Exercise Manual
For

SC2104/CE3002
Sensors, Interfacing and Digital Control

Practical Exercise #1:
Familiarization with
STM32CubeIDE
and
STM32F4 Board

Venue: SCSE Labs

COMPUTER ENGINEERING COURSE

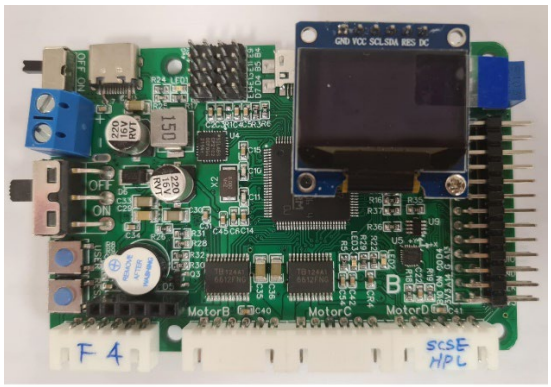
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY

Learning Objectives

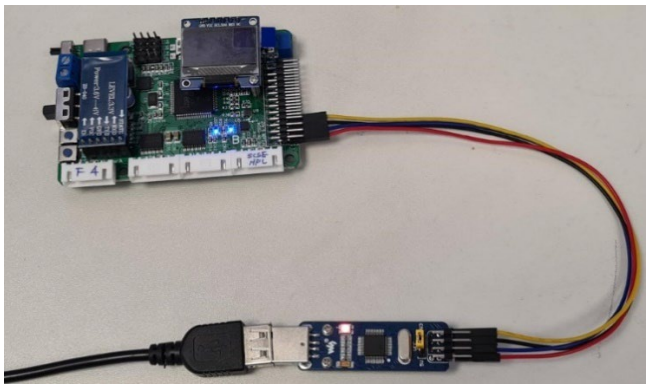
These exercises are to introduce students to the STM32F4 board and its software development platform STM32CubeIDE. The student will learn how to use the development platform to configure the STM32F4 board, and how to code application programs to access the hardware features provided on the board.

Equipment and accessories required

i) One desktop computer installed with STM32CubeIDE and PuTTY.
ii) One STM32F4 board
iii) ONE ST-LINK SWD board



STM32F4 board



ST-Link for downloading and debugging code

Introduction

You will go through a sequence of tasks designed to let you be familiarized with using the STM32CubeIDE platform to develop programs for the STM32F4 processor based systems. The applications that you develop in this practical exercise #1 will be used in subsequent lab exercises, while the procedures learnt in here will not be reiterate in the subsequent lab exercises (we will assume that you are already familiar with it after this lab).

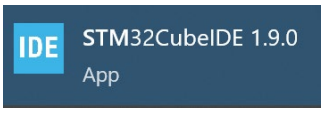
1. STM32CubeIDE and STM32F4 Board

STM32CubeIDE is a C (and C++) development platform that supports GUI based configuration, automatic code generation for configured peripherals, code compilation and debugging for STM32 based microcontrollers and microprocessors.

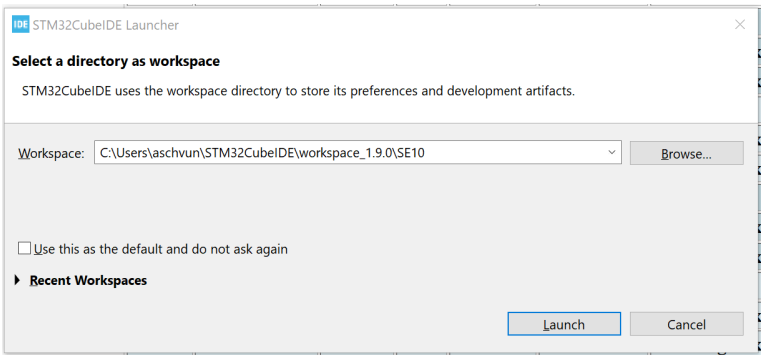
This IDE is based on the Eclipse/CDT framework and GCC toolchain for code development, and the computer in the lab should already be installed with the latest version of the STMCubeIDE.

1.1. Setting up STM32CubeIDE

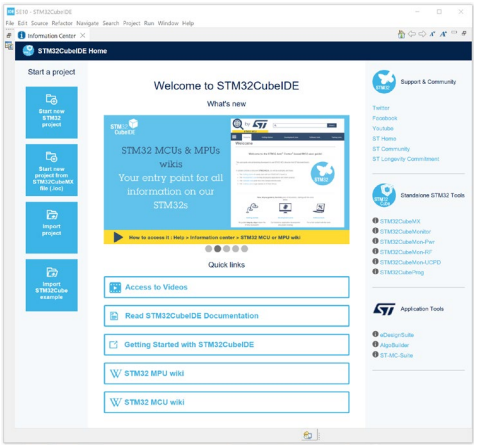
Launch the STM32CubeIDE on the computer



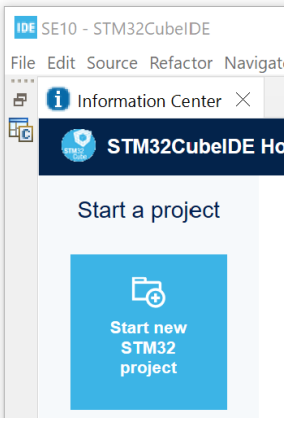
Create a sub-directory for the workspace (based on your lab group number, e.g. SE10)



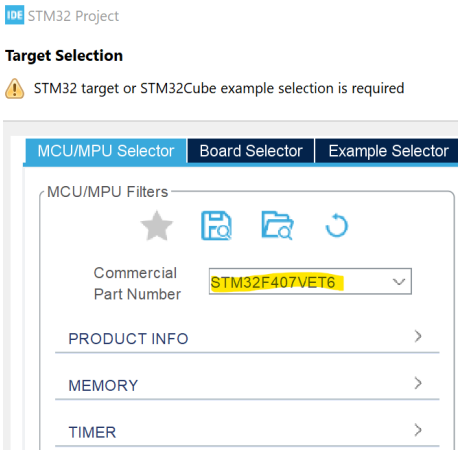
You will see the IDE startup interface as follows



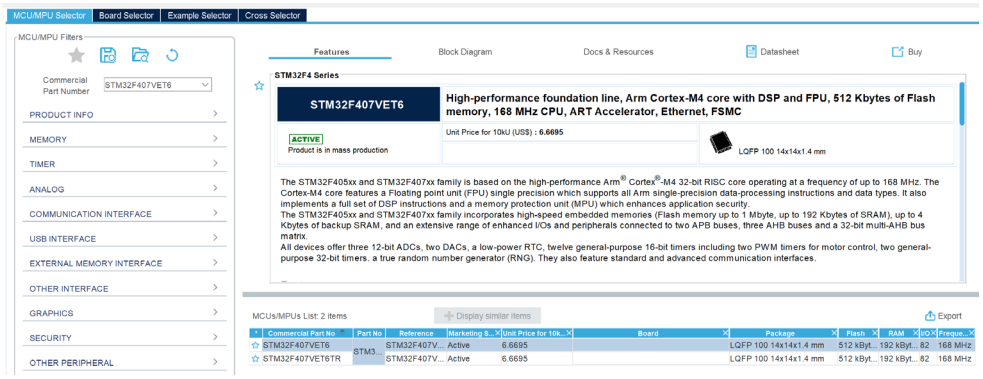
Select "Start new STM32 project" on the left panel.



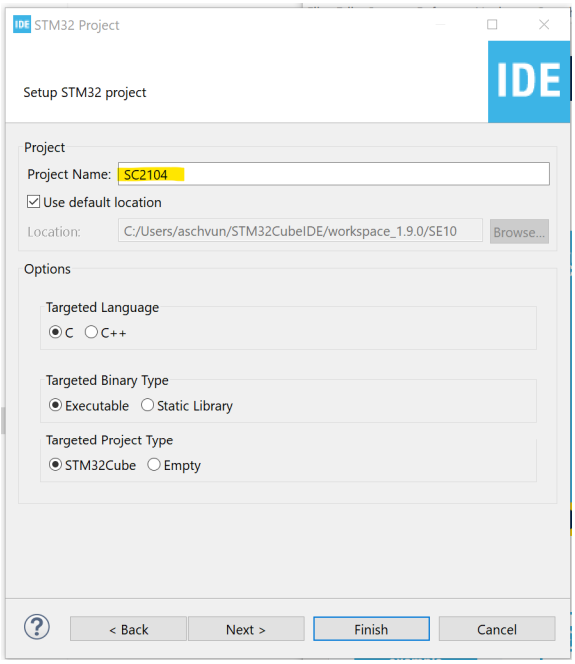
Key in the microcontroller's Part Number used in the STM32F4 board: STM32F407VET6



The IDE will list the corresponding part available in its library, which you can then select.

