UMSAE Software Tutorials

Contents

1 Introduction 3
- 1.1 Software Systems . . . 3
  - 1.1.1 Vehicle Control Software (VCS) . . . 3
  - 1.1.2 Data Acquisition Lead (DAQ) . . . 3
  - 1.1.3 Software Testing Lead (SWT) . . . 3
- 1.2 Vehicle Control Systems . . . 3
  - 1.2.1 Accumulator Control Unit (ACU) . . . 3
  - 1.2.2 Vehicle Control Unit (VCU) . . . 4
  - 1.2.3 Sensor Control Unit (SCU) . . . 4
  - 1.2.4 Motor Controller (MC) . . . 4
  - 1.2.5 Battery Management System (BMS) . . . 4
  - 1.2.6 Driver Display (Dash) . . . 4
- 1.3 Downloading CubeMx and CubeIDE . . . 4
  - 1.3.1 CubeMx . . . 4
  - 1.3.2 CubeIDE . . . 4
- 1.4 Hardware . . . 4

2 Tutorial Projects 5

3 Tutorial 1: LED Blink 5

4 Tutorial 2: HAL USART 10

5 Tutorial 3: HAL UART Interrupts 12

6 Tutorial 4: HAL FreeRTOS 13

1 Introduction

Hi! Welcome to the UMSAE Formula Electric Software System! We have created the following tutorials so new members can learn some of the basics to start working on our vehicle. This document outlines the structure of the software system, provides some common terminology we use on our team, and walks you through each tutorial. The earlier you finish the tutorials, the earlier we can give you a task! If you need help with any of the tutorials the Software System Officer (SSO), Vehicle Control Software Lead (VCS), Data Acquisition Lead (DAQ), or Software Testing Lead (SWT) for help, and we'll guide you in the right direction!

1.1 Software Systems

The UMSAE Formula Electric Software System is broken down into multiple sections. Each section has a team lead, and they are all overseen by the Software Systems Officer (SSO).

- Vehicle Control Systems (VCS)
- Data Acquisition Lead (DAQ)
- Software Testing Lead (SWT)

As of 2025, these are our software team leads!

- Systems Officer (SSO) → Caleb Pollreis
- Vehicle Control Systems (VCS) → Mason Pronger
- Data Acquisition Lead (DAQ) → Ethan Alexander
- Software Testing Lead (SWT) → Evan Mack

1.1.1 Vehicle Control Software (VCS)

This section is responsible for writing all the embedded software to control the car, which includes the following:

- Accumulator Control Unit (ACU)
- Vehicle Control Unit (VCU)
- Motor Controller (MC)
- Battery Management System (BMS)
- Driver Display (Dash)

1.1.2 Data Acquisition Lead (DAQ)

This section is responsible for gathering and transferring data from the car. Our goal is to log data from the our Sensor Control Unit (SCU), with the data logger, to validate other systems. A future goal for this section is to write an app to monitor data collection in real time, as well as visualize our data collection against other sensors on our car. Accurate data collection is an important step for this year's car and our goal is to start collecting data in when the car is ready to drive!

1.1.3 Software Testing Lead (SWT)

This section focuses on ensuring the quality and reliability of software and system components through comprehensive testing, automation and analyzing data. Responsibilities include writing documentation and enforcing test cases, unit testing, performing system, functional and integration testing on car software such as ACU, VCU, SCU, and CAN communication.

1.2 Vehicle Control Systems

1.2.1 Accumulator Control Unit (ACU)

The ACU includes all the circuitry behind the firewall (a fireproof barrier that separates the driver from the accumulator). The ACU controls all the auxiliary functions for the accumulator (safety, control, and charging) and also manages/distributes low voltage (LV) power. The code base includes opening the AIRs (Accumulator Isolation Relays), accumulator charging, turning on the radiator fans etc.

1.2.2 Vehicle Control Unit (VCU)

The VCU is responsible for interfacing with the pedal sensors, determining and sending torque commands to the motor controller, keeping track of the car state using a state machine, and interfacing with the dash screen.

1.2.3 Sensor Control Unit (SCU)

The SCU is responsible for gathering data from many sensors and sending that data to a CAN bus for the logging board to collect the data. Some examples of data we collect are:

- Wheel speed data
- Cooling loop temperature
- Flow meter readings (for cooling loop)
- Shock potentiometer data
- IMU (Inertial Measurement Unit) data

1.2.4 Motor Controller (MC)

We use a PM100Dx motor controller to control our Emrax 228 motor. It's important that the motor controller is calibrated properly to ensure the motor can output a higher maximum torque value! We calibrate the motor by using the RMS GUI software.

1.2.5 Battery Management System (BMS)

We use a EMUS G1 BMS to monitor the cells in our accumulator. A BMS will increase the safety factor that comes with high voltage (HV) components such as our accumulator. The BMS can collect data about the cells (balancing levels, tempurature, voltage etc.) and we can read that data via the EMUS G1 software. We are also able to communicate to the BMS via CAN (Controller Area Network) communication.

1.2.6 Driver Display (Dash)

Our dash screen is used to display important information to the driver in real time. This includes:

- Vehicle speed
- Safety loop status
- TSA (Tractive System Active) indicator light
- RTD (Ready to Drive) indicator light

1.3 Downloading CubeMx and CubeIDE

We will do this as a group because we will need to download a specific version (the same version that our GitHub repositories run on) for both CubeMx and CubeIDE.

1.3.1 CubeMx

STM32CubeMX is a way to start some of your projects. You need to download **6.11.1**, which you can find on the STM32CubeMX page: `https://www.st.com/en/development-tools/stm32cubemx.html` and create a free account to download!

1.3.2 CubeIDE

CubeIDE is the IDE we use to program our STM32 microcontrollers on our various PCBs (ACU, VCU, and SCU) on our car. You need to download version **1.15.1**, go to the STM32CubeIDE page: `https://www.st.com/en/development-tools/stm32cubeide.html#get-software`

1.4 Hardware

The STM32-F446RE is the development board we use to flash on all the PCBs on the car (ACU, VCU, and SCU). Unlike normal programming that can be compiled and ran on any hardware, embedded systems software is usually written for a specific chip. Here we are using an ARM Cortex M4 bases chip, and we need to buy the corresponding dev board to develop for it. It costs about $30, so the SSO will get a list of new members and make a mass order of boards.

