Use of an I2C LCD Display

rev1.0 24/03/2020

GOAL

Write on a LCD Display connected via I2C protocol

PREREQUISITES

Software needed:

- STM32IDE

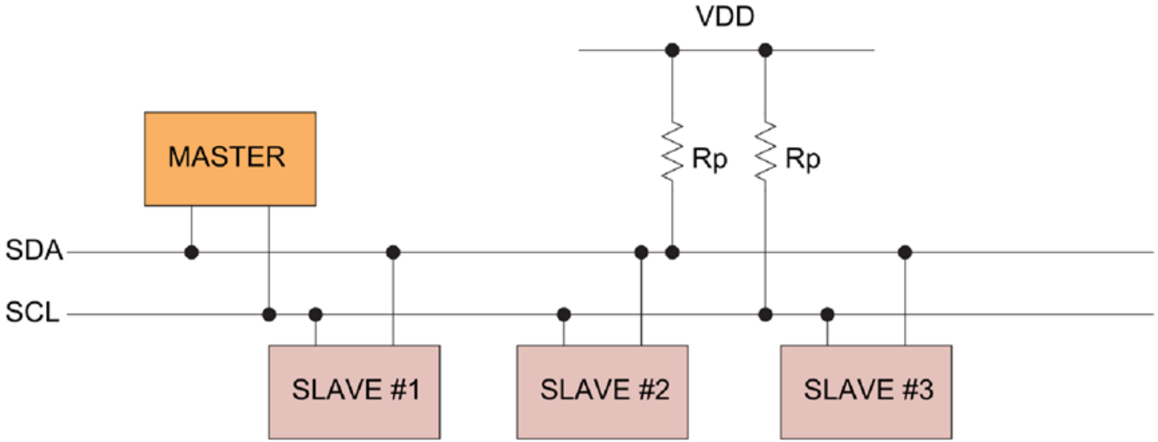
Hardware used in this example:

- **NUCLEO-F446ZE**
- **I2C Display**

I²C (Inter-Integrated Circuit)

I²C is a synchronous, multi-master, multi-slave bus invented in 1982 by Philips Semiconductor. It is widely used for attaching lower-speed peripheral ICs to processors and uC in short-distance, intra-board communication.
This protocol use only two data wires:

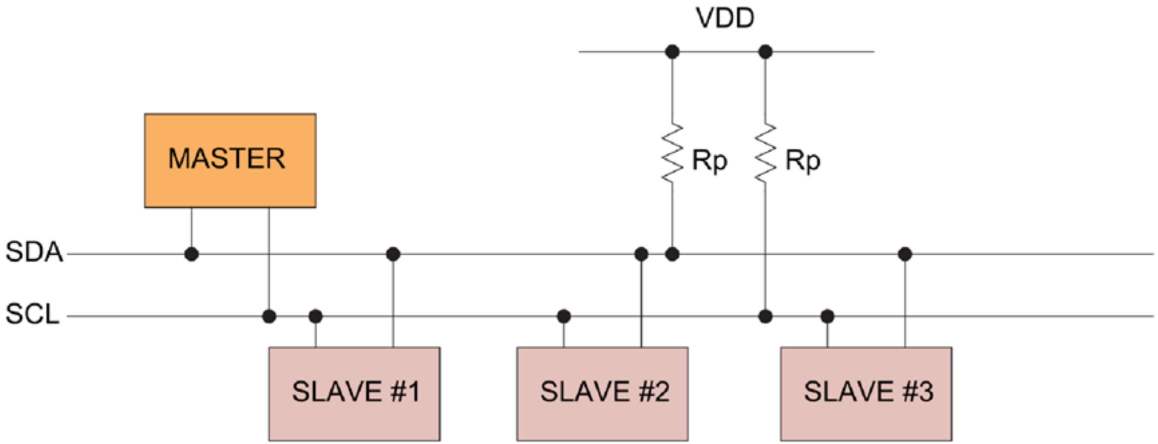
- *SDA* (Serial Data)
- *SCL* (Serial Clock)



I²C (Inter-Integrated Circuit)

Masters and Slaves shares the same data wires:

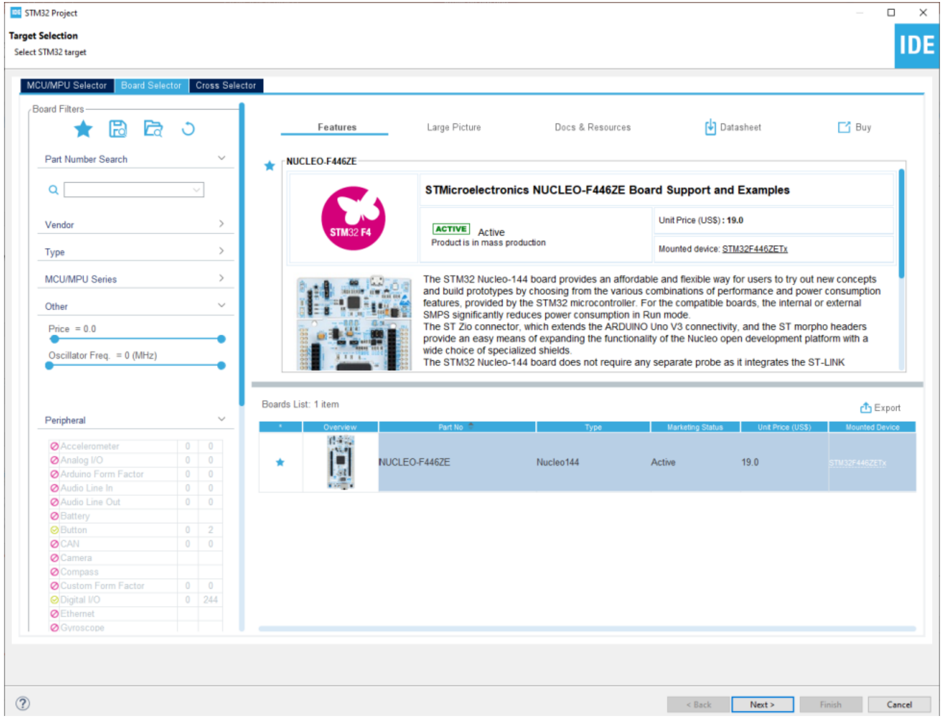
In order to communicate each slave has an address in order to differentiate from the others: so each Slave should have an *UNIQUE* address, different from the other ones.



Start a new project

From the stm32IDE software click on File -> New -> STM32 Project.