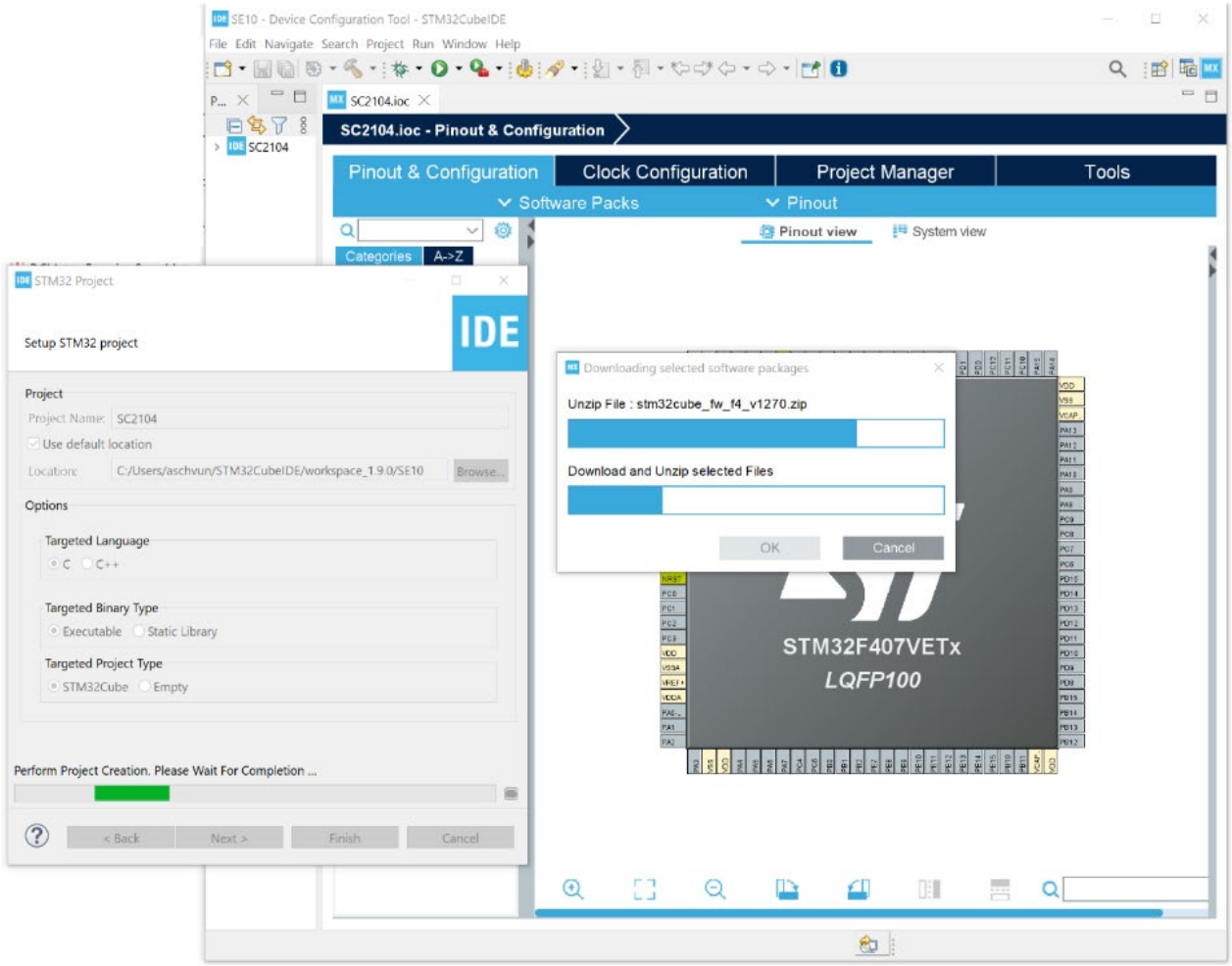


Click "Next" and enter the Project Name (e.g. SC2104).



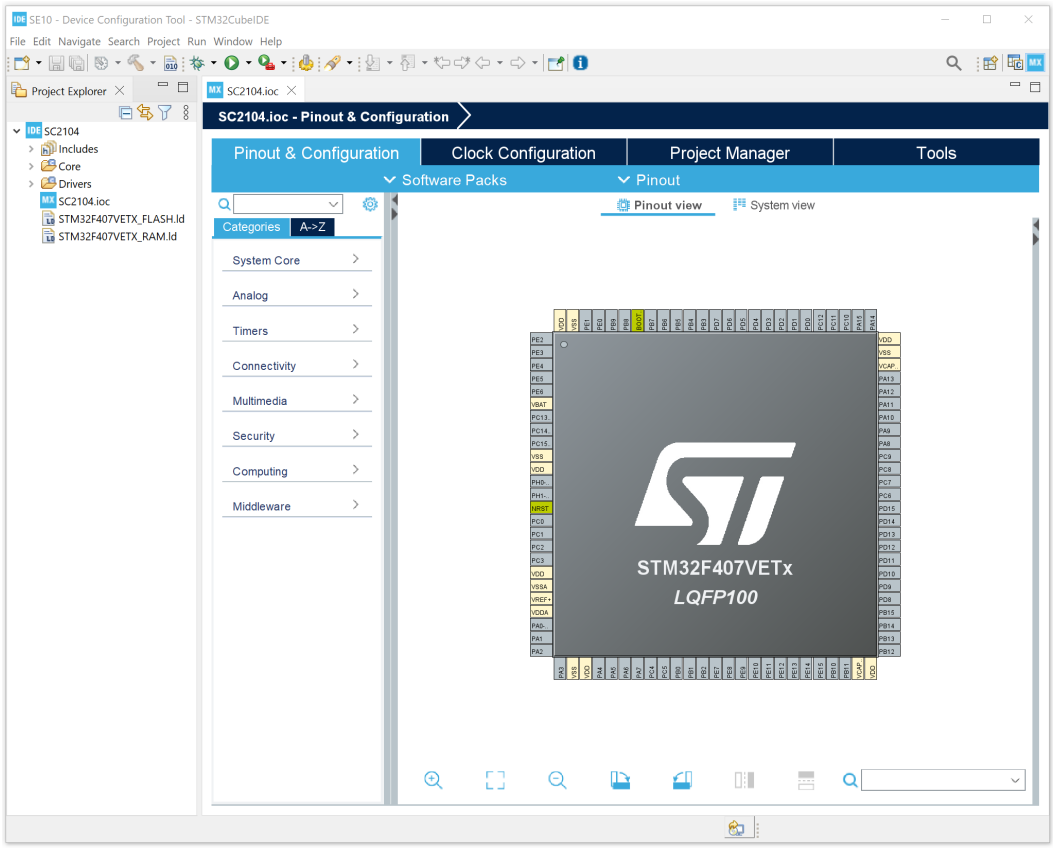
Click "Finish".

The IDE will then do the necessary setup and download the appropriate files related to the selected microcontroller (which may take a while for a new project).

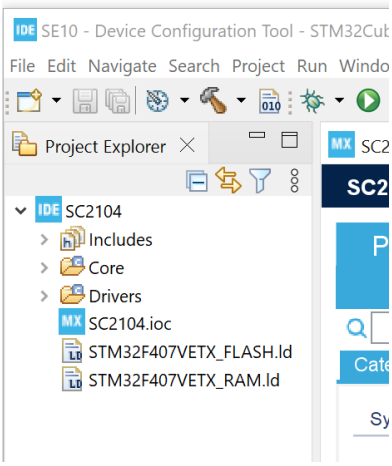


1.2. Microcontroller Configuration

Once the setup of the microcontroller by the IDE is completed, you will be presented with the screen showing a GUI of the microcontroller selected.

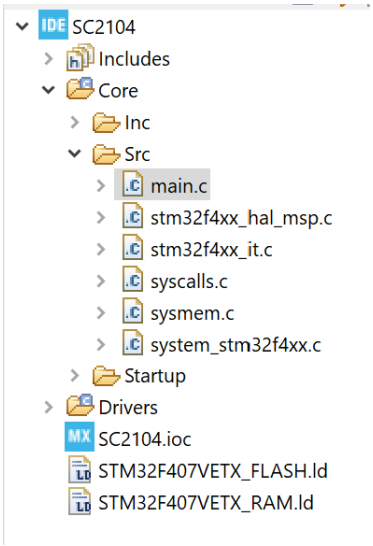


You will also see some files on its left panel, which have been setup by the IDE corresponds to the microcontroller selected.