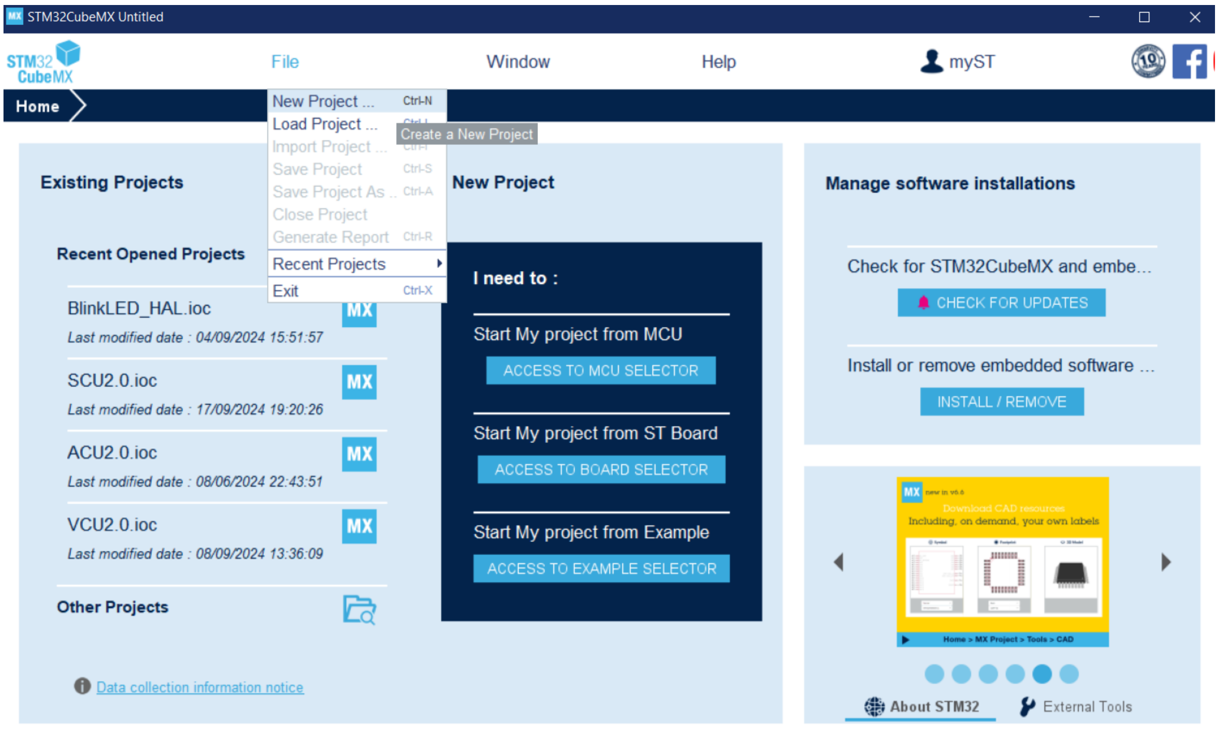
2 Tutorial Projects

These projects will give you an introduction to embedded systems development. Our code base uses the Hardware Abstraction Layer (HAL) library for the STM32. Hardware Abstraction Layer (HAL) is one layer above Standard Peripheral Library (SPL), and as the name suggests, HAL abstracts SPL functions. Once you finish a tutorial, tell your SSO. They will likely want to see your code to make sure everything is done correctly, or if they have any tips on good practice/optimization.

NOTE: After you finish each tutorial please reference the **Uploading Tutorials to GitHub** section. A section lead or the SSO will review a few pull requests (PRs) from each new member. The goal is to teach you how to submit PRs and give you guys feedback on your work :)

