University of Hull
School of Engineering and Computer Science

UNIVERSITY
OF HULL

Final Design Report

Group J

Embedded Systems
Module 601071

Aleksejs Nikitins
201914453

Kevin Mancini
202140581

Tomáš Krupička
202140574

Inês Carvalho
202140662

Rostislav Sorokin
201807421

Turan Hudasoy
201602421

Submission Date: Friday 17th December 2021

Contents

1. Introduction

The aim of this system development project was to develop a product that incorporated a mixture of I2C sensors (temperature/humidity, accelerometer/gyroscope, magnetometer and barometer) and SPI devices (microphone, speaker, SD card, GPIO expander, LCD display and thumb joystick). The chosen product concept was an interactive game where the user would utilize all sensors built into the STM32 Discovery Board B-L475E-IOT01A to achieve preset tasks/levels. The SPI hardware and software were designed and constructed to aid navigation and enhance the user's experience. For example, an LCD display showing menu options, a joystick enabling the user to move between those options and an SD card to store game data as well as allow the user to save their progress. The software was programmed in C and tested using the STM32CubeIDE program which allowed for easy debugging.

2. Corporate Deliverables

The following sections include all the corporate deliverables.

2.1. Overall Design of System Development Project

This section addresses the overview design of the system development project, more specifically the input/output memory map with the registers used, the two interfaces overview (I^2C and SPI) with reference to used SPI buses, the interrupt line assignments and the interrupt assignments.

Input/Output Memory Map

I2C Registers and definitions:
0x0F - Who Am I register for I2C components

Temperature:
0x20 – Control Register 1
0x27 – Status Register
0x2A and 0x2B – LSB and MSB Temperature Data

Barometer:
0x28, 0x29, 0x2A (PressureOut) 0x2B, 0x2C (TempOut)

Humidity:
0x20 – control register, 0x27 – status register,
0x28-29 – humidity value

Gyroscope:
0x20 and 0x21 temperature of Gyroscope
From 0x22 to 0x27 – X,Y,Z data of Gyroscope

Magnetometer:
0x3C address, 0x28-0x2D XYZ values.

Accelerometer:
0x28 – x LSB data of Accelerometer (MSB – 0x29)
0x2A – y LSB data of Accelerometer (MSB – 0x2B)
0x2C – z LSB data of Accelerometer (MSB – 0x2D)
0x10 – Control perimeters for the device

SPI Registers and definitions:

Sound In:
Addresses between 0x000h to 0x200h

Sound Out:
DAC I2C address 0x60, DAC channel 1.

Joystick:
X-axis, Y-axis

SD Card:
SD card is controlled by sending commands with arguments over SPI, not by accessing registers

GPIO Expander:
0x00 – direction setting of I/O
0x08 – capturing GPIO value when the interrupt occurs
0x09 – reading from the port, writing to the port

LCD Display:
From 0x00 to 0xEFh (239) Addresses range of X
From 0x00 to 0x13Fh (319) Addresses range of Y
* The RAM addresses depend on with Display Data Direction is selected

Fig 1: Input/Output Memory Map

I^2C and SPI Interfaces Overview

Team Member	I^2C Sensor	SPI Device	Chip Select (CS)
Aleksejs Nikitins	Magnetometer	Sound Out	7
Inês Carvalho	Temperature	Sound In	6
Kevin Mancini	Gyroscope	Display	2
Rostislav Sorokin	Accelerometer	IO Expander	3
Tomáš Krupička	Humidity	SD Card	5
Turan Hudasoy	Barometer	Joystick	4

Table 1: Interface Allocation and Chip Select (CS) Pins
used for the SPI Communication

There are some pins on the STM32 board which are shared between all the SPI devices used for this project.

- System Clock - Pin PA_5
- MISO - Pin PA_6
- MOSI - Pin PA_7
- Ground
- Voltage for some devices - 5V
- Voltage for some devices - 3V3.
- Reset Pin.

Interrupts

Regarding the interrupts, it is important to note in this section that, in general, these were not used in the individual code by each of the group members.

On the other hand, these were used in one of the familiarisation exercises, the timer interrupt exercise. As such, these will be discussed in detail in the section corresponding to this familiarisation exercise, section 2.4.3.

The interrupt exercises have allowed the use of a specific timer for accommodating the process of blinking LED lights on the board. One interrupt exercise deals with the pressed and unpressed blue service button interrupt which manages the frequency of blinking leds. The other exercise deals with the particular timing of parallel LED lights blinking adhered to a specific interval of idle measured in seconds. For the purposes of this exercise one timing handler TIM2 is used.

2.2. State of the Present Development

In this section is going to be illustrated the state of the present development, including all that have been achieved in chronological order. In addition to this, are shown what still has to be achieved and how this could be done.

It follows a detailed diagram including all the stages of the process and their state.

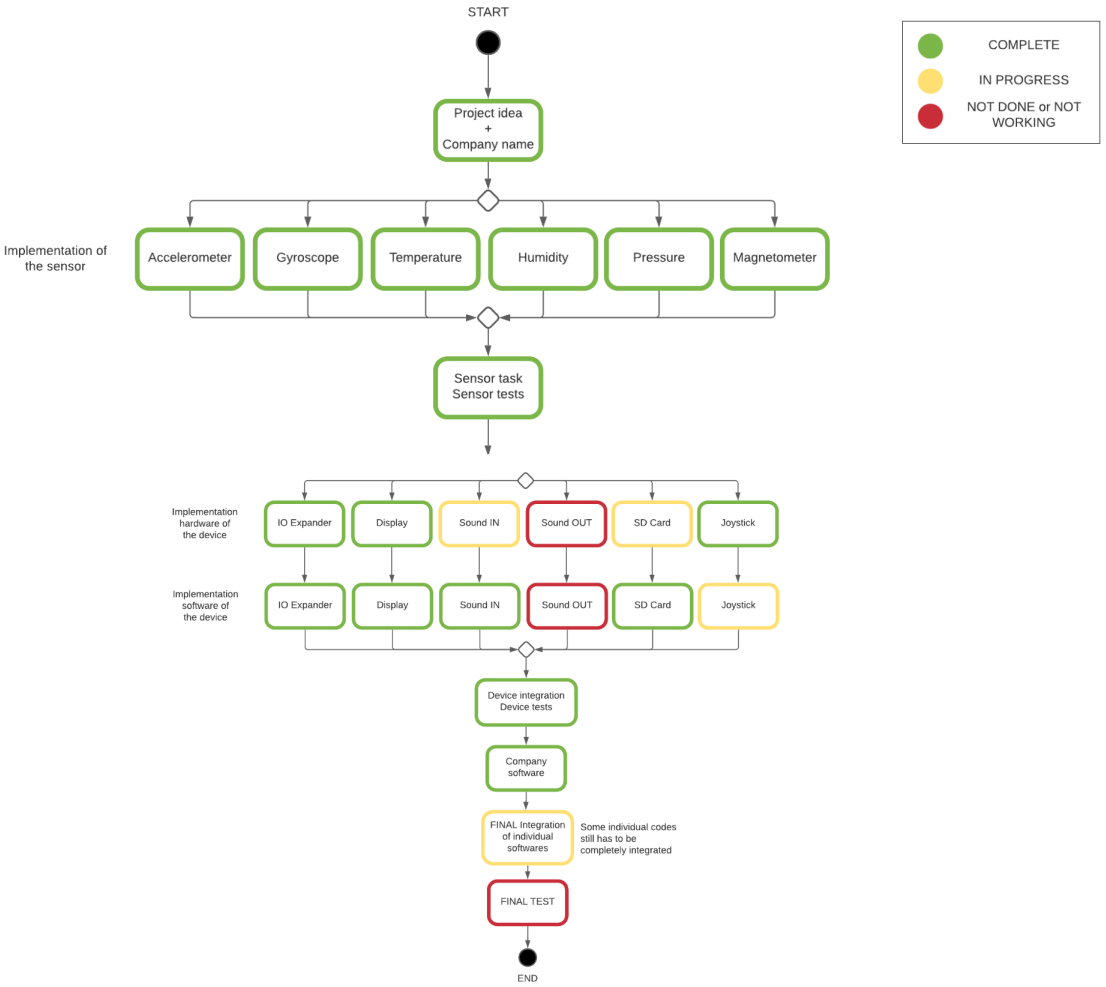


Fig 1: Diagram of the State of the Present Development

At first, we all together defined the project idea and the company name. Then we successfully implemented our sensors creating sensor tasks and the sensor tests which assure that the code is properly working. Then we soldered on the Arduino Proto Shield creating the circuits for our devices.

As could be expected, due to the great difficulty of the sound in and sound out hardware both presented some technical problems. It was necessary for the team members responsible for these two devices to allocate more time than initially thought for the hardware development.

The sound out hardware and software is not working and it will be discussed later in sections 3.2.1 and 3.2.2 .

The sound in hardware is not 100% functional, as there is a small problem in the last state of the circuit. This will be discussed later (section 4.2.1.), however it is important to mention that as this problem is not a severe one, it is not a hindrance to the development and success of this device's software.

Then we successfully wrote the company software and we integrated some individual codes (Display and Temperature) and merged them all using GitHub. Finally, we connected some of the shields and executed the integrated code (Display, Sound IN, Joystick, SD Card). We had no time to do a final test with all the shields connected.

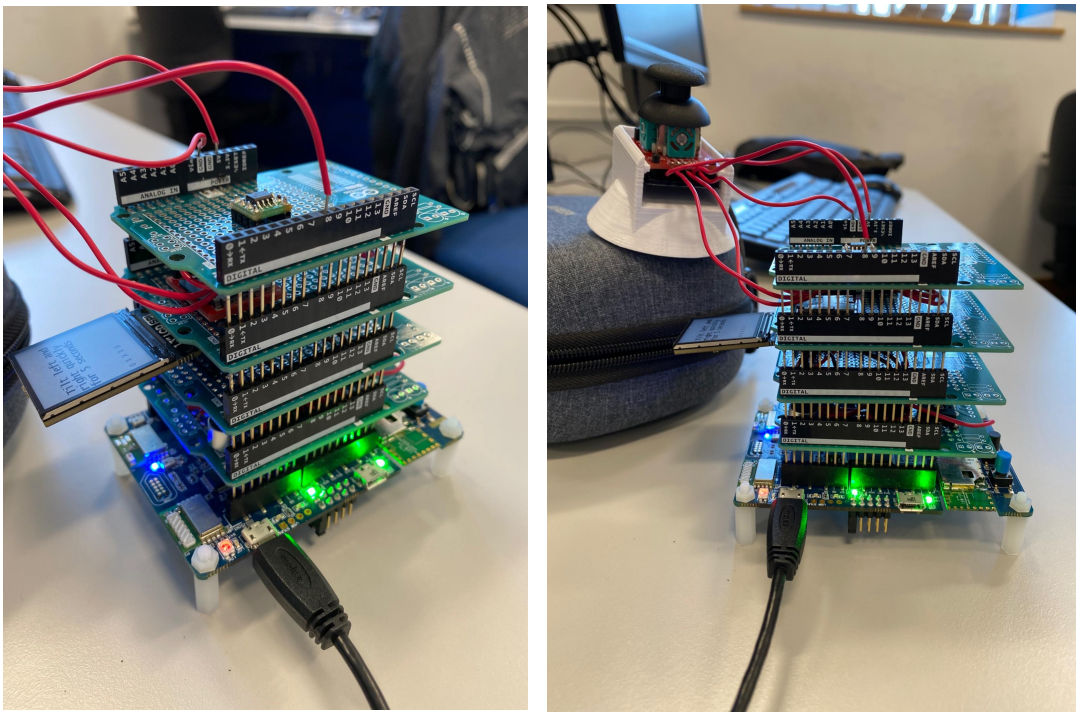


Fig. 2: Integration of Display, Temperature, Joystick and SD Card

2.3. Company Software

Firstly the device turns on. The first state is to run all tests on I2C sensors and SPI devices one after the other. If any tests fail the device will display an error message as can be seen below.

```
int runGame(){
     int result = runAllTests();


     if(result != 0){
          showStartupError(result);
          return 1;
     }
```

Once this is completed successfully the menu is displayed and the user will have to navigate the menu options using the thumb joystick. If a new game is selected using the button available on the joystick, the task begins. A random task will be chosen which makes use of each of the six I2C sensors and the user gets an allocated amount of time to complete the task. If they pass they gain a score of +1 and move on to the next task. If they fail then they lose a life and have the chance to retry the task/level. Once they run out of lives the game reaches the end state.

The full code can be seen in the 'game.c' file in the appendices within the int runGame() function which shows all states and outputs of each.

2.4. Familiarisation Exercises

This section contains the three familiarisation exercises developed throughout this module: password, NFC and timer interrupt exercise. These are explained in the subsections below, with evidence of its success being provided.

2.4.1. Password Exercise

This program deals with reading the entries from the keyboard, changing and storing values in the arrays. Figure xxxx1 shows the main menu of thi program created by the “printf” function from the “stdio.h” library.

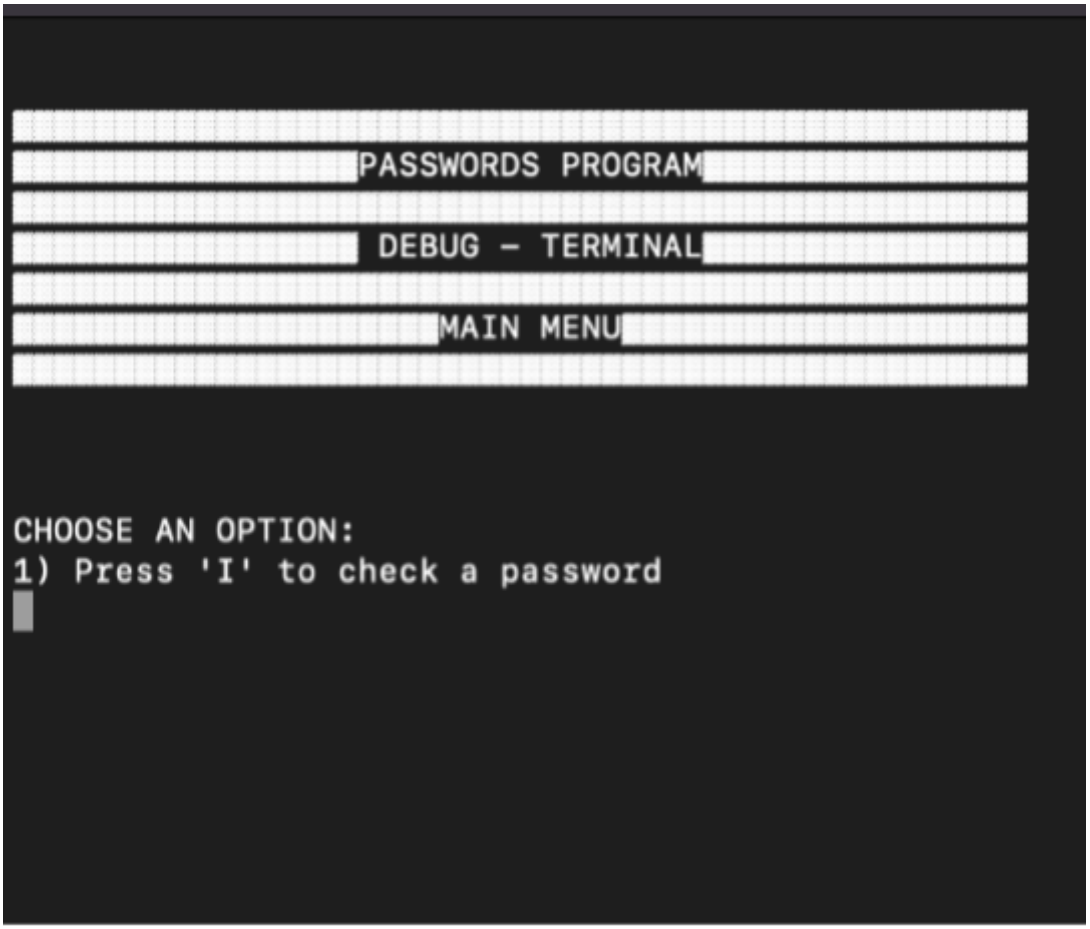


Figure xxxx1. Main menu of the program

“Main.c” source file contains all the variables and functions needed to run the program. Software code snippet xa shows the declared variables and functions:

```
1   void identify_menu(void);
2   void change_password(void);
3   char COMMAND_LOGIN = 'C';
4   char COMMAND_LOGIN_small = 'c';
5   char COMMAND_LOGOUT = 'L';
6   char COMMAND_LOGOUT_small = 'l';
7   char COMMAND_INSERT = 'I';
8   char COMMAND_INSERT_small = 'i';
9   char COMMAND_NEWPASS = 'N';
10  char COMMAND_NEWPASS_small = 'n';
```

```
11  char COMMAND_ESCAPE = '\033';
12  char command;
13  char password[10];
14  char correct_passwords[6][10] = {
15  "202140662\0",
16  "201602421\0",
17  "202140574\0",
18  "202140581\0",
19  "201914453\0",
20  "201807421\0"};
21  int status = 1;
22  int correct = 0;
23  int my_account = -1;
```

Software code xa. Declared variables and functions for the “Passwords” program

Character variables at the lines 3-12 are responsible for dealing with pressed keyboard buttons, either lowercase or uppercase or even “escape”, which would allow to go through the program’s procedures. The other character variables store initial information for student numbers, the array is used for such purposes. Finally, two more variables from line 21-22 manage the “while” loops and the “my_account” variable deal with the identification number of a particular student number.
After all the variables are initialized, the main “while” loop is activated and the main menu is printed out. Following Software Code snippet xb shows all the procedures.

```
1         while(1){
2   printf("\e[1;1H\e[2J");//clears screen
3   printf("\n\n\r                                                  \n");
4   printf("\r                  PASSWORDS PROGRAM                   \n");
5   printf("\r                                                      \n");
6   printf("\r                   DEBUG - TERMINAL                   \n");
7   printf("\r                                                      \n");
8   printf("\r                       MAIN MENU                      \n");
9   printf("\r                                                      \n");
10  printf("\n\n\n");
11  printf("\rCHOOSE AN OPTION:\n\r1) Press 'I' to check a password\n\r");
12  scanf("%c",&command);
13  if ((COMMAND_INSERT==command)||(COMMAND_INSERT_small==command))
14  {
15  identify_menu();
16  password[9] = '\0';
17  for (int i=0; i<6;i++){
18  if (strcmp(correct_passwords[i],password)==0){
19  correct = 1;
20  my_account = i;
21  }
22  }
23  if (!correct){
24  printf("\r*** ERROR - RETRY ***\n");
25  for (int z=0; z<5;z++){
26  HAL_GPIO_WritePin(GPIOB,GPIO_PIN_14,GPIO_PIN_SET);
27  HAL_Delay (500u);
28  HAL_GPIO_WritePin(GPIOB,GPIO_PIN_14,GPIO_PIN_RESET);
29  HAL_Delay (500u);
30  }
```

```
}
else {
printf("\r*** LOGIN - DONE ACC. N: %d ***\n",     );
for (int x=0; x<5;x++){
HAL_GPIO_WritePin(GPIOB,GPIO_PIN_14,GPIO_PIN_SET);
HAL_Delay (150u);
HAL_GPIO_WritePin(GPIOB,GPIO_PIN_14,GPIO_PIN_RESET);
HAL_Delay (150u);
}
while (status){ // Infinite-loop
change_password();
}
// Reset variables
status = 1;
correct = 0;
my_account = -1;
}
}
else{
main();
}
}
```

Software code xb. main “while” loop code

Once the letter “i” (lowercase or uppercase) is pressed, the function “identify_menu()” is called. With its help a particular password is identified and passed further. If the password is correct, then the rapid blinking of the board’s LED lights is instigated and the “change_password” menu is called in the “while” loop within the first “else” statement. If , however, the typed password is incorrect, then the program simply restarts as “main()” is called.

The software code snippet xc below presents two functions which deal with identifying and changing the student numbers .

```
void identify_menu(void)
{
printf("\e[1;1H\e[2J");//clears screen
printf("\r▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒\n");
printf("\r▒▒▒▒▒TYPE THE PASSWORD TO IDENTIFY ACCOUNT▒▒▒▒▒▒\n");
printf("\r▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒\n\r\n");
scanf("%c",&command);//character is registered for command option
if(COMMAND_ESCAPE==command){
main();//if ESCAPE button is pressed, the program goes to the main
}
else{
printf("%c",command);
password[0]=command;
for (int i=1; i<9;i++){
scanf("%c",&password[i]);
if (password[i]=='\r'){
password[i] = '\0';
i=9;
}
```

```
20  else
21  printf("%c",password[i]);
22  //printf("*"); Hidden password
23  }
24  }
25  }
26  void change_password(void){
27  printf("\e[1;1H\e[2J");//clears screen
28  printf("\r                                                                    \n");
29  printf("\r          PRESS 'C' TO CHANGE PASSWORD for account #%d      \n",
30  my_account);
31  printf("\r                                                                    \n");
32  printf("\r                    Press 'L' TO LOGOUT                           \n");
33  printf("\r                                                                    \n\r"
34  );
35  scanf("%c",&command);
36  if ((COMMAND_LOGIN==command)||
37  (COMMAND_LOGIN_small==command)){ // CHANGE PASSWORD
38  printf("\e[1;1H\e[2J");//clears screen
39  printf("\r                                                                    \n");
40  printf("\r        ENTER THE NEW PASSWORD for account #%d              \n",
41  my_account);
42  printf("\r                                                                    \n\r"
43  );
44  for (int i=0; i<9;i++){
45  scanf("%c",&correct_passwords[my_account][i]);
46  if (correct_passwords[my_account][i]=='\r'){
47  correct_passwords[my_account][i] = '\0';
48  i=9;
49  }
50  else
51  printf("%c",correct_passwords[my_account][i]);
52  //printf("*"); Hidden password
53  }
54  correct_passwords[my_account][9] = '\0';
55  }
56  if ((COMMAND_LOGOUT==command)||
57  (COMMAND_LOGOUT_small==command)){ // LOGOUT
58  printf("\e[1;1H\e[2J");//clears screen
59  printf("\r***************** LOGOUT - DONE*****************\n");
60  status = 0;
61  }
62  }
63
64
65
```

Software Code xc. Functions for identifying and changing student numbers

The “identify_menu()” function just reads 9 typed keyboard entries and provides back the value typed. And when the “change_password()” function is called, then if the changing procedure is instigated, it scans the new keyboard entry for the student number and stores it into the specific place in the array which was identified by the identification function.