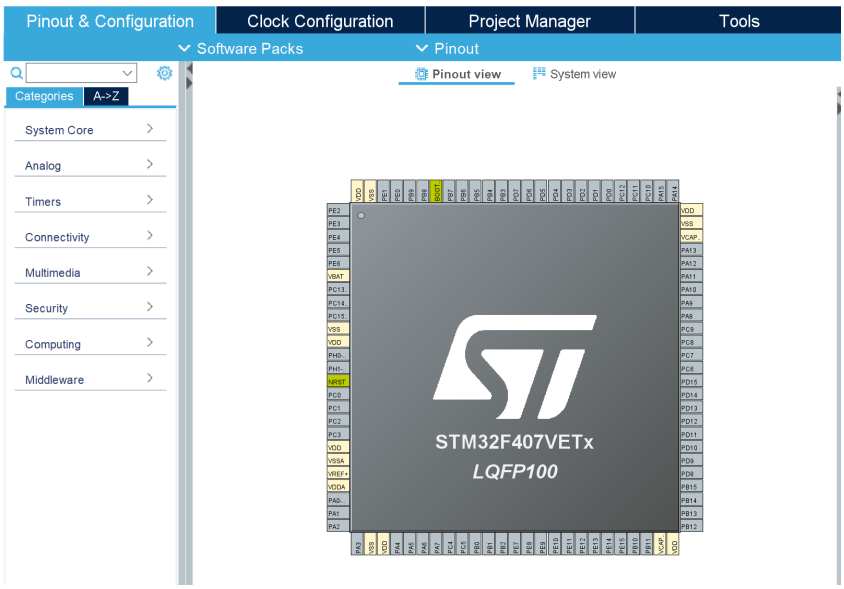
Take note of the file that has extension "ion" (which contains the GUI of the selected microcontroller seen above). You will use this file to configure the microcontroller (e.g. I/O pins) when needed.

There are also some c source files generated during the setup, including a "main.c" which contain the necessary system related code for initialization and start-up. More code will be added to these files after you do the configuration of the microcontroller. (This is also the file that you will later add your own application code).



1.3 Configuration the I/O interface

To configure the microcontroller, select the ioc file which will display the following GUI screen.

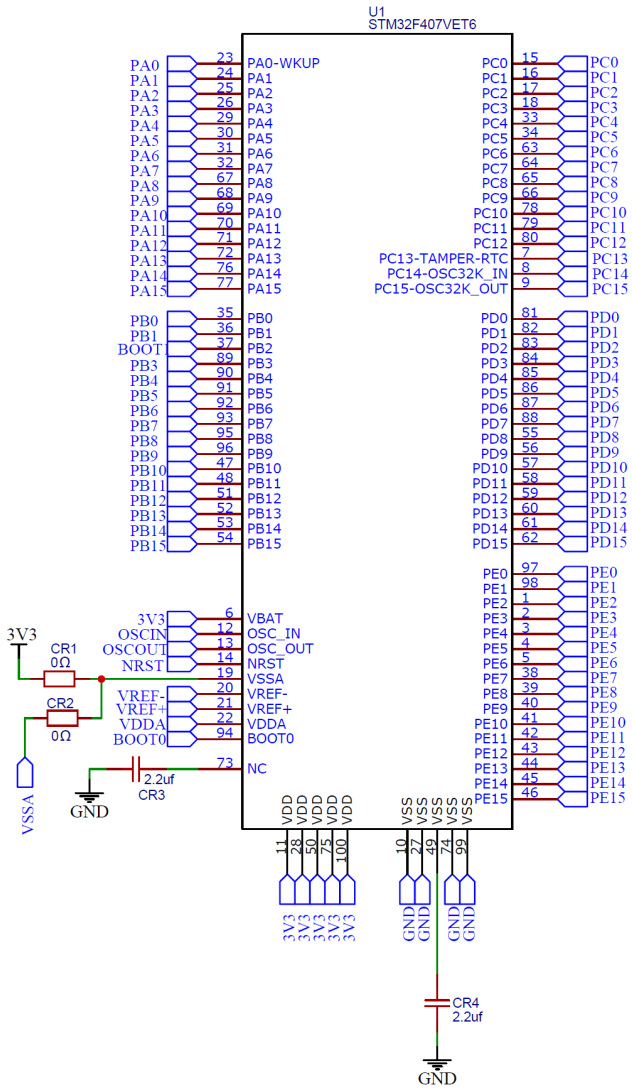


In this exercise, you will first configure the relevant I/O interfaces for the following operations:

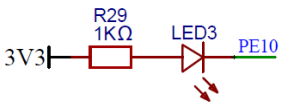
a. A LED that will be blinked by your program
b. A buzzer that will control by your program
c. A Serial Wire Debugger (SWD) to download your code to the microcontroller

To do so, you will need to refer to the schematics of the STM32F4 board that you are using. You then select the corresponding I/O pins of the microcontroller and configure them accordingly.

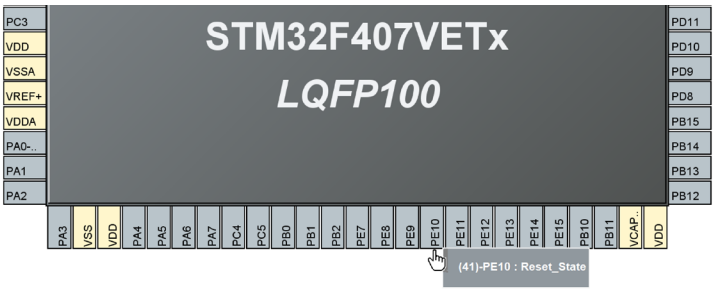
First, the following shows the overall I/O pins of the microcontroller on the circuit board.



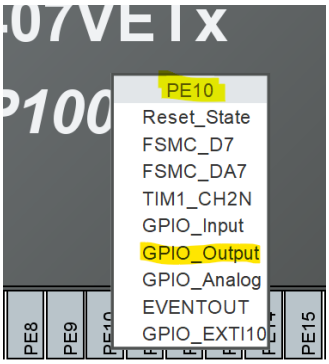
a. To begin, you will configure the LED3 on the board that is controlled through PE10 (meaning pin 10 of Port E) as shown below.



Select PE10 of the microcontroller in the IDE

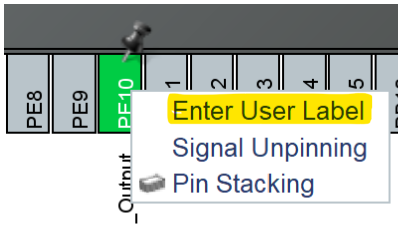


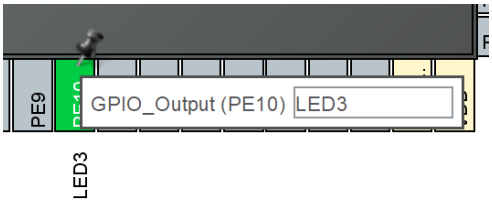
Click on the pin, and a list of options will be displayed.



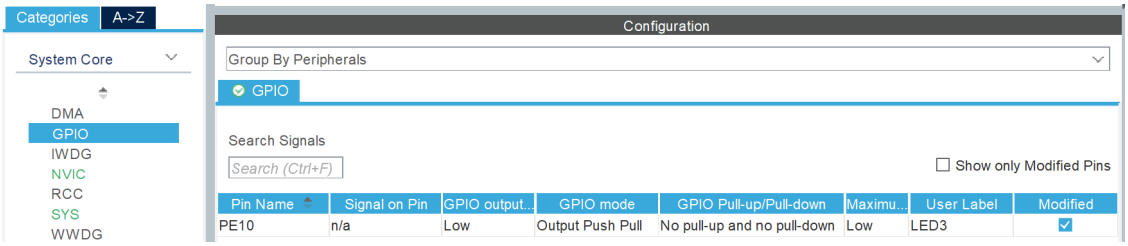
Select the configuration that you need for pin PE10, which in this case, you will select it to be an output pin: GPIO_Output

You can also change the label of the pin to make it more descriptive by right clicking on the pin. E.g., change it to “LED3”.



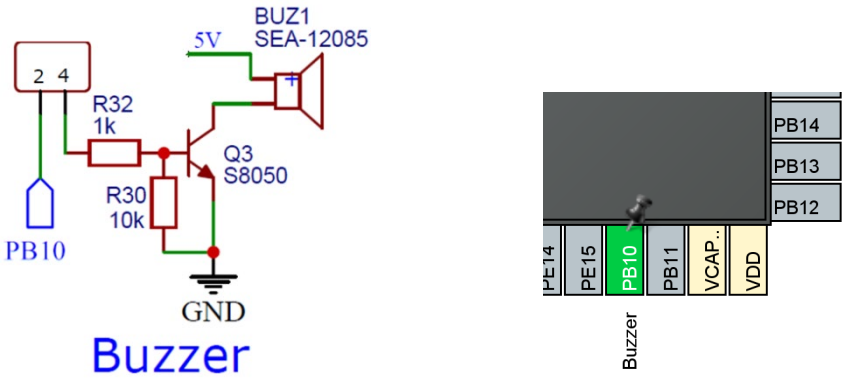


You can see more detail about PE10 that you have setup through the “System Core → GPIO” tab on the left panel . (E.g. the type of output this pin is configured with.)