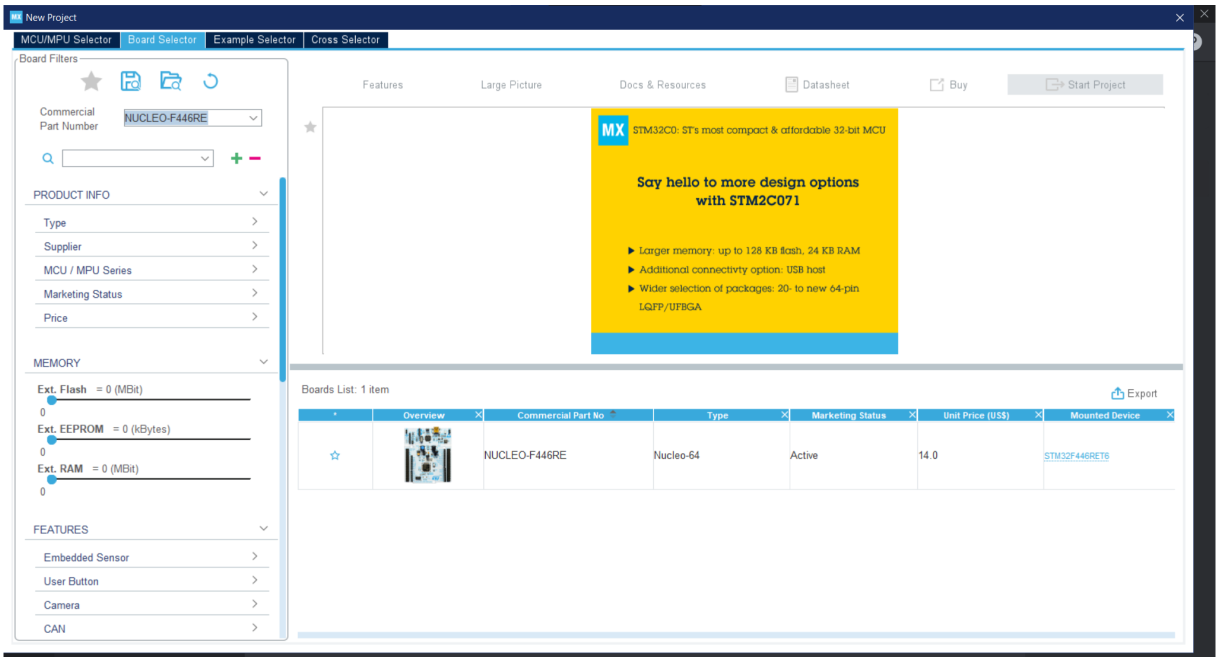
3 Tutorial 1: LED Blink

1. Open CubeMx
2. Click on file → New project


STM32CubeMX Untitled
STM32 CubeMX
File
Window
Help
myST
Home
New Project ... Ctrl-N
Load Project ... Ctrl-L
Create a New Project
Import Project ... Ctrl-I
Save Project Ctrl-S
Save Project As .. Ctrl-A
Close Project
Generate Report Ctrl-R
Recent Projects
Exit Ctrl-X
Existing Projects
Recent Opened Projects
BlinkLED_HAL.ioc
Last modified date : 04/09/2024 15:51:57
SCU2.0.ioc
Last modified date : 17/09/2024 19:20:26
ACU2.0.ioc
Last modified date : 08/06/2024 22:43:51
VCU2.0.ioc
Last modified date : 08/09/2024 13:36:09
Other Projects
Data collection information notice
New Project
I need to :
Start My project from MCU
ACCESS TO MCU SELECTOR
Start My project from ST Board
ACCESS TO BOARD SELECTOR
Start My project from Example
ACCESS TO EXAMPLE SELECTOR
Manage software installations
Check for STM32CubeMX and embe...
CHECK FOR UPDATES
Install or remove embedded software ...
INSTALL / REMOVE
About STM32
External Tools

   (a) This will take you to all the different types of STM MCUs and boards that you can buy. We use the STM NUCLEO-F446RE board to write embedded software and program the microcontrollers (VETx) on our custom PCBs (ACU, SCU, and VCU).

3. Type "F446RE" in the "commercial part number" section



Commercial Part Number: NUCLEO-F446RE

Ext. Flash = 0 (MBit)

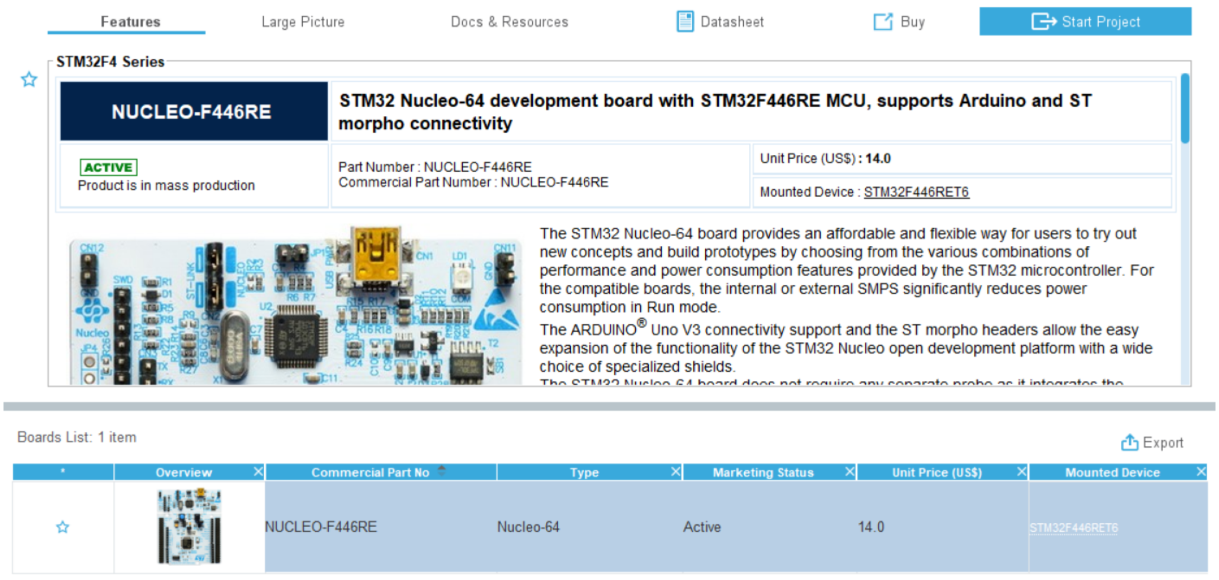
Ext. EEPROM = 0 (kBytes)

Ext. RAM = 0 (MBit)

Boards List: 1 item

*	Overview	Commercial Part No	Type	Marketing Status	Unit Price (US$)	Mounted Device
☆		NUCLEO-F446RE	Nucleo-64	Active	14.0	STM32F446RET6

4. Click the STM NUCLEO-F446RE board when it pops up, then click the blue "Start Project" button



STM32F4 Series

NUCLEO-F446RE

STM32 Nucleo-64 development board with STM32F446RE MCU, supports Arduino and ST morpho connectivity

ACTIVE
Product is in mass production

Part Number : NUCLEO-F446RE
Commercial Part Number : NUCLEO-F446RE

Unit Price (US$) : **14.0**

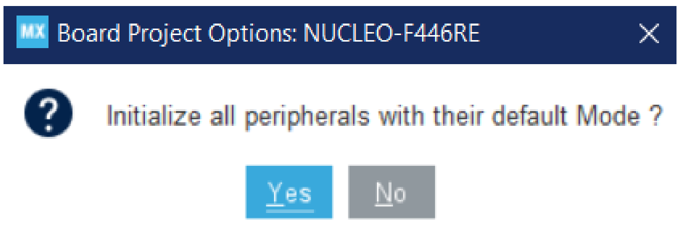
Mounted Device : STM32F446RET6

The STM32 Nucleo-64 board provides an affordable and flexible way for users to try out new concepts and build prototypes by choosing from the various combinations of performance and power consumption features provided by the STM32 microcontroller. For the compatible boards, the internal or external SMPS significantly reduces power consumption in Run mode.
The ARDUINO® Uno V3 connectivity support and the ST morpho headers allow the easy expansion of the functionality of the STM32 Nucleo open development platform with a wide choice of specialized shields.