After all is done, the terminal screen is cleared, the “while” loop counter is put down to 0, and everything is back to the main menu.

2.4.2. NFC Exercise

To do this exercise we used built in function such as:

- **NFC_IO_Init()**: To initialize the interface
- **NFC_IO_IsDeviceReady()**: To check whether the sensor is ready to be used
- **NFC_IO_ReadMultiple()**: To read bytes from the sensor
- **NFC_IO_WriteMultiple()**: To write bytes in the sensor memory
- **NFC_IO_DeInit()**: To deinitialize the interface

After the initialization of the NFC interface the code permits the user to read and write into the NFC sensor through the terminal. In addition, is possible to do the same with a smartphone, in one of the following screenshots a IOS device has been used.

It follows the code of the familiarization exercise and some screenshots which illustrate how the program works.

```
/**
  * @brief  Main program
  * @param  None
  * @retval None
  */
int main(void)
{

  /* STM32L4xx HAL library initialization:
       - Configure the Flash prefetch, Flash preread and Buffer caches
       - Systick timer is configured by default as source of time base, but user
             can eventually implement his proper time base source (a general
purpose
             timer for example or other time source), keeping in mind that Time
base
             duration should be kept 1ms since PPP_TIMEOUT_VALUEs are defined and
             handled in milliseconds basis.
       - Low Level Initialization
     */
  HAL_Init();

  /* Configure the System clock to have a frequency of 80 MHz */
  SystemClock_Config();


  /* Configure the User LED */
  BSP_LED_Init(LED2);

  /* Configure the User Button in GPIO Mode */
  BSP_PB_Init(BUTTON_USER, BUTTON_MODE_GPIO);

  /* Initialize all configured peripherals */
  hDiscoUart.Instance = DISCOVERY_COM1;
  hDiscoUart.Init.BaudRate = 115200;
  hDiscoUart.Init.WordLength = UART_WORDLENGTH_8B;
```

```
  hDiscoUart.Init.StopBits = UART_STOPBITS_1;
  hDiscoUart.Init.Parity = UART_PARITY_NONE;
  hDiscoUart.Init.Mode = UART_MODE_TX_RX;
  hDiscoUart.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  hDiscoUart.Init.OverSampling = UART_OVERSAMPLING_16;
  hDiscoUart.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  hDiscoUart.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;

  BSP_COM_Init(COM1, &hDiscoUart);

  NFC_IO_Init(0);
  NFC_GPIO_CLK_ENABLE();

  uint8_t Read_Addr = 0xAC;
  uint8_t Write_Addr = 0xAD;
  uint8_t *write_pBuffer = malloc(8*sizeof(uint8_t));
  uint8_t *read_pBuffer = malloc(8*sizeof(uint8_t));
  uint16_t Length = 8*sizeof(char);
  char mode = 'R'; // The default mode is the reading mode

  printf("\r************************************************************\n");
  printf("\r***************** NFC Reader ***************\n");
  printf("\r************************************************************\n\n");

  while (1)
  {
        printf("\r*** Press: W to write or R to read ***\n");
        scanf("%c", &mode);
        if (mode=='R' && NFC_IO_IsDeviceReady(Read_Addr,1)==0){
              // READING MODE
              printf("\r*** READING MODE  ***\n");
              NFC_IO_ReadMultiple(Read_Addr,read_pBuffer,Length);
              printBuffer(read_pBuffer);
        } else if (mode=='W' && NFC_IO_IsDeviceReady(Read_Addr,1)==0) {
              // WRITING MODE
              printf("\r*** WRITING MODE  ***\n");
              scanBuffer(write_pBuffer);
              NFC_IO_WriteMultiple (Write_Addr, write_pBuffer,Length);
              printBuffer(write_pBuffer);
        }
  }
  NFC_IO_DeInit();
}

void printBuffer(uint8_t *pBuffer){
        for(int loop = 0; loop < 8; loop++)
              printf("%d ", pBuffer[loop]);
      printf("\n");
}

void scanBuffer(uint8_t *pBuffer){
        for(int loop = 0; loop < 8; loop++)
        {
              char tmp;
              scanf("%c",&tmp);
              printf("%c ", tmp);
              pBuffer[loop] = (uint8_t) tmp;
        }
```

```
        printf("\n");
}
```

Software Code 1: NFC code

```
************************************************************
***************** NFC Reader *****************
************************************************************

*** Press: W to write or R to read ***
*** READING MODE  ***
                    255 255 255 255 255 255 255 255
*** Press: W to write or R to read ***
*** WRITING MODE  ***
                    a s d a s d a s
                                    97 115 100 97 115 100 97 115
*** Press: W to write or R to read ***
```

Fig. 3: NFC Exercise - Terminal Screenshot

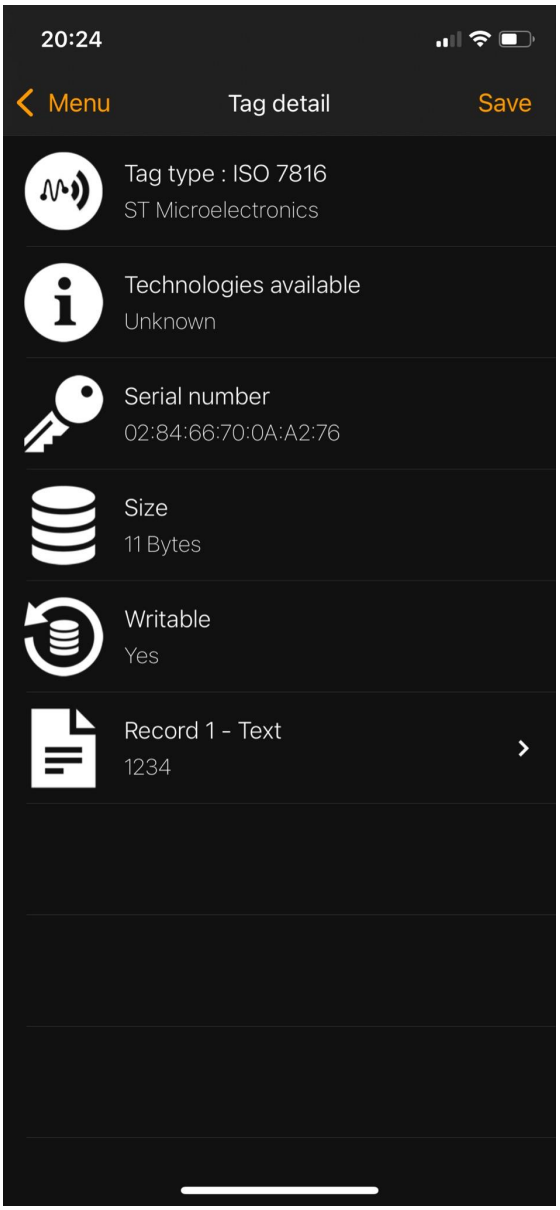


Fig. 4: NFC Exercise: Iphone Screenshot

2.4.3. Timer Interrupt Exercise

The first Timer Interrupt exercise deals with the pressing of the blue button which changes the frequency of the blinking light on the board.
The first task shown in the Software Code Snippet x_001 deals with the function button pressed on the B-L475E-IOT01A itself. All the processes happen in the main and interrupt source codes. Firstly, a special external and volatile variable responsible for being changed at a particular condition is defined in the main file -
`volatile delaychanged=1000;` This variable sets back the delay time of 1 second once the delay is less than 100ms (Software Code snippet xxx1). The delay applies to the flashing of LED2 green light on the board.

```
while (1)
{
        if (delaychanged<100){
              delaychanged=1000;
        }
HAL_Delay(delaychanged);
HAL_GPIO_WritePin(GPIOB,GPIO_PIN_14,GPIO_PIN_SET);
HAL_Delay(delaychanged);
HAL_GPIO_WritePin(GPIOB,GPIO_PIN_14,GPIO_PIN_RESET);
}
}

void Hull_Uni_Init(char Pin_Mode){
      GPIO_InitTypeDef Hull_Uni_Structure;
      __HAL_RCC_GPIOC_CLK_ENABLE();
      Hull_Uni_Structure.Pin = GPIO_PIN_14;
      Hull_Uni_Structure.Mode = GPIO_MODE_OUTPUT_PP;
      Hull_Uni_Structure.Pull = GPIO_NOPULL;
      Hull_Uni_Structure.Speed = GPIO_SPEED_FREQ_LOW;
      HAL_GPIO_Init(GPIOB, &Hull_Uni_Structure);
      if (Pin_Mode==1)
      {
            Hull_Uni_Structure.Pin = GPIO_PIN_13;
            Hull_Uni_Structure.Mode = GPIO_MODE_INPUT;
            Hull_Uni_Structure.Pull = GPIO_PULLUP;
            Hull_Uni_Structure.Speed = GPIO_SPEED_FREQ_HIGH;
            HAL_GPIO_Init(GPIOC, &Hull_Uni_Structure);
      }
if (Pin_Mode==2){
      Hull_Uni_Structure.Pin = GPIO_PIN_13;
      Hull_Uni_Structure.Pull = GPIO_PULLUP;
      Hull_Uni_Structure.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
      Hull_Uni_Structure.Mode = GPIO_MODE_IT_FALLING;
      HAL_GPIO_Init(GPIOC, &Hull_Uni_Structure);
/*Enable and set Button EXTI Interrupt to the lowest priority*/
      HAL_NVIC_SetPriority((IRQn_Type)(EXTI15_10_IRQn),0x0F,0x00);
      HAL_NVIC_EnableIRQ((IRQn_Type)(EXTI15_10_IRQn));
      }
}
```

Software code xxx1

The procedures simply set up the blinking LED light which starts blinking slowly for 1 second.,

The function “Hull_Uni_Unit()” sets up the reaction of the program when the blue button is pushed down and pulled up automatically.

Whenever the blue button on the board is pressed anytime, an interrupt occurs and the signal is sent to the interrupt source file called “stm32l4xx_hal_gpio.c” as the interrupt involves hardware manipulation - pressing the non-reset button on the board. Software code xxx2 shows the code associated with manipulating the blinking frequency of the integrated LED2 light on the board.

```
1   void HAL_GPIO_EXTI_IRQHandler(uint16_t GPIO_Pin)
2   {
3   /* EXTI line interrupt detected */
4   if(__HAL_GPIO_EXTI_GET_IT(GPIO_Pin) != 0x00u)
5   {
6           delaychanged=delaychanged/1.15;
7   __HAL_GPIO_EXTI_CLEAR_IT(GPIO_Pin);
8   HAL_GPIO_EXTI_Callback(GPIO_Pin);
9   }
10  }
```

Software code xxx2. Interrupt function which changes the frequency of LED blinking

The original value of the untouched and non-modified pin responsible for the button is 0x00u. If the register value of this pin is changed, then automatically the function speeds up the blinking rate of the LED2 by dividing the original 1000ms interval by 1.15. Once the interval becomes less than 100ms, the value of "delaychanged" is reinstated for 1000ms.

The second interrupt task requires more logical operators to be used. The "main.c" file contains simply a global loop for blinking blue and orange LED lights on the board and the delay is 100ms (Software code snippet xxx3). It also triggers the interrupt handler with the execution of the following command: HAL_TIM_BASE_START_IT(&htim2);

Please click here for the video of the first interrupt exercise process.

```
1   HAL_TIM_BASE_START_IT(&htim2);
2   …
3   …
4
5   while (1)
6   {
7               HAL_GPIO_WritePin(GPIOC, GPIO_PIN_9, GPIO_PIN_SET);
8               HAL_Delay(100);
9               HAL_GPIO_WritePin(GPIOC, GPIO_PIN_9, GPIO_PIN_RESET);
10              HAL_Delay(100);
    }
```

Software code xxx3. Blue and Orange LEDs interchanging blinking

However, the interrupt source file "stm32l4xx_it.c" organises a completely parallel process of the LED2 light blinking in a particular order as seen in the software code snippet xxx4.

Please click here for the video of the second interrupt exercise process.

```
1   extern TIM_HandleTypeDef htim2;
2   int on_tim = 0;
3   int off_tim = 0;
4   int start_cond=0;
5   int start_cond_off=0;
6   …
7   …
```

```
8   …
9   void TIM2_IRQHandler(void)
10  {
11        if (start_cond==0){
12              HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_SET);
13              start_cond=1;
14        }
15        if (on_tim<10){
16              on_tim=on_tim+1;
17        }
18        else if (on_tim==10){
19              if (start_cond_off==0){
20              HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_RESET);
21              start_cond_off=1;
22              }
23                    if (off_tim<300){
24              off_tim=off_tim+1;
25                    }
26                    else if (off_tim==300){
27                          HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14,
28  GPIO_PIN_SET);
29                          start_cond=0;
30                          start_cond_off=0;
31                          on_tim=0;
32                          off_tim=0;
33                    }
34        }
35         HAL_TIM_IRQHandler(&htim2);
36  }
```

Software code xxx4. Code for parallel processes of blinking LED lights in the particular order

The external interrupt handler and initial condition values for the timer being off and on are declared (lines 1-5) and in the interrupt function which is triggered a set of “if… else” statements is used with the helps of simple counting of values responsible for activating and deactivating LED2 light. As the prescaler has been particularly set up for the timer so that the time of the loop execution equals 100ms, once the “on_tim” counter reaches value of 10 (line 15), it means that one second has passed, and the activated LED2 light is deactivated for 30 seconds until the “off_tim” counter value reaches 300. Once this happens, the initial conditions are restarted and the LED2 light is activated for a second once again and so on.

2.5. Integration Plan and Implementation

Initially all group members were allocated an individual chip select pin to avoid two members using the same pin. Any extra GPIO pins that were needed were assigned during the schematic design phase of the project. This was done to avoid compatibility issues that may have arisen once soldering was completed. This ensured that all six proto shield boards would work together and run properly once connected to the STM32 Discovery Board B-L475E-IOT01A.

Secondly, a GitHub repository was created in order to merge all individual code files which helped to isolate different threads of c code using branches. This kept the group organised and kept everyone working within their own allocated spaces. The files were then able to be downloaded from GitHub using GitBash which allowed members to always have the current files available to them at university or on their home computers. This also gave the option to restore old working versions of the code incase of errors.

The code inside the main function of the main.c file has been decided to be as minimal as possible to increase the flexibility of the project and at the same time assure solidity, for example all the return values of the principal individual functions and of the testing functions were defined.

Lastly, the main framework of the game has been designed in a way that allows easy integration of individual tasks using different sensors, not needing to modify the actual game framework (backend, main frontend - menus), and still allow space for individualization of tasks as wide as possible.

To conclude, following these measures and expedients the code could be successfully integrated without any large problems relating to compatibility.

3. Individual Deliverables - Aleksejs Nikitins

3.1. Magnetometer Sensor

Magnetometer Sensor LIS3MDL is integrated into the Discovery Kit B-L475E-IOT01A1. It can be accessed and controlled via an I2C interface and does not require any GPIO pins assigned or connected externally.

All information input and output during testing is completed via terminal using UART. This is supposed to be changed to another function to use display via the SPI interface.

Following the group concept, a simple game was created to demonstrate use of a magnetometer in the project. Separate LIS3MDL.C code file containing functions needed and LIS3MDL.H header file containing all defines needed were created.

The aim of the game is to find the position in which a magnetometer reading XYZ values will be near enough to randomly generated values. This can be achieved by placing a magnet or strong conductive metal object near the sensor. During testing it was discovered that the sensor is very sensitive, so values required cannot be achieved precisely. Some adjustments were made, such as read values were divided by 10 to decrease sensitivity, required values could be near enough in +-20 (120 and 80 are acceptable for 100). Also absolute value counts (-100 or 100). These adjustments made the game easier. When all three coordinate values are close enough the game ends. There is no time limit,so it can be played and reset until won or fed up.
To start the game, the Blue Button must be pressed to generate a random position, which is needed to find.

3.1.1. Magnetometer Sensor - Functions

Created LIS3MDL.C file contains multiple functions.

- Error function which switches off LED2 if error acquired.

```
void Error(void)
{
	LD2_OFF;
}
```

- HAL I2C read/write data functions.

```
static uint8_t I2Cx_ReadData(uint16_t Addr, uint8_t Reg)
{
	HAL_StatusTypeDef status = HAL_OK;
	uint8_t value = 0;
	status = HAL_I2C_Mem_Read(&hi2c2, Addr, Reg,
I2C_MEMADD_SIZE_8BIT, &value, 1, 0x10000);
	if(status != HAL_OK) Error();
	return value;
}
//-----------------------------------------
static void I2Cx_WriteData(uint16_t Addr, uint8_t Reg, uint8_t Value)
{
	HAL_StatusTypeDef status = HAL_OK;
	status = HAL_I2C_Mem_Write(&hi2c2, Addr, (uint16_t)Reg,
I2C_MEMADD_SIZE_8BIT, &Value, 1, 0x10000);
	if(status != HAL_OK) Error();
}
//-----------------------------------------
uint8_t Mag_IO_Read(uint16_t DeviceAddr, uint8_t RegisterAddr)
{
	return I2Cx_ReadData(DeviceAddr, RegisterAddr);
}
//-----------------------------------------
void Mag_IO_Write(uint16_t DeviceAddr, uint8_t RegisterAddr, uint8_t
Value)
{
	I2Cx_WriteData(DeviceAddr, RegisterAddr, Value);
}
```

- Magnetometer ID reading function.

```
uint8_t Mag_ReadID(void)
{
	uint8_t ctrl = 0x00;
	ctrl = Mag_IO_Read(MAG_I2C_ADDRESS,LIS3MDL_MAG_WHO_AM_I_REG);
	return ctrl;
}
```

- Magnetometer who_am_i check function.

```
void Mag_Ini(void)
{
uint16_t ctrl = 0x0000;
HAL_Delay(2000u);
if(Mag_ReadID()==LIS3MDL_MAG_WHO_AM_I) LD2_ON;
```

```
else Error();
LD2_OFF;
MagInit(ctrl);
LD2_ON;
}
```

- Magnetometer initialization function.

```
void MagInit(uint16_t InitStruct)
{
	uint8_t value = 0;
//Disable magnetometer (MD = 0x02)
	value = Mag_IO_Read(MAG_I2C_ADDRESS,LIS3MDL_MAG_CTRL_REG3);
	value&=~LIS3MDL_MAG_MD_MASK;
	value|=LIS3MDL_MAG_MD_POWER_DOWN;
	Mag_IO_Write(MAG_I2C_ADDRESS,LIS3MDL_MAG_CTRL_REG3,value);
//Enable BDU
	value = Mag_IO_Read(MAG_I2C_ADDRESS,LIS3MDL_MAG_CTRL_REG5);
	value&=~LIS3MDL_MAG_BDU_MASK;
	value|=LIS3MDL_MAG_BDU_ENABLE;
	Mag_IO_Write(MAG_I2C_ADDRESS,LIS3MDL_MAG_CTRL_REG5,value);
//Enable Data Rate 80 Гц
	value = Mag_IO_Read(MAG_I2C_ADDRESS,LIS3MDL_MAG_CTRL_REG1);
	value&=~LIS3MDL_MAG_DO_MASK;
	value|=LIS3MDL_MAG_DO_80Hz;
	Mag_IO_Write(MAG_I2C_ADDRESS,LIS3MDL_MAG_CTRL_REG1,value);
//Full scale selection 4 gauss
	value = Mag_IO_Read(MAG_I2C_ADDRESS,LIS3MDL_MAG_CTRL_REG2);
	value&=~LIS3MDL_MAG_FS_MASK;
	value|=LIS3MDL_MAG_FS_12Ga;
	Mag_IO_Write(MAG_I2C_ADDRESS,LIS3MDL_MAG_CTRL_REG2,value);
//High Performance Mode
	value = Mag_IO_Read(MAG_I2C_ADDRESS,LIS3MDL_MAG_CTRL_REG1);
	value&=~LIS3MDL_MAG_OM_MASK;
	value|=LIS3MDL_MAG_OM_HIGH;
	Mag_IO_Write(MAG_I2C_ADDRESS,LIS3MDL_MAG_CTRL_REG1,value);
//Disable temperature sensor
	value = Mag_IO_Read(MAG_I2C_ADDRESS,LIS3MDL_MAG_CTRL_REG1);
	value&=~LIS3MDL_MAG_TEMP_EN_MASK;
	value|=LIS3MDL_MAG_TEMP_EN_DISABLE;
	Mag_IO_Write(MAG_I2C_ADDRESS,LIS3MDL_MAG_CTRL_REG1,value);
//Enable magnetometer (MD = 0x00)
	value = Mag_IO_Read(MAG_I2C_ADDRESS,LIS3MDL_MAG_CTRL_REG3);
	value&=~LIS3MDL_MAG_MD_MASK;
	value|=LIS3MDL_MAG_MD_CONTINUOUS;
	Mag_IO_Write(MAG_I2C_ADDRESS,LIS3MDL_MAG_CTRL_REG3,value);
}
```