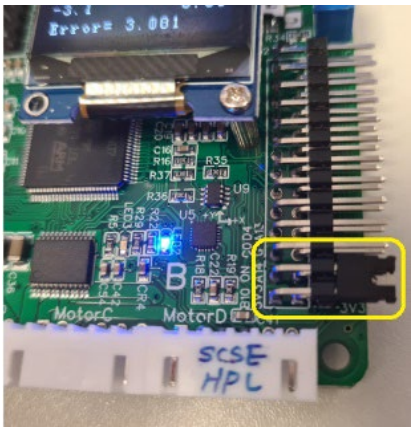


This completes the configuration setup of PE10 to drive the LED3 device. But before you proceed to generate the initialization code for this I/O pin, you will setup two other interfaces that are to be used in this exercise.

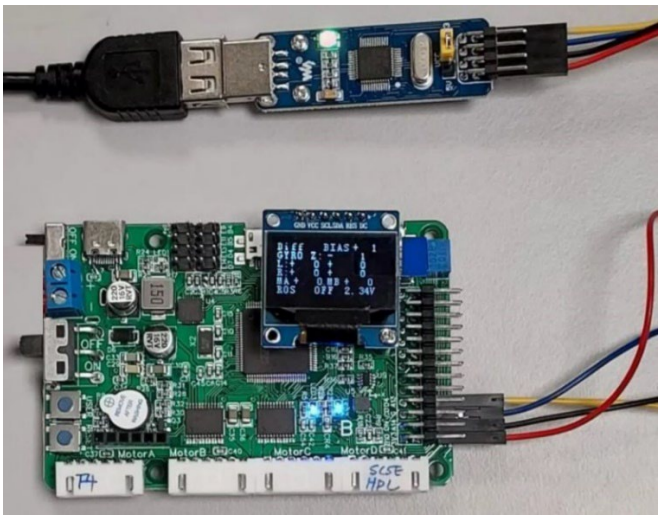
b. Using the same procedure, setup the buzzer that is connected to PB10 as shown below



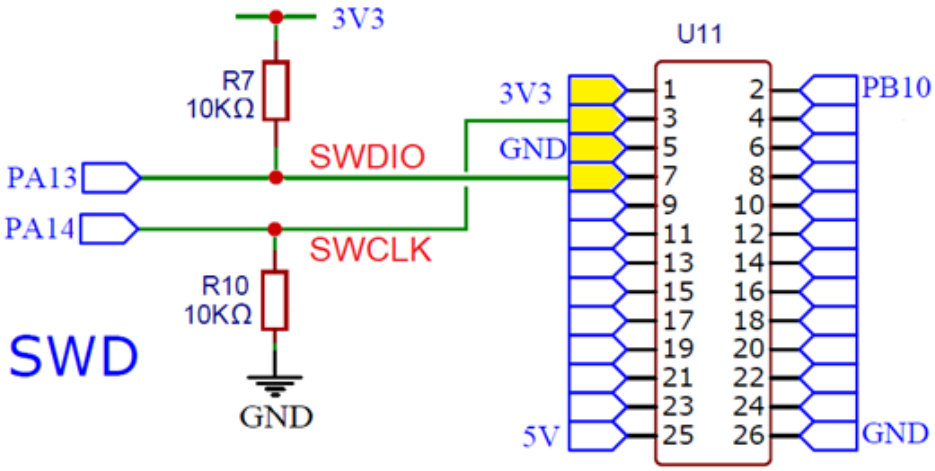
Note that a header connector is needed between pin 2 and pin 4 of the side header pins in order for the buzzer to be driven by PB10 signal to produce the audio sound.



c. You will also be using a Serial Wire Debug (SWD) ST-Link board to download your code onto the board. It also enables you to debug your program - such as to set breakpoints in your code and single step your code and inspect the values of the variables used in your program.

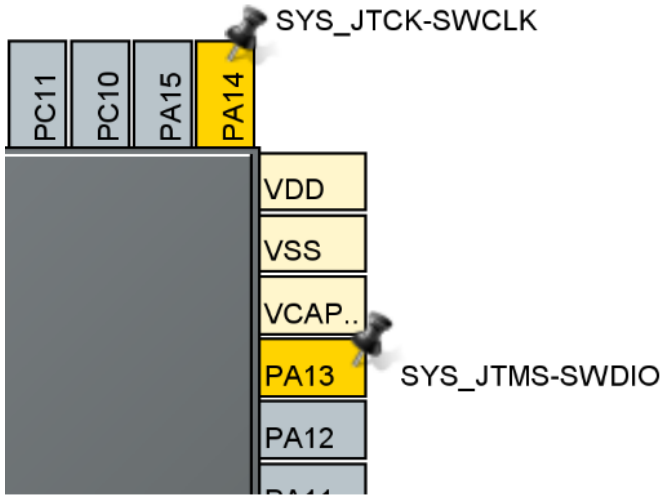


The SWD interface consists of two wires (excluding the power and ground wires): SWCLK and SWDIO.



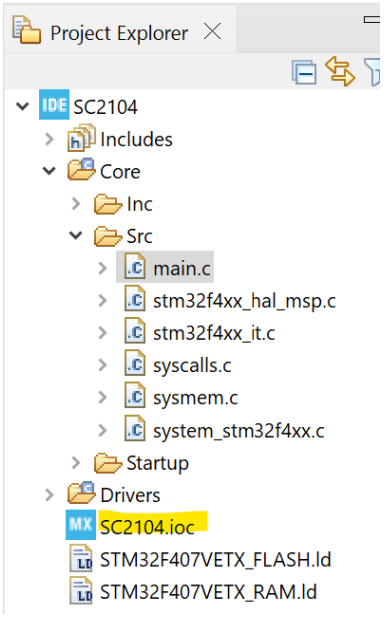
On the STM32F4 circuit board, these two signals are to be generated by PA13 and PA14 as indicated in the above schematic.

Furthermore, PA13 and PA14 have built-in SWD support functions which you can select accordingly as shown below.

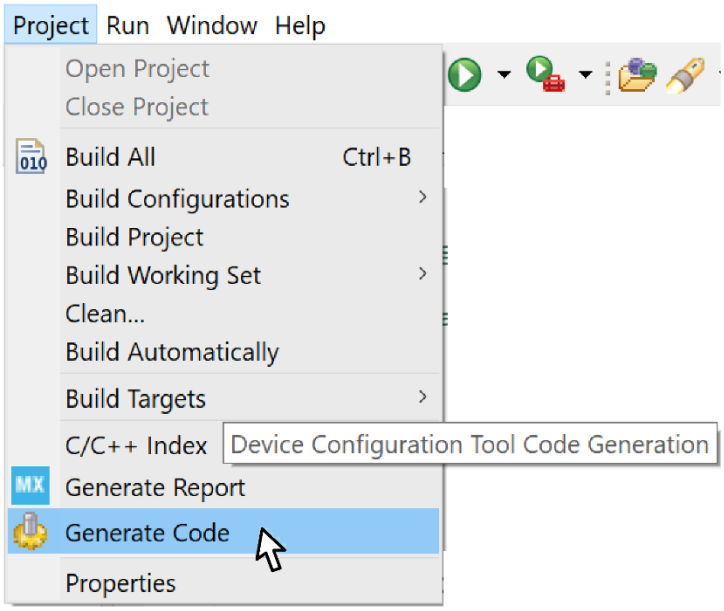


1.3 Generating the initialization code for the configured I/O pins

At this stage, what you have configured are stored in the ioc file. The next step is to let the IDE generates the corresponding initialization code correspond to the configured hardware. These autogenerated code will be added to the **main.c** file (which you can further add your own code to implement specific applications.)



To generate the initialization code for the configured hardware, click on the "Project" Tab and choose "Generate Code".



(Alternatively, just execute "File → Save" using the "File" tab.)

Once the code generation is completed, you will notice new initialization code and functions are added inside the main.c file, corresponding to the various pins configurations that you have specified.

```
143 /**
144   * @brief GPIO Initialization Function
145   * @param None
146   * @retval None
147   */
148 static void MX_GPIO_Init(void)
149 {
150   GPIO_InitTypeDef GPIO_InitStruct = {0};
151
152   /* GPIO Ports Clock Enable */
153   __HAL_RCC_GPIOE_CLK_ENABLE();
154   __HAL_RCC_GPIOB_CLK_ENABLE();
155   __HAL_RCC_GPIOA_CLK_ENABLE();
156
157   /*Configure GPIO pin Output Level */
158   HAL_GPIO_WritePin(LED3_GPIO_Port, LED3_Pin, GPIO_PIN_RESET);
159
160   /*Configure GPIO pin Output Level */
161   HAL_GPIO_WritePin(Buzzer_GPIO_Port, Buzzer_Pin, GPIO_PIN_RESET);
162
163   /*Configure GPIO pin : LED3_Pin */
164   GPIO_InitStruct.Pin = LED3_Pin;
165   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
166   GPIO_InitStruct.Pull = GPIO_NOPULL;
167   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
168   HAL_GPIO_Init(LED3_GPIO_Port, &GPIO_InitStruct);
169
170   /*Configure GPIO pin : Buzzer_Pin */
171   GPIO_InitStruct.Pin = Buzzer_Pin;
172   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
173   GPIO_InitStruct.Pull = GPIO_NOPULL;
174   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
175   HAL_GPIO_Init(Buzzer_GPIO_Port, &GPIO_InitStruct);
176
177 }
178
```

You can now add your own application code to the main.c file to operate the interfaces.