Boards List: 1 item

*	Overview	Commercial Part No	Type	Marketing Status	Unit Price (US$)	Mounted Device
☆		NUCLEO-F446RE	Nucleo-64	Active	14.0	STM32F446RET6

5. You will then get prompted to "Initialize all peripherals with their default mode"

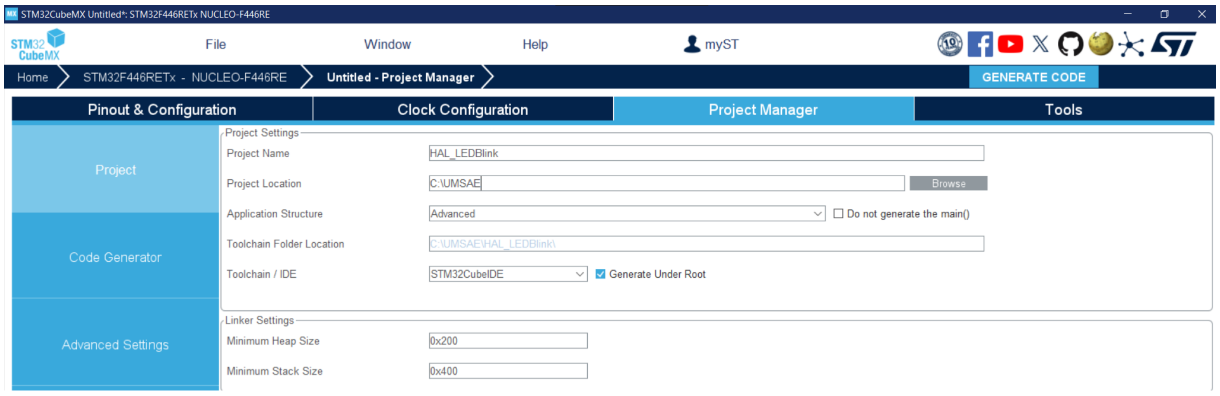


Board Project Options: NUCLEO-F446RE

Initialize all peripherals with their default Mode ?

Yes No

6. Click yes, then CubeMX will take you to the microcontroller "Pinout & Configuration" section!
   (a) We use the Pinout & Configuration section to write the preprocessor directives for our embedded C programs. Throughout the tutorials we will come back to this page frequently to and explain how to configure our microcontrollers with different communication protocols.
   (b) We will leave all of the settings as default for this tutorial
7. Now we can move to the blue "Project Manager" tab



Project Settings

Project Name: HAL_LEDBlink

Project Location: C:\UMSAE

Application Structure: Advanced ☐ Do not generate the main()

Toolchain Folder Location: C:\UMSAE\HAL_LEDBlink\

Toolchain / IDE: STM32CubeIDE ☑ Generate Under Root

Linker Settings

Minimum Heap Size: 0x200

Minimum Stack Size: 0x400

8. Under the Project name section type HAL_LEDBlink
   (a) This will be the name of the project folder for our HAL_LEDBlink project
9. Under the "Project Location" section and choose the place you want the file to be located on your computer
   (a) Personally, I made a UMSAE folder and inside that folder I have all the different project folders related to our car
10. Next, click the drop down window for the "Toolchain/IDE" section and select "STM32CubeIDE
    (a) This will be the IDE we use to program our microcontrollers

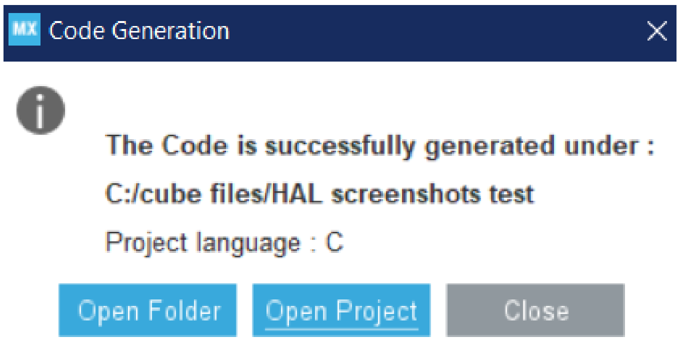
11. Now click on the "Code Generator" tab. Under the "Generated Files" section, enable the "Generate peripheral initialization as a pair of 'c./.h' files per peripheral" setting. This will generate a lot of the board level definitions in the .h file instead.

12. Now click the "Generate Code" in the top right of CubeMX

    (a) This will configure the microcontroller for the project file based on the pinout configurations page and other settings

13. When prompted, click "Open Project" then click "Launch"

    (a) Once CubeIDE opens you should see your project appear on the left-hand side of the screen under "Project Explorer"



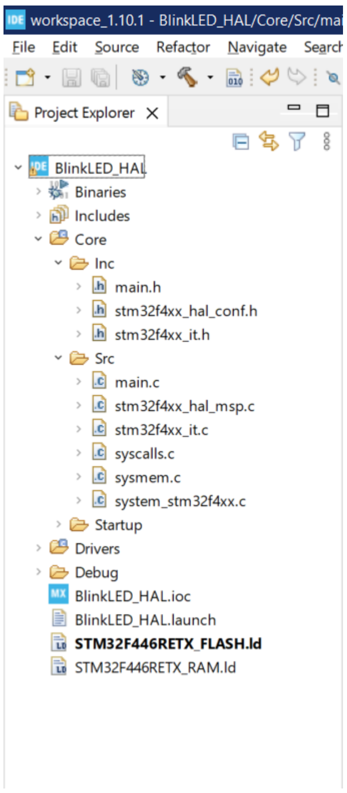
Code Generation

The Code is successfully generated under :

C:/cube files/HAL screenshots test

Project language : C

14. Click on the project then the only folders we need to look at for these tutorials are the "Inc" (Include) and "Src" (Source) files