- Magnetometer values reading function

```
void Mag_GetXYZ(int16_t* pData)
{
	uint8_t buffer[6];
	uint8_t i=0;
	buffer[0]=Mag_IO_Read(MAG_I2C_ADDRESS,LIS3MDL_MAG_OUTX_L);
	buffer[1]=Mag_IO_Read(MAG_I2C_ADDRESS,LIS3MDL_MAG_OUTX_H);
	buffer[2]=Mag_IO_Read(MAG_I2C_ADDRESS,LIS3MDL_MAG_OUTY_L);
```

```
            buffer[3]=Mag_IO_Read(MAG_I2C_ADDRESS,LIS3MDL_MAG_OUTY_H);
            buffer[4]=Mag_IO_Read(MAG_I2C_ADDRESS,LIS3MDL_MAG_OUTZ_L);
            buffer[5]=Mag_IO_Read(MAG_I2C_ADDRESS,LIS3MDL_MAG_OUTZ_H);
            for(i=0;i<3;i++)
            {
                  pData[i] =
      ((int16_t)((uint16_t)buffer[2*i+1]<<8)+buffer[2*i]);
            }
      }
```

- Send to terminal function

```
    void SendToTerminal(char* msg)
      {
            HAL_UART_Transmit(&huart1, (uint8_t*)msg,strlen(msg),0x1000);
      }
```

- Value comparison function

```
int validateValue(int16_t value, int16_t target, int16_t threshold)
      {
            return (abs(value - target) < threshold);
      }
```

- Main game function which uses functions above.

```
void Mag_Game(void)
{
      int16_t buffer[3] = {0};
      static int16_t val[3], nullif[3], game_rand[3], tmp16;
      tmp16=HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13);
      int win[3] ={0};
      void SendToTerminal(char* msg)
      {
            HAL_UART_Transmit(&huart1, (uint8_t*)msg,strlen(msg),0x1000);
      }
      int validateValue(int16_t value, int16_t target, int16_t threshold)
      {
            return (abs(value - target) < threshold);
      }
      while (1)
      {
if ((win[0]!=1) || (win[1]!=1) || (win[2]!=1))
{
      Mag_GetXYZ(buffer);
      val[0]=buffer[0]-nullif[0];
      val[1]=buffer[1]-nullif[1];
      val[2]=buffer[2]-nullif[2];
      tmp16=HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13);
            if(tmp16==0)
            {
                  nullif[0]=buffer[0];
                  nullif[1]=buffer[1];
                  nullif[2]=buffer[2];
                  game_rand[0]= (rand() % 100);
                  game_rand[1]= (rand() % 100);
                  game_rand[2]= (rand() % 100);
//          game_rand[0]= 100;
```

```
//            game_rand[1]= 100;
//            game_rand[2]= 100;
              }
              val[0] = abs(val[0])/10;
              val[1] = abs(val[1])/10;
              val[2] = abs(val[2])/10;
              sprintf(str1,"Find the position: X:%06d Y:%06d Z:%06d
",game_rand[0], game_rand[1], game_rand[2]);
              SendToTerminal(str1);
              SendToTerminal("     Current position: ");
              if (validateValue(val[0], game_rand[0], 20))
              {
                    sprintf(str1, "X:%s", "  OK   ");
                    win[0]=1;
              }
              else
              {
                    sprintf(str1, "X:%06d ", val[0]);
                    win[0]=0;
              }
              SendToTerminal(str1);

              if (validateValue(val[1], game_rand[1], 20))
              {
                    sprintf(str1, "Y:%s", "  OK   ");
                    win[1]=1;
              }
              else
              {
                    sprintf(str1, "Y:%06d ", val[1]);
                    win[1]=0;
              }
              SendToTerminal(str1);

              if (validateValue(val[2], game_rand[2], 20))
              {
                    sprintf(str1, "Z:%s", "  OK   ");
                    win[2]=1;
              }
              else
              {
                    sprintf(str1, "Z:%06d ", val[2]);
                    win[2]=0;
              }
              SendToTerminal(str1);
              SendToTerminal("\r");
}
              else
      {
      SendToTerminal("You WIN\r\n");
                    }
HAL_Delay(50u);
                  }
```

}

This main function checks win array elements and outputs messages to the terminal in constant while(1) loop. Refresh time of coordinates reading, message sending can be changed with HAL_Delay(50u);.

3.1.2. Magnetometer Sensor - Global Variables

Header file LIS3MDL.H was created and included in the project. It contains defines, which are more convenient to use. Most of them are self-explaining and separated into groups.

```
#ifndef LIS3MDL_H_
#define LIS3MDL_H_

#include "stm32l4xx_hal.h"
#include <string.h>
//-----------------------------------------------
#define ABS(x)         (x < 0) ? (-x) : x
//-----------------------------------------------
#define LD2_Pin          GPIO_PIN_5
#define LD2_GPIO_Port    GPIOA
#define LD2_ON           HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET)
//GREEN
#define LD2_OFF          HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET)
//-----------------------------------------------
#define MAG_I2C_ADDRESS        0x3C
//-----------------------------------------------
#define     LIS3MDL_MAG_WHO_AM_I_REG 0X0F
#define          LIS3MDL_MAG_CTRL_REG1          0X20
#define     LIS3MDL_MAG_CTRL_REG2          0X21
#define     LIS3MDL_MAG_CTRL_REG3               0X22
#define          LIS3MDL_MAG_CTRL_REG5    0X24
//-----------------------------------------------
#define     LIS3MDL_MAG_WHO_AM_I           0x3D
//-----------------------------------------------
#define     LIS3MDL_MAG_MD_CONTINUOUS                  0x00
#define     LIS3MDL_MAG_MD_SINGLE                      0x01
#define     LIS3MDL_MAG_MD_POWER_DOWN            0x02
#define     LIS3MDL_MAG_MD_POWER_DOWN_AUTO 0x0
#define     LIS3MDL_MAG_MD_MASK                        0x03
//-----------------------------------------------
#define     LIS3MDL_MAG_BDU_DISABLE              0x00
#define     LIS3MDL_MAG_BDU_ENABLE               0x40
#define     LIS3MDL_MAG_BDU_MASK                       0x40
//-----------------------------------------------
#define     LIS3MDL_MAG_DO_0_625Hz               0x00
#define     LIS3MDL_MAG_DO_1_25Hz                      0x04
#define     LIS3MDL_MAG_DO_2_5Hz                       0x08
#define     LIS3MDL_MAG_DO_5Hz                   0x0C
#define     LIS3MDL_MAG_DO_10Hz                        0x10
#define     LIS3MDL_MAG_DO_20Hz                        0x14
#define     LIS3MDL_MAG_DO_40Hz                        0x18
#define     LIS3MDL_MAG_DO_80Hz                        0x1C
#define     LIS3MDL_MAG_DO_MASK                        0x1C
//-----------------------------------------------
#define     LIS3MDL_MAG_FS_4Ga                   0x00
#define     LIS3MDL_MAG_FS_8Ga                   0x20
```

```
#define     LIS3MDL_MAG_FS_12Ga                         0x40
#define     LIS3MDL_MAG_FS_16Ga                         0x60
#define     LIS3MDL_MAG_FS_MASK                         0x60
//-----------------------------------------------
#define     LIS3MDL_MAG_OM_LOW_POWER            0x00
#define     LIS3MDL_MAG_OM_MEDIUM               0x20
#define     LIS3MDL_MAG_OM_HIGH                         0x40
#define     LIS3MDL_MAG_OM_ULTRA_HIGH                   0x60
#define     LIS3MDL_MAG_OM_MASK                         0x60
//-----------------------------------------------
#define     LIS3MDL_MAG_TEMP_EN_DISABLE         0x00
#define     LIS3MDL_MAG_TEMP_EN_ENABLE                  0x80
#define     LIS3MDL_MAG_TEMP_EN_MASK            0x80
//-----------------------------------------------
#define LIS3MDL_MAG_OUTX_L                              0X28
#define LIS3MDL_MAG_OUTX_H                              0X29
#define LIS3MDL_MAG_OUTY_L                              0X2A
#define LIS3MDL_MAG_OUTY_H                              0X2B
#define LIS3MDL_MAG_OUTZ_L                              0X2C
#define LIS3MDL_MAG_OUTZ_H                              0X2D
//-----------------------------------------------
void Mag_Ini(void);
void Mag_Read(void);
//-----------------------------------------------
#endif /* LIS3MDL_H_ */
```

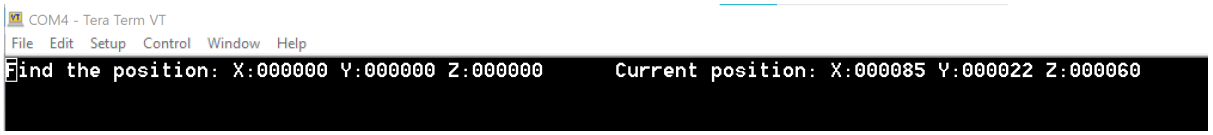
3.1.3. Magnetometer Sensor - Testing

When the main game function starts:

1. **MagInit()** function initializes the sensor with chosen settings
2. **Mag_Ini()** function checks who_am_i value and light up LED2 if correct
3. **Mag_Game()** function starts a constant loop. In this loop it checks win condition, reads blue button state, reads current XYZ values, and sends messages to the terminal. If the blue button is pressed, the program nullifies readings and defines required coordinates.

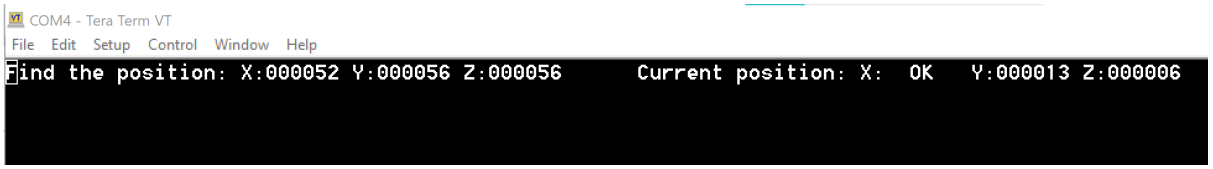
If any of the coordinates fit the conditions, its value is displayed as "OK". Game stops when all three coordinates match the condition and the function sends a "YOU WIN" message.

3.1.4. Magnetometer Sensor - Practical Example

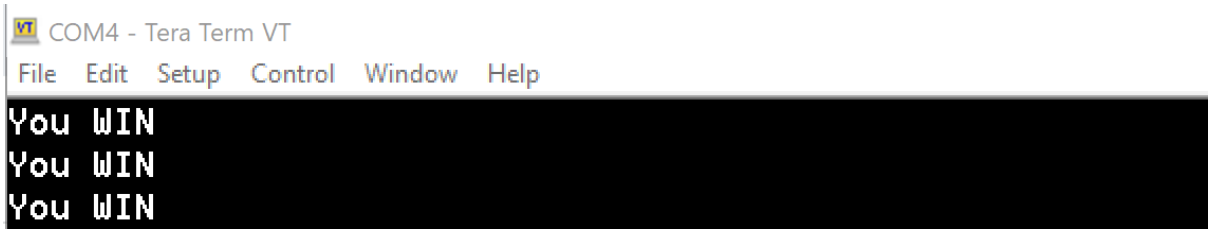


When the game starts the terminal outputs zero required coordinates and current XYZ values.

When the Blue Button is pressed, required coordinates are randomly generated. Also current position nullified.



When a magnet is placed near the sensor. X value already matches.



When all three coordinates match.

3.2. Sound Out Device

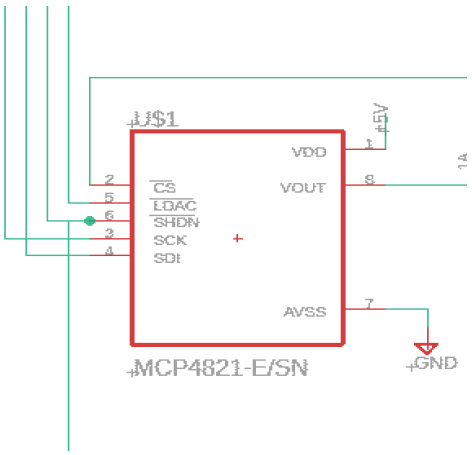
Following group concept sound out device is supposed to be built on the provided Arduino ProtoShield and to be controlled via SPI interface. The Chip Select PIN assigned was PIN7(PB1 signal), which corresponds to D6 on the Arduino shield. When the D6 pin is active, the Sound Out Device receives signals through MOSI PIN D11(PA7 signal), using CLK PIN D13(PA5 signal).

Sound out function converts the audio file to an array of values and sends them to the MOSI pin with the required frequency. Frequency can be controlled with delay in the send cycle. Lower and higher tone BEEP sound contains sine wave signals with different frequencies.

3.2.1. Sound Out Hardware

Hardware part of the sound out device consists of MCP4811 Digital to Analogue Converter, filter to pass only AC signal (capacitor), variable resistor to maintain low voltage on the sound amplifier input (not to oversaturate it), PAM8301AAF sound amplifier and finally speaker.

- **MCP4811 DAC connection.**



The **CS** pin is connected to the shield PIN7 (D6).Used to activate the device. Inverse signal, so must be low when active. PD6 must be initialized as active in software, then switched off when needed to be used.

The **LDAC** pin is connected to shield PIN2 (D1). Used to latch data transmission. Inverse signal, so must be low when active. D1 must be initialized as active in software,then switched off when needed to be used.

The **SHDN** pin is connected to the shield PIN1(D0). Used to switch off output, even if data is incoming. Inverse signal, so must be low when active. D0 must be initialized as active in software, then switched off when needed to be used.

The **SCK** pin is connected to the shield D13 Clock speed for transfer data. Must be initialized with software via SPI1.

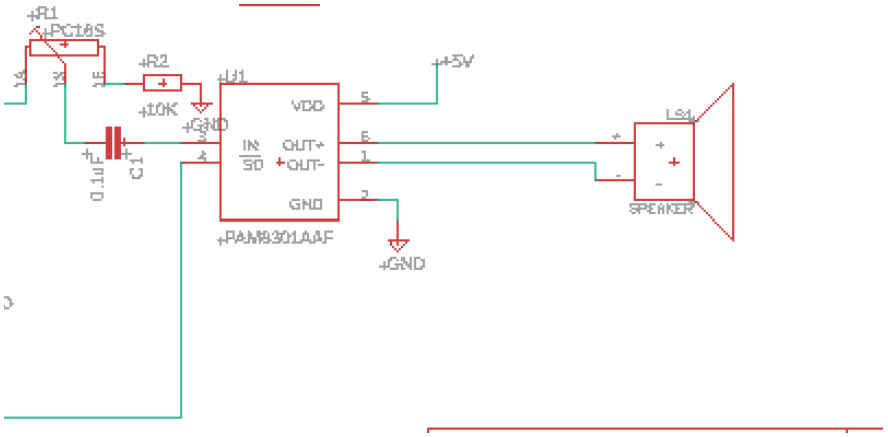
The **SDI** pin is connected to shield D11. Data input MOSI. Must be initialized with software via SPI1.

The **VDD** pin is connected to a shield 5V pin. Power supply.

The **VOUT** pin is connected to the audio amplifier via variable resistor (potentiometer) and 10uF capacitor. Variable resistor works as a voltage divider, connected to ground via a 10K resistor. Max output should be 0.28V (from lectures). Capacitor passes only the AC signal, and works as filter and level shifter.

The **AVSS** pin is connected to a shield GND pin. Ground.

- **PAM8301AAF Audio amplifier connection.**



The **IN** pin is connected to DAC output with some passive elements.

The **SD** pin is connected to the same line as the SHDN pin in the DAC. Same function.

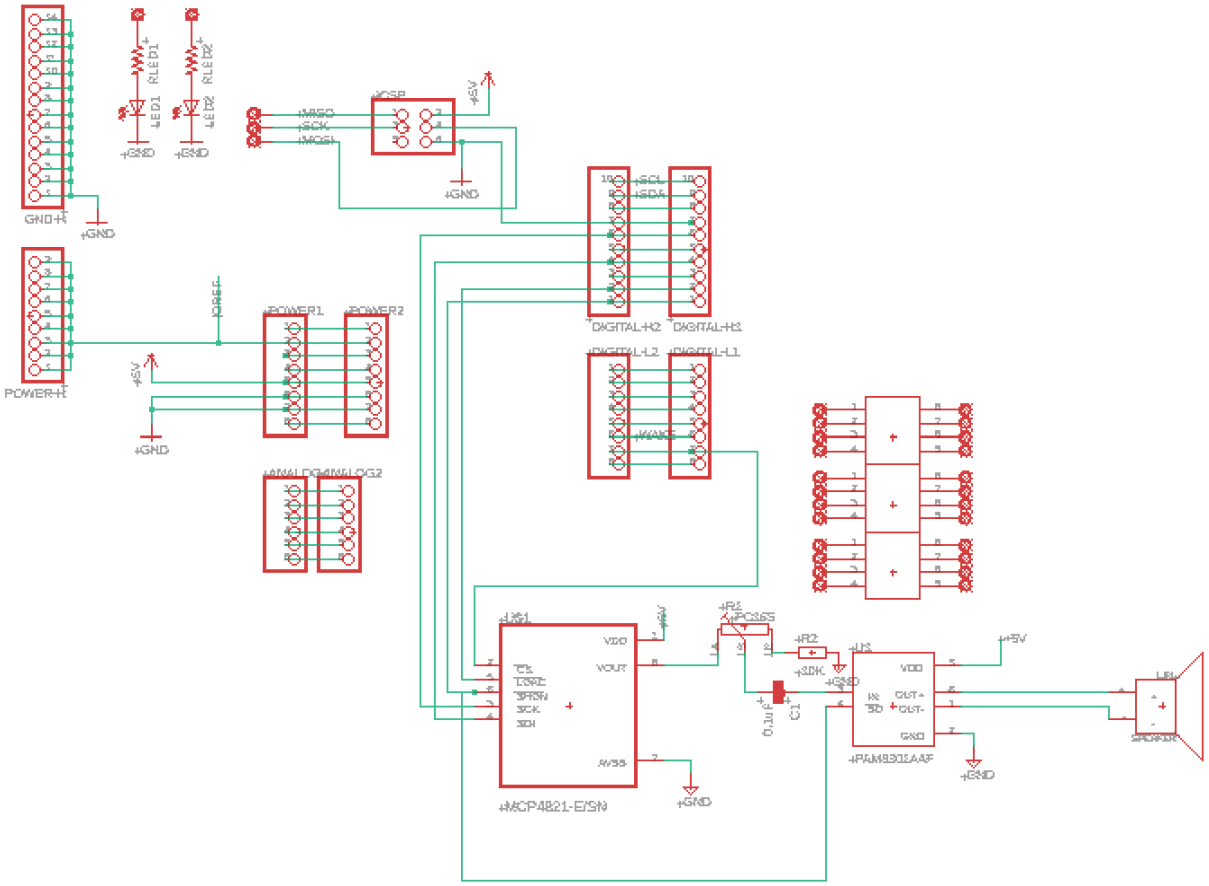
The **VDD** pin is connected to a shield 5V pin. Power supply.

The **Out+** pin is connected to the speaker + pin. Sound output.

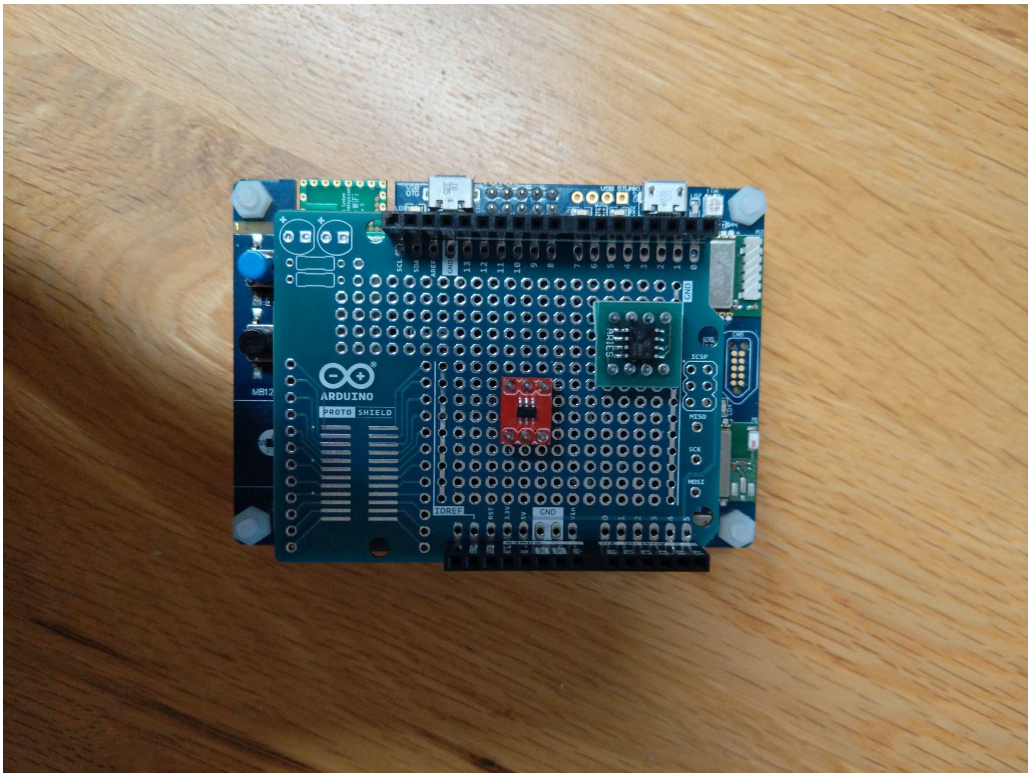
The **Out-** pin is connected to the speaker - pin. Sound output.