Select your board or your uC and click *next.*



Target Selection

Select STM32 target

Price = 0.0

Oscillator Freq. = 0 (MHz)

Accelerometer	0	0
Analog I/O	0	0
Arduino Form Factor	0	0
Audio Line In	0	0
Audio Line Out	0	0
Battery		
Button	0	2
CAN	0	0
Camera		
Compass		
Custom Form Factor	0	0
Digital I/O	0	244
Ethernet		
Gyroscope		

NUCLEO-F446ZE

STMicroelectronics NUCLEO-F446ZE Board Support and Examples

Active

Product is in mass production

Unit Price (US$) : 19.0

Mounted device: STM32F446ZETx

The STM32 Nucleo-144 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 ST Zio connector, which extends the ARDUINO Uno V3 connectivity, and the ST morpho headers provide an easy means of expanding the functionality of the Nucleo open development platform with a wide choice of specialized shields.
The STM32 Nucleo-144 board does not require any separate probe as it integrates the ST-LINK

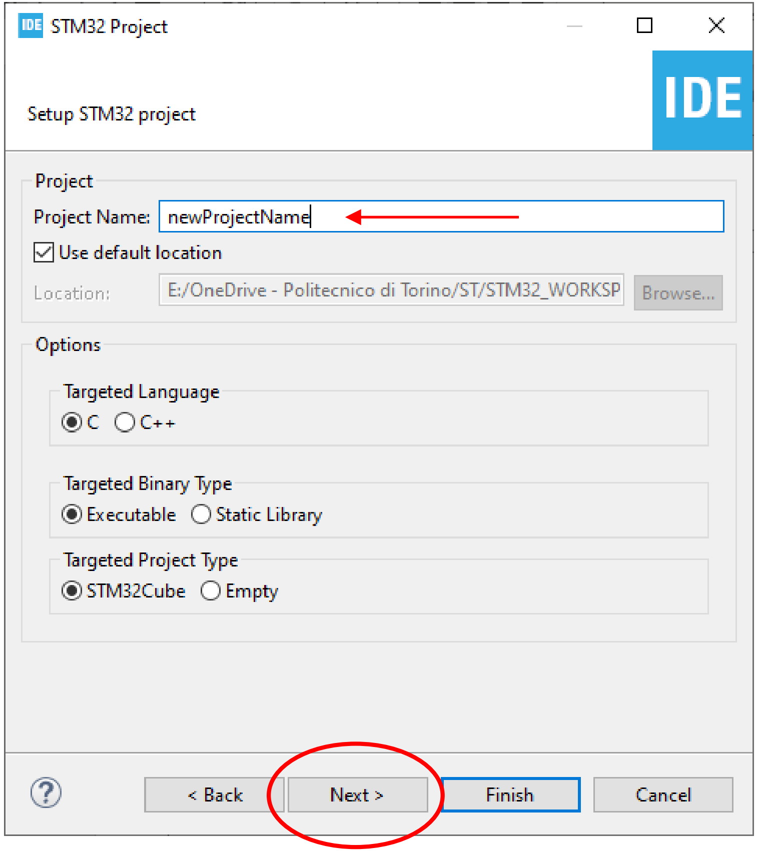
Boards List: 1 item

*	Overview	Part No	Type	Marketing Status	Unit Price (US$)	Mounted Device
★		NUCLEO-F446ZE	Nucleo144	Active	19.0	STM32F446ZETx

Start a new project

Type the name of your project and click next.

By default the project will be created in the workspace folder.



Setup STM32 project

Project

Project Name: newProjectName

☑ Use default location

Location: E:/OneDrive - Politecnico di Torino/ST/STM32_WORKSP Browse...

Options

Targeted Language

● C ○ C++

Targeted Binary Type

● Executable ○ Static Library

Targeted Project Type

● STM32Cube ○ Empty

Start a new project

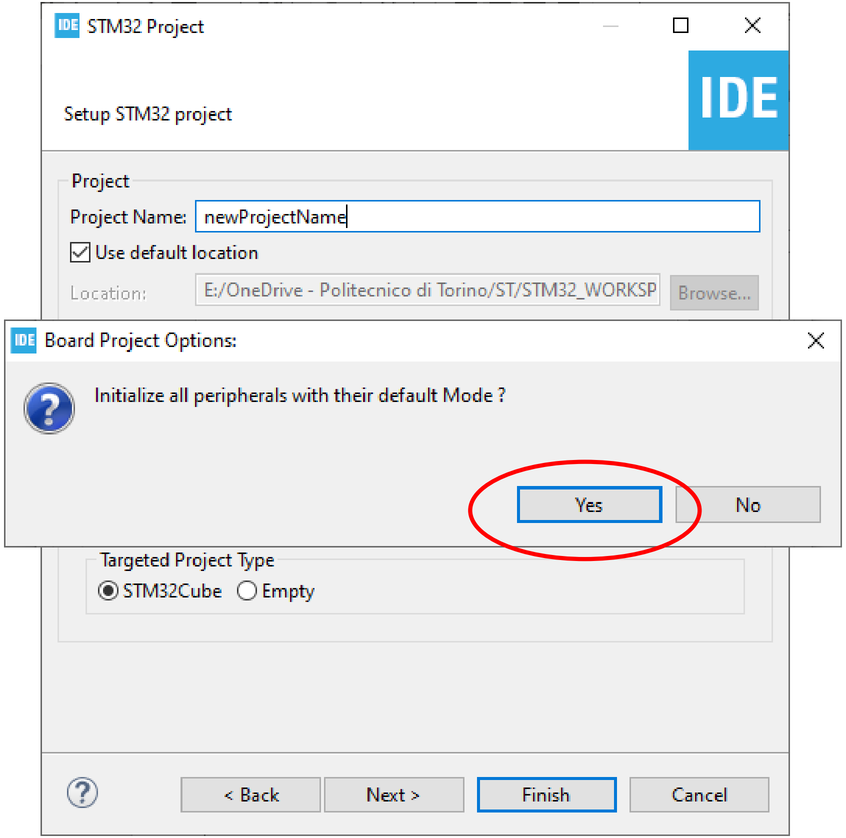
Type the name of your project and click next.

By default the project will be created in the *workspace* folder.

The *STM32IDE* has the option to initialize all the peripheral with their ***default*** mode:

Clicking *Yes* the *USART3*, all the *LEDs* and the blue *UserButton* will be configured as default.

Click *Yes*.



STM32 Project

Setup STM32 project

Project

Project Name: newProjectName

☑ Use default location

Location: E:/OneDrive - Politecnico di Torino/ST/STM32_WORKSP

Board Project Options:

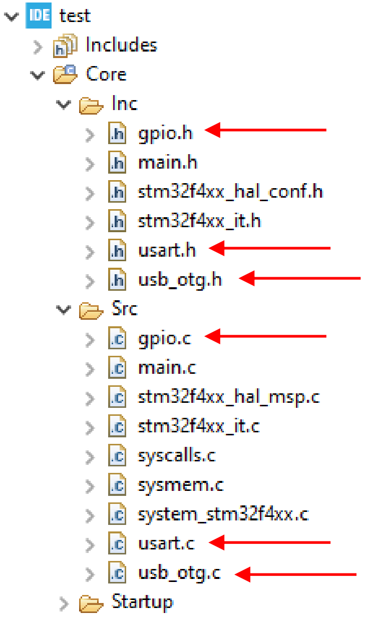
Initialize all peripherals with their default Mode ?