2. Writing your application code

Click and open the main.c file, and look for the following lines of code within its "main" function.

```
63  int main(void)
64  {
65    /* USER CODE BEGIN 1 */
66
67    /* USER CODE END 1 */
68
69    /* MCU Configuration--------------------------------------------------------*/
70
71    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
72    HAL_Init();
73
74    /* USER CODE BEGIN Init */
75
76    /* USER CODE END Init */
77
78    /* Configure the system clock */
79    SystemClock_Config();
80
81    /* USER CODE BEGIN SysInit */
82
83    /* USER CODE END SysInit */
84
85    /* Initialize all configured peripherals */
86    MX_GPIO_Init();
87    /* USER CODE BEGIN 2 */
88
89    /* USER CODE END 2 */
90
91    /* Infinite loop */
92    /* USER CODE BEGIN WHILE */
93    while (1)
94    {
95      /* USER CODE END WHILE */
96
97      /* USER CODE BEGIN 3 */
98    }
99    /* USER CODE END 3 */
100 }
```

Notice that there are various comments indicating where user code should be added. Code that are added by you within the indicated "USER CODE BEGIN-END" sections will be preserved during future code generation by the IDE (i.e., when you need to re-configure the I/O pins through the ioc file, which you will be doing later). User code that are not within the "USER CODE" sections will be deleted when new initialization code are generated through the ioc file.

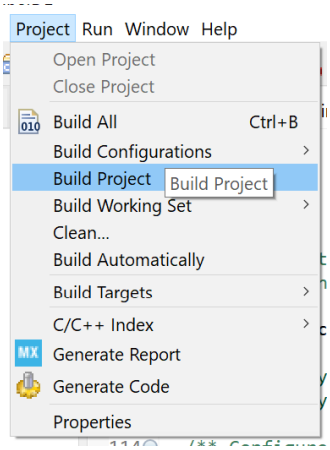
The STM32CubeIDE platform also provides various Hardware Abstraction Layer (HAL) set of APIs that can be used by upper layer software (such as the user application) to access the hardware. (Google for "STM32F4 HAL Documentation" for complete list of the HAL APIs.)

The following shows the two lines of HAL instructions that can be used to blink the LED3 you configured earlier. (It should be obvious how these two lines of code drive the LED.)

```
91    /* Infinite loop */
92    /* USER CODE BEGIN WHILE */
93    while (1)
94    {
95        HAL_GPIO_TogglePin(GPIOE, GPIO_PIN_10);
96        HAL_Delay(1000);
97      /* USER CODE END WHILE */
98
99      /* USER CODE BEGIN 3 */
100   }
101   /* USER CODE END 3 */
```

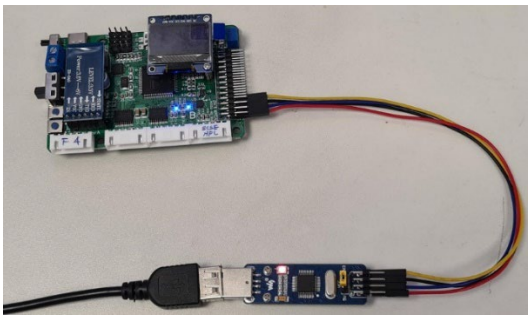
3. Executing your application program

To run the C program, you will need to first compile the program. Go to the "Project" tab and select "Project → Build Project" option in the IDE.

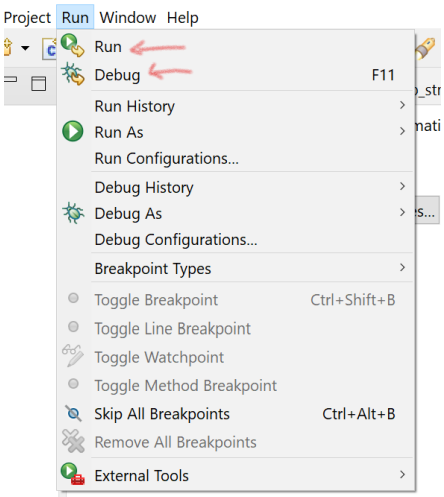


If there is no error, you can then download the code to the STM32F4 board by using the SWD ST-Link debugger board.

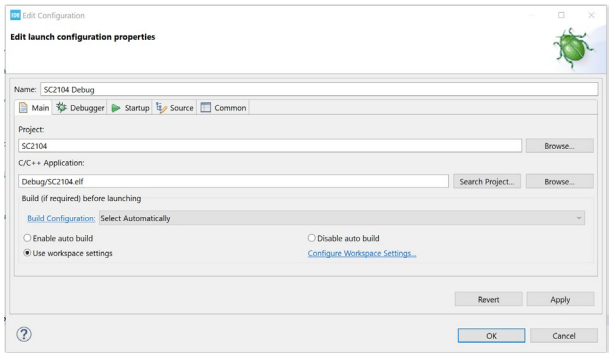
First connect the ST-Link board between the computer and the STM32F4 board. (Note that with this setup, the ST-Link board is able to power up the STM32F4 board without the need of external power source.)

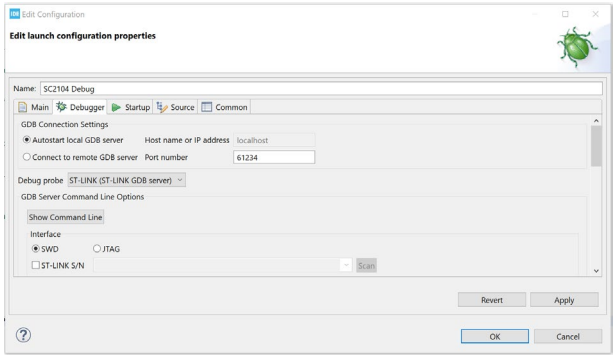


To download the code, on the "Run" tab, select either "Run → Run", or "Run → Debug".



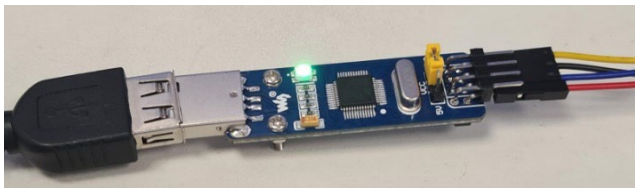
The first time you download the code, a pop-up message will appear, requesting you to confirm the ST-Link debug board setup.



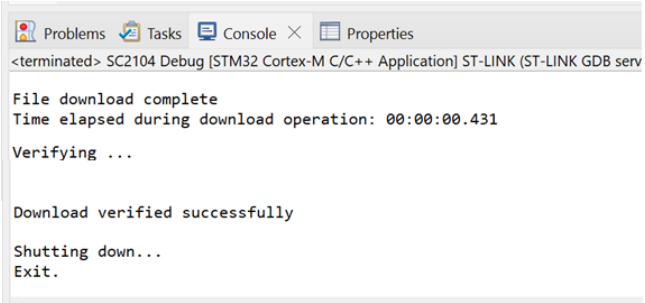


Typically, there is no change needed to be made with the default setup, and you can just proceed to download the code.