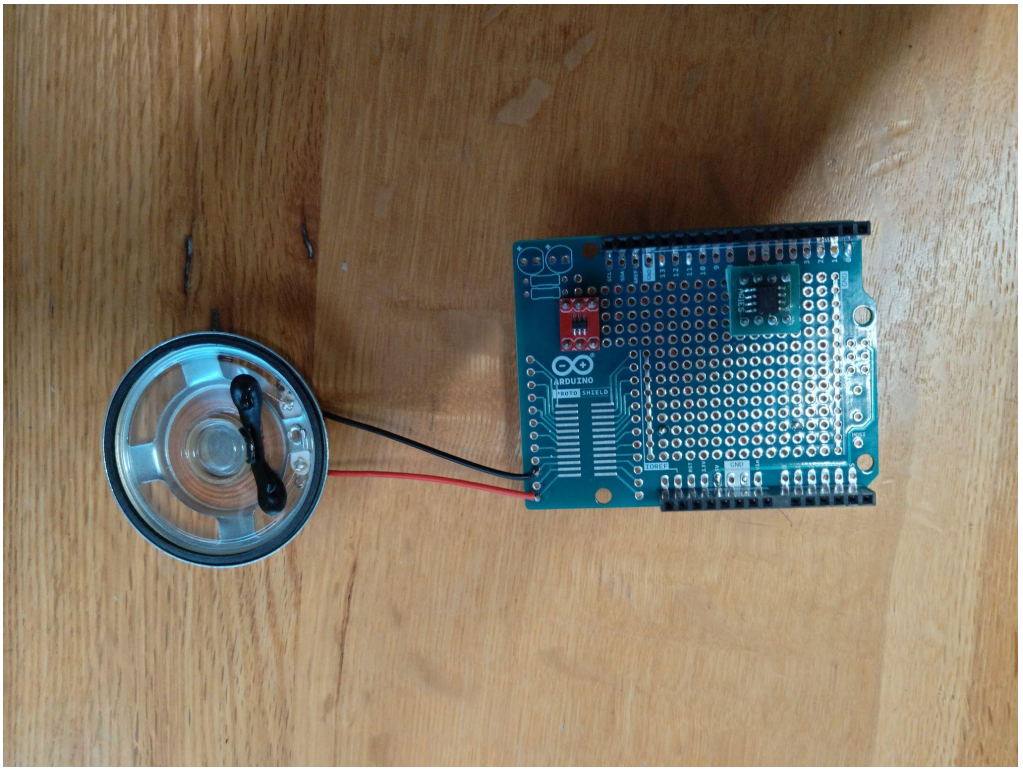
The **GND** pin is connected to a shield GND pin. Ground.

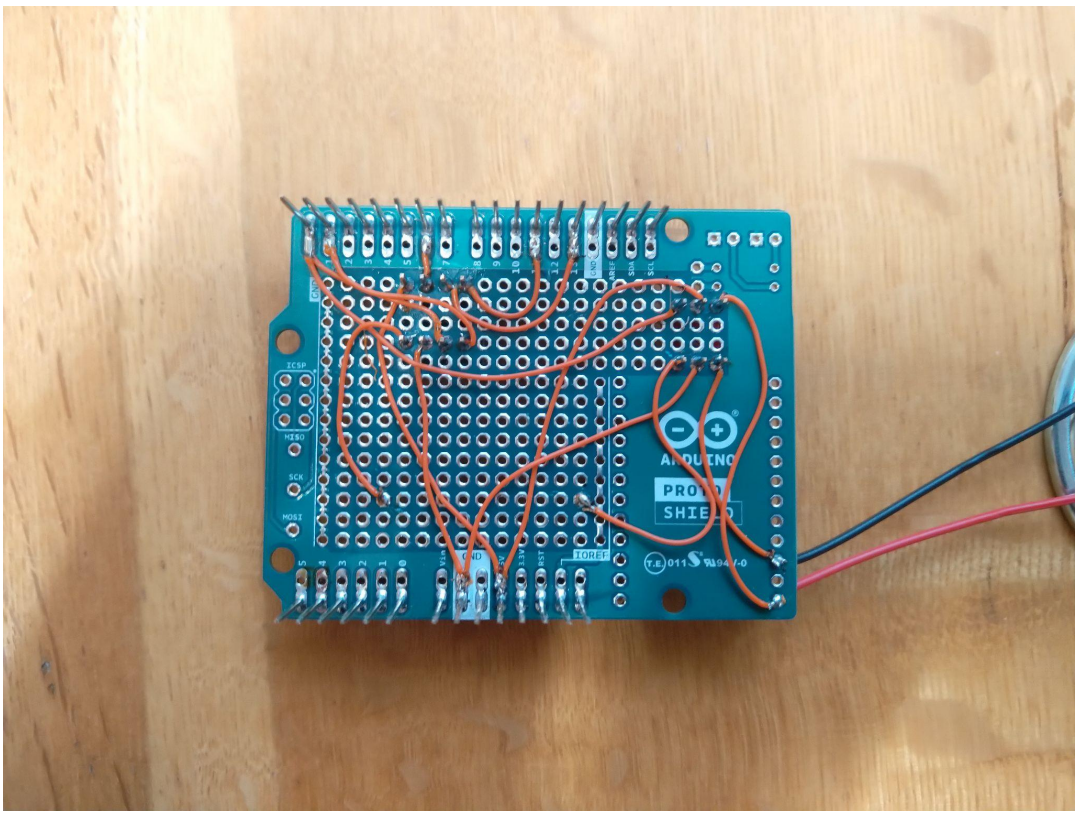
Sound out overall schematics.

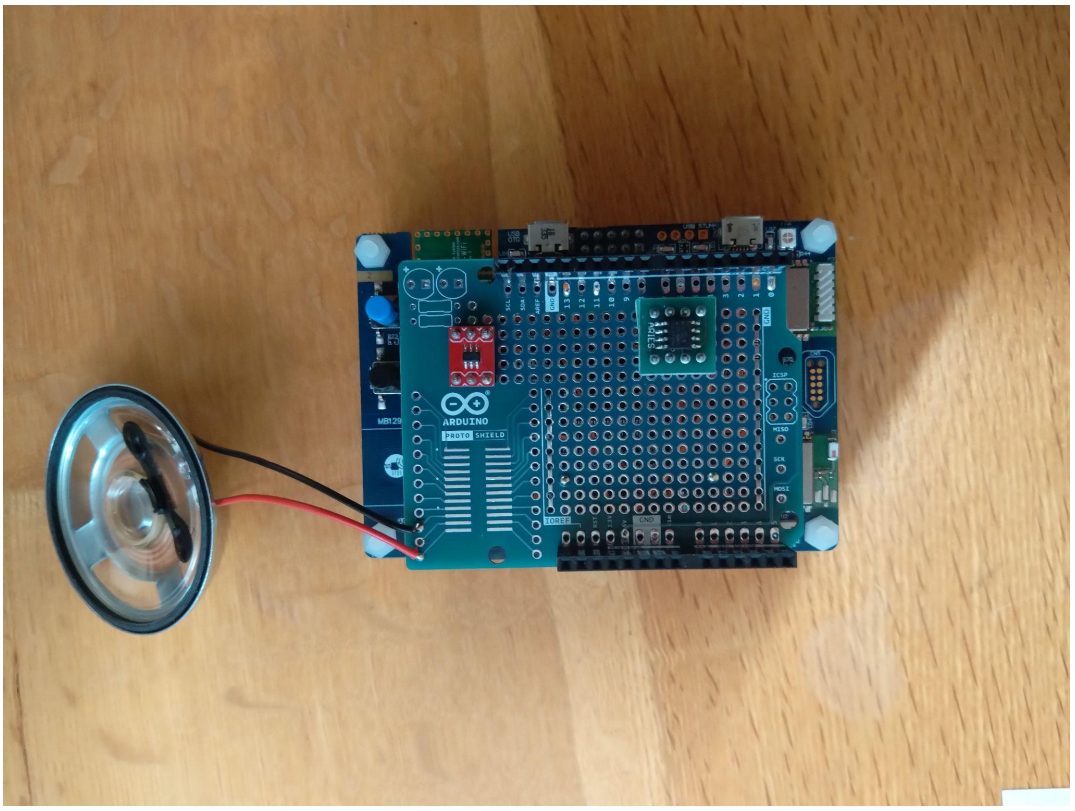


Unfortunately I was not able to obtain several components needed (capacitor, resistor and potentiometer), so the shield board is not completed yet.


ARDUINO
PROTO SHIELD


ARDUINO
PROTO SHIELD





3.2.2. Sound Out Software

Sound_out.c file was created containing several functions:

- DelayMicro() function to use microsecond delays.

```
__STATIC_INLINE void DelayMicro(__IO uint32_t micros)

{
        micros *=(SystemCoreClock / 1000000) / 5;
        while (micros—);

}
```

- Switching on the DAC

```
/* Switching on the DAC
HAL_GPIO_WritePin(AUDIO_RESET_GPIO, AUDIO_RESET_PIN, GPIO_PIN_RESET);
HAL_GPIO_WritePin(AUDIO_RESET_GPIO, AUDIO_LATCH_PIN, GPIO_PIN_RESET);
HAL_GPIO_WritePin(AUDIO_RESET_GPIO, AUDIO_SHDN_PIN, GPIO_PIN_RESET);
```

- Send the data to pin

```
HAL_SPI_Transmit(hspi1, pData, sizeof(pData),SPI_TRANSFER_TIMEOUT);
```

- Beeping noise function

```
for (int i = 0; i < 1000; ++i)
{
    // pData enter 1
    pData[0] = 0x5F;
    pData[1] = 0xFF;
    HAL_SPI_Transmit(hspi1, pData,
sizeof(pData),SPI_TRANSFER_TIMEOUT);
    delay(1)
    // pData enter 0
    pData[0] = 0x50;
    pData[1] = 0x00;
    HAL_SPI_Transmit(hspi1, pData,
sizeof(pData),SPI_TRANSFER_TIMEOUT);
}
```

- The pData format is 16 bytes, the first 4 are configuration, and the remaining 12 are data.
- Latch and shutdown pins can be connected directly to GND avoiding use of extra pins.
- Software coding was stuck due to a not fully completed hardware part. Unable to test code.

3.3. Sound Out - Integration with Another Interface

Sound Out can be integrated with other devices in many ways. First of all use a lower and higher tone beeping sound when the magnet moves closer or further in the magnetometer game. It will simulate a motion tracker radar system.

Any sounds can be played when a game is lost or won in other sensor games. Joypad movement can be accompanied with sound effects. Sounds can be loaded from an SD card. All these integrations require more experience and practice in microcontroller programming.

4. Individual Deliverables - Inês Carvalho

4.1. Temperature Sensor

The temperature sensor is integrated into our project as part of a level in our game. In this level, the player has to reach a certain temperature in a certain period of time.

The temperature of the board can be increased by the player, for example by placing it closer to a heater. It can also be decreased, for example, by placing the board near an open refrigerator door. One of the aims of this level is to leave it to the player's imagination how to increase or decrease the temperature of the board.

4.1.1. Temperature Sensor - Functions

TSensor_Who_Am_I Function

The TSensor_Who_Am_I function reads from the Who_Am_I register (0Fh) and returns the device identifier. In this case, the return number should be equal to 188=BCh.

```
/**
  * @brief  Read Who_Am_I register.
  * @retval Temperature Sensor Device Identifier
  */
 uint8_t TSensor_Who_Am_I(void){

       uint8_t Device_ID;

       HAL_I2C_Master_Transmit(&hi2c2, TN1218_Temperature_WriteAddress,
&WhoAmI_Address, 1, 100);
       HAL_I2C_Master_Receive(&hi2c2, TN1218_Temperature_ReadAddress, &Device_ID,
1, 100);
       HAL_UART_Transmit(&huart1, &Device_ID, 1, 100);

       return Device_ID;
 }
```

Software Code 2: Temperature Sensor - TSensor_Who_Am_I Function

BSP_TSENSOR_Init Function

The BSP_TSENSOR_Init function can be found in the official BSP Library from STM32.

This function initializes the temperature sensor and returns 0 if the initialization was successful or 1 if the initialization failed.

```
/**
  * @brief  Initializes peripherals used by the I2C Temperature Sensor driver.
  * @retval TSENSOR status
  */
uint32_t BSP_TSENSOR_Init(void)
{
  uint8_t ret = TSENSOR_ERROR;

#ifdef USE_LPS22HB_TEMP
  tsensor_drv = &LPS22HB_T_Drv;
#else /* USE_HTS221_TEMP */
  tsensor_drv = &HTS221_T_Drv;
#endif

  /* Low level init */
  SENSOR_IO_Init();

  /* TSENSOR Init */
  tsensor_drv->Init(TSENSOR_I2C_ADDRESS, NULL);

  ret = TSENSOR_OK;

  return ret;
}
```

Software Code 3: Temperature Sensor - BSP_TSENSOR_Init Function

BSP_TSENSOR_ReadTemp Function

The BSP_TSENSOR_ReadTemp function can be found in the official BSP Library from STM32.

This function reads from the temperature register and returns the measured temperature float value.

```
/**
  * @brief  Read Temperature register of TS751.
  * @retval STTS751 measured temperature value.
  */
float BSP_TSENSOR_ReadTemp(void)
{
  return tsensor_drv->ReadTemp(TSENSOR_I2C_ADDRESS);
}
```

Software Code 4: Temperature Sensor - BSP_TSENSOR_ReadTemp Function

TSensor_Task Function

The TSensor_Task function is the only function to be called in the infinite while loop inside the main, regarding the temperature sensor. It is also important to note, as can be seen in the code below, that this function calls, within it, the BSP_TSensor_ReadTemp function.

This function is responsible for all the main action of the temperature sensor, i.e. for reading, in cycle, the temperature values with an interval of approximately 1 second between each reading. It is also responsible for building the level of this sensor, displaying different messages for the two possible cases:

- Level Passed: if the player does reach the required temperature within the time limit, the level is completed and a congratulatory message will be displayed.
- Level Failed: if the player does not reach the required temperature within the time limit, a message will be displayed informing the player that the level has been failed and he has to try again.

It is also important to note that this function returns the value 0 if the level was passed.

```
/**
   * @brief  Periodically read the temperature value from the register, checking
if the temperature level was passed or failed.
   * @retval 0 if the level was passed
   */
 uint8_t TSensor_Task(void){

              int i;  // Aux variable for the cycles
              uint8_t temperature1;  // Integer part of the temperature value
              uint8_t temperature2;  // Fractional part of the temperature value
              float temperature_fraction;  // Aux variable for the preparation of
the fractional part of the temperature value

              //Formatted Messages
              char str_temperature[100] = ""; // Formatted message to display the
temperature value

              //Messages
              uint8_t message3[] = "***** Temperature Level Initialized *****
\r\n\n ";
              uint8_t message4[] = " Level Completed! :) \r\n ";
              uint8_t message6[] = " Level Failed! :( \r\n ";
              uint8_t message7[] = "Press the Black Button to Try Again \r\n\n ";

       while(1){
        for(i=0; i<10; i++) {  // For cycle to count how many times the
temperature value is read. This can be translated in seconds (an approximation)

                temperature_value = BSP_TSENSOR_ReadTemp();  // Get the value from
the register
                temperature1 = temperature_value;  // Get the integer part of the
temperature value
                temperature_fraction = temperature_value - temperature1;  //
Prepare the fraction part
                temperature2 = trunc(temperature_fraction * 100);  // Get the
fraction part of the temperature value

                snprintf(str_temperature,100," Temperature = %d.%02d\n\r",
temperature1, temperature2);  // Print the temperature value
```

```
                HAL_UART_Transmit(&huart1,( uint8_t *
)str_temperature,sizeof(str_temperature),1000);
                HAL_Delay(1000);  // Wait 1 second

                if(i == seconds_temperature_level - 1) { // If the player has not
reached the temperature value during the given time

                HAL_UART_Transmit(&huart1,message6,sizeof(message6),1000);  //
Message: level failed

                HAL_UART_Transmit(&huart1,message7,sizeof(message7),1000);  //
Message: press the blue button to try again
                while(BSP_PB_GetState(BUTTON_USER) == GPIO_PIN_SET);  // Function
to press the set button
                while(BSP_PB_GetState(BUTTON_USER) == GPIO_PIN_RESET);  //
Function to press the reset button (good practice)
                }

                if(temperature1 == value_temperature_level) {  // If the player
has reached the temperature value in time
                        i = seconds_temperature_level - 1 ;

                        HAL_UART_Transmit(&huart1,message4,sizeof(message4),1000);
// Message: level passed

                        return 0;
                }
              }
       }
 }
```

Software Code 5: Temperature Sensor - TSensor_Task Function

Temperature Tests Functions

It is also important to mention in this section that three more functions were created, each one corresponding to a test to be performed before the start of the temperature level: TSensor_Who_Am_I_Test, TSensor_Initiation_Test and TSensor_Admitted_Current_Temperature_Value_Test. Since an integration of all the project's code will be carried out later, it was necessary to organise the tests into functions so that they can be easily integrated.

All three of these functions are of type void, so they have no return.

These tests will be discussed in section 4.1.3.

4.1.2. Temperature Sensor - Global Variables

The global variables, i.e. the variables that can be accessed by any function throughout the entire program, are present in the code below. As you can see, all the variables are properly commented, so it is possible to understand their function in the code and, consequently, which function are they used by.

The three first variables were created in order to store the register's addresses. Typically, this would be done with defines. However, as pointers to these values were needed, it became necessary to create these variables.

The following two variables (value_temperature_level and seconds_temperature_level) are only used by the TSensor_Task function in the pre-integration phase. These variables were created with the post-integration phase in mind, i.e. they were created so that in the future it would be possible for the SD Card device to define their value. In practice, this means that it would be the responsibility of the SD Card device to define what temperature value the player has to reach in this level, and how long he has to do it.

```
/* Global Variables */

uint8_t WhoAmI_Address = 0x0F;  // Who_Am_I Address
uint8_t TN1218_Temperature_WriteAddress = 0xBE; // Write Address
uint8_t TN1218_Temperature_ReadAddress = 0xBF;  // Read Address

uint8_t value_temperature_level = 28;  // Value the player must reach on the
temperature level
uint8_t seconds_temperature_level = 10;  // Seconds the player has to reach the
temperature value

float temperature_value = 0;   // Aux variable to the measured temperature value
float current_temperature_value = 0;   // Aux variable to the measured current
temperature value
```

Software Code 6: Temperature Sensor - Global Variables of the Main File

Other global variables were used, which are included in the BSP library as can be seen below.

```
/* Global Variables of the BSP library */

static TSENSOR_DrvTypeDef *tsensor_drv;
```

Software Code 7: Temperature Sensor - Global Variables of the BSP Library

4.1.3. Temperature Sensor - Testing

For the temperature sensor, three tests were written that are performed before the temperature level starts. As mentioned before (section 4.1.1.), these three tests are organised within three functions.

- **Who_Am_I Test** → TSensor_Who_Am_I_Test Function
- **Sensor Initiation Test** → TSensor_Initiation_Test Function
- **Admitted Current Temperature Value Test** → TSensor_Admitted_Current_Temperature_Value_Test Function

Who_Am_I Test

The Who_Am_I test is the first test to be performed and includes the Who_Am_I function mentioned above in order to get the device identifier.

The test checks if the temperature sensor device identifier is what is supposed to be, this is 188=BCh. Three messages are always displayed: the first indicates the name of the test, "Who_Am_I Test", the second the value returned by the function, the device identifier, and the last one indicates whether the test was successful or not.

For the test to succeed, the value returned by the function must be 188=BCh. Otherwise, the test fails and an error message will be displayed.

```
/**
  * @brief  Who_Am_I Test
  * @retval -
  */
void TSensor_Who_Am_I_Test(){

       uint8_t Device_ID_aux;

       //Formatted Messages
       char str[100] = ""; // Formatted message to display

       //Debug Messages
       uint8_t message8[] = "\n\n\n\n\n ***** Who_Am_I TEST ***** \r\n ";
       uint8_t message9[] = " ***** Who_Am_I TEST Successfully Performed *****
\r\n ";
       uint8_t message10[] = " ***** ERROR - Who_Am_I TEST ***** \r\n ";


         /* Who_Am_I TEST*/
         HAL_UART_Transmit(&huart1,message8,sizeof(message8),1000);  // Message:
name of the test
         Device_ID_aux = TSensor_Who_Am_I();   // Get the Device Identifier

         if(Device_ID_aux == 188){  //If the device identifier is what it's
supposed to be (188)
                snprintf(str,100,"Device ID = %d\n\r", Device_ID_aux);  //Print
the device identifier obtained from the register
                HAL_UART_Transmit(&huart1,( uint8_t * )str,sizeof(str),1000);
                HAL_UART_Transmit(&huart1,message9,sizeof(message9),1000);  //
Message: test was successful
         }

         else{   // If the device identifier is different from 188
                snprintf(str,100,"Device ID = %d\n\r", Device_ID_aux);  // Print
the number obtained from the register
                HAL_UART_Transmit(&huart1,message10,sizeof(message10),1000);  //
Message: test failed
         }
}
```

Software Code 8: Temperature Sensor - Who_Am_I Test
(TSensor_Who_Am_I_Test Function)

Fig 5: Temperature Sensor - Who_Am_I Test on the Terminal

Sensor Initiation Test

The sensor initiation test is the second test to be performed and includes the BSP_TSENSOR_Init function mentioned above.

This second test is responsible for initialising the sensor and checking whether the initialisation was successful or not. Two messages are always displayed: the first indicates that the sensor initialisation is going to start, "Initialize Temperature Sensor" and the second and last indicates whether the test was successful or not.