Targeted Project Type

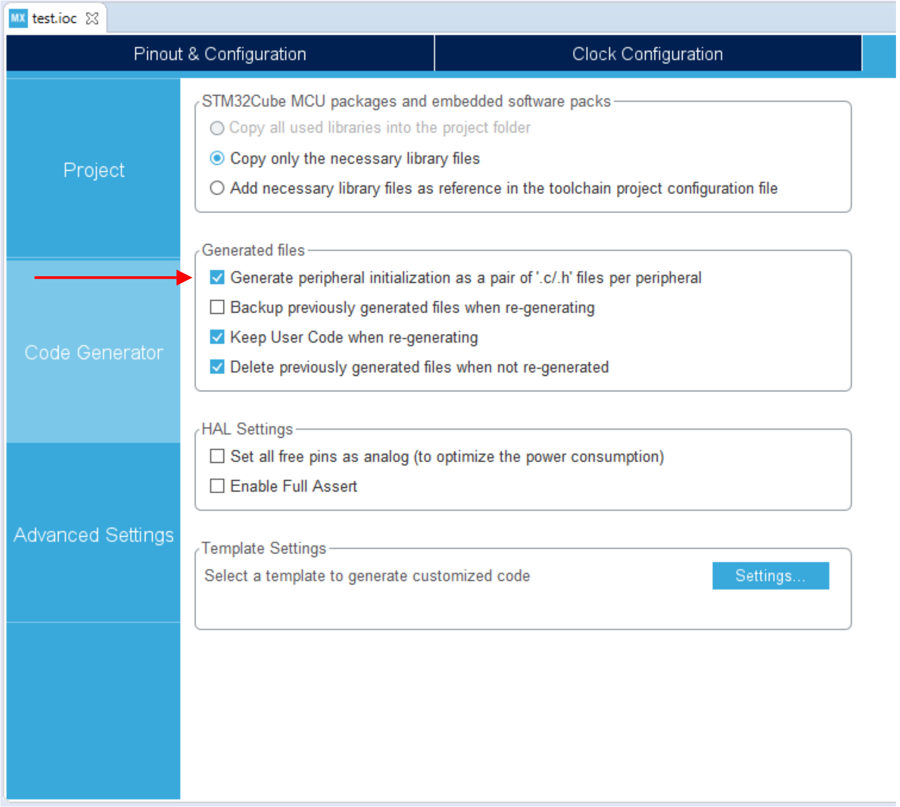
● STM32Cube ○ Empty

Project Manager

In the Code Generator Tab check the ***Generate peripheral initialization […]*** box: each periperhal will have a disting *periph.c* and *periph.h* files.



```
test
  Includes
  Core
    Inc
      gpio.h
      main.h
      stm32f4xx_hal_conf.h
      stm32f4xx_it.h
      usart.h
      usb_otg.h
    Src
      gpio.c
      main.c
      stm32f4xx_hal_msp.c
      stm32f4xx_it.c
      syscalls.c
      sysmem.c
      system_stm32f4xx.c
      usart.c
      usb_otg.c
    Startup
```



STM32Cube MCU packages and embedded software packs

○ Copy all used libraries into the project folder
● Copy only the necessary library files
○ Add necessary library files as reference in the toolchain project configuration file

Generated files

☑ Generate peripheral initialization as a pair of '.c/.h' files per peripheral
□ Backup previously generated files when re-generating
☑ Keep User Code when re-generating
☑ Delete previously generated files when not re-generated

HAL Settings

□ Set all free pins as analog (to optimize the power consumption)
□ Enable Full Assert

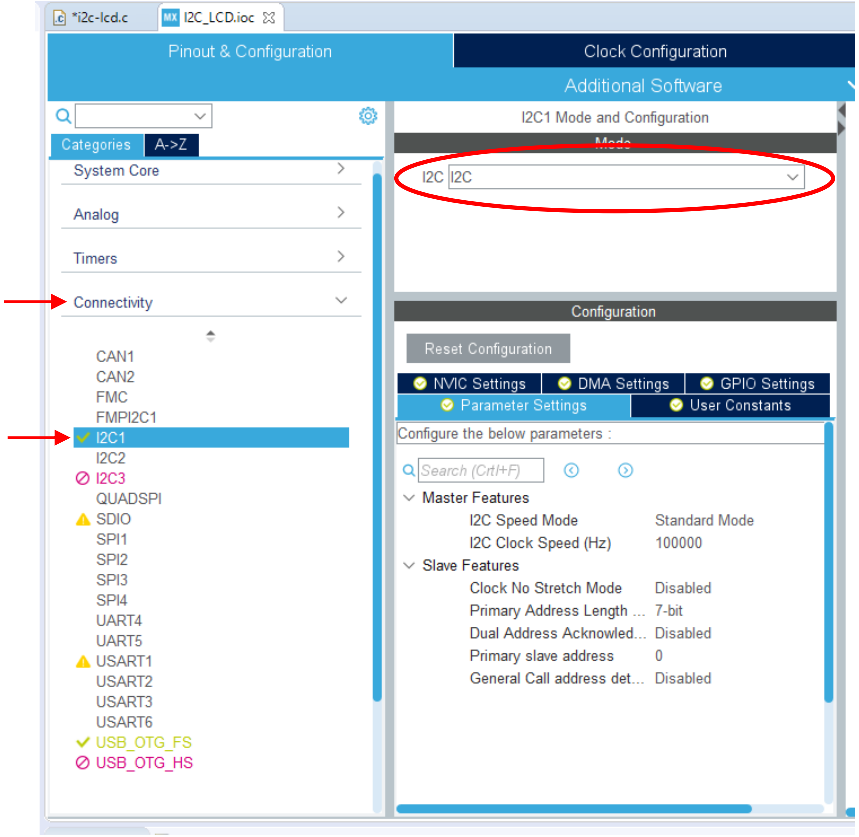
Template Settings

Select a template to generate customized code

Configure the peripherals

For this project we’ll need to enable the I2C bus, so go under *connectivity*, select ***I2C1*** and ***enable*** it.

All the configuration ends here, so now you can generate the code (click on the generate icon).



I2C1 Mode and Configuration

I2C I2C

Configure the below parameters :

Master Features

I2C Speed Mode	Standard Mode
I2C Clock Speed (Hz)	100000

Slave Features

Clock No Stretch Mode	Disabled
Primary Address Length ...	7-bit
Dual Address Acknowled...	Disabled
Primary slave address	0
General Call address det...	Disabled

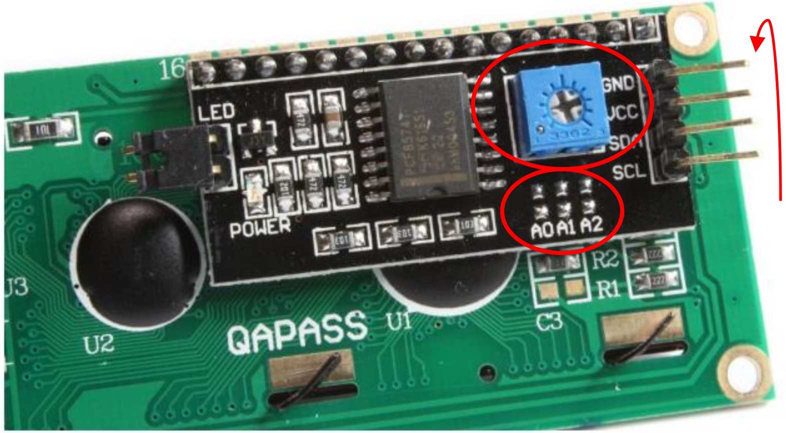
Connect the LCD

On the back of the LCD we can find a black board: it’s function is to let us communicate with the LCD via I2C. This board is based on the pcf8574 (or eventually pcf8574A, check your board).