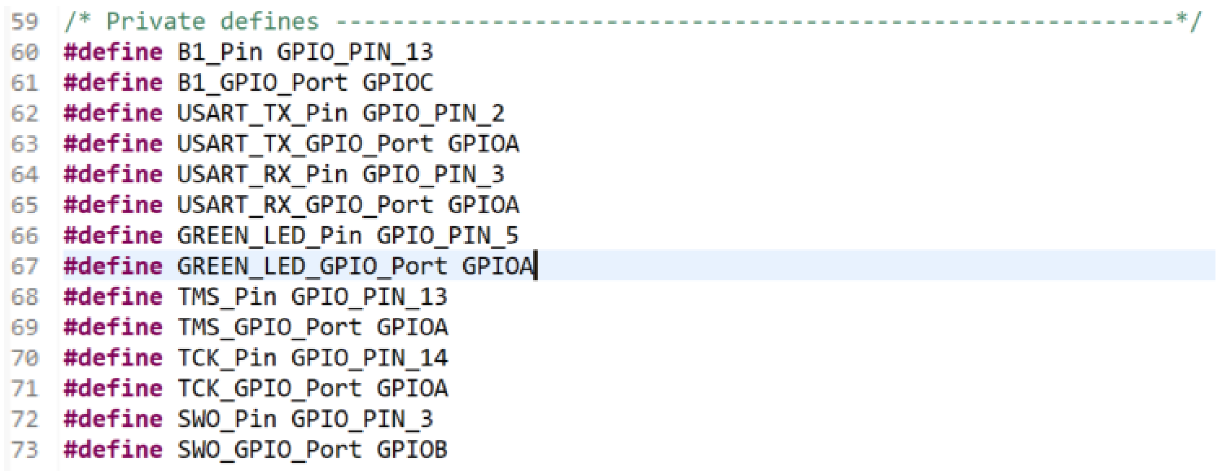
15. Next, open both the "main.c" and "main.h" files in the Src and Inc folders respectively

    (a) Files ending with .c are files where we write our source code for the car, .h files (header files) are used to store preprocessor directives

16. Open "main.h" and scroll down to the code block labelled "Private defines"

    (a) These defines were generated by CubeMX specifically the "Pinout & Configuration" section generates these defines

(b) To get the LED to blink we need two things, the Port and Pin number of the microcontroller. The defines that CubeMX generated for us gave the port and pin values custom labels so it's easier for us to read. You'll need to use LD2_GPIO_Port and LD2_GPIO_Pin to turn the LED on and off.



```
/* Private defines -----------------------------------------------------------*/
#define B1_Pin GPIO_PIN_13
#define B1_GPIO_Port GPIOC
#define USART_TX_Pin GPIO_PIN_2
#define USART_TX_GPIO_Port GPIOA
#define USART_RX_Pin GPIO_PIN_3
#define USART_RX_GPIO_Port GPIOA
#define GREEN_LED_Pin GPIO_PIN_5
#define GREEN_LED_GPIO_Port GPIOA
#define TMS_Pin GPIO_PIN_13
#define TMS_GPIO_Port GPIOA
#define TCK_Pin GPIO_PIN_14
#define TCK_GPIO_Port GPIOA
#define SWO_Pin GPIO_PIN_3
#define SWO_GPIO_Port GPIOB
```

17. Now open "main.c", and scroll down the the "main" function, this is where we write the code to toggle the LED on and off

    (a) Inside the braces of the infinite while loop and above the comment /* USER CODE END WHILE */ we are going to use two HAL (Hardware Abstraction Layer) functions: HAL_GPIO_TogglePin() and HAL_Delay()

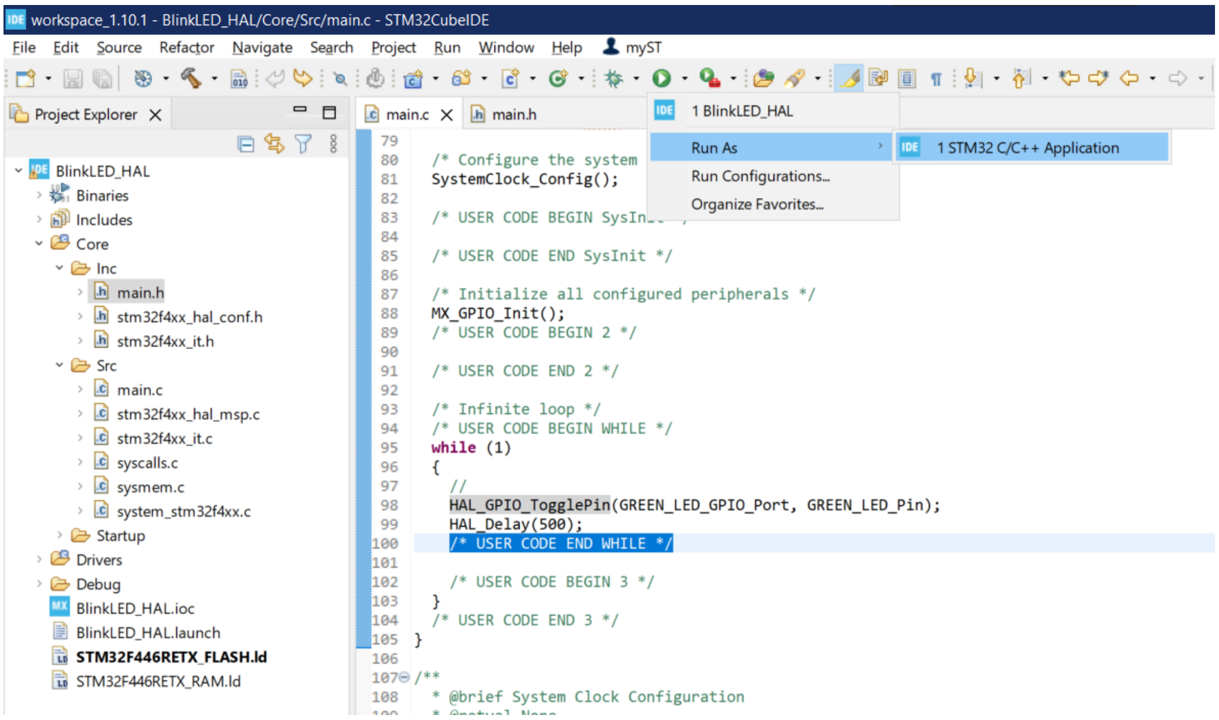
    (b) HAL_GPIO_TogglePin takes in two parameters, the port and pin of the microcontroller used to turn the LED on and off. This is LD2_GPIO_Port and LD2_GPIO_Pin from the "main.h" file

18. On the next line, use HAL_Delay and hold left-ctrl then left click on the function again to see it's parameters. Go back to main.c and write 500 as the function parameter for now! This is the amount of time we want the LED to be on and off for.