As mentioned before, for the test to succeed the value returned by the BSP_TSENSOR_Init function must be 0. If the return number is 1, the test fails and an error message will be displayed.

```
/**
  * @brief  Sensor Initiation and Test
  * @retval -
  */
void TSensor_Initiation_Test(){

	uint8_t Sensor_Initiation_aux;

	//Debug Messages
	uint8_t message1[] = "\n***** Initialize Temperature Sensor and TEST *****
\r\n ";
	uint8_t message2[] = "***** Temperature Sensor Initialized ***** \r\n\n ";
	uint8_t message11[] = "***** ERROR - Sensor Initiation TEST ***** \r\n\n
";

	  /* Sensor Initiation and TEST */
	  HAL_UART_Transmit(&huart1,message1,sizeof(message1),1000);  // Message:
name of the test
	  Sensor_Initiation_aux = BSP_TSENSOR_Init();  // Sensor initialisation

	  if(Sensor_Initiation_aux == 0) {  // If the variable returned by the
initialization function is 0
	  HAL_UART_Transmit(&huart1,message2,sizeof(message2),1000);  // Message:
test was successful
	  }

	  if(Sensor_Initiation_aux == 1) { // If the variable returned by the
initialization function is 1
		  HAL_UART_Transmit(&huart1,message11,sizeof(message11),1000);  //
Message: test failed
	  }
}
```

Software Code 9: Temperature Sensor - Sensor Initiation and Test

(TSensor_Initiation_Test Function)

Fig 6: Temperature Sensor - Sensor Initiation and Test on the Terminal

Admitted Current Temperature Value Test

The admitted current temperature value test is the last and most complex test performed and includes the BSP_TSENSOR_ReadTemp and an excerpt of code present in the TSensor_Task function, both mentioned above.

The test checks if we can get an admitted temperature value, giving us the current temperature value. According to the datasheet, this sensor operates in a temperature range of -40 °C to +120 °C. Therefore, it is considered that an admissible value must be within this range.

Three messages are always displayed: the first indicates the name of the test, the second indicates the current temperature value obtained, and the last one indicates whether the test was successful or not, i.e. if the value is within the range mentioned above.

```
/**
  * @brief  Admitted Current Temperature Value Test
  * @retval -
  */
void TSensor_Admitted_Current_Temperature_Value_Test(){ //Admitted Current
Temperature Value TEST

       uint8_t current_temperature1;
       uint8_t current_temperature2;
       float current_temperature_fraction;

       //Formatted Messages
       char str_current_temperature[100] = ""; // Formatted message to display
the current temperature value

       //Debug Messages
       uint8_t message12[] = "***** Admitted Current Temperature Value TEST *****
\r\n ";
       uint8_t message13[] = " ***** Admitted Current Temperature Value TEST
Successfully Performed ***** \r\n\n ";
       uint8_t message14[] = " ***** ERROR - Admitted Current Temperature Value
TEST ***** \r\n\n ";

         HAL_UART_Transmit(&huart1,message12,sizeof(message12),1000);  //
Message: name of the test

         current_temperature_value = BSP_TSENSOR_ReadTemp();  // Get the value
from the register
         current_temperature1 = current_temperature_value;  // Get the integer
part of the temperature value
         current_temperature_fraction = current_temperature_value -
```

```
current_temperature1;   // Prepare the fraction part
        current_temperature2 = trunc(current_temperature_fraction * 100);    //
Get the fraction part of the temperature value

        snprintf(str_current_temperature,100," Current Temperature =
%d.%02d\n\r", current_temperature1, current_temperature2);  // Print the current
temperature value
        HAL_UART_Transmit(&huart1,( uint8_t *
)str_current_temperature,sizeof(str_current_temperature),1000);

        if(current_temperature1>-40 && current_temperature1<120){  // If the
temperature value is within an admissible range
               HAL_UART_Transmit(&huart1,message13,sizeof(message13),1000);  //
Message: test was successful
        }

        else{  // If the temperature value is not within an admissible range
               HAL_UART_Transmit(&huart1,message14,sizeof(message14),1000);  //
Message: test failed
        }

}
```

Software Code 10: Temperature Sensor - Admitted Current Temperature Value Test (TSensor_Admitted_Current_Temperature_Value_Test Function)

```
***** Admitted Current Temperature Value TEST *****
 Current Temperature = 27.23
***** Admitted Current Temperature Value TEST Successfully Performed *****
```

Fig 7: Temperature Sensor - Admitted Current Temperature Value Test on the Terminal

4.1.4. Temperature Sensor - Practical Example

For this temperature level example, the player had to reach a temperature of 28 degrees, in 10 seconds. The two figures below show what happens if the level was passed (Figure 8) and if the level was failed (Figure 9).

It is important to note that for this demonstration the temperature of the board was increased by moving it closer to a heater.

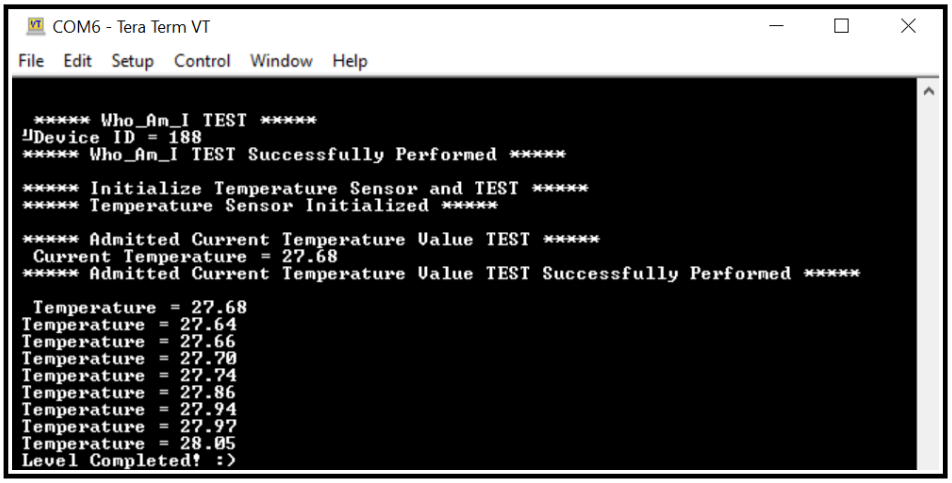


Fig. 8: Temperature Level Passed

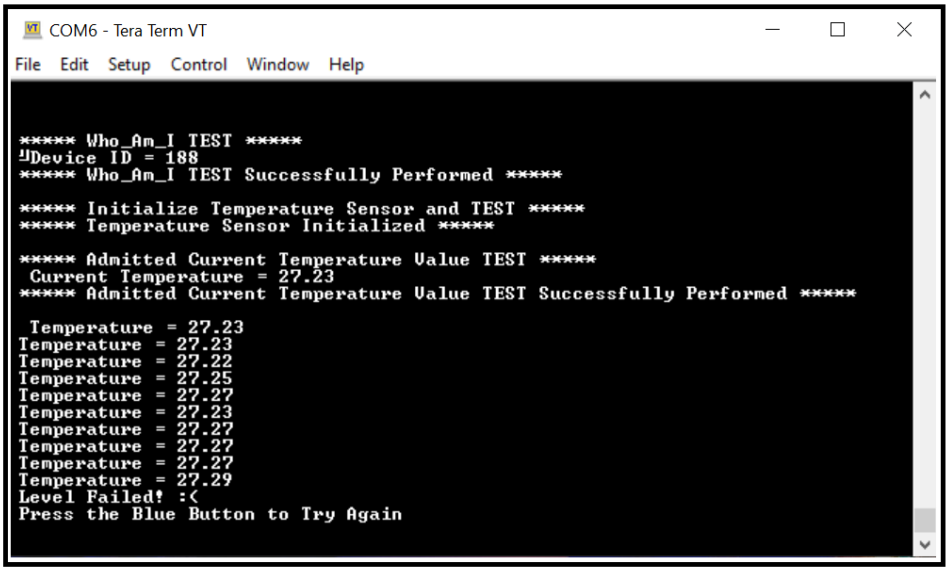


Fig. 9: Temperature Level Failed

4.2. Sound In Device

The purpose of the sound in device in our project is to record voice for future playback through another device (sound out).

The following subsections will explain the procedure used to develop the hardware, and the software, of this SPI device.

4.2.1. Sound In Hardware

The sound in hardware requires an LF353 operational amplifier, an MCP3002 10-Bit Analog-to-Digital Converter and, of course, the Arduino Proto Shield Rev3 in order to integrate this interface with the board.

This SPI device needs to have an operational amplifier connected to an ADC in order to provide the gain and level-shifting. Coupling capacitors and a low-pass filter will also be required, as will be explained later.

It is also important to note that the reference voltage value chosen, in accordance with the datasheet, was 5V.

The two figures below illustrate the schematic and the board of this device. Note that both were developed using the EAGLE software.

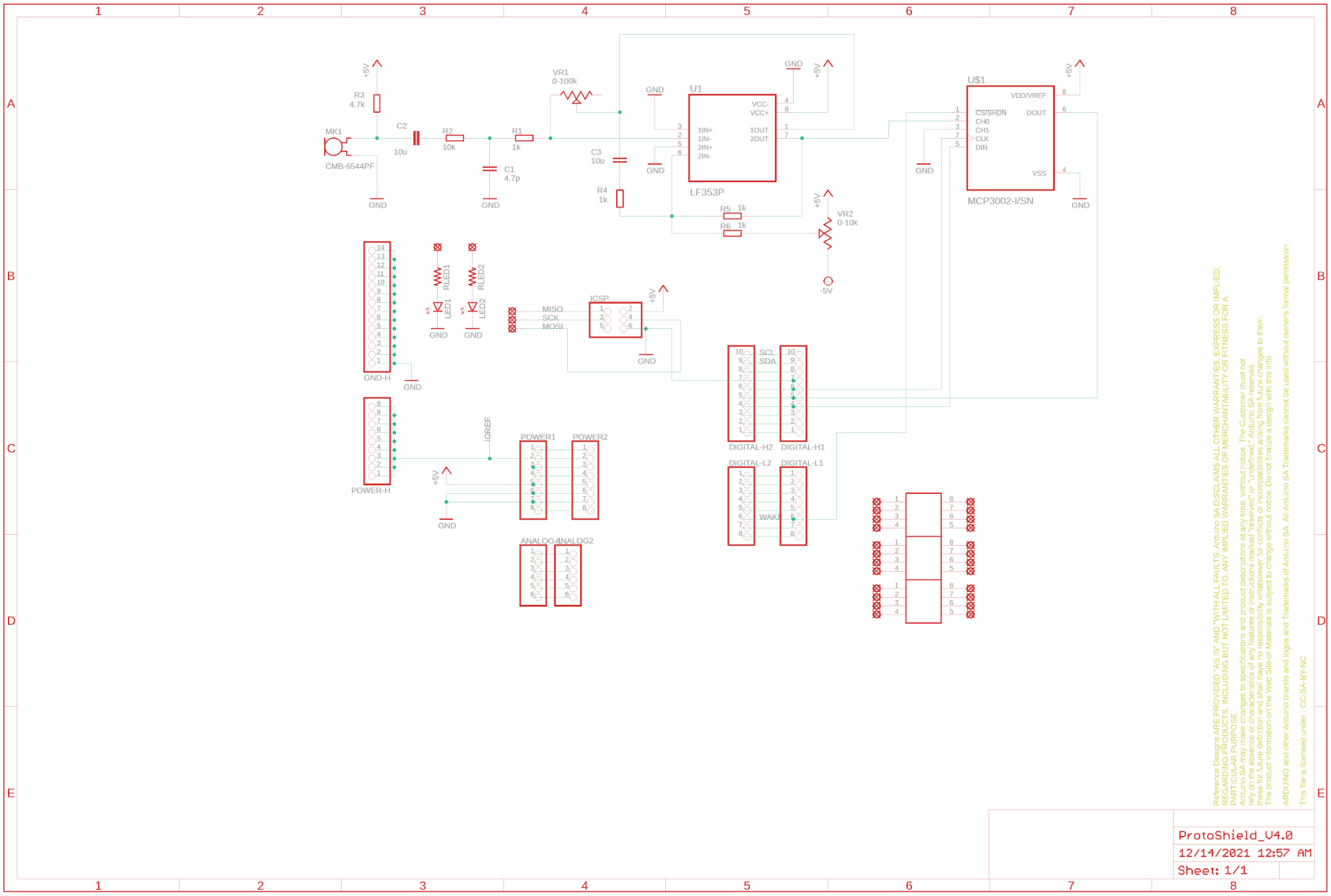


Fig. 10: Sound In Schematic

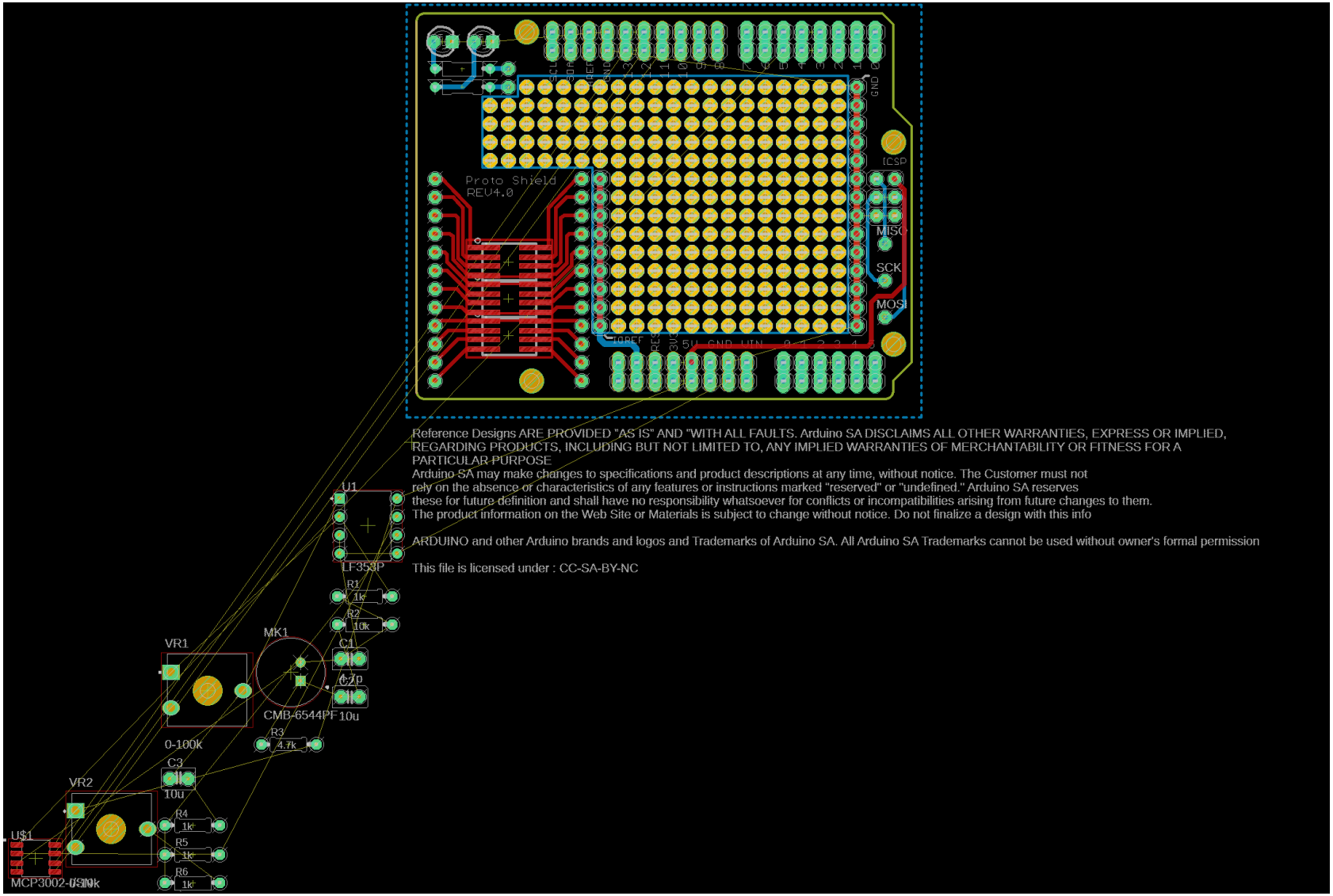


Fig. 11: Sound In Board

Firstly, as you can see in the figures above, it was necessary to bised the microphone using a 4.7kΩ resistor.

After the biasing, a 10µF coupling capacitor was used in order to block the DC signal, letting only the AC signal pass through.

Following that, a 2º order low-pass filter is required to eliminate unwanted high-frequency noise. The passband frequency is 3kHz because the software of this device is designed to record a sample of a human voice. Therefore, after a 10k resistor was chosen, the value of the capacitor was calculated using the following formula: $f = \frac{1}{2\pi RC}$, with $f = 3\text{kHz}$ and $R = 10\text{k}\Omega$. The value of the capacitor is approximately 4.7pF.

After the low-pass filter, a gain needs to be provided: $G = \frac{R_f}{R_{in}}$. Therefore, an 1kΩ resistor and a 0-100kΩ potentiometer were used since the value of the last one depends on the input signal of the microphone. This part of the schematic is intended to amplify the signal. It should be noted that all the parts mentioned so far were achieved using one of the LF353's amplifiers.

Following the gain, and using the LF353's second amplifier, it was once again necessary to use a 10µF coupling capacitor. Then, a level-shifting is required in order for the input signal to match the input range of the ADC. This circuit was achieved using 1k resistors and a 0-10kΩ potentiometer.

The connection between the LF353 operational amplifier and the MCP3002 ADC Converter is achieved through the connection between the output pin of the LF353's second amplifier and the input pin of the MCP3002'S channel zero.

According to the MCP3002 datasheet, the connection between the MCP3002 ADC Converter and the Arduino Proto Shield Rev3 is achieved using the following MCP3002's pins: Chip Select (CS), SPI Serial Clock (CLK), SPI Serial Data Input (Din) and SPI Serial Data Output (Dout). The SPI connections, made according to the MCP3002 datasheet, are represented in the table below:

MCP3002 ADC Converter	Arduino Proto Shield Rev3
CS	6 Digital Pin
CLK	SCLK
Din	MOSI
Dout	MISO

Table 2: SPI Connection between MCP3002 and Arduino Proto Shield

In the figure below, it is possible to confirm that the interface hardware of this SPI device has been successfully completed:

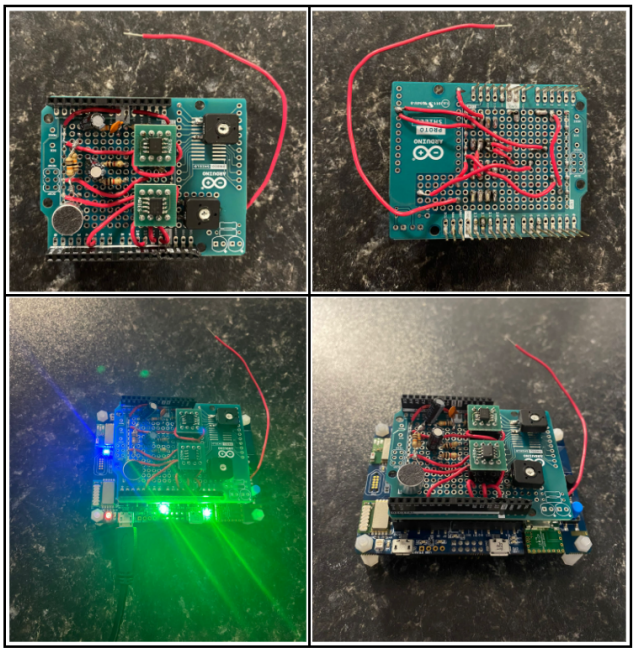


Fig. 12: Real Images of the Sound In Interface Hardware

It is important to note that to facilitate the passage from theory to practice, the -5V of the 0-10kΩ potentiometer for the level-shifting, were connected to the ground (0V) as it was considered that this procedure would not have a negative impact on the circuit.

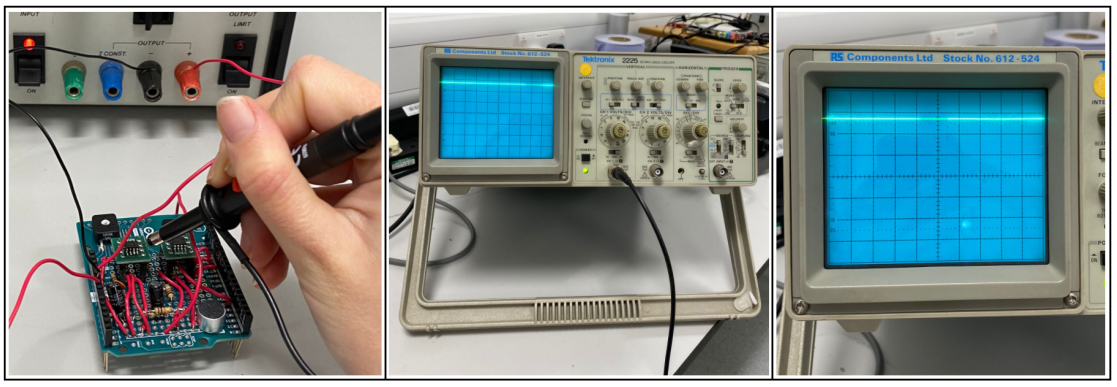


Fig. 13: Real Images of the Test performed on LF353's Pin 7
(LF353's output pin connected to the MCP3002's input pin)

It should be noted that the hardware of this device is not 100% functional. Although the theoretical approach is correct, there is a small problem with the last state of the circuit, the level_shifting. It would take a bit more time and research to readjust the values of the resistors and potentiometer in this state. However, as can be seen from

the figure above, voltage can be obtained throughout the circuit, thus confirming that the problem with this device is not severe.

Therefore, the development and success of the sound in software was not affected as can be seen in the following section.

4.2.2. Sound In Software

The software of this device was developed to read the voltage of the MCP3002 ADC in order to record voice, for this to be later saved and use to output voice. Therefore, this software is only responsible for getting the data stream out of the microphone, whereby the processing of the voltage obtained is a higher level design.

The figure below depicts the SPI Bus, present on page 15 of the MCP3002 datasheet, which will be used to communicate between the hardware and software of this device.