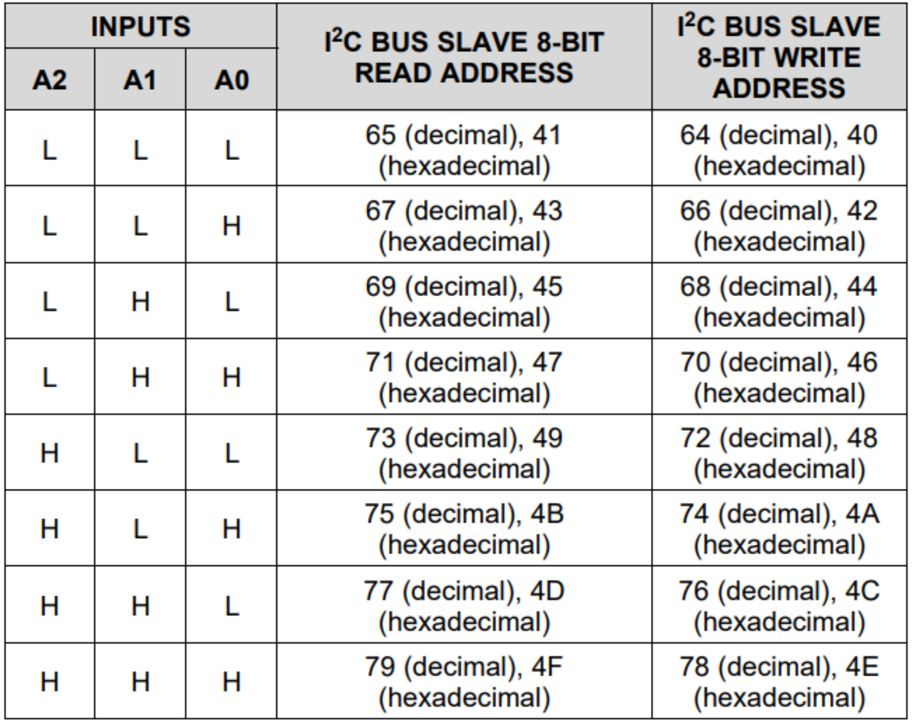
The “blue Box” on the back is a trimmer, rotating it with a screwdriver we can adjust the contrast.

It’s possible to configure the address of the LCD using the 3 jumper A0, A1 and A2.

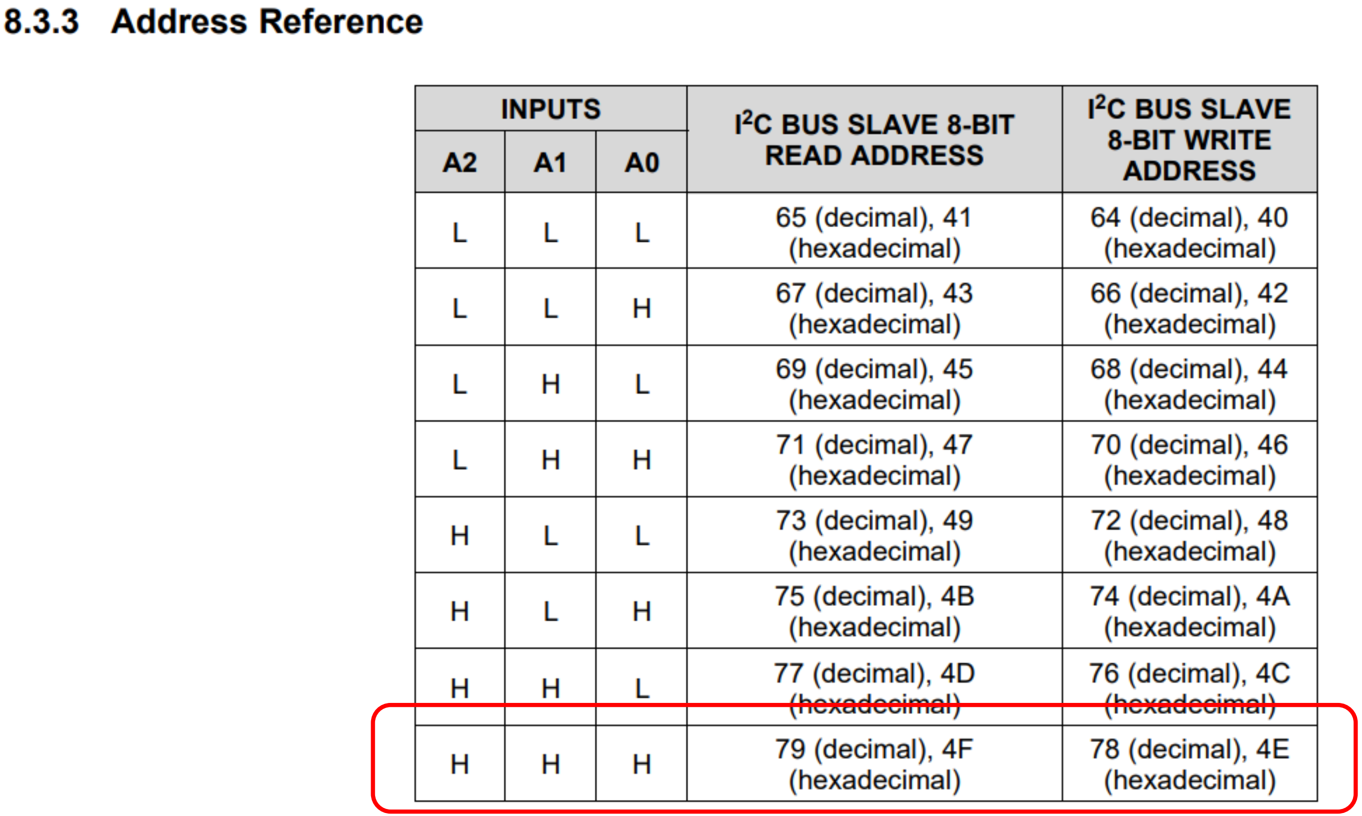
In the next page you will find all the configuration with all the respective addresses, take in mind that when the jumper is open the state us ‘H’ .



8.3.3 Address Reference



INPUTS			I^{2}C BUS SLAVE 8-BIT READ ADDRESS	I^{2}C BUS SLAVE 8-BIT WRITE ADDRESS
A2	A1	A0		
L	L	L	65 (decimal), 41 (hexadecimal)	64 (decimal), 40 (hexadecimal)
L	L	H	67 (decimal), 43 (hexadecimal)	66 (decimal), 42 (hexadecimal)
L	H	L	69 (decimal), 45 (hexadecimal)	68 (decimal), 44 (hexadecimal)
L	H	H	71 (decimal), 47 (hexadecimal)	70 (decimal), 46 (hexadecimal)
H	L	L	73 (decimal), 49 (hexadecimal)	72 (decimal), 48 (hexadecimal)
H	L	H	75 (decimal), 4B (hexadecimal)	74 (decimal), 4A (hexadecimal)
H	H	L	77 (decimal), 4D (hexadecimal)	76 (decimal), 4C (hexadecimal)
H	H	H	79 (decimal), 4F (hexadecimal)	78 (decimal), 4E (hexadecimal)



8.3.3 Address Reference

INPUTS			I^2C BUS SLAVE 8-BIT READ ADDRESS	I^2C BUS SLAVE 8-BIT WRITE ADDRESS
A2	A1	A0		
L	L	L	65 (decimal), 41 (hexadecimal)	64 (decimal), 40 (hexadecimal)
L	L	H	67 (decimal), 43 (hexadecimal)	66 (decimal), 42 (hexadecimal)
L	H	L	69 (decimal), 45 (hexadecimal)	68 (decimal), 44 (hexadecimal)
L	H	H	71 (decimal), 47 (hexadecimal)	70 (decimal), 46 (hexadecimal)
H	L	L	73 (decimal), 49 (hexadecimal)	72 (decimal), 48 (hexadecimal)
H	L	H	75 (decimal), 4B (hexadecimal)	74 (decimal), 4A (hexadecimal)
H	H	L	77 (decimal), 4D (hexadecimal)	76 (decimal), 4C (hexadecimal)
H	H	H	79 (decimal), 4F (hexadecimal)	78 (decimal), 4E (hexadecimal)