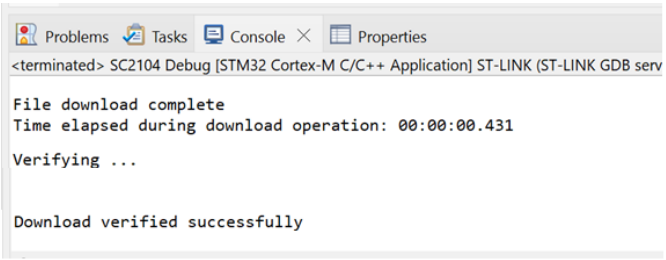
During the download, the LED on the ST-Link board will blink between red and green color. (If not, unplug its USB cable on the PC and re-insert it to retry)



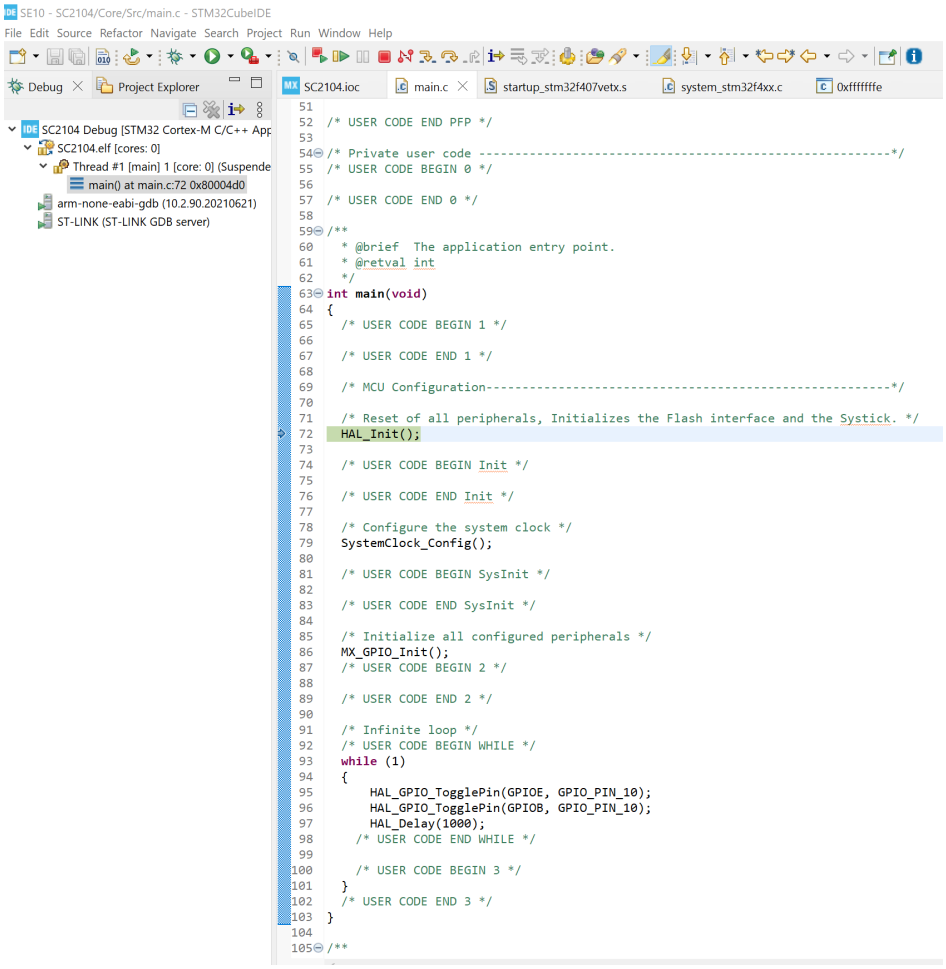
a. If you use "Run → Run", option, the debugger will exit once the program is downloaded successfully (with its LED remain red color). The STM32F4 board should immediately execute the program.



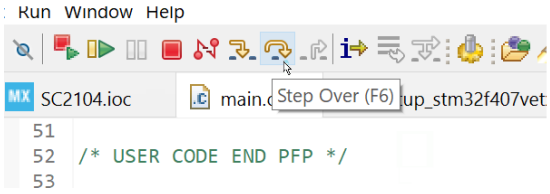
b. If you use "Run → Debug", the debugger will pause the execution of the program after it is downloaded (with the LED on the ST-Link board continue to blink between red and green color)



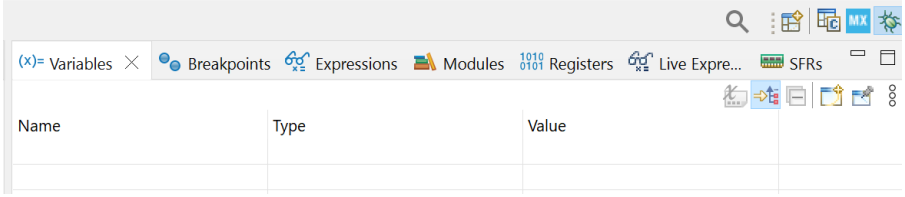
The following screenshot shows the program that is paused at the `HAL_Init()` statement when executing in Debug mode.



You can proceed to single step the program by using "Step Over" function, and monitor the execution of the code line-by-line (and observe the contents of any Variables as the code is executed)

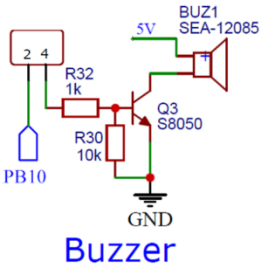


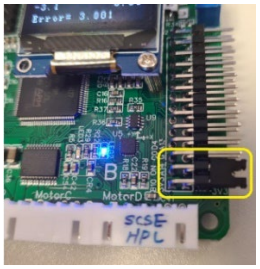
The panel shown below will allow you to select and observe the values of Variables etc. when you single step the program.



4. Practice #1 - Buzzer

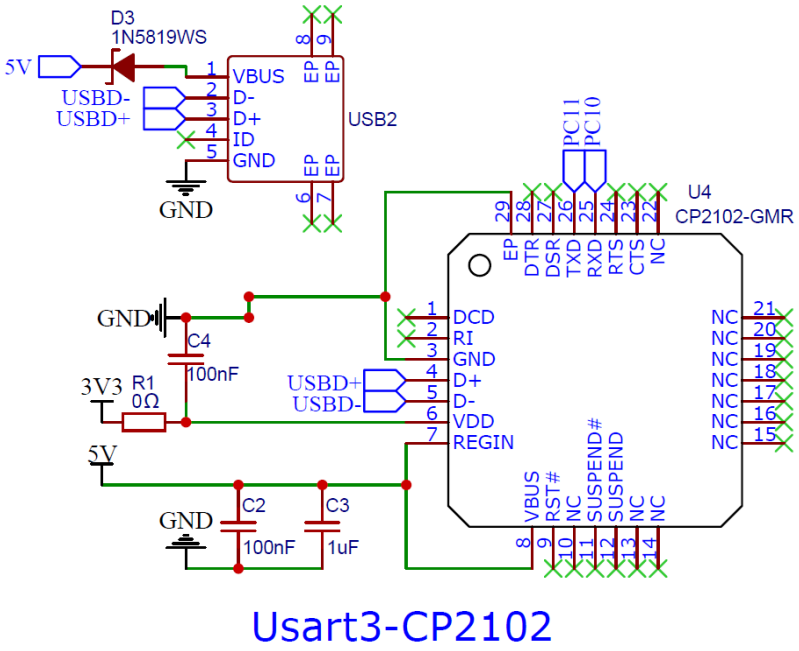
Add the code needed to operate the buzzer based on the configuration you have done earlier. The Buzzer will sound when it is turned on.





5. Practice #2 - Serial Port Communication

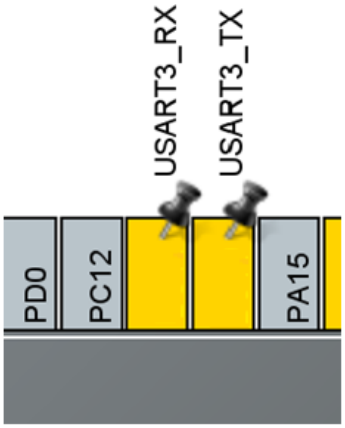
You will next develop an application to send data through the serial port (to the PC). This is to be done through **USART3** serial port based on the circuit shown below.