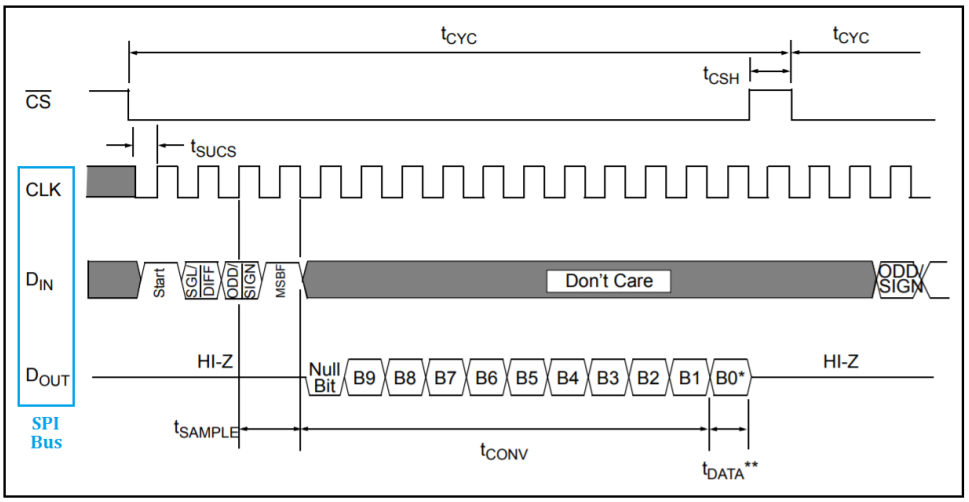


Fig. 14: Sound In - SPI Bus
(MCP3002 Datasheet)

As previously mentioned, the chip select pin (CS) was previously decided by all members of the group in order to avoid members using the same pin. This is an important point since this pin is the communication indicator with our device. As a result, it is important to remember that the chip select pin of this device is pin 6.

Regarding the SPI Bus, according to the datasheet of the ADC, the bit 0 (Start) needs to be logic high (1) in order to read and write from the SPI bus. Also, the bit 1 (SGL/DIFF) is 1 in order to select the single mode and the bit 2 (ODD/SIGN) is 0 since in the sound in hardware only channel 0 of the ADC was used, as you can see in the table below:

	CONFIG BITS		CHANNEL SELECTION		GND
	SGL/DIFF	ODD/SIGN	0	1	
Single-Ended Mode	1	0	+		—
	1	1		+	—
Pseudo-Differential Mode	0	0	IN+	IN-	—
	0	1	IN-	IN+	—

Fig. 15: Sound In - Table of the SPI Bits Configuration (MCP3002 Datasheet)

When programming the SPI device in the STM32CubeIDE software, it was necessary to start by defining the “Pinout and Configuration” of the SPI communication, as you can see in the figure below. The settings that are important to note are highlighted in red in the figure.

The pin selected in the pinout configuration of the SPI mode was the PA7 pin as this is the pin corresponding to the MOSI pin on the board (D11). Therefore, as you can see in the figure below, the mode chosen for this pin was obviously the “Full-Duplex Master”. It is also important to note that it was necessary to disable the "Hardware NSS Signal" option, since the chip select pin was previously assigned.

Regarding the “Parameter Settings”, more precisely the “Basic Parameters”, it was necessary to change the number of bits in the “Data Size” to 15 bits since this is the total number of bits of the Din and Dout in the SPI Bus, as you can see by the figure of the SPI bus.

Regarding the “Clock Parameters”, it was necessary to increase the “Prescaler” to 16 bits since when we integrate all the devices, a decrease of the baud rate along the device stack becomes indispensable.

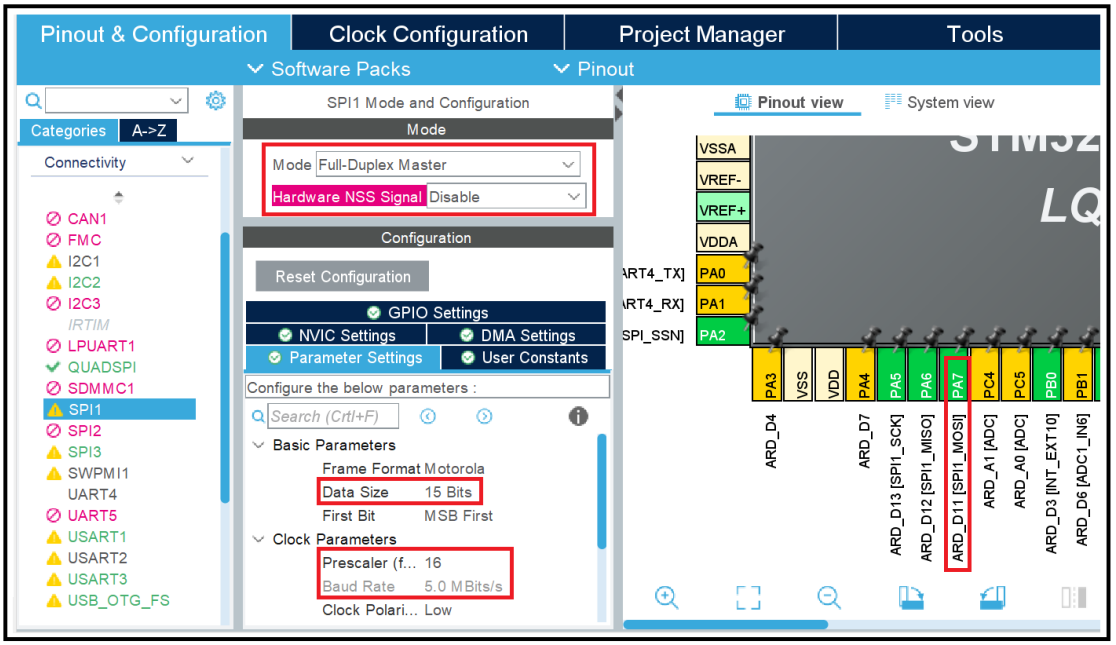


Fig. 16: Sound In - Pinout and SPI Configuration
(STM32CubeIDE Software)

After this initial configuration, which allows a good SPI communication, a single function was created that is responsible for the task of this device. The reason why there is only one function, i.e. the code for this device is all organised in one function, is to make the integration with the other devices simpler.

ReadADC_MCP3002 Function

The ReadADC_MCP3002 function is the only function to be called in the infinite while loop inside the main, regarding the sound in device. It is important to note that this function was developed using the MCP3002 ADC datasheet.

This function is responsible for all the main action of the sound in device, i.e. for reading, in cycle, the ADC voltage values with an interval of approximately 200 milliseconds between each reading, 50 times. Therefore, this function allows voice recording for approximately 10 seconds (200ms × 50 = 10s).

It is also important to note that this function returns the value 0 after the recording is done, as you can see by the code and figure below.

```
/**
   * @brief Read the ADC voltage during approximately 10 seconds, for a future
playback of the recorded voice.
   * @retval 0 when the recording is over
   */
uint8_t ReadADC_MCP3002(){

      //Hardware Variables
      uint8_t voltage_ref_ADC = 5;
```

```
      //Sound In
      uint8_t k;  // Aux varibled for the cycle
      uint8_t TxData_ADC[2];
      uint8_t RxData_ADC[2];
      uint16_t RxData_ADC_shifted;
      char buf[10];
      float voltage_ADC;

       // Message
    uint8_t message15[] = " \r\n ***** MCP3002 ADC Voltage  ***** \r\n ";
       uint8_t message16[] = "Press the Button to Record Voice \r\n\n ";
       uint8_t message17[] = "Recording is Over \r\n\n ";

      // Construct the message according to the Sound In Hardware and the
MCP3002 datasheet
       TxData_ADC[0] = 0b11;
       TxData_ADC[0] = ((TxData_ADC[0] << 1) + 0) << 5;
       TxData_ADC[1] = 0;

        HAL_UART_Transmit(&huart1,message15,sizeof(message15),1000);  //
Message: name of the task

        HAL_UART_Transmit(&huart1,message16,sizeof(message16),1000);  //
Message: press the blue button to start recording
        while(BSP_PB_GetState(BUTTON_USER) == GPIO_PIN_SET);  // Function to
press the set button
        while(BSP_PB_GetState(BUTTON_USER) == GPIO_PIN_RESET);  // Function to
press the reset button (good practice)

       for(k=0; k<50; k++) { // For cycle to read the ADC voltage 50 times at
200 ms intervals, i.e. for approximately 10 seconds

       //Get the reply
       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_RESET);  //Chip Select
(CS): 6 Digital Pin
       HAL_SPI_TransmitReceive(&hspi1, TxData_ADC, RxData_ADC, 2, 1000);  //Get
the data
       HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_SET);  //Chip Select (CS):
6 Digital Pin

       // Combining 2 bytes into 1 byte - Get ADC value by shifting right once
and masking out all but last 10 bits
       RxData_ADC_shifted = ((uint16_t)RxData_ADC[0] << 8) + RxData_ADC[1];
       RxData_ADC_shifted >>= 1;
       RxData_ADC_shifted &= 0b0000001111111111;

       // Calculate the ADC voltage
       voltage_ADC = (5 * RxData_ADC_shifted) / 1024; //Multiply by 5 because it
is the reference voltage and Divided by 1024 because MCP3002 is a 10-bit ADC

       /* Convert voltage to integer and decimal format */
       voltage_ADC = voltage_ADC * 100;
       sprintf(buf, "%u.%02u V\r\n",((unsigned int)voltage_ADC / 100),((unsigned
int)voltage_ADC % 100)); //Recast the float into two parts (2 integers) and print
decimal figure
       HAL_UART_Transmit(&huart1, (uint8_t *)buf, strlen(buf), 1000);
```

```
        HAL_Delay(200);   // Wait 200 ms
        }

        HAL_UART_Transmit(&huart1,message17,sizeof(message17),1000);  // Message:
recording is over
        return 0;
}
```

Software Code 11: Sound In - ReadADC_MCP3002 Function

To demonstrate the functionality of the developed software of this device, a screenshot of the terminal is shown in the figure below. As you can see, the player has to press the blue button of the board in order to start recording voice for approximately 10 seconds.

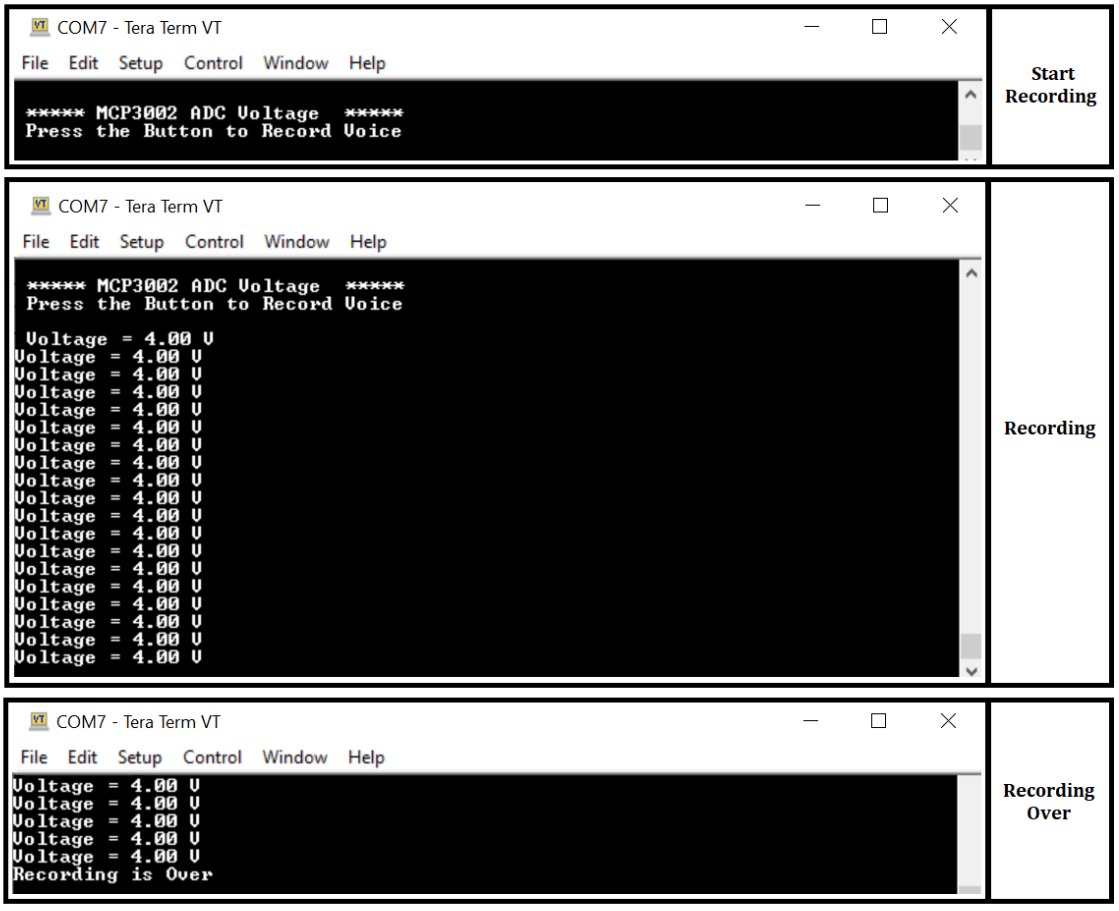


Fig. 17: Sound In - Recording Voice (Terminal)

In the figure below is illustrated the real image of the hardware and software final test of the sound in device.

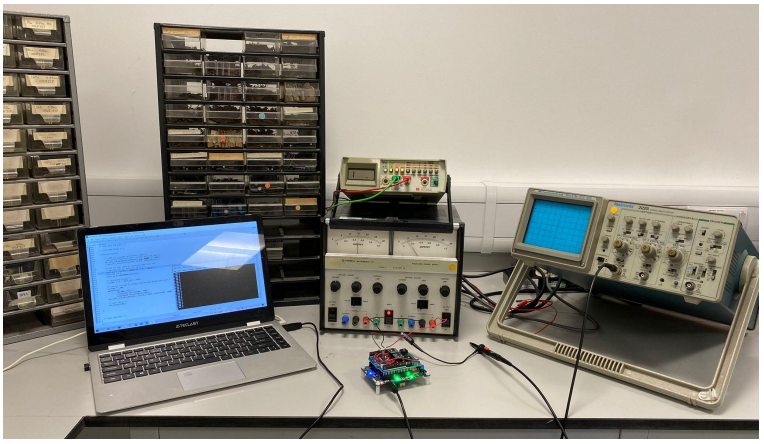


Fig. 18: Sound In - Real Image of the Hardware and Software Testing

4.3. Sound In - Integration with Another Interface

The integration of this SPI device, the sound in, with other SPI devices could be done in several ways. Two of them will be addressed, as they are the ones that, in practice, make most sense to be developed:

Integration with the SD Card (and Sound Out)

The integration between these two devices should exist in order to output a voice recording. The voltage data obtained from the MCP3002 ADC, needs to be processed like it was mencionan above (section 4.2.2.) and then passed to the SD Card for storage, before being routed to the Sound Out device. This consists of a higher level design.

Integration with the Display

In the pre-integration phase, everything was displayed on Tera Terminal. Now, in the integration phase, it is necessary to display the messages on the LCD Display. This is achieved by using the ST7789_WriteString(uint16_t x, uint16_t y, const char *str, FontDef font, uint16_t color, uint16_t bgcolor) function, mentioned in the display section below (section 5.2.2.), instead of the HAL_UART_Transmit

5. Individual Deliverables - Kevin Mancini

5.1. Gyroscope Sensor

A Gyroscope sensor is a component that is able to capture the angular acceleration of the three axes (X, Y and Z) of the main board where the sensor is soldered. Included in the component there are several pins, those are useful to communicate with it through the board and the code.

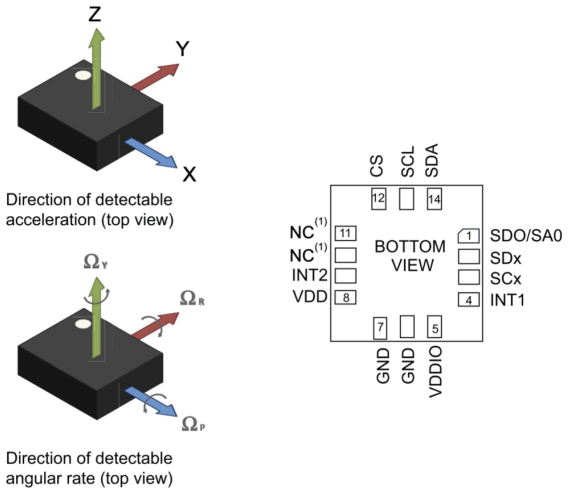


Fig 19: Gyroscope/Accelerometer pins and axes

The following sections analyze how the software is being structured giving importance to the functions used, their return value, the global variables included in the code and last but not least the testing code developed. It will be also shown how the gyroscope and the display are integrated in the project from a software perspective.

5.1.1. Gyroscope Sensor - Functions

The functions whose names start with BSP belong to the Gyroscope BSP library.

BSP_GYRO_Init

This function initializes the gyroscope and returns a *uint8_t* value that represents the status of the gyroscope. First, the function checks if the *WHO_AM_I* register's value corresponds to *0x6A*. If this first control is passed it sets the power mode, the output data rate, the axes and other parameters. After all this the sensor is being initialized and the correct return value is given back.

The possible return values are *GYRO_OK* and *GYRO_ERROR*, defined as following:

```
typedef enum
{
  GYRO_OK = 0,
  GYRO_ERROR = 1,
  GYRO_TIMEOUT = 2
}
GYRO_StatusTypeDef;
```

Software Code 12: Implementation of *GYRO_StatusTypeDef*

```
/** @defgroup STM32L475E_IOT01_GYROSCOPE_Private_Functions GYROSCOPE Private
Functions
  * @{
  */
/**
  * @brief  Initialize Gyroscope.
  * @retval GYRO_OK or GYRO_ERROR
  */
uint8_t BSP_GYRO_Init(void)
{
  uint8_t ret = GYRO_ERROR;
  uint16_t ctrl = 0x0000;
  GYRO_InitTypeDef LSM6DSL_InitStructure;

  if(Lsm6dslGyroDrv.ReadID() != LSM6DSL_ACC_GYRO_WHO_AM_I)
  {
    ret = GYRO_ERROR;
  }
  else
  {
    /* Initialize the gyroscope driver structure */
    GyroscopeDrv = &Lsm6dslGyroDrv;

    /* Configure Mems : data rate, power mode, full scale and axes */
    LSM6DSL_InitStructure.Power_Mode = 0;
    LSM6DSL_InitStructure.Output_DataRate = LSM6DSL_ODR_52Hz;
    LSM6DSL_InitStructure.Axes_Enable = 0;
    LSM6DSL_InitStructure.Band_Width = 0;
    LSM6DSL_InitStructure.BlockData_Update = LSM6DSL_BDU_BLOCK_UPDATE;
    LSM6DSL_InitStructure.Endianness = 0;
    LSM6DSL_InitStructure.Full_Scale = LSM6DSL_GYRO_FS_2000;

    /* Configure MEMS: data rate, full scale  */
    ctrl = (LSM6DSL_InitStructure.Full_Scale |
LSM6DSL_InitStructure.Output_DataRate);

    /* Configure MEMS: BDU and Auto-increment for multi read/write */
    ctrl |= ((LSM6DSL_InitStructure.BlockData_Update |
LSM6DSL_ACC_GYRO_IF_INC_ENABLED) << 8);

    /* Initialize component */
    GyroscopeDrv->Init(ctrl);

    ret = GYRO_OK;
  }

  return ret;
}
```

Software Code 13: Implementation of *BSP_GYRO_Init*

BSP_GYRO_DeInit

This simple function just deinitialize the gyroscope sensor without returning any value.

```
void BSP_GYRO_DeInit(void)
{
  /* DeInitialize the Gyroscope IO interfaces */
  if(GyroscopeDrv != NULL)
  {
    if(GyroscopeDrv->DeInit!= NULL)
    {
      GyroscopeDrv->DeInit();
    }
  }
}
```

Software Code 14: Implementation of *BSP_GYRO_DeInit*

The definition and implementation of the variable *GyroscopeDrv* is included in the following section *5.1.2 Global Variables*.

BSP_GYRO_LowPower

This function sets the gyroscope power mode. The parameter status states whether or not the power mode has to be activated. No return value is given.

```
/**
  * @brief  Set/Unset Gyroscope in low power mode.
  * @param  status 0 means disable Low Power Mode, otherwise Low Power Mode is
enabled
  */
void BSP_GYRO_LowPower(uint16_t status)
{
  /* Set/Unset component in low-power mode */
  if(GyroscopeDrv != NULL)
  {
    if(GyroscopeDrv->LowPower!= NULL)
    {
      GyroscopeDrv->LowPower(status);
    }
  }
}
```

Software Code 15: Implementation of *BSP_GYRO_LowPower*

BSP_GYRO_GetXYZ

This function gets the angular acceleration of all *X, Y* and *Z* axes by reading the gyroscope registers. These values are pasted in the given parameter *pfData* (array of three *floats*). No return value is given.

```
/**
  * @brief  Get XYZ angular acceleration from the Gyroscope.
  * @param  pfData: pointer on floating array
```

```
  */
void BSP_GYRO_GetXYZ(float* pfData)
{
  if(GyroscopeDrv != NULL)
  {
    if(GyroscopeDrv->GetXYZ!= NULL)
    {
      GyroscopeDrv->GetXYZ(pfData);
    }
  }
}
```

Software Code 16: Implementation of *BSP_GYRO_GetXYZ*

GYRO_Get_WHOAMI

This function returns the value of the *WHOAMI* register. It has been used for preliminary checks and test coding.

```
/**
  * @brief  Get the value of the WHO_AM_I register of the gyroscope sensor
  * @retval WHO_AM_I register's value
  */
uint8_t GYRO_Get_WHOAMI(void){
       uint8_t ID;
       HAL_I2C_Master_Transmit(&hi2c2, GYRO_WRITE_ADDRESS, &WhoAmI_Address, 1,
100);
       HAL_I2C_Master_Receive(&hi2c2, GYRO_READ_ADDRESS, &ID, 1, 100);
       return ID;
}
```

Software Code 17: Implementation of *BSP_GYRO_WHOAMI*

GYRO_PrintTerminal

This function just prints on the terminal a string message, this utility makes the code more clear and readable.

```
/**
  * @brief  Print on the terminal messages, used for debug only
  * @param  char array of the message
  */
void GYRO_PrintTerminal(char* msg){
       HAL_UART_Transmit(&huart1,( uint8_t * )msg,sizeof(msg),1000);
}
```

Software Code 17: Implementation of *GYRO_PrintTerminal*

GYRO_Get_Temp

This C function calculates the current temperature of the gyroscope sensor reading and combining the two registers: *OUT_TEMP_L* and *OUT_TEMP_H*. His return value is the sensor's temperature saved into a two bytes unsigned int variable.

```
/**
  * @brief  Get the value of the WHO_AM_I register
  * @param  NONE
  * @retval WHO_AM_I register's value
  */
uint16_t GYRO_Get_Temp(void){
	uint8_t cur_temp_l;
	uint8_t cur_temp_h;

	HAL_I2C_Master_Transmit(&hi2c2, GYRO_WRITE_ADDRESS, &OutTempL_Address, 1,
100);
	HAL_I2C_Master_Receive(&hi2c2, GYRO_READ_ADDRESS, &cur_temp_l, 1, 100);

	HAL_I2C_Master_Transmit(&hi2c2, GYRO_WRITE_ADDRESS, &OutTempL_Address, 1,
100);
	HAL_I2C_Master_Receive(&hi2c2, GYRO_READ_ADDRESS, &cur_temp_h, 1, 100);

	return ((cur_temp_h << 8) | cur_temp_l);
}
```

Software Code 18: Implementation of *GYRO_Get_Temp*

GYRO_Check_temp

This C function controls whether or not the current temperature is between the admitted limits. Those limits respectively MIN_TEMP and MAX_TEMP are macros and have been obtained from the sensor's datasheet.