For our example we have a PCF8574 board based, so the address will be 0x4E

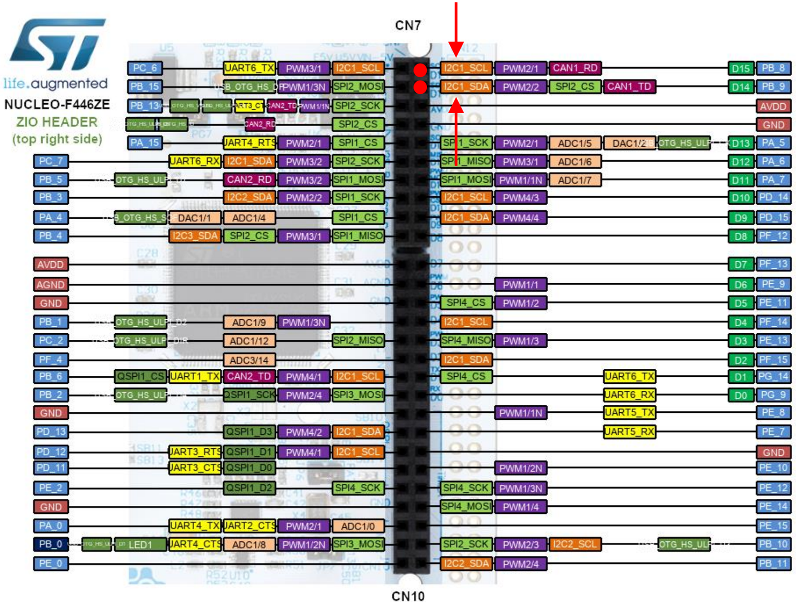
Connect the lcd

In order to communicate with the LCD you should wire it to the board.

For this example we are using a NUCLEO-F446ZE, you can find all the detailed pinout here.

Connect the SDA and the SCL pins of the LCD with the I2C1_SDA and the I2C1_SCL pins of the board (the 2 red dots).

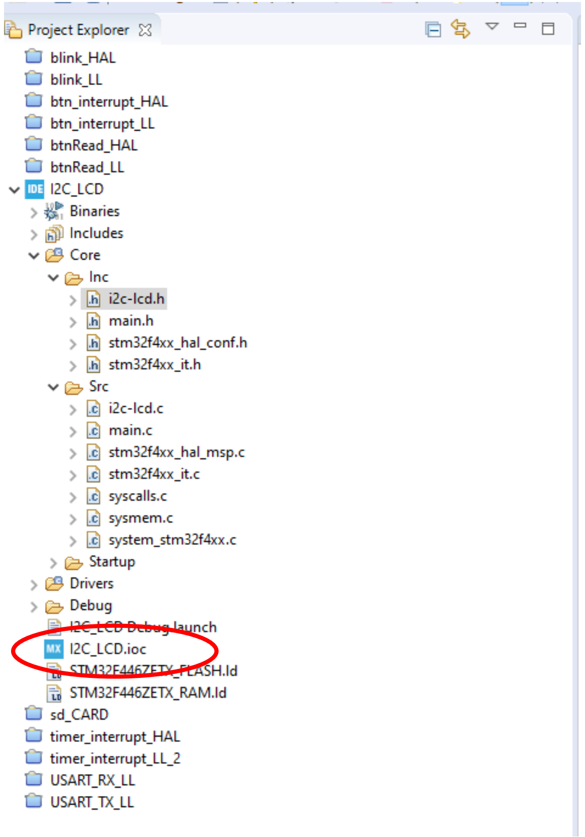
Do not forget to connect 5V and GND pins too.



I2C pins

In the NUCLEO boards there are some pins that can do the same thing: so there could be more than one pin that can be configured as *I2C1_SDA*, *I2C1_SCL* and so.

To check which are the correct pins simply go to the ***I2C_LCD.ioc*** file under the project explorer and search for the i2c pins.

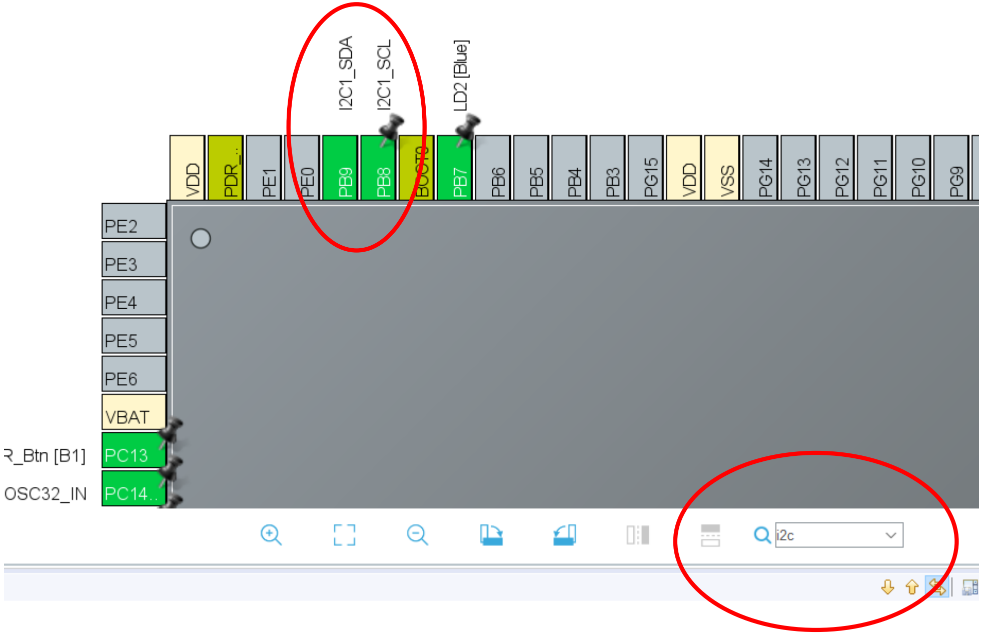


```
blink_HAL
blink_LL
btn_interrupt_HAL
btn_interrupt_LL
btnRead_HAL
btnRead_LL
I2C_LCD
  Binaries
  Includes
  Core
    Inc
      i2c-lcd.h
      main.h
      stm32f4xx_hal_conf.h
      stm32f4xx_it.h
    Src
      i2c-lcd.c
      main.c
      stm32f4xx_hal_msp.c
      stm32f4xx_it.c
      syscalls.c
      sysmem.c
      system_stm32f4xx.c
    Startup
  Drivers
  Debug
  I2C_LCD Debug.launch
  I2C_LCD.ioc
  STM32F446ZETX_FLASH.ld
  STM32F446ZETX_RAM.ld
sd_CARD
timer_interrupt_HAL
timer_interrupt_LL_2
USART_RX_LL
USART_TX_LL
```

I2C pins

As you can see in our case SDA and SCL pins are PB8 and PB9.

If you can’t find your pins you can look for them typing in the search lens field: the found pins will blink black.