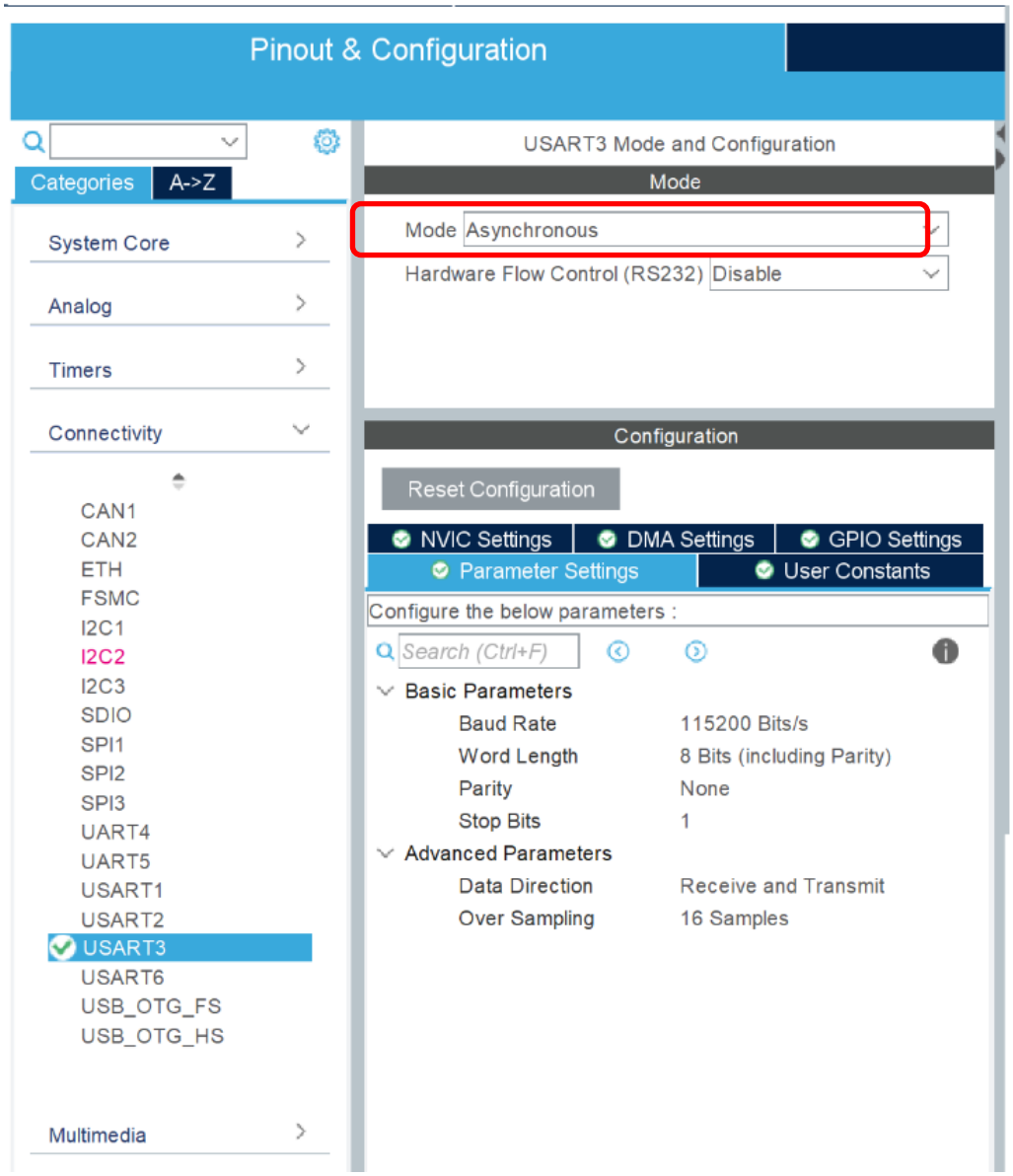
The USART3 serial communication is interfaced through the two pins: *Pxxx?* and *Pxxx?* as shown in the above schematic.

To use this serial interface, you need to first configure these two pins. This is to be done through the ioc file as before.

Select these two pins of the microcontroller, and assign them the USART3's RX and TX functions (these serial port functions are built-in for these two pins).



In addition to setting up the RX and TX function to the pins, you also need to enable the USART3 serial port through the “Connectivity” entry on the left panel as shown below, by changing USART3’s Mode to “Asynchronous”



As before, do a “File → Save” to let the IDE auto-generates the code needed to activate the serial port.

Extra lines of code should now appear in the GPIO_Init function showing the configuration of the two pins for the serial port operation.

Another function, MX_USART3_UART is also automatically generated which is used to initialize the serial port operation.

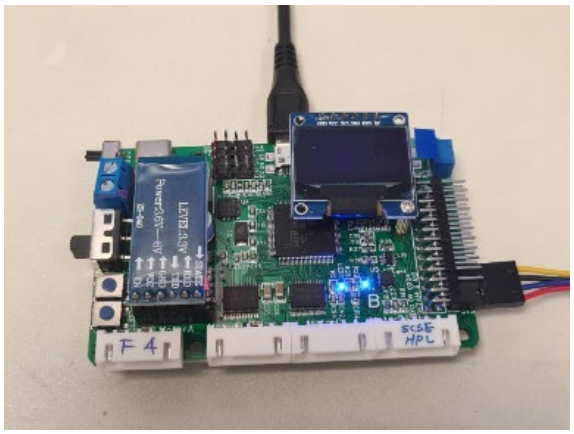
```
154 /**
155   * @brief USART3 Initialization Function
156   * @param None
157   * @retval None
158   */
159 static void MX_USART3_UART_Init(void)
160 {
161
162   /* USER CODE BEGIN USART3_Init 0 */
163
164   /* USER CODE END USART3_Init 0 */
165
166   /* USER CODE BEGIN USART3_Init 1 */
167
168   /* USER CODE END USART3_Init 1 */
169   huart3.Instance = USART3;
170   huart3.Init.BaudRate = 115200;
171   huart3.Init.WordLength = UART_WORDLENGTH_8B;
172   huart3.Init.StopBits = UART_STOPBITS_1;
173   huart3.Init.Parity = UART_PARITY_NONE;
174   huart3.Init.Mode = UART_MODE_TX_RX;
175   huart3.Init.HwFlowCtl = UART_HWCONTROL_NONE;
176   huart3.Init.OverSampling = UART_OVERSAMPLING_16;
177   if (HAL_UART_Init(&huart3) != HAL_OK)
178   {
179     Error_Handler();
180   }
181   /* USER CODE BEGIN USART3_Init 2 */
182
183   /* USER CODE END USART3_Init 2 */
184
185 }
```

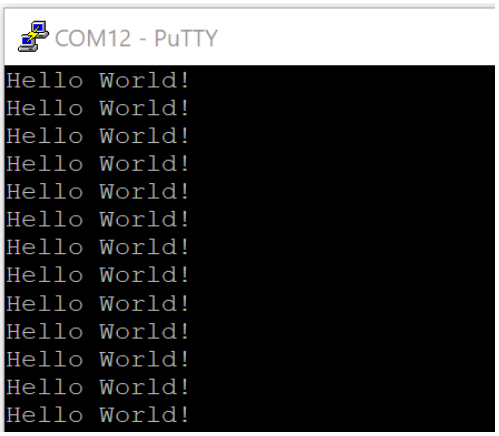
5.1 Coding and executing the serial program

The following two lines of code are examples of how messages can be sent through the serial port. Include them in the appropriate places in your STM32F4 program.

```
uint8_t sbuf[] = "Hello World!\n\r";
HAL_UART_Transmit(&huart3, sbuf, sizeof(sbuf), HAL_MAX_DELAY);
```

Connect the STM32F4 board USART3 serial port to the computer using a USB cable.



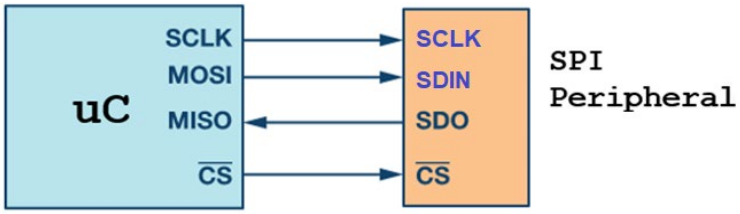


Run the PuTTY program on the computer to display the messages sent by your STM32F4 program.

6. Practice #3 – SPI interface to the OLED Display

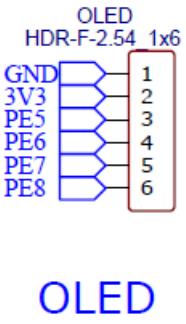
The on-board OLED Display is driven by a controller chip (SSD1306) to display various messages sent by the STM32F4 microcontroller using SPI based transferring protocol.

Standard SPI interface uses 3 signals, the SCLK (Clock), SDIN (Data in) and SDO (Data out), plus a CS# (Chip Select)



As the OLED Display only accepts and displays data sent by the STM32F4 microcontroller, and does not send data to the microcontroller, SDO is not used in the STM32F4 circuit board interface. Furthermore, the CS# signal is tied low on the OLED Display such that it is always enabled, ready to accept data and command from the microcontroller.

The OLED Display on the STM32F4 board is to be driven by PE5 to PE8 using the SPI signaling protocol.

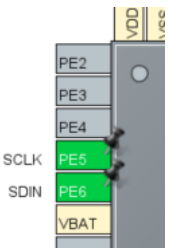


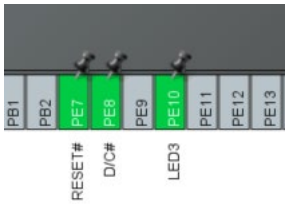
However, PE5 to PE8 do not have built-in SPI function. As such, in this exercise, you will need to code a function to perform the SPI signaling using the “bit banging” approach.

6.1 Configurating the interface pins

Firstly, configure the pins PE5-PE8 as general output to serve the following functions:

- PE5 = SCLK - the clock signal
- PE6 = SDIN - the data in signal
- PE7 = RESET# - reset signal, asserted to reset the OLED display
- PE8 = DATA/COMMAND# - signal to indicate whether the SDIN signal is Data (to be displayed) or Command (to access the display’s built-in registers).

