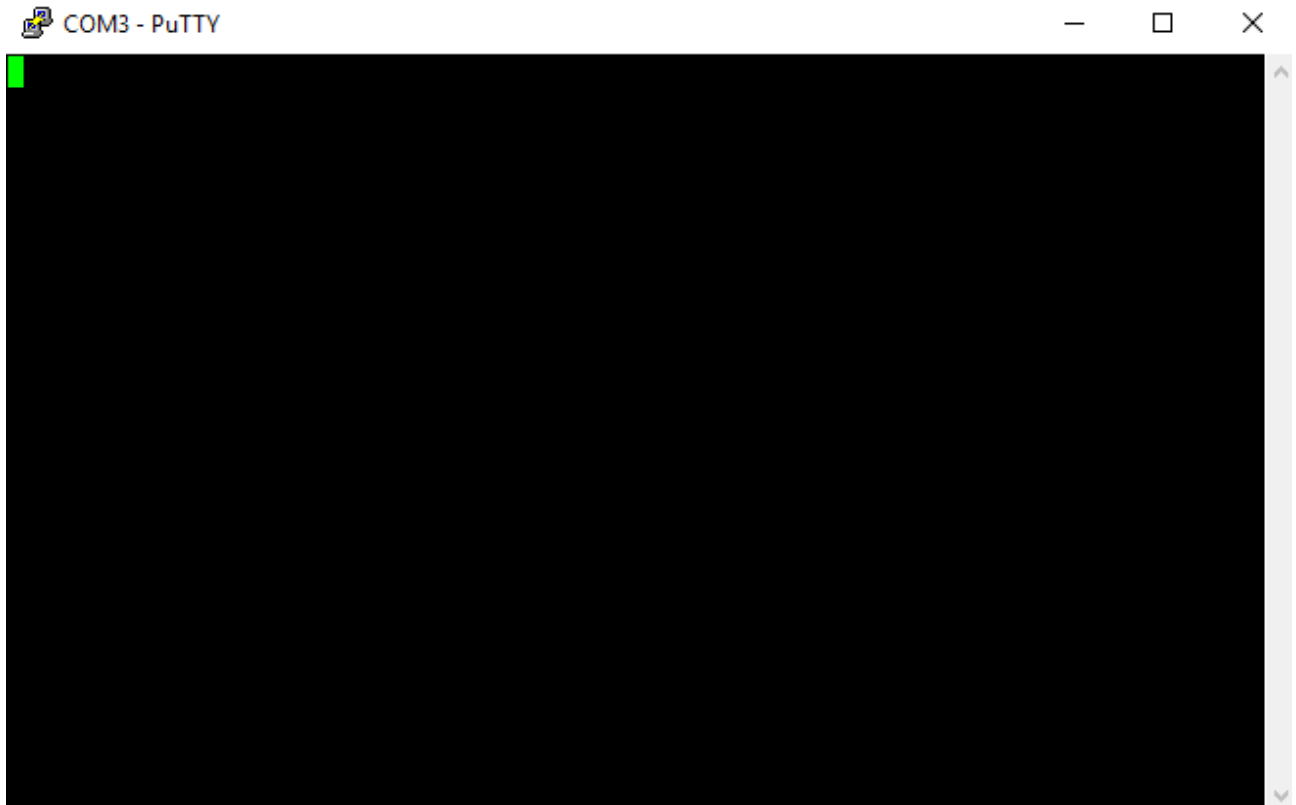
Now the Discovery is connected, it is needed to discover what COM-port it uses. To do that, go to “device manager” (apparaatbeheer in Dutch) in windows. Click “Ports (COM and LTP)” and find “serieel USB-apparaat”. The number behind it is the COM-port the stm uses.



Now go to Putty. Choose the “serial” connection type. Type COMx (with x the number of you COM-port) as your serial line, and pick a baute-rate of 9600. Then, press “Open”.