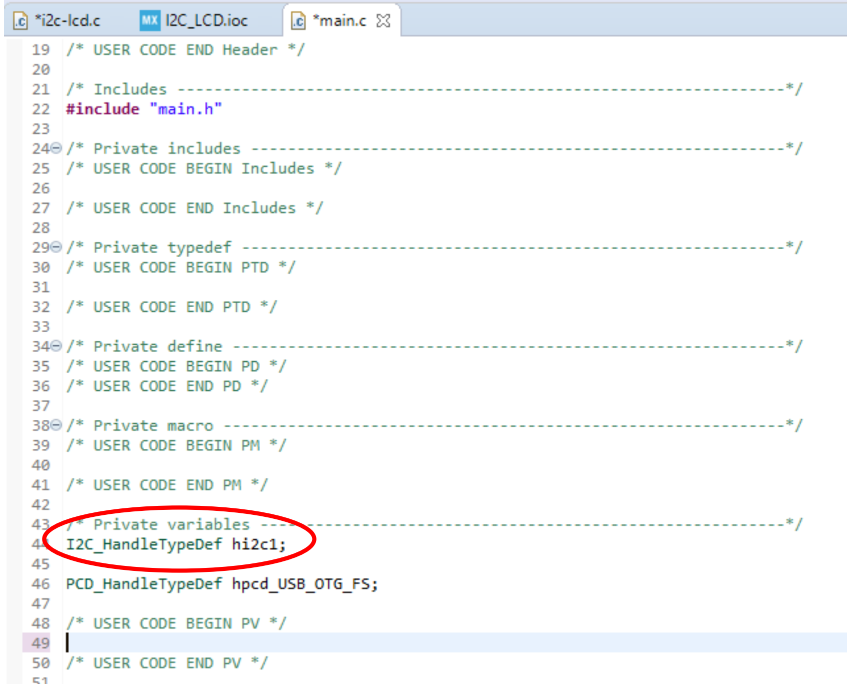
I2C Initialization

Let’s have a look in the I2C initialization function.

It’s automatically generated by *CubeMX* and we can find it in the ***main.c*** file.

Since we are using HAL Libraries for this example, CubeMX create first the I2C1 Handler under the private variables section.

Then we can scroll down and see the actual i2c initialization function.

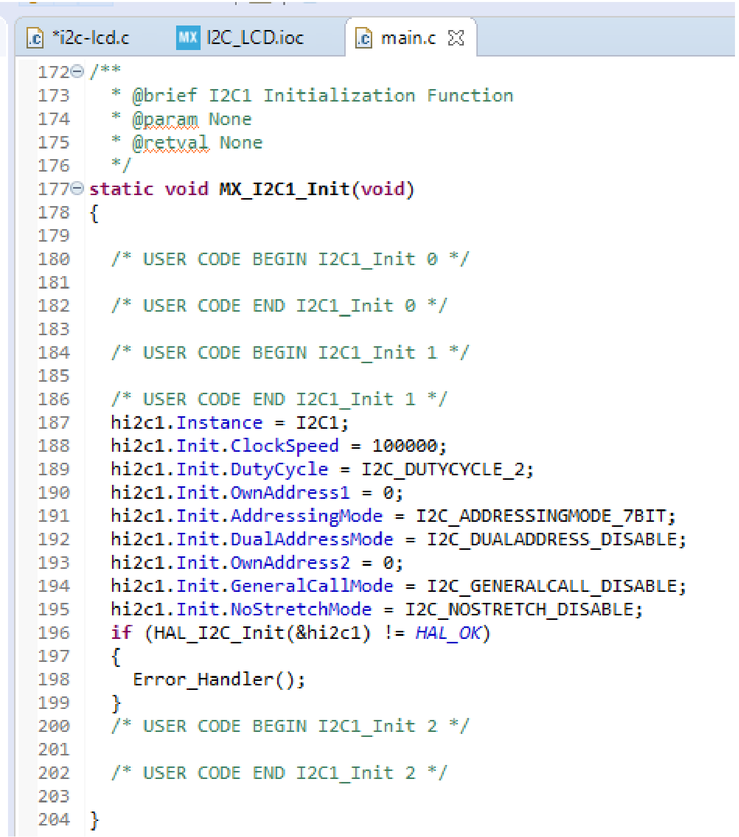


```
/* USER CODE END Header */

/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
I2C_HandleTypeDef hi2c1;

PCD_HandleTypeDef hpcd_USB_OTG_FS;

/* USER CODE BEGIN PV */

/* USER CODE END PV */
```

I2C Initialization

The code is very simple:

CubeMX takes the handler structure previously defined and fill with the parameters configured earlier.

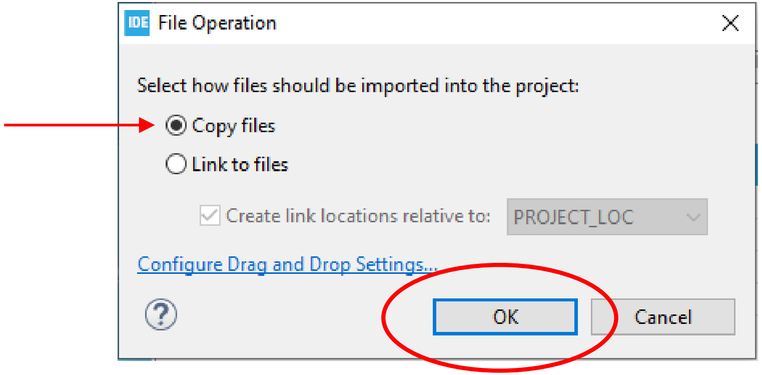


```
/**
  * @brief I2C1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */

  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 100000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */

}
```

I2C lcd Library

The last step to be able to send data to our LCD screen is to include the library that will let us send strings and numbers very easily to the screen.

Simply drag and drop the **i2c-lcd.h** and **i2c-lcd.c** file in the *inc* and *src* folders as shown in the picture. If asked, copy the file in the workspace.



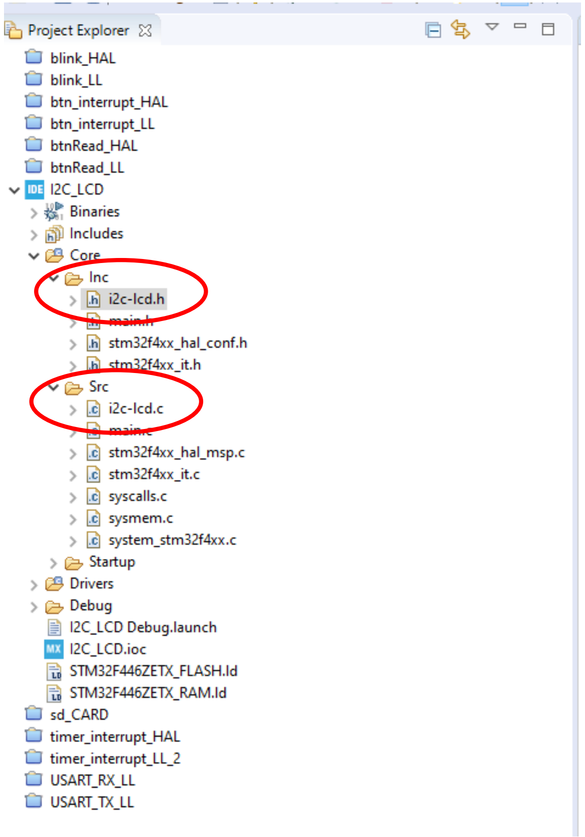
File Operation

Select how files should be imported into the project:

● Copy files

○ Link to files

☑ Create link locations relative to: PROJECT_LOC

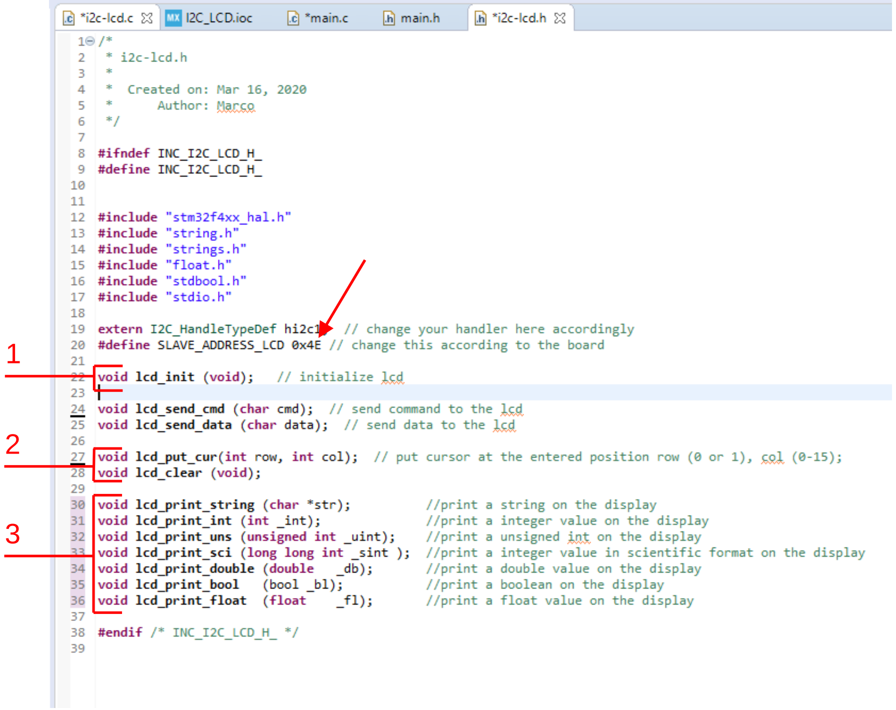


I2c-lcd library functions

In the file i2c-lcd.h we can find the prototypes of the library functions:

1. *Lcd_init():* Initialize the LCD in order to work in 4 bit mode.
2. *Lcd_put_cur():* put the cursor in a specific point of the LCD.
3. *Lcd_clear():* clean the LCD, the screen will appear empty.
4. *Lcd_print_xx():* these function allow to write any kind of data type on the screen.

Check Slave address, has to match your configuration.



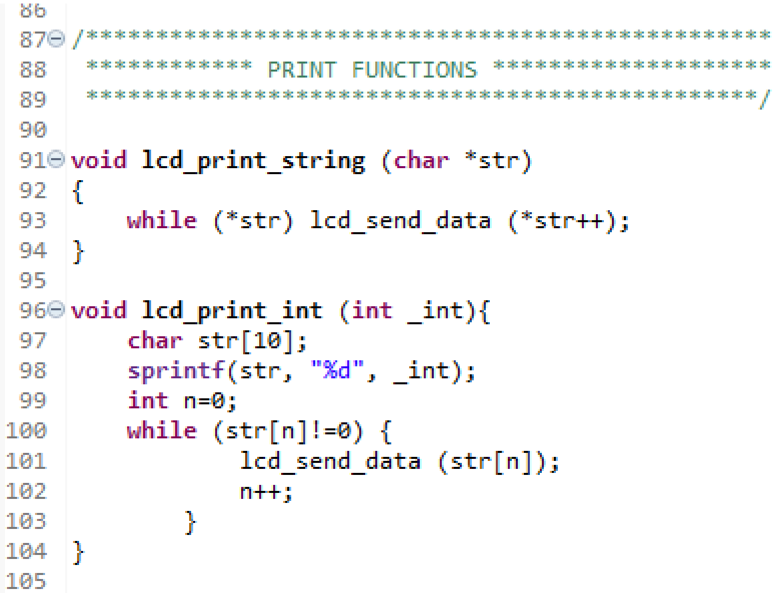
```c
/*
 * i2c-lcd.h
 *
 *  Created on: Mar 16, 2020
 *      Author: Marco
 */

#ifndef INC_I2C_LCD_H_
#define INC_I2C_LCD_H_


#include "stm32f4xx_hal.h"
#include "string.h"
#include "strings.h"
#include "float.h"
#include "stdbool.h"
#include "stdio.h"

extern I2C_HandleTypeDef hi2c1;  // change your handler here accordingly
#define SLAVE_ADDRESS_LCD 0x4E // change this according to the board

void lcd_init (void);   // initialize lcd

void lcd_send_cmd (char cmd);  // send command to the lcd
void lcd_send_data (char data);  // send data to the lcd

void lcd_put_cur(int row, int col);  // put cursor at the entered position row (0 or 1), col (0-15);
void lcd_clear (void);

void lcd_print_string (char *str);          //print a string on the display
void lcd_print_int (int _int);              //print a integer value on the display
void lcd_print_uns (unsigned int _uint);    //print a unsigned int on the display
void lcd_print_sci (long long int _sint );  //print a integer value in scientific format on the display
void lcd_print_double (double   _db);       //print a double value on the display
void lcd_print_bool   (bool _bl);           //print a boolean on the display
void lcd_print_float  (float    _fl);       //print a float value on the display

#endif /* INC_I2C_LCD_H_ */
```

LCD print functions

In the ***i2c-lcd.c*** file you can find the definition of all the library functions.

About the print function, they work all in the same way:

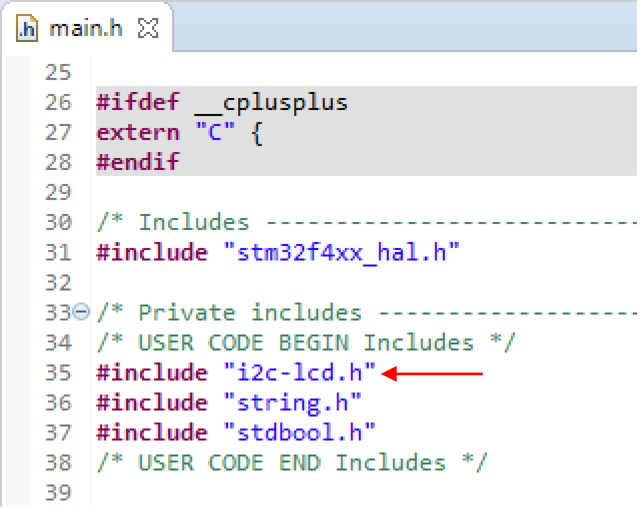
1. Creation of a char array: it contains 10 elements since the max integer value in the architecture has 10 digits (32bit architecture)
2. Copy the value passed by parameters (the integer number for this case) in the array using the ***sprintf()*** function
3. Send each character of the array with the function ***lcd_send_data().*** Notice that the value 0 (not the number ‘0’) represents the termination of the array.



```
/**************************************************
 ************ PRINT FUNCTIONS ********************
 **************************************************/

void lcd_print_string (char *str)
{
	while (*str) lcd_send_data (*str++);
}

void lcd_print_int (int _int){
	char str[10];
	sprintf(str, "%d", _int);
	int n=0;
	while (str[n]!=0) {
			lcd_send_data (str[n]);
			n++;
		}
}
```