19. Save your work and it's almost time to run the program!

20. Plug in your STM-F446RE board (if you don't have one yet, ask the SSO to borrow one for this tutorial)

21. Next click the drop down arrow beside the green play button at the top of the screen then select "Run as" → "STM32 C/C++ Application"



```

/* Configure the system
SystemClock_Config();

/* USER CODE BEGIN SysIn

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
/* USER CODE BEGIN 2 */

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  //
  HAL_GPIO_TogglePin(GREEN_LED_GPIO_Port, GREEN_LED_Pin);
  HAL_Delay(500);
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
}

/**
 * @brief System Clock Configuration
```

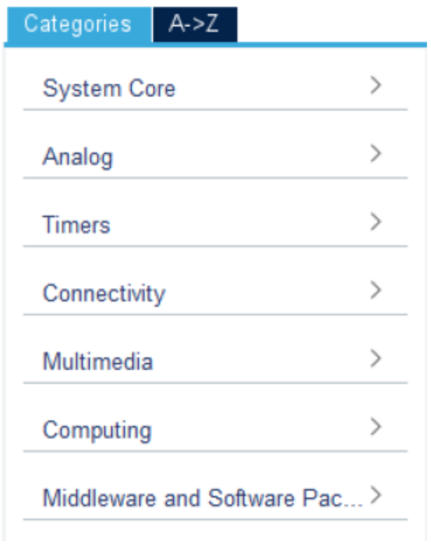
22. When prompted hit "ok". This is just to set up the run profile for the project

23. Your project should now automatically build then run!

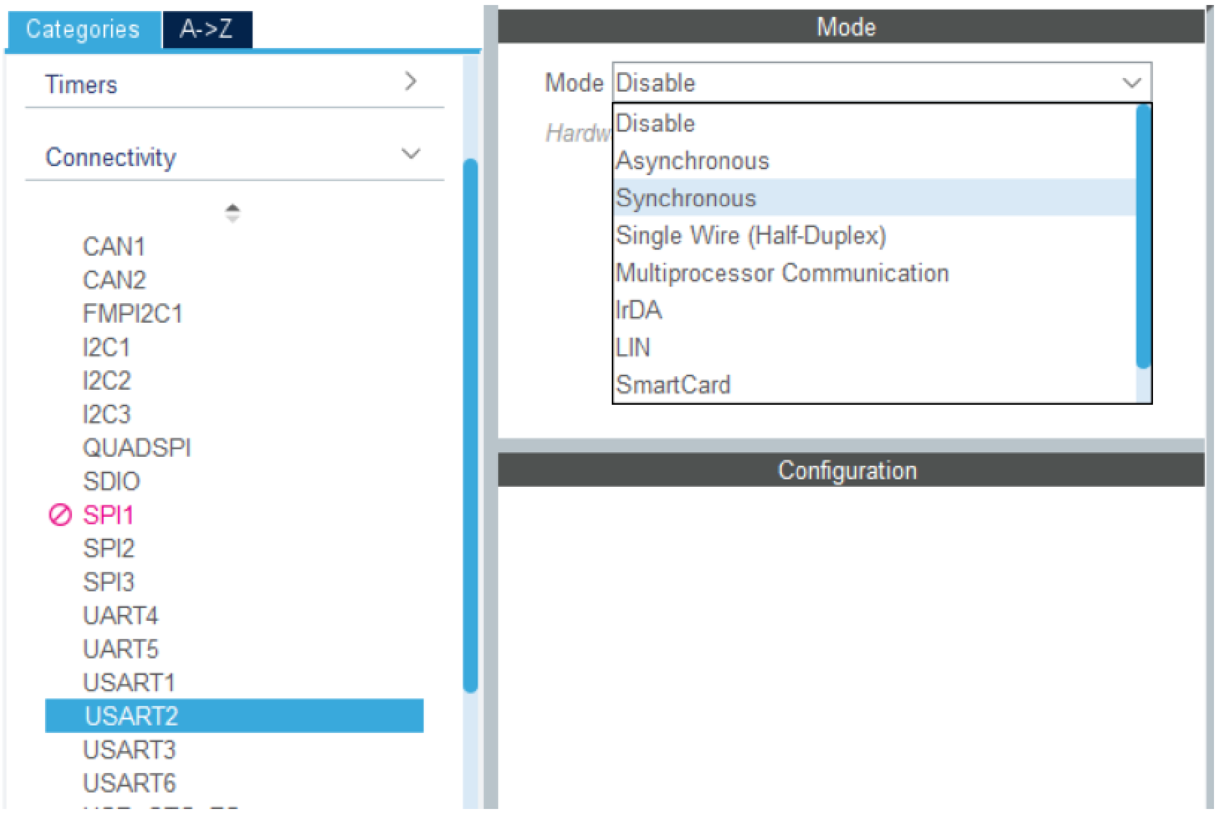
    (a) Try changing the delay value, this will change how fast the LED flashes on the board!!