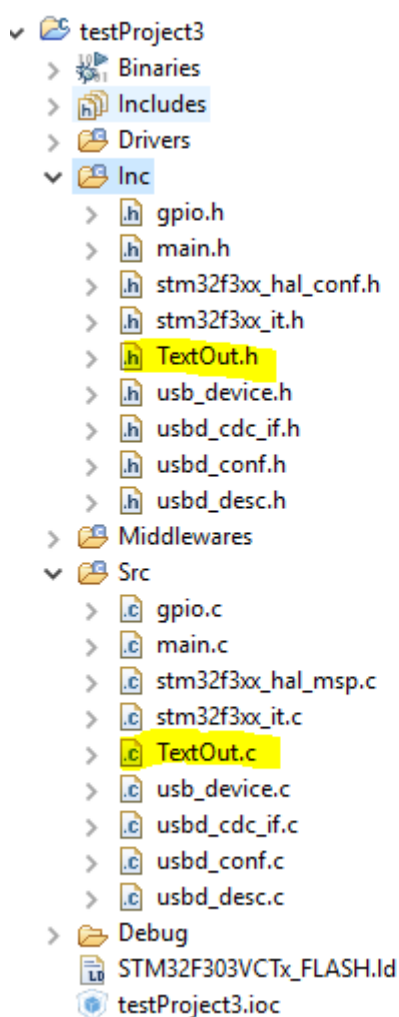
Now, you should get a black screen like this:



If you press 't', nothing on the screen should happen, but putty sends the 't' over USB to the STM. This should toggle one of Led 3 on the board.

Sending

Next, sending text to the terminal should be printed. To do this, copy TextOut.c to the scr folder of the project, and TextOut.h to the inc folder. This can be done in the left menu of eclipse.



TextOut contains three function: TextOut, backslashNfixer and HexOut. HexOut outputs hexadecimal numbers over USB, and can be used for debugging Linux devices. TextOut outputs chars represented in ASCII. BackslashNfixer puts a /r after a /n. If it was to be omitted, the terminal would tab after each newline.

TextOut can be used directly if a plain string is used. If variables needs to be outputed, sprintf should be used to copy a string to the smallStrBuffer, which needs to be put in TextOut. Both methods are demonstrated in the example below.

First include TextOut.h and <string.h> Note that usb_cdc_if.h is included in TextOut.h and therefore does not need to be included here.

```
/* USER CODE BEGIN Includes */  
#include "TextOut.h"  
#include <string.h>  
  
/* USER CODE END Includes */
```