Table 1. Device summary

Part number	Temp. range [°C]	Package	Packing
LSM6DSL	-40 to +85	LGA-14L (2.5x3x0.83mm)	Tray
LSM6DSLTR	-40 to +85		Tape & Reel

Table 3: Gyroscope temp. range

The return value is simply the result of the logical operation. The importance of this function is to assure that the gyroscope is working in ideal environmental conditions during the execution of the program.

```
/**
  * @brief  Check if the current temperature is between the admitted boundaries
  * @retval 1 if correct temperature, otherwise 0
  */
```

```
uint8_t GYRO_Check_Temp(void){
	current_temp = GYRO_Get_Temp();
	return current_temp>MIN_TEMP && current_temp<MAX_TEMP;
}
```

Software Code 19: Implementation of *GYRO_Check_Temp*

GYRO_Compare_Values

This C function checks the given user's values and the criteria boundaries. The parameters requested are: the type of criteria to consider and the user's input (floating array). The return value is just the result of the logical operations done, it is 1 if the control is passed and 0 otherwise.

```
/**
  * @brief  Compare the user's values  with the criteria of the task
  * @param  n indicates which criteria should be used, pfData user's values
  * @retval 1 CORRECT
  */
uint8_t GYRO_Compare_Values(uint8_t n, float *pfData){
	if (n==0){
		// X check
		return ( pfData[0]<X_MIN || pfData[0]>X_MAX ) && pfData[1]>Y_MIN &&
pfData[1]<X_MAX && pfData[2]>X_MIN && pfData[2]<X_MAX;
	} else if (n==1){
		// Y check
		return pfData[0]>X_MIN && pfData[0]<X_MAX && ( pfData[1]<Y_MIN ||
pfData[1]>X_MAX ) && pfData[2]>X_MIN && pfData[2]<X_MAX;
	} else if (n==2){
		// Z check
		return pfData[0]>X_MIN && pfData[0]<X_MAX && pfData[1]>Y_MIN &&
pfData[1]<X_MAX && ( pfData[2]<X_MIN || pfData[2]>X_MAX );
	} else {
		return 1;
	}
}
```

Software Code 20: Implementation of *GYRO_Compare_Values*

*One more function, called *GYRO_TASK*, includes gyroscope operations but it is only explained at the end of the display section*

5.1.2. Gyroscope Sensor - Global Variables

In this section will be illustrated which global variables have been used in the code. Here I tried to reduce the number of those trying to include inside functions that don't have to be necessarily shared and as defined those that have constant values. Follows a list of the global variables and an explanation that clarifies how they are used and by which functions.

```
uint8_t GYRO_WhoAmI_Address = 0x0F;
uint8_t GYRO_OutTempL_Address = 0x20;
uint8_t GYRO_OutTempH_Address = 0x21;
I2C_HandleTypeDef GYRO_hi2c2;
UART_HandleTypeDef GYRO_huart1;
int16_t GYRO_current_temp;
```

Software Code 21: Global variables

As can be seen on the code snippet some variables have been created to store register's addresses. This should normally be done using defines, however here I had the need to obtain pointers to those values and this can be done only by creating variables. Then a *I2C_HandleTypeDef* is used to read and write to gyroscope sensor's registers using *HAL_I2C_Master_Receive* and *HAL_I2C_Master_Transmit*. In addition, a *UART_HandleTypeDef* variable is used to write on the terminal for debug purposes only by using the function: *HAL_UART_Trasmit*. In conclusion, a *current_temp* variable is used to store the current temperature of the gyroscope sensor.

Several other values have been included in the code by using defines instead of variables, as already said. It follows a list of those defines.

```
#define GYRO_NUMBER_OF_ATTEMPTS 10
#define GYRO_NUMBER_OF_TASKS 6
#define GYRO_WHO_AM_I_ADDRESS 0x0F
#define GYRO_OUT_TEMP_L_ADDRESS 0x20
#define GYRO_OUT_TEMP_H_ADDRESS 0x21
#define GYRO_MAX_TEMP 85
#define GYRO_MIN_TEMP -30
#define GYRO_OUT_TEMP_H_ADDRESS 0x21
#define GYRO_WHO_AM_I_VALUE 0x6A
#define GYRO_WRITE_ADDRESS 0xD5
#define GYRO_READ_ADDRESS 0xD4
#define GYRO_TEST_OK 0
#define GYRO_TEST_ERROR 1
#define GYRO_X_MAX 40000
#define GYRO_X_MIN -40000
#define GYRO_Y_MAX 40000
#define GYRO_Y_MIN -40000
#define GYRO_Z_MAX 40000
#define GYRO_Z_MIN -40000
```

Software Code 22: Gyroscope Sensor - Defines

Another global variable has also been used. This is included in the BSP library and is listed below.

```
typedef struct
{
  void       (*Init)(uint16_t);
  void       (*DeInit)(void);
```

```
  uint8_t    (*ReadID)(void);
  void       (*Reset)(void);
  void       (*LowPower)(uint16_t);
  void       (*ConfigIT)(uint16_t);
  void       (*EnableIT)(uint8_t);
  void       (*DisableIT)(uint8_t);
  uint8_t    (*ITStatus)(uint16_t, uint16_t);
  void       (*ClearIT)(uint16_t, uint16_t);
  void       (*FilterConfig)(uint8_t);
  void       (*FilterCmd)(uint8_t);
  void       (*GetXYZ)(float *);
}GYRO_DrvTypeDef;

static GYRO_DrvTypeDef *GyroscopeDrv;
```

Software Code 23: BSP global variable and struct definition

5.1.3. Gyroscope Sensor - Testing

In this section are listed the test functions written to check the gyroscope.

```
/* TEST: Gyroscope WHOAMI check
 * RETURN: TEST_OK ( 0 ) or TEST_ERROR ( 1 )
 * */
uint8_t WHO_AM_I_TEST(void){
	return GYRO_Get_WHOAMI()==WHO_AM_I_VALUE?TEST_OK:TEST_ERROR;
}

/* TEST: Gyroscope Initialization test
 * RETURN: TEST_OK ( 0 ) or TEST_ERROR ( 1 )
 * */
uint8_t GYRO_INIT_TEST(void){
	return BSP_GYRO_Init();
}

/* TEST: READ-WRITE registers test
 * RETURN: TEST_OK ( 0 ) or TEST_ERROR ( 1 )
 * */
uint8_t READ_WRITE_TEST(void){
	uint8_t res1,res2,res3;

	HAL_I2C_Master_Transmit(&hi2c2, GYRO_WRITE_ADDRESS, &Test_Address1, 1,
100);
	HAL_I2C_Master_Receive(&hi2c2, GYRO_READ_ADDRESS, &res1, 1, 100);

	HAL_I2C_Master_Transmit(&hi2c2, GYRO_WRITE_ADDRESS, &Test_Address2, 1,
100);
	HAL_I2C_Master_Receive(&hi2c2, GYRO_READ_ADDRESS, &res2, 1, 100);

	HAL_I2C_Master_Transmit(&hi2c2, GYRO_WRITE_ADDRESS, &Test_Address3, 1,
100);
	HAL_I2C_Master_Receive(&hi2c2, GYRO_READ_ADDRESS, &res3, 1, 100);
	return ( res1==GYRO_TEST_EXPECTED_1 && res2==GYRO_TEST_EXPECTED_2 &&
res3==GYRO_TEST_EXPECTED_3 )?TEST_OK:TEST_ERROR;
}
```

```
/* TEST: Task test
 * RETURN: TEST_OK ( 0 ) or TEST_ERROR ( 1 )
 * */
uint8_t GYRO_TASK_TEST(void){
	uint8_t res = 0;

	float test_pfData_1[] = {0.0,0.0,0.0};
	float test_pfData_2[] = {40001.0,0.0,0.0};
	float test_pfData_3[] = {0.0,40001.0,0.0};
	float test_pfData_4[] = {0.0,0.0,40001.0};
	float test_pfData_5[] = {40001.0,40001.0,0.0};
	float test_pfData_6[] = {0.0,40001.0,40001.0};
	float test_pfData_7[] = {40001.0,0.0,40001.0};
	float test_pfData_8[] = {40001.0,40001.0,40001.0};
	float test_pfData_9[] = {-40001.0,0.0,0.0};
	float test_pfData_10[] = {0.0,-40001.0,0.0};
	float test_pfData_11[] = {0.0,0.0,-40001.0};
	float test_pfData_12[] = {-40001.0,-40001.0,0.0};
	float test_pfData_13[] = {0.0,-40001.0,-40001.0};
	float test_pfData_14[] = {-40001.0,0.0,-40001.0};
	float test_pfData_15[] = {-40001.0,-40001.0,-40001.0};

	res += GYRO_Compare_Values(1,test_pfData_1);
	res += !GYRO_Compare_Values(1,test_pfData_2);
	res += GYRO_Compare_Values(1,test_pfData_3);
	res += GYRO_Compare_Values(1,test_pfData_4);
	res += GYRO_Compare_Values(1,test_pfData_5);
	res += GYRO_Compare_Values(1,test_pfData_6);
	res += GYRO_Compare_Values(1,test_pfData_7);
	res += GYRO_Compare_Values(1,test_pfData_8);
	res += !GYRO_Compare_Values(1,test_pfData_9);
	res += GYRO_Compare_Values(1,test_pfData_10);
	res += GYRO_Compare_Values(1,test_pfData_11);
	res += GYRO_Compare_Values(1,test_pfData_12);
	res += GYRO_Compare_Values(1,test_pfData_13);
	res += GYRO_Compare_Values(1,test_pfData_14);
	res += GYRO_Compare_Values(1,test_pfData_15);

	res += GYRO_Compare_Values(2,test_pfData_1);
	res += GYRO_Compare_Values(2,test_pfData_2);
	res += !GYRO_Compare_Values(2,test_pfData_3);
	res += GYRO_Compare_Values(2,test_pfData_4);
	res += GYRO_Compare_Values(2,test_pfData_5);
	res += GYRO_Compare_Values(2,test_pfData_6);
	res += GYRO_Compare_Values(2,test_pfData_7);
	res += GYRO_Compare_Values(2,test_pfData_8);
	res += GYRO_Compare_Values(2,test_pfData_9);
	res += !GYRO_Compare_Values(2,test_pfData_10);
	res += GYRO_Compare_Values(2,test_pfData_11);
	res += GYRO_Compare_Values(2,test_pfData_12);
	res += GYRO_Compare_Values(2,test_pfData_13);
	res += GYRO_Compare_Values(2,test_pfData_14);
	res += GYRO_Compare_Values(2,test_pfData_15);

	res += GYRO_Compare_Values(3,test_pfData_1);
	res += GYRO_Compare_Values(3,test_pfData_2);
	res += GYRO_Compare_Values(3,test_pfData_3);
```

```
        res += !GYRO_Compare_Values(3,test_pfData_4);
        res += GYRO_Compare_Values(3,test_pfData_5);
        res += GYRO_Compare_Values(3,test_pfData_6);
        res += GYRO_Compare_Values(3,test_pfData_7);
        res += GYRO_Compare_Values(3,test_pfData_8);
        res += GYRO_Compare_Values(3,test_pfData_9);
        res += GYRO_Compare_Values(3,test_pfData_10);
        res += GYRO_Compare_Values(3,test_pfData_11);
        res += !GYRO_Compare_Values(3,test_pfData_12);
        res += GYRO_Compare_Values(3,test_pfData_13);
        res += GYRO_Compare_Values(3,test_pfData_14);
        res += GYRO_Compare_Values(3,test_pfData_15);

        return res;
}
```

Software Code 24: Gyroscope Sensor - Testing

Other test functions will be shown in the display section.

5.1.4. Gyroscope Sensor - Practical Example

Now there are shown screenshots and photos demonstrating the correctness of the previous code snippets.

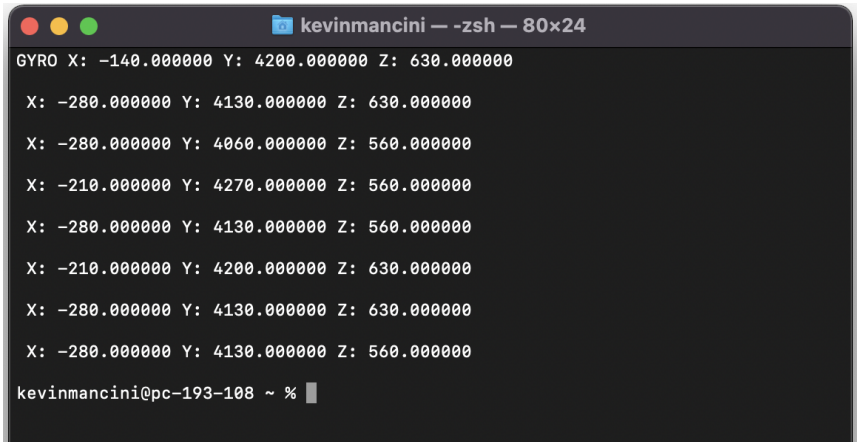


Fig. 20: Terminal with XYZ values

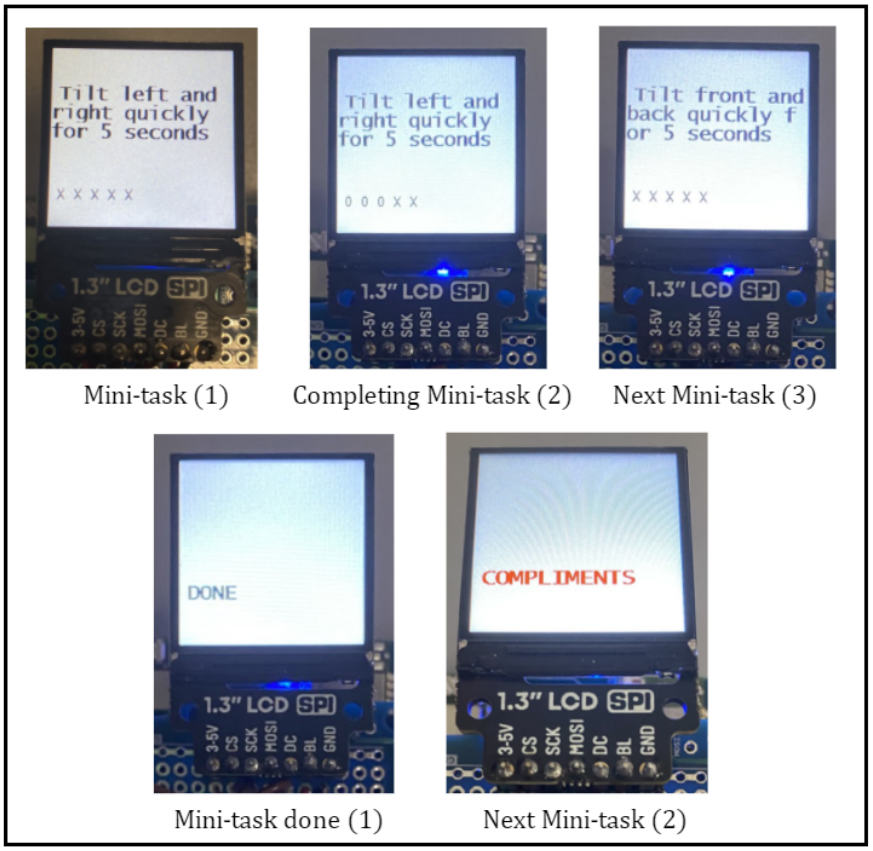


Fig. 21: Display - Tasks

5.2. Display Device

The following sections will introduce the hardware and software done to integrate the *ST7789V LCD* Display in the project.

5.2.1. Display Hardware

The *ST7789V LCD* Display is being installed on an Arduino Proto Shield. The device has 7 pins: *3-5 Volt*, *Chip Select*, *System Clock*, *MOSI*, *Data-Control*, *Backlight*, *Ground*. In the following section is discussed the way how these pins are connected to the shield and the main board and the reasons beside these decisions.

A figure of the schematic and the board made using EAGLE follow.