4 Tutorial 2: HAL USART

1. We will need to install PuTTY for this tutorial. For our purposes, PuTTY monitors a specific COM port for serial communication and displays the data to the user through the PuTTY terminal.
   (a) We'll configure the PuTTY settings later on in this tutorial
2. Follow steps 1 → 6 of the LED Blink tutorial to set up a new CubeMX project. You should end up on the "Pinout & Configuration" section again
3. For this tutorial we want to enable USART2 on the board.
   (a) USART stands for Universal Synchronous Asynchronous Receiver Transmitter. It is sometimes called the Serial Communications Interface or SCI. For our purposes we use USART to send messages to the COM port on our laptop through the cable connecting the STM32 Nucleo board to your laptop
4. To enable USART, go to the "Connectivity" drop down menu on the left hand side of CubeMX



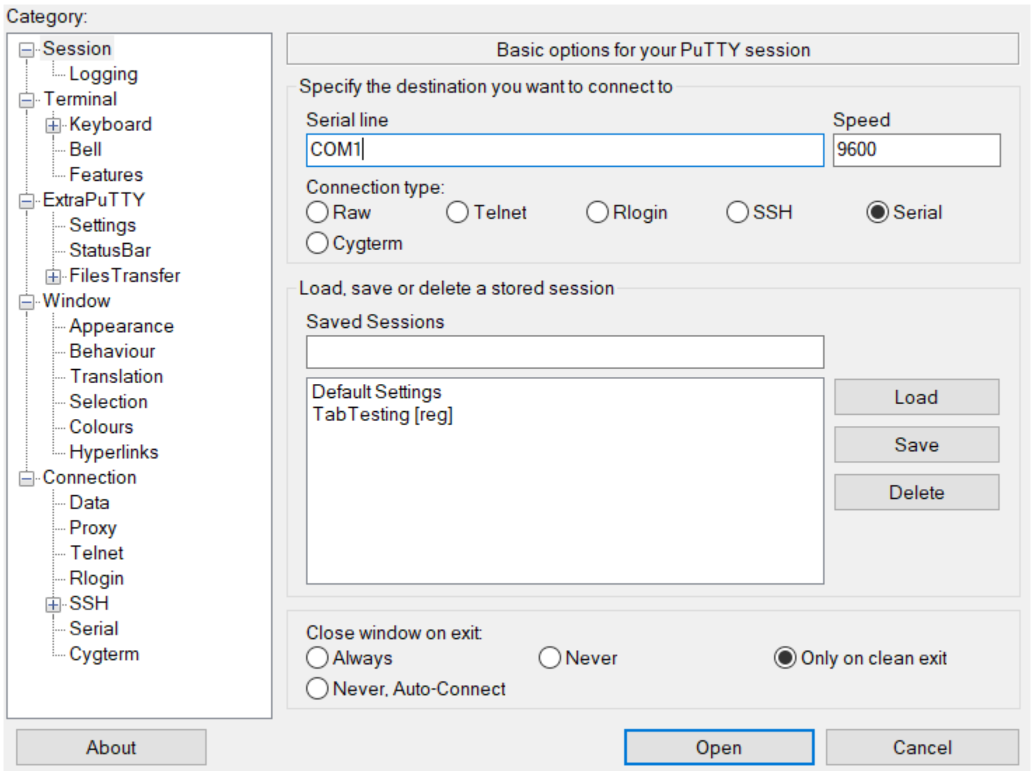
5. Click on USART2 and change the mode from "Disable" to "Synchronous"



6. Now look at the "Configuration" section that just opened in CubeMX and open the "Parameter Settings" tab
   (a) For this tutorial we will use a Baud rate of 115200 Bits/s
   (b) Later on, we will tell PuTTY to monitor our laptop's COM port at the same baud rate. This is important because if PuTTY and our program have a baud rate mismatch, then the PuTTY terminal will just display "garbage" (not human readable data) to the PuTTY terminal
7. Now follow steps 7 → 13 of the LED Blink tutorial, but name the project "HAL_USART", and save the project in the same "UMSAE" root folder as last time
8. Once the project is open, find USART.h and main.c in the project directory
9. We will need to use husart2 from USART.h. This was generated when we enabled USART2 in the connectivity tab in CubeMX
10. Next, open main.c and scroll down to the infinite while loop
    (a) For this tutorial, we will be using the HAL_USART_Transmit() and HAL_Delay() functions
    (b) Using ctrl + click, click on the HAL_USART_Transmit() function and read what parameters the function takes. I want you to try to understand this independently, but feel free to ask the SSO or team leads a question! It will definitely benefit your learning if you try to understand the function first though.
    (c) Here's the code I would like you to use for the parameters! Place the variables in the function parameters, followed by HAL_Delay() with a delay value of 1000.

```
uint8_t message[] = "Welcome to UMSAE Software!\r\n";
uint16_t size = sizeof(message);
uint32_t timeOut = 10;

```

11. Once you've set up the parameters correctly, open PuTTY on your computer! You should see a window that looks similar to this:



Basic options for your PuTTY session

Specify the destination you want to connect to

Serial line: COM1

Speed: 9600

Connection type: Raw Telnet Rlogin SSH [✓ Serial] Cygterm

Load, save or delete a stored session

Saved Sessions

Default Settings
TabTesting [reg]

Close window on exit: Always Never [✓ Only on clean exit] Never, Auto-Connect

12. We need to change the Serial line and Speed settings. These are the COM port on your computer and PuTTY's baud rate.
    (a) Open "Device Manager" on your computer and go to the "Ports (COM & LPT)" drop down menu. Here you'll see which COM port your NUCLEO is connected to on your computer.
    (b) Change the "Serial line" setting to match your COM port and change the "Speed" to 115200 to match our USART baud rate
13. Click "open" to open the PuTTY terminal
14. Run the program the same way we did in the Blink LED tutorial. You should see a message in the terminal!