Now type the following snippet in the infinite while loop:

```

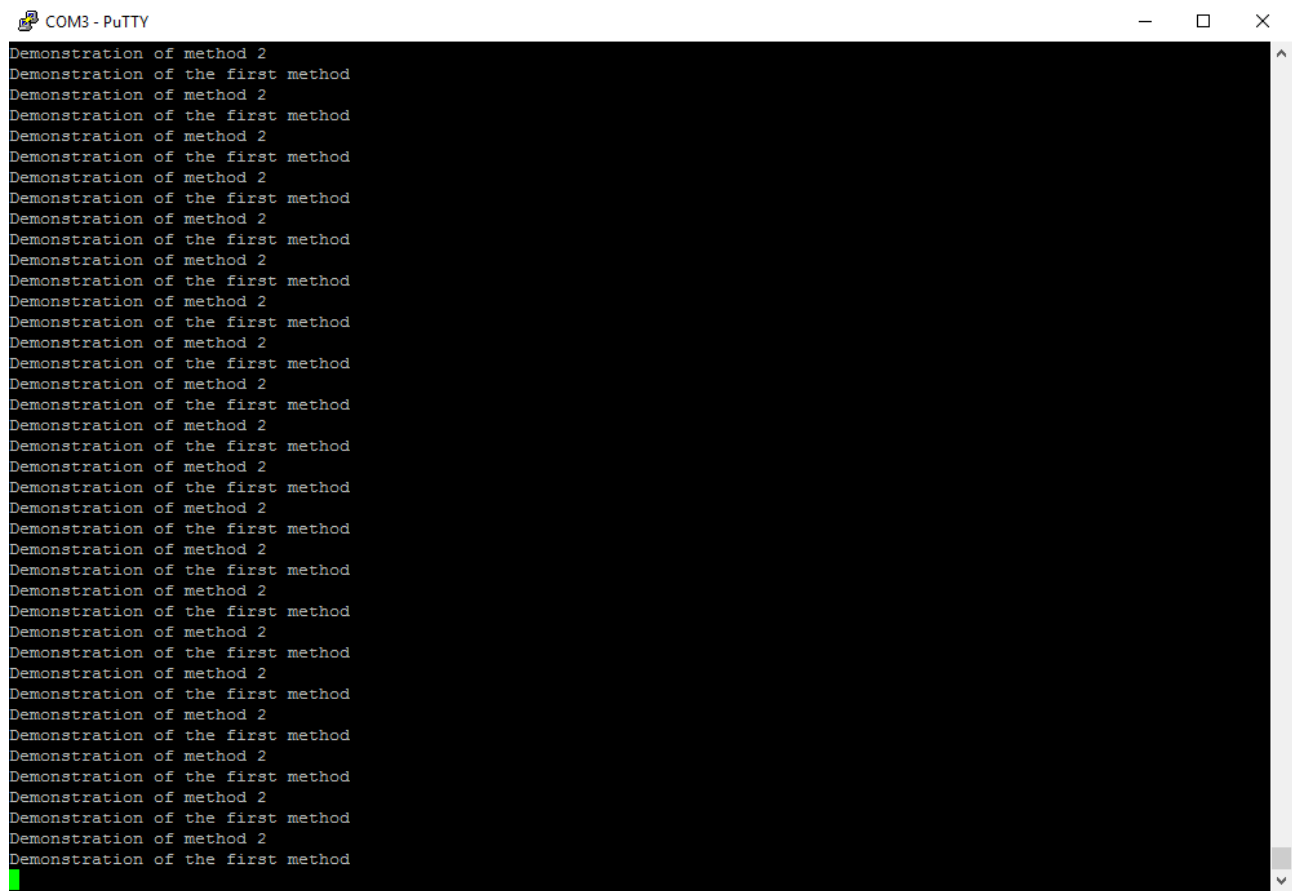
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    TextOut("Demonstration of the first method\n");

    int demoNumber = 2;
    sprintf(smallStrBuffer, "Demonstration of method %i\n", demoNumber);
    TextOut(smallStrBuffer);
/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */

```

If putty is opened via the method above, the output should look like this.



The screenshot shows a PuTTY window titled 'COM3 - PuTTY'. The window contains a repeating pattern of two lines of text: 'Demonstration of method 2' followed by 'Demonstration of the first method'. This pattern is repeated approximately 25 times, filling the window. The text is white on a black background. The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

Known issues

TextOut is not implemented in a very precise way. First of all, the smallStrBuffer is only 1024 bytes long, which means that longer strings can not be outputted. Secondly, \n will be immediately followed by \r\0, which means it can only be used at the end of a string. Thirdly, if no \n is used no zero terminator is added, which means that

```
sprintf(smallStrBuffer, "This is amazing\n");
```

```
sprintf(smallStrBuffer, "Rick");
```

```
TextOut(smallStrBuffer);
```