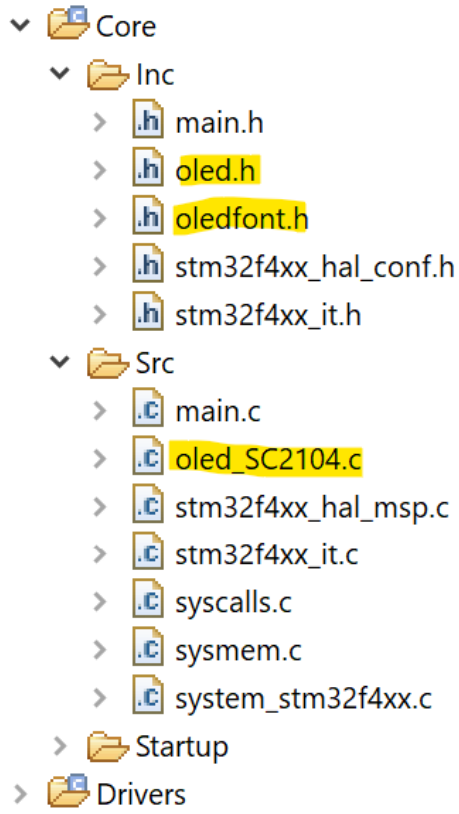




6.2 Library functions and sample program

For this OLED Display interface, library functions to display messages on the Display are provided, together with the necessary driver codes to operate the Display, except the function "`OLED_WR_Byte`" that emulate the SPI signaling which you will develop.

These OLED Display related functions and driver code are to be added to the project as highlighted below. (These three files are available in the NTULearn courses site, under the "Laboratory" folder).



The following shows the prototypes of the various library functions that are used to operate the OLED Display (these are declared in the "oled.h" file)

```
//OLED Control Functions
void OLED_WR_Byte(uint8_t dat,uint8_t cmd);  // to be implemented
void OLED_Display_On(void);
void OLED_Display_Off(void);
void OLED_Refresh_Gram(void);
void OLED_Init(void);
void OLED_Clear(void);
void OLED_DrawPoint(uint8_t x,uint8_t y,uint8_t t);
void OLED_ShowChar(uint8_t x,uint8_t y,uint8_t chr,uint8_t size,uint8_t mode);
void OLED_ShowNumber(uint8_t x,uint8_t y,uint32_t num,uint8_t len,uint8_t size);
void OLED_ShowString(uint8_t x,uint8_t y,const uint8_t *p);
void OLED_Set_Pos(unsigned char x, unsigned char y);
```

The followings show the sample code of a program (in the main.c file) to display messages on the OLED Display, using some of the library functions provided.

```
  ******************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------
#include "main.h"

/* Private includes ----------------------
/* USER CODE BEGIN Includes */
#include "oled.h"

/* USER CODE END Includes */
```

:

```
int main(void)
{
  /* USER CODE BEGIN 1 */
  uint8_t *oled_buf; // buffer to store value to be displayed on OLED
  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();
```

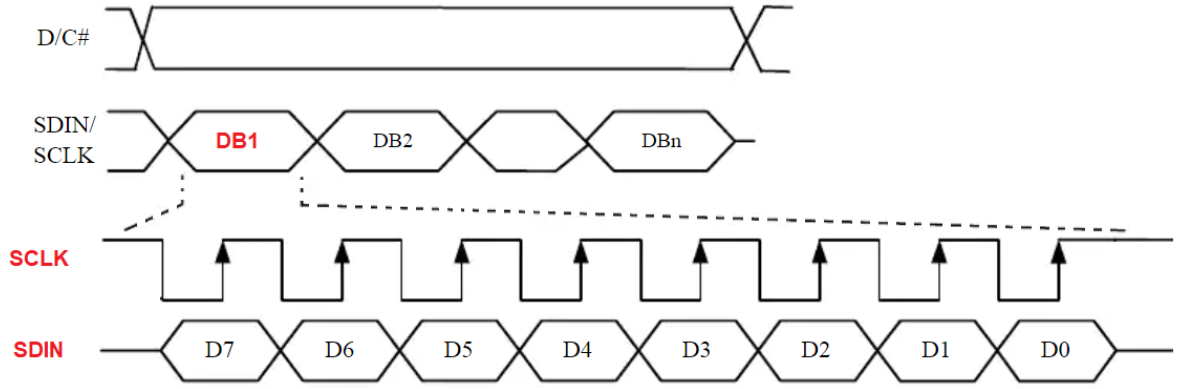
:

```
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART3_UART_Init();
  MX_USART2_UART_Init();

  /* USER CODE BEGIN 2 */
  OLED_Init(); // Initialize the display
  OLED_ShowString(0, 5, "Hello World"); // display message on OLED display at line 5
  OLED_Refresh_Gram(); // show message

  HAL_Delay(2000);    // display for 2 second
  OLED_Clear(); // clear the display

  oled_buf = "STM32 Board"; // another way to display message through buffer
  OLED_ShowString(10,30, oled_buf); //message at line 30
  OLED_Refresh_Gram(); // show message

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  uint8_t sbuf[] = "Hello World!\n\r";
  while (1)
  {
      HAL_GPIO_TogglePin(GPIOE, GPIO_PIN_10); // LED3
```

6.3 The `OLED_WR_Byte()` function

The `OLED_WR_Byte()` in the oled_SC2104.c is the key function called by all the OLED library APIs to generate the SPI signalling for sending the data to the OLED display.

The following timing diagram shows how the SCLK and the SDIN are to perform the SPI signalling to transfer the data.



Below is the skeleton code of the `OLED_WR_Byte()` function that you will need to complete in this exercise, to generate the two signals SCLK and SDIN as shown above.

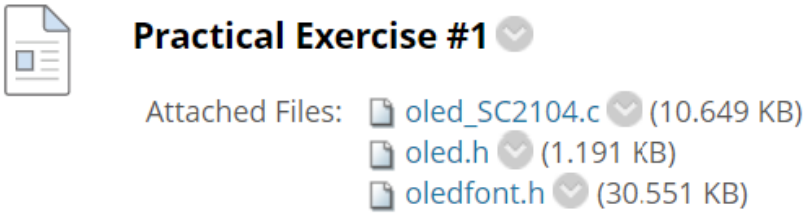
```
/**********************************************************************************
Function: Send the data/command to the OLED Display controller using SPI bit-banging
Input   : dat: data/command on SDIN pin
          DataCmd: data/command# on D/C# pin
               1 => sending data
               0 => sending command
Output  : none

**********************************************************************************/

void OLED_WR_Byte(uint8_t dat,uint8_t DataCmd)
{
    uint8_t i;

    if(DataCmd == 1)          // Data write
      OLED_RS_Set();          // Set the D/C# line
    else                      // Command write
      OLED_RS_Clr();          // Clear the D/C# line

    for(i=0;i<8;i++)
    { // Complete the code below
            :
            :
            :
    }

    OLED_RS_Set();          // Keep the D/C# line high upon exit
}
```

You can call the following functions in order to toggle the two signals:

```
OLED_SCLK_Set();    // set SCLK
OLED_SCLK_Clr();    // clear SCLK
OLED_SDIN_Set();    // Set SDIN
OLED_SDIN_Clr();    // clear SDIN
```

6.4 Program development

(a) Download the following files from the NTULearn course site (under the "Laboratory → Practical Exercises" folder) and save it to a local directory on the computer.



(b) Import (see Appendix) all three files into your project.

(c) Complete the **`OLED_WR_Byte()`** function in the oled_SC2104.c file as described in 6.3.

(d) Add the message displaying code shown in 6.2 to your **`main.c`**.

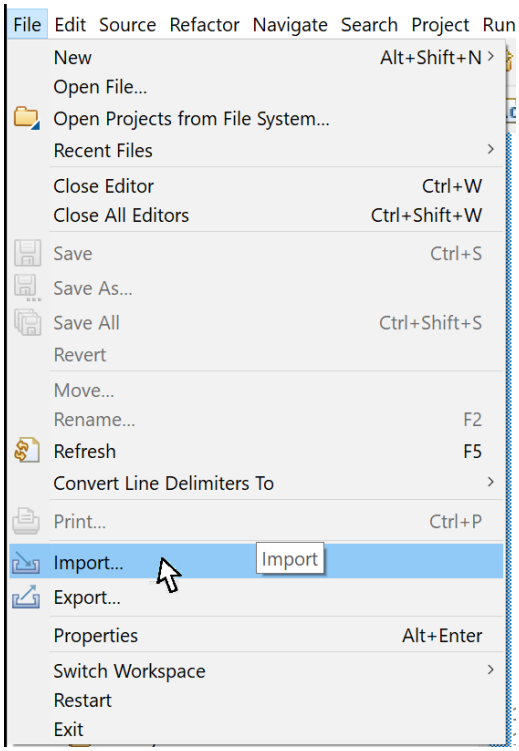
(e) Compile and download your program to the STN32F4 board to see that it is executing as expected.

If it does not work, this is the good opportunity for you to learn how to use the single stepping feature of the debugger to debug your program.

Appendix

File can be added to the project by using the "Import" function.

(a) Under the "File" tab, select the "Import" function



(b) Then select "General → File System", and follow the instructions to select the directory to import the file needed.