Include the libraries

Open the main.h file and include the ***i2c-lcd.h*** file as shown.



```

#ifdef __cplusplus
extern "C" {
#endif

/* Includes ---------------------------
#include "stm32f4xx_hal.h"

/* Private includes -------------------
/* USER CODE BEGIN Includes */
#include "i2c-lcd.h"
#include "string.h"
#include "stdbool.h"
/* USER CODE END Includes */

```

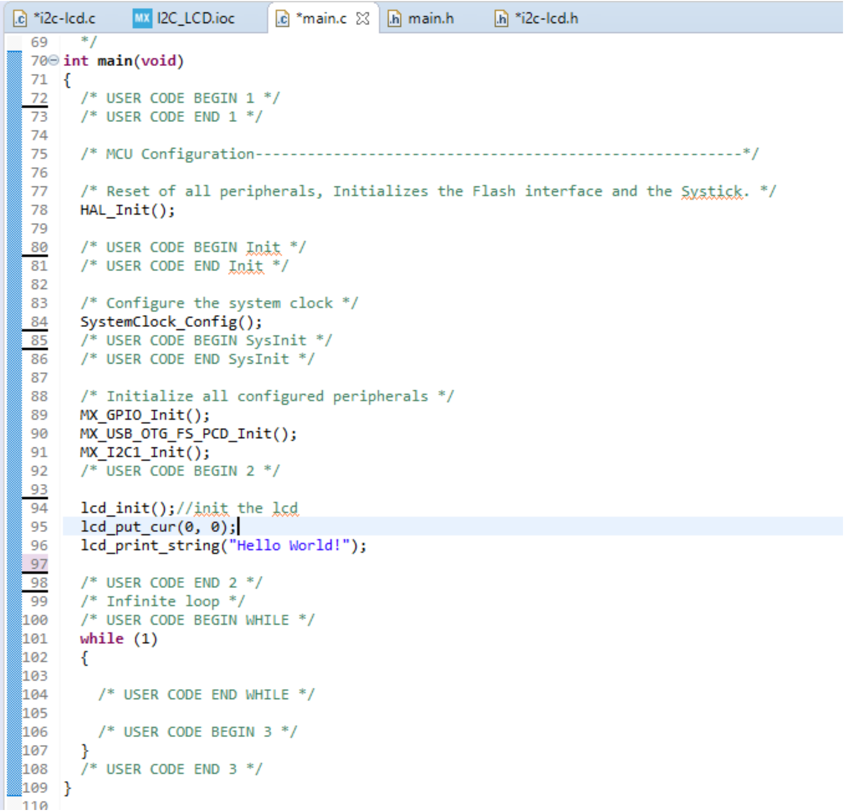
Print “HELLO WORLD”

With all of this knowledge we can try to print some texts on the display.

For this example we’ll display the string «Hello World!» after all the peripherals configurations.

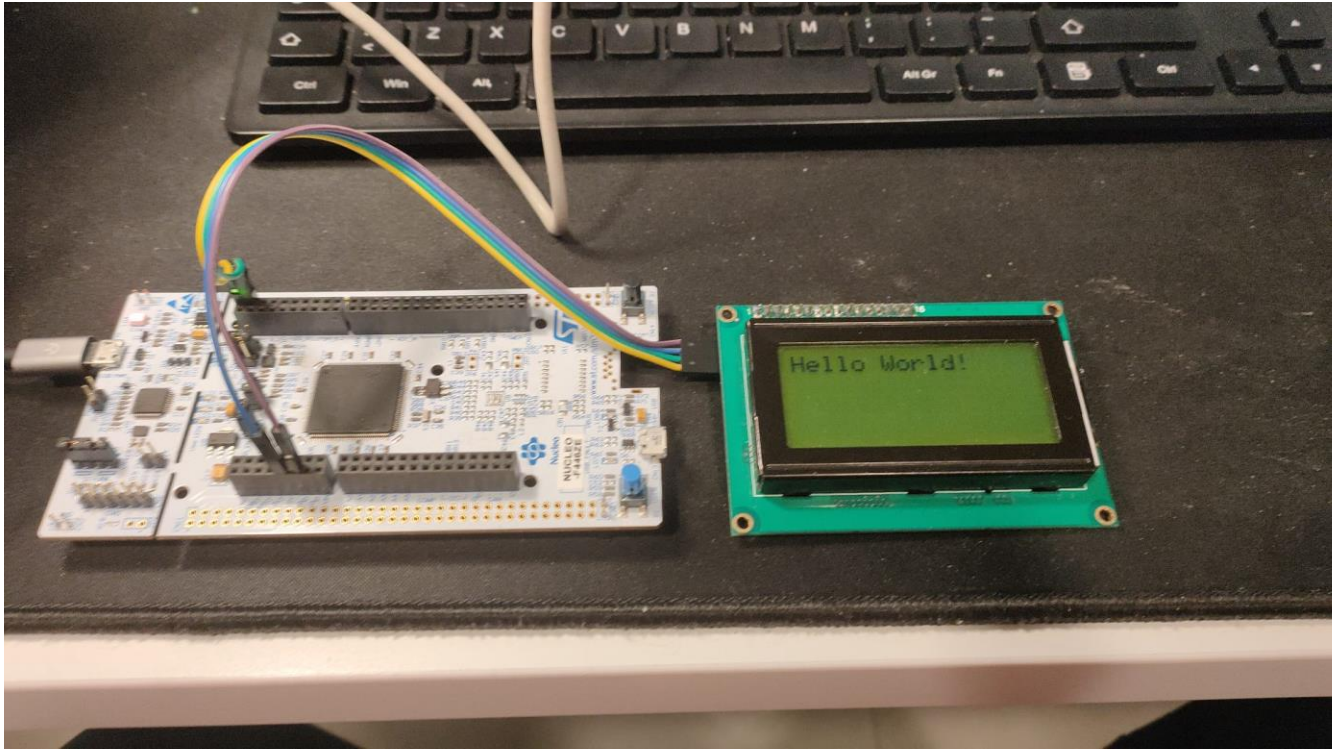
Let’s write some code:

- ***Initialize*** the lcd
- ***Set the cursor*** at first row and first line. Remember to check the size of your display, in our case we have a 16 rows by 4 lines display (16x4 or 1604)
- ***Print*** the desired text.



```c
  */
int main(void)
{
  /* USER CODE BEGIN 1 */
  /* USER CODE END 1 */

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

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

  /* USER CODE BEGIN Init */
  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();
  /* USER CODE BEGIN SysInit */
  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USB_OTG_FS_PCD_Init();
  MX_I2C1_Init();
  /* USER CODE BEGIN 2 */

  lcd_init();//init the lcd
  lcd_put_cur(0, 0);
  lcd_print_string("Hello World!");

  /* USER CODE END 2 */
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {

    /* USER CODE END WHILE */

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

The result should be something like that:



Useful links

I2C introduction: **https://www.youtube.com/watch?v=qeJN_80CiMU**

I2C display: **https://www.youtube.com/watch?v=rfRJGfK2t-A&t=473s**

Oled display via i2c: **https://controllerstech.com/oled-display-using-i2c-stm32/**

Other links:

https://controllerstech.com/lcd-20x4-using-i2c-with-stm32/

https://controllerstech.com/i2c-lcd-in-stm32/