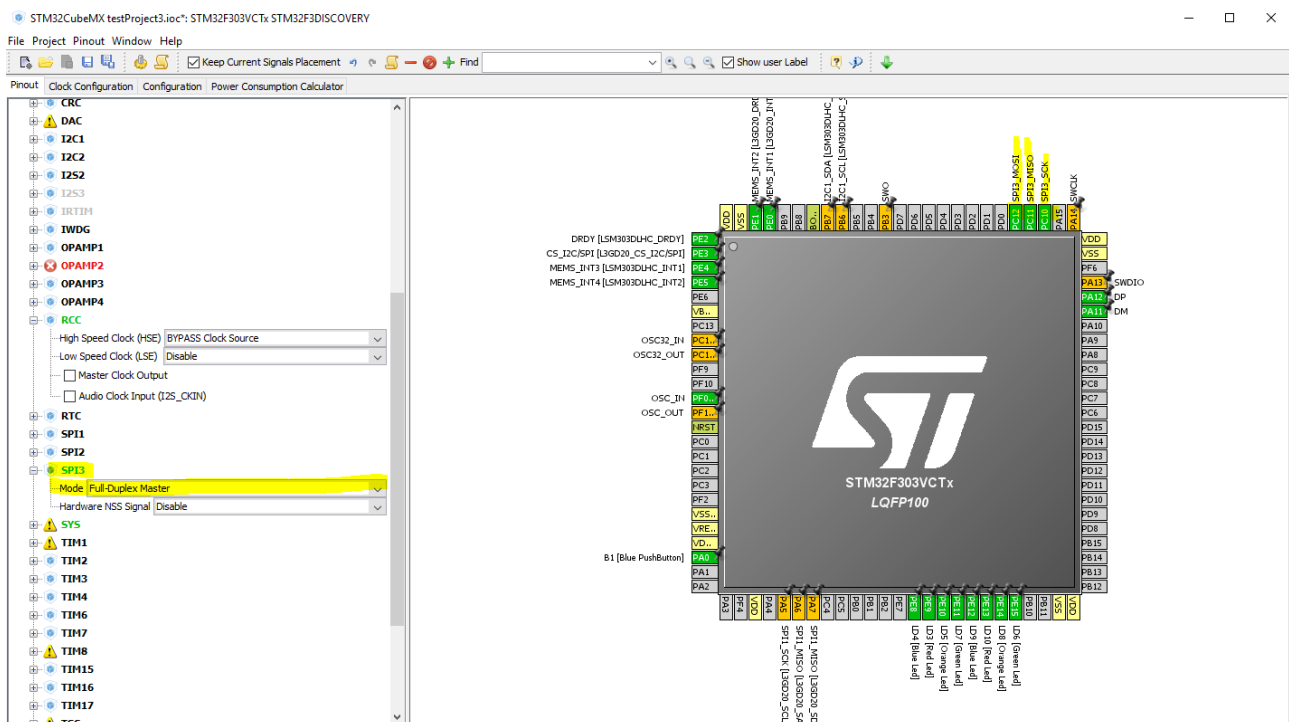
Will result in "Rick is amazing \n\r\0" instead of "Rick\0".

The second two problems can be fixed by putting more though in BackslashNfixer, but it can also just be avoided.

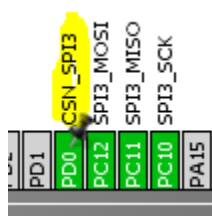
SPI

SPI is a communication protocol. It is used for communication between the microcontroller and the FPGA, and between the microcontroller and the communication chip. Compared to I2C, SPI had less communication overhead, but it uses a slave select line for each slave, which makes it inappropriate for systems with lots of slaves. For the internal working, read [wikipedia](https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus). For debugging purposes, SPI can be measured with a scope. Knowing how the protocol works is useful. Read wikipedia.

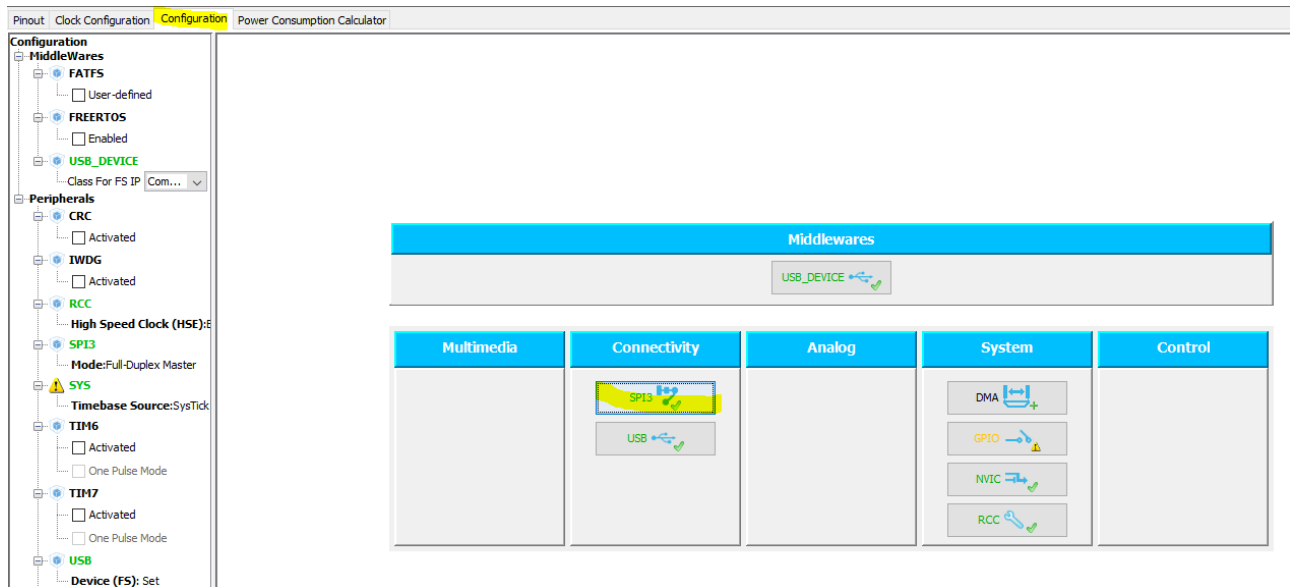
This tutorial will show how to connect it to the NRF-chip. We will write to and read from a register. More information about the NRF24- chip will be given later in this document. Communicating with the FPGA works in a similar fashion. First go to cube. In the left menu choose SPI3 → full duplex master. Keep “hardware NSS signal disabled”, we will do that manually. Note that this automatically created 3 outputs pins in the picture below:



Now, a couple of extra pins are needed for the NRF24 chip to work. First the slave select pin (called NSS or SS or CSN), as GPIO output. Secondly a chip enable (CE) pin as GPIO output. This pin is used by the NRF to diable or enable the antenna. Thirdly, an IRQ pin as GPIO input. This gives interrupts when data is sended or received. Because this is a SPI tutorial, only the SPI part will be covered, so the later two are not nessecary. Make PD0 a gpio-Output pin, and call it “CSN_SPI3” using the “user label option” (right-click on PD0 after making it an output pin);



Now, go to the configuration tab, and then to SPI3.



Put data size to 8-bit and press ok. Generate code and open the project in st workbench (eclipse).

There are three important SPI-functions:

```
HAL_SPI_Transmit(spiHandle, &sendData, 1, 100);
```

```
HAL_SPI_Receive(spiHandle, &receiveData, 1, 100);
```

```
HAL_SPI_TransmitReceive(spiHandle, &sendData, &receiveData, 1, 100);
```

Where the first tells the function what spi to use, the sendData and receiveData are pointers to arrays from where to read or where to write to, 1 is the amount of data send/received (in bytes) and 100 is the timeout in milliseconds.

For SPI to work, it is needed to put the slave select low before sending/receiving as the master, and pulling it up again when sending/receiving is done.