5 Tutorial 3: HAL UART Interrupts

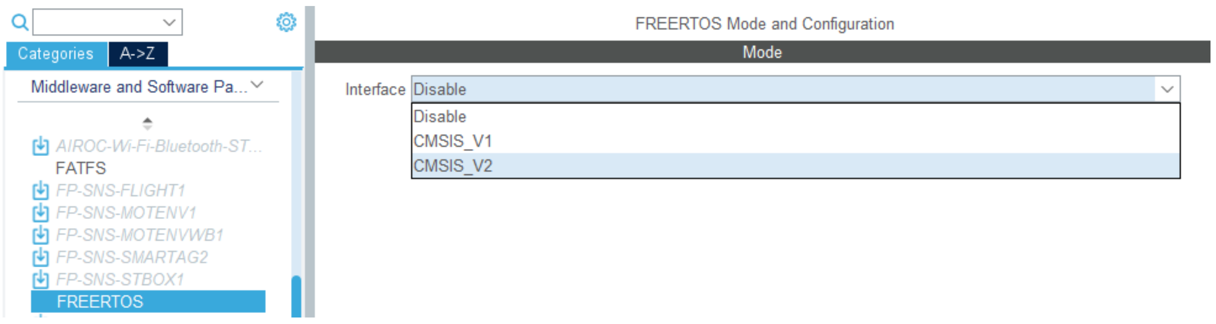
1. Follow steps 2 $\rightarrow$ 6 of the HAL USART tutorial to set up a new CubeMX project. This time we are going to enable UART
   (a) To do this, set the mode from "Synchronous" to "Asynchronous"
   (b) Set the baud rate to 9600 Bits/s
2. Go to the "NVIC Settings" tab and enable "USART2 global interrupt"
3. Go to the "Project Manger" tab and name the project "HAL_UART_IRQ_Tutorial" and set the your project location and Toolchain/IDE to STM32CubeIDE
4. First you'll need to write the function HAL_UART_Receive_IT()
   (a) Use some of the following parameters when "enabling" this function
   (b) Write the function in the "USER CODE BEGIN 2" comment block of the code

```
uint8_t msg[] = "Welcome to Software!!!\r\n"; // Data to be sent
uint16_t msg_size = sizeof(msg); // Length of the msg

uint8_t recvd_data; // Receive buffer
uint16_t recvd_data_size = 1; // Size of the character 't'

```

5. You'll need to call the function HAL_UART_RxCpltCallback() to receive a user interrupt using PuTTY
6. Finally we want the LED to blink whenever the character 't' is received and transmit the message assigned to the user_data pointer by using HAL_UART_Transmit()!
7. You'll know if your code works when the LED toggles on when you press 't' and off when you press 't' again with the PuTTY terminal open. Every time you press 't' you should also see the message "Welcome to Software!!!"!

6 Tutorial 4: HAL FreeRTOS

1. FreeRTOS is a form of co-operative multitasking. Basically the microprocessor (STM32) will switch between the tasks as each one finishes its processing
2. Follow steps 1-6 of the LED Blink tutorial to set up a new CubeMX project
3. Now enable UART (same as last tutorial) but with a baud rate of 115200 Bits/s
   (a) No need to enable the global interrupt for this tutorial
4. Now, scroll down to the "Middleware and Software Packs" drop down menu and click on "FreeRTOS"
   (a) Select "CMSIS_V2" on the "Interface" drop down menu



5. Go to the "Tasks and Queues" tab and click the "Add" button
   (a) Here is where we will make two FreeRTOS tasks to add (append), and remove from a queue
6. Here are the settings you need to change for the first task:
   (a) Call the first task "queueAppendTask"
   (b) Set it's priority to "osPriorityLow"
   (c) Leave Stack Size (Words) as 128
   (d) Change the name of the entry function to "StartQueueAppendTask"
   (e) Leave Code Generation Option as "Default"
   (f) Leave Parameter as "NULL"
   (g) Leave Allocation as "Dynamic"
7. Now, create another task according to these settings:
   (a) Call the first task "queueRemoveTask"
   (b) Set it's priority to "osPriorityLow"
   (c) Leave Stack Size (Words) as 128
   (d) Change the name of the entry function to "StartQueueRemoveTask"
   (e) Leave Code Generation Option as "Default"
   (f) Leave Parameter as "NULL"
   (g) Leave Allocation as "Dynamic"
8. Next, we want to make the queue that these two tasks will use to send and receive messages
9. Click "Add" under the Queues section
10. Here are the queue settings:
    (a) Call the queue "coolQueue"
    (b) Set the queue size to 8
    (c) Set the item size to "uint16_t"
    (d) Set the allocation to "Dynamic"
11. Now, go to the "Project manager" tab and call the project "HAL_FreeRTOS_Tutorial", set the project file location and set the Toolchain/IDE to STM32CubeIDE
12. Click generate code

13. The functions you will need to use in this tutorial come from the CMSIS-RTOS2 (CMSIS_V2) version of FreeRTOS

    (a) You can find these functions in the "Middlewares" folder of our project. CubeMX automatically generated these files when we enabled the CMSIS_V2 FreeRTOS functionality

14. Here's a hyperlink to some documentation on these functions: CMSIS_V2 FreeRTOS Queue Functions

15. In main.c, we want to add (append) data to the queue using the StartQueueAppendTask() and remove data from the queue using StartQueueRemoveTask().

16. Use the following code inside StartQueueAppendTask(). We want to increment dataSend inside the infinite loop and append it to the queue

```
uint16_t dataSend = 0;
const uint16_t MAX = 100;

```

17. Once the count reaches a max of 100 we want to reset the data sent value to 0.

18. Use the following code inside StartQueueRemoveTask():

```
uint16_t dataReceived = 0;
char new_char[5];

```

19. In StartQueueRemoveTask(), you want to remove the data from the queue and transmit the data you get to your COM port so PuTTY can view it. Use the HAL_UART_Transmit() function to transmit the message!

    (a) Hint: You'll need to use the following code to convert the data to the correct type to transmit via UART!

    ```
    snprintf(new_char,"%d\r\n",dataReceived);

    ```

20. At the end of the FreeRTOS tasks, StartQueueAppendTask() and StartQueueRemoveTask() you'll need to use osDelay() to guarantee the function has enough time to fully execute its processing

    (a) Pass the function the following code as its parameter converting the number you enter in milliseconds to the equivalent number of ticks:

    ```
    pdMS_TO_TICKS(100)

    ```

21. Once you run PuTTY with the correct baud rate, you'll know if your code works if you get the output that counts to 99 then resets to 0 and continues counting!