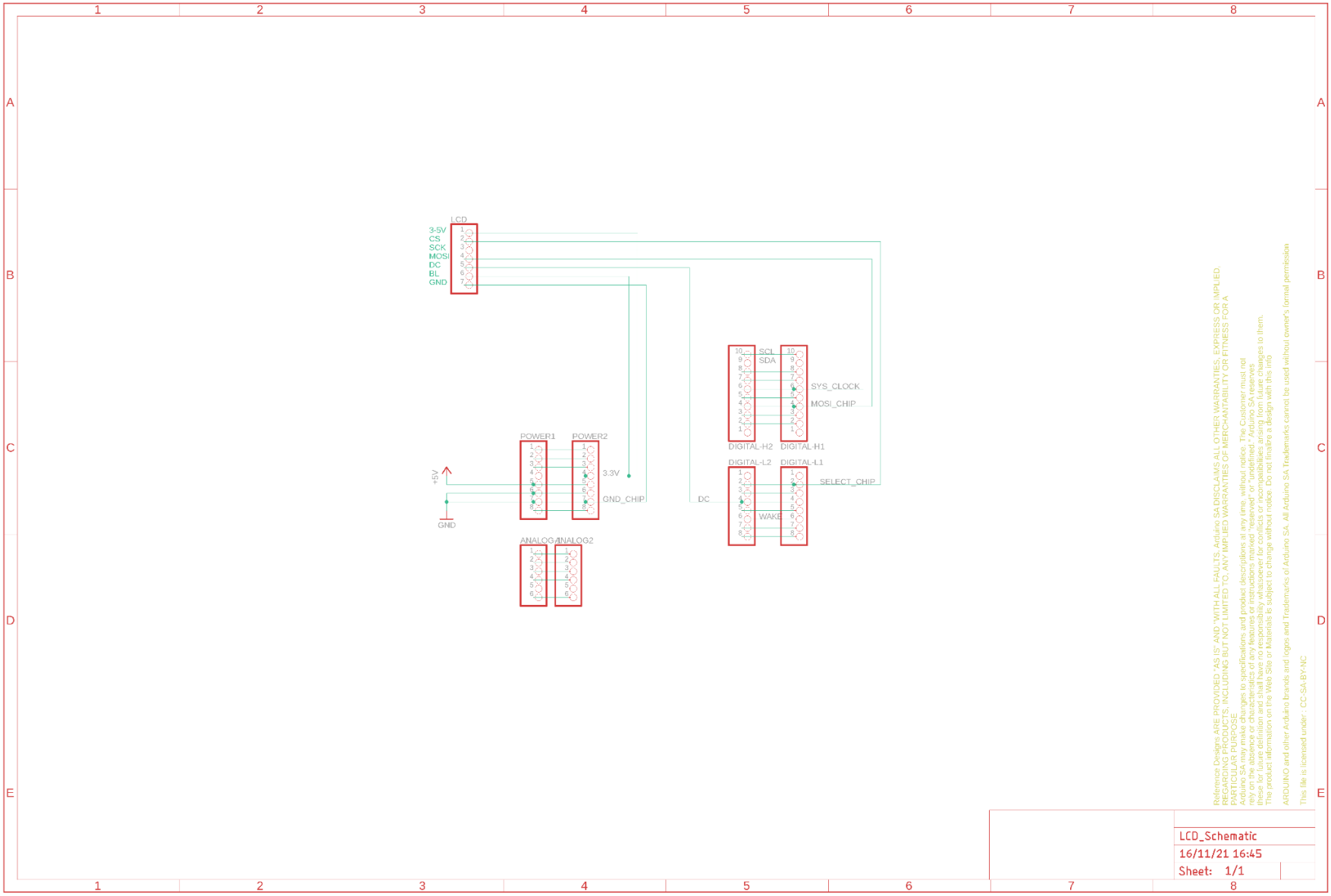


Fig. 22: Display Schematic

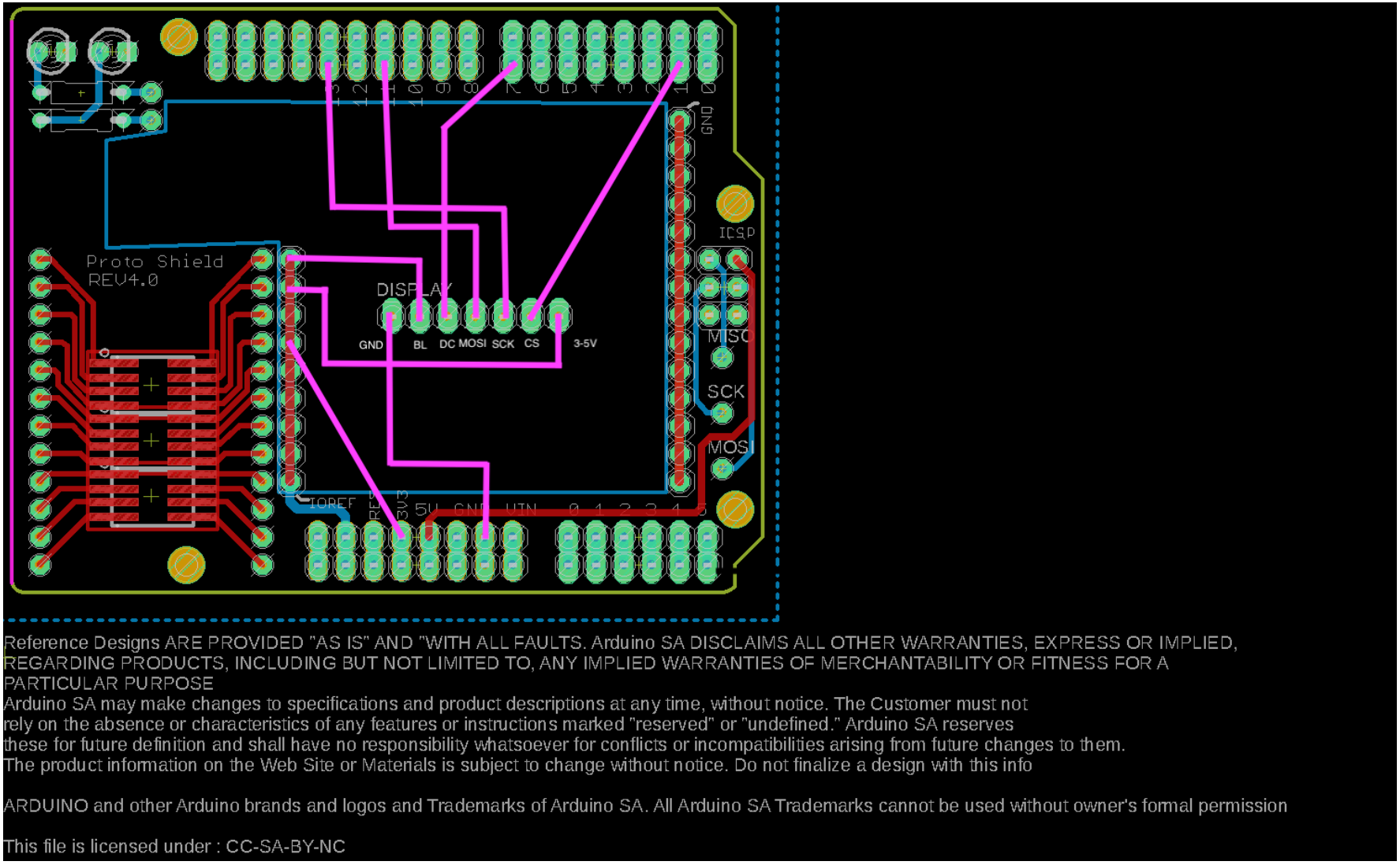


Fig. 23: Display Board

3-5V PIN: This pin has been connected to the 3.3v shield pin according to the sensor's datasheet voltage values.
CS PIN: This pin has been connected to DIGITAL 1, DIGITAL 1 is reserved only for this connection, this means that no other Aduino shield uses it.
SCK PIN: This pin has been connected to DIGITAL 13 so that he could read the clock from the main board.
MOSI PIN: This pin has been connected to DIGITAL 11 so that he could access the MOSI pin of the board.
DC PIN: This pin has been connected to DIGITAL 7, DIGITAL 7 is also reserved.
BL PIN: This pin has been connected to the 3.3v shield pin together with the 3-5v pin. Connecting the BL PIN directly to this pin we lose the possibility to turn the backlight off when desired. A possible alternative is to connect it to one of the DIGITAL pins in the shield. Doing this is possible to turn on or off the backlight setting the pin respectively to 1 or 0.
GND PIN: This pin has been connected to POWER 7 ground.

Now it is shown the real implementation showing the soldering and the wiring done.

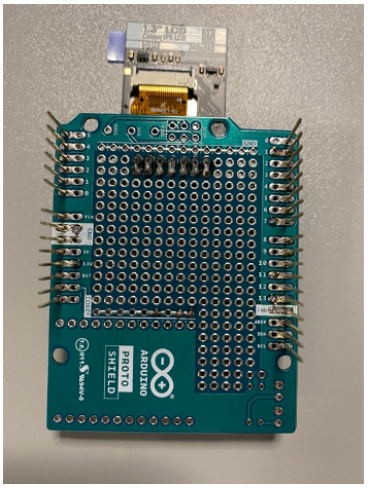


Fig. 24:Back of the shield

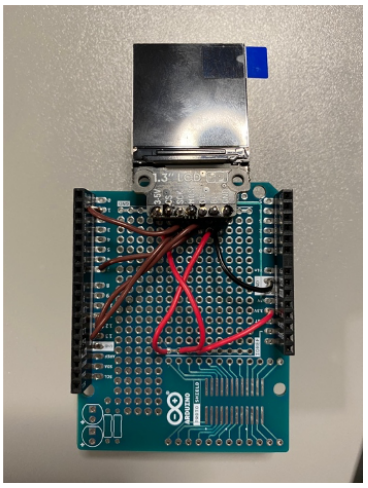


Fig. 25: Front of the shield

5.2.2. Display Software

To integrate this device I decided to use the *SPI1* bus and to do this I had to set the mode to *Full-Duplex master*, the *Data Size* to 8 bits and the *Prescaler* (for Baud Rate) at 64. Then to write into the display memory I used a 18bit/pixel format rapperented in the following diagram.

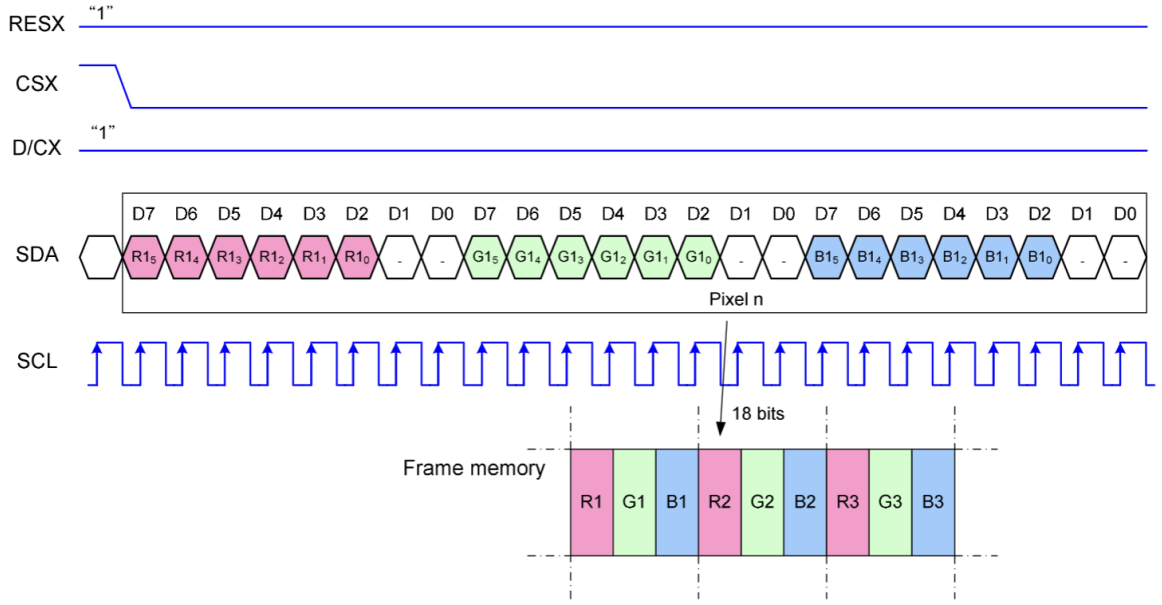


Fig 26: Write data for 18-bit/pixel (RGB-6-6-6-bit input)

The functions whose names start with ST7789 belong to the display library.

ST7789_Init

This function initializes the display and is part of the display library. This function must be called before any other operation on the display. No value is returned.

```
/**
 * @brief Initialize ST7789 controller
 * @param none
 * @return none
 */
void ST7789_Init(void)
{
       HAL_Delay(25);
    ST7789_RST_Clr();
    HAL_Delay(25);
    ST7789_RST_Set();
    HAL_Delay(50);

    ST7789_WriteCommand(ST7789_COLMOD);        //     Set color mode
    ST7789_WriteSmallData(ST7789_COLOR_MODE_16bit);
      ST7789_WriteCommand(0xB2);                //     Porch control
      {
            uint8_t data[] = {0x0C, 0x0C, 0x00, 0x33, 0x33};
            ST7789_WriteData(data, sizeof(data));
      }
      ST7789_SetRotation(ST7789_ROTATION);     //     MADCTL (Display Rotation)

      /* Internal LCD Voltage generator settings */
    ST7789_WriteCommand(0XB7);                  //     Gate Control
```

```
    ST7789_WriteSmallData(0x35);                //     Default value
    ST7789_WriteCommand(0xBB);                  //     VCOM setting
    ST7789_WriteSmallData(0x19);                //     0.725v (default 0.75v for
0x20)
    ST7789_WriteCommand(0xC0);                  //     LCMCTRL
    ST7789_WriteSmallData (0x2C);               //     Default value
    ST7789_WriteCommand (0xC2);                 //     VDV and VRH command
Enable
    ST7789_WriteSmallData (0x01);               //     Default value
    ST7789_WriteCommand (0xC3);                 //     VRH set
    ST7789_WriteSmallData (0x12);               //     +-4.45v (defalut +-4.1v for
0x0B)
    ST7789_WriteCommand (0xC4);                 //     VDV set
    ST7789_WriteSmallData (0x20);               //     Default value
    ST7789_WriteCommand (0xC6);                 //     Frame rate control in
normal mode
    ST7789_WriteSmallData (0x0F);               //     Default value (60HZ)
    ST7789_WriteCommand (0xD0);                 //     Power control
    ST7789_WriteSmallData (0xA4);               //     Default value
    ST7789_WriteSmallData (0xA1);               //     Default value
      /**************** Division line ****************/

      ST7789_WriteCommand(0xE0);
      {
            uint8_t data[] = {0xD0, 0x04, 0x0D, 0x11, 0x13, 0x2B, 0x3F, 0x54,
0x4C, 0x18, 0x0D, 0x0B, 0x1F, 0x23};
            ST7789_WriteData(data, sizeof(data));
      }

    ST7789_WriteCommand(0xE1);
      {
            uint8_t data[] = {0xD0, 0x04, 0x0C, 0x11, 0x13, 0x2C, 0x3F, 0x44,
0x51, 0x2F, 0x1F, 0x1F, 0x20, 0x23};
            ST7789_WriteData(data, sizeof(data));
      }
    ST7789_WriteCommand (ST7789_INVON);         //     Inversion ON
      ST7789_WriteCommand (ST7789_SLPOUT);      //     Out of sleep mode
      ST7789_WriteCommand (ST7789_NORON);       //     Normal Display on
      ST7789_WriteCommand (ST7789_DISPON);      //     Main screen turned on

      HAL_Delay(50);
      ST7789_Fill_Color(BLACK);                 //     Fill with Black.
}
```

Software Code 25: Implementation of *ST7789_Init*

ST7789_Fill_Color

This function sets all the display's pixels of a certain color. The parameter is the color. No value is returned.

```
/**
 * @brief Fill the DisplayWindow with single color
 * @param color -> color to Fill with
 * @return none
 */
```

```
void ST7789_Fill_Color(uint16_t color)
{
	uint16_t i, j;
	ST7789_SetAddressWindow(0, 0, ST7789_WIDTH - 1, ST7789_HEIGHT - 1);
	ST7789_Select();
	for (i = 0; i < ST7789_WIDTH; i++)
		for (j = 0; j < ST7789_HEIGHT; j++) {
			uint8_t data[] = {color >> 8, color & 0xFF};
			ST7789_WriteData(data, sizeof(data));
		}
	ST7789_UnSelect();
}
```

Software Code 26: Implementation of *ST7789_Fill_Color*

ST7789_WriteString

This function prints on the display a given array of char. It follows his implementation and the one of all the main functions called in it related to the display.

```
/**
 * @brief Write a string
 * @param  x&y -> cursor of the start point.
 * @param str -> string to write
 * @param font -> fontstyle of the string
 * @param color -> color of the string
 * @param bgcolor -> background color of the string
 * @return  none
 */
void ST7789_WriteString(uint16_t x, uint16_t y, const char *str, FontDef font,
uint16_t color, uint16_t bgcolor)
{
	ST7789_Select();
	while (*str) {
		if (x + font.width >= ST7789_WIDTH) {
			x = 0;
			y += font.height;
			if (y + font.height >= ST7789_HEIGHT) {
				break;
			}

			if (*str == ' ') {
				// skip spaces in the beginning of the new line
				str++;
				continue;
			}
		}
		ST7789_WriteChar(x, y, *str, font, color, bgcolor);
		x += font.width;
		str++;
	}
	ST7789_UnSelect();
}

/**
```

```
 * @brief Write a char
 * @param  x&y -> cursor of the start point.
 * @param ch -> char to write
 * @param font -> fontstyle of the string
 * @param color -> color of the char
 * @param bgcolor -> background color of the char
 * @return  none
 */
void ST7789_WriteChar(uint16_t x, uint16_t y, char ch, FontDef font, uint16_t
color, uint16_t bgcolor)
{
	uint32_t i, b, j;
	ST7789_Select();
	ST7789_SetAddressWindow(x, y, x + font.width - 1, y + font.height - 1);

	for (i = 0; i < font.height; i++) {
		b = font.data[(ch - 32) * font.height + i];
		for (j = 0; j < font.width; j++) {
			if ((b << j) & 0x8000) {
				uint8_t data[] = {color >> 8, color & 0xFF};
				ST7789_WriteData(data, sizeof(data));
			}
			else {
				uint8_t data[] = {bgcolor >> 8, bgcolor & 0xFF};
				ST7789_WriteData(data, sizeof(data));
			}
		}
	}
	ST7789_UnSelect();
}

/**
 * @brief Write data to ST7789 controller
 * @param buff -> pointer of data buffer
 * @param buff_size -> size of the data buffer
 * @return none
 */
static void ST7789_WriteData(uint8_t *buff, size_t buff_size)
{
	ST7789_Select();
	ST7789_DC_Set();

	// split data in small chunks because HAL can't send more than 64K at once

	while (buff_size > 0) {
		uint16_t chunk_size = buff_size > 65535 ? 65535 : buff_size;
		HAL_SPI_Transmit(&ST7789_SPI_PORT, buff, chunk_size, HAL_MAX_DELAY);
		buff += chunk_size;
		buff_size -= chunk_size;
	}

	ST7789_UnSelect();
}
```

Software Code 27: Implementation of *ST7789_WriteString*

ST7789_Test

This is a test function that can be run after the initialization of the display. It tests several operations and it shows graphically if the device is working properly.

```
/**
 * @brief A Simple test function for ST7789
 * @param  none
 * @return  none
 */
void ST7789_Test(void)
{
       ST7789_Fill_Color(WHITE);
       HAL_Delay(1000);
       ST7789_WriteString(10, 20, "Speed Test", Font_11x18, RED, WHITE);
       HAL_Delay(1000);
       ST7789_Fill_Color(CYAN);
    HAL_Delay(500);
       ST7789_Fill_Color(RED);
    HAL_Delay(500);
       ST7789_Fill_Color(BLUE);
    HAL_Delay(500);
       ST7789_Fill_Color(GREEN);
    HAL_Delay(500);
       ST7789_Fill_Color(YELLOW);
    HAL_Delay(500);
       ST7789_Fill_Color(BROWN);
    HAL_Delay(500);
       ST7789_Fill_Color(DARKBLUE);
    HAL_Delay(500);
       ST7789_Fill_Color(MAGENTA);
    HAL_Delay(500);
       ST7789_Fill_Color(LIGHTGREEN);
    HAL_Delay(500);
       ST7789_Fill_Color(LGRAY);
    HAL_Delay(500);
       ST7789_Fill_Color(LBBLUE);
    HAL_Delay(500);
       ST7789_Fill_Color(WHITE);
       HAL_Delay(500);

       ST7789_WriteString(10, 10, "Font test.", Font_16x26, GBLUE, WHITE);
       ST7789_WriteString(10, 50, "Hello Steve!", Font_7x10, RED, WHITE);
       ST7789_WriteString(10, 75, "Hello Steve!", Font_11x18, YELLOW, WHITE);
       ST7789_WriteString(10, 100, "Hello Steve!", Font_16x26, MAGENTA, WHITE);
       HAL_Delay(1000);

       ST7789_Fill_Color(RED);
       ST7789_WriteString(10, 10, "Rect./Line.", Font_11x18, YELLOW, RED);
       ST7789_DrawRectangle(30, 30, 100, 100, WHITE);
       HAL_Delay(1000);

       ST7789_Fill_Color(RED);
       ST7789_WriteString(10, 10, "Filled Rect.", Font_11x18, YELLOW, RED);
       ST7789_DrawFilledRectangle(30, 30, 50, 50, WHITE);
       HAL_Delay(1000);
```

```
	ST7789_Fill_Color(RED);
	ST7789_WriteString(10, 10, "Circle.", Font_11x18, YELLOW, RED);
	ST7789_DrawCircle(60, 60, 25, WHITE);
	HAL_Delay(1000);

	ST7789_Fill_Color(RED);
	ST7789_WriteString(10, 10, "Filled Cir.", Font_11x18, YELLOW, RED);
	ST7789_DrawFilledCircle(60, 60, 25, WHITE);
	HAL_Delay(1000);

	ST7789_Fill_Color(RED);
	ST7789_WriteString(10, 10, "Triangle", Font_11x18, YELLOW, RED);
	ST7789_DrawTriangle(30, 30, 30, 70, 60, 40, WHITE);
	HAL_Delay(1000);

	ST7789_Fill_Color(RED);
	ST7789_WriteString(10, 10, "Filled Tri", Font_11x18, YELLOW, RED);
	ST7789_DrawFilledTriangle(30, 30, 30, 70, 60, 40, WHITE);
	HAL_Delay(1000);

	//     If FLASH cannot storage anymore datas, please delete codes below.
	ST7789_Fill_Color(WHITE);
	ST7789_DrawImage(0, 0, 128, 128, (uint16_t *)saber);
	HAL_Delay(3000);
}
```

Software Code 28: Implementation of *ST7789_Test*

GYRO_Task

This function starts the gyroscope task, all the sub-tasks and calls some of the functions previously illustrated. Then prints on the display the user's instructions and the current progress of the task. The return value is 0 if the task is passed, 1 if failed.

```
/**
  * @brief  Run the gyroscope task
  * @retval 0 if PASSED, otherwise 1 FAILED
  */
uint8_t GYRO_Task(void){
	// START TASK
	//char str_tmp[100] = ""; // Formatted message to display the value
	char points0[] = "X X X X X ";
	char points1[] = "O X X X X ";
	char points2[] = "O O X X X ";
	char points3[] = "O O O X X ";
	char points4[] = "O O O O X ";
	char points5[] = "O O O O O ";
	// Init
	HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_RESET);
	ST7789_Init();
	uint8_t GYRO_init = BSP_GYRO_Init();
	float *pfData = malloc(3*sizeof(float));
	float *gyro_value = malloc(3*sizeof(float));
	HAL_Delay(1000);
	BSP_GYRO_GetXYZ(gyro_value);
```

```
	int p = 0;
	ST7789_Fill_Color(WHITE);
	if (GYRO_init == 0)
	while(1){
		// MINI-TASK
		for(int n = 0; n<3; n++){
			if(n==0)
				ST7789_WriteString(10, 50, "Tilt left and right
quickly for 5 seconds", Font_16x26, 100, WHITE);
			else if(n==1)
				ST7789_WriteString(10, 50, "Tilt front and back
quickly for 5 seconds", Font_16x26, 100, WHITE);
			else if(n==2)
				ST7789_WriteString(10, 50, "Tilt clockwise and
opposite quickly for 5 seconds", Font_16x26, 100, WHITE);
			while(p<5){
				if (p==0)
					ST7789_WriteString(10, 190, points0, Font_11x18,
100, WHITE);
				if (p==1)
					ST7789_WriteString(10, 190, points1, Font_11x18,
100, WHITE);
				if (p==2)
					ST7789_WriteString(10, 190, points2, Font_11x18,
100, WHITE);
				if (p==3)
					ST7789_WriteString(10, 190, points3, Font_11x18,
100, WHITE);
				if (p==4)
					ST7789_WriteString(10, 190, points4, Font_11x18,
100, WHITE);
				if (p==5)
					ST7789_WriteString(10, 190, points5, Font_11x18,
100, WHITE);
				HAL_Delay(1000);
				if (GYRO_Check_temp())
					BSP_GYRO_GetXYZ(pfData);
				if (GYRO_Compare_Values(n,pfData))
					p++;
			}
			ST7789_Fill_Color(WHITE);
			ST7789_WriteString(10, 160, "DONE", Font_16x26, 100, WHITE);
			HAL_Delay(1000);
			ST7789_Fill_Color(WHITE);
			p = 0;
		}
		ST7789_Fill_Color(WHITE);
		ST7789_WriteString(10, 160, "COMPLIMENTS", Font_16x26, RED, WHITE);
		HAL_Delay(1000);
		ST7789_Fill_Color(WHITE);
		return 0;
	}
	HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_SET);
	return 1;
}
```

Software Code 29: Implementation of *GYRO_Task*