In chapter 8.3.1 and 8.3.2 of the [manual](#) of the NRF24, it is described how to write data to a register. To write to a register, write 000A AAAA where AAAA is the register number in binary, followed by the data you want to write to that register. To read, write 001A AAAA followed by a clock signal to read the data. Go to main.c and find the infinite loop. Write the code below (it is assumed that you have USB working, see above.) It will read the USB input. If it is a number between 0 and 9, it will write that to register 5. If it is a space, it will read register 5 and output it to the terminal

```

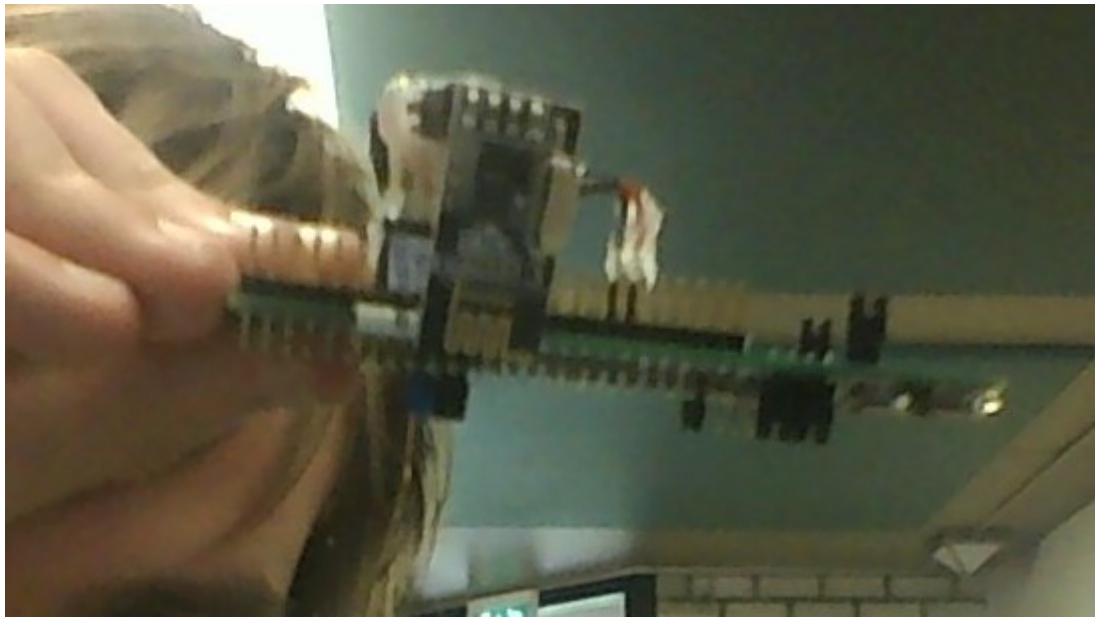
100  /* Infinite loop */
101  /* USER CODE BEGIN WHILE */
102  while (1)
103  {
104      //if there is something in the USB-buffer
105      if(usbLength != 0){
106          //for every byte in the USB buffer
107          for(int i = 0; i < usbLength; i++){
108              //if there is a number between 0 and 9 in the buffer,
109              if(usbData[i] <= '9' && usbData[i] >= '0'){
110                  //put the slave select low to start the communication
111                  HAL_GPIO_WritePin(GPIOD, CSN_SPI3_Pin, GPIO_PIN_RESET);
112
113                  //write command = 000A AAAA where AAAA is the register to write to.
114                  //in this case, we will write to register 5 so AAAA = 00101
115                  //total command is 0000 0101 or 0x05 in hex.
116                  //note that for numbers above 15, the implementation 0x0Y - where Y is the register number - will not work
117                  uint8_t command = 0x05;
118
119                  //write the command. This is 1 byte, and we use a timeout of 100 ms.
120                  //&hspi3 is a pointer to a struct cube uses to store information about SPI.
121                  HAL_SPI_Transmit(&hspi3, &command, 1, 100);
122
123                  //transmit the received number to put in the register
124                  HAL_SPI_Transmit(&hspi3, &usbData[i], 1, 100);
125
126                  //pull the slave select low to stop communication
127                  HAL_GPIO_WritePin(GPIOD, CSN_SPI3_Pin, GPIO_PIN_RESET);
128              }
129
130              //if there is a space in the buffer
131              else if(usbData[i] == ' '){
132                  //put the slave select low to start the communication
133                  HAL_GPIO_WritePin(GPIOD, CSN_SPI3_Pin, GPIO_PIN_RESET);
134
135                  //write command = 001A AAAA where AAAA is the register to read from.
136                  // = 0010 0101 for reg 5; = 0x25 in Hex
137                  uint8_t command = 0x05;
138
139                  //write the command. This is 1 byte, and we use a timeout of 100 ms.
140                  //&hspi3 is a pointer to a struct cube uses to store information about SPI.
141                  HAL_SPI_Transmit(&hspi3, &command, 1, 100);
142
143                  //reserve space to write the output of the read command to
144                  uint8_t data;
145
146                  //send a clock and read the data from the NRF24
147                  HAL_SPI_Receive(&hspi3, &data, 1, 100);
148
149                  //pull the slave select low to stop communication
150                  HAL_GPIO_WritePin(GPIOD, CSN_SPI3_Pin, GPIO_PIN_RESET);
151
152                  //output data to screen
153                  sprintf(smallStrBuffer, "register 5 = %i\n", data);
154                  TextOut(smallStrBuffer);
155              }
156
157              //reset the buffer counter because we handled the input
158              usbLength = 0;
159          }
160      }
161  }
162  /* USER CODE END WHILE */

```

Now connect the nrf-chip to the discovery using pin pinout in cube, and the following picture. Note that CE and IRQ may be left floating.



We have headers that do this automatically for you.



Now, connect the board to your computer and test the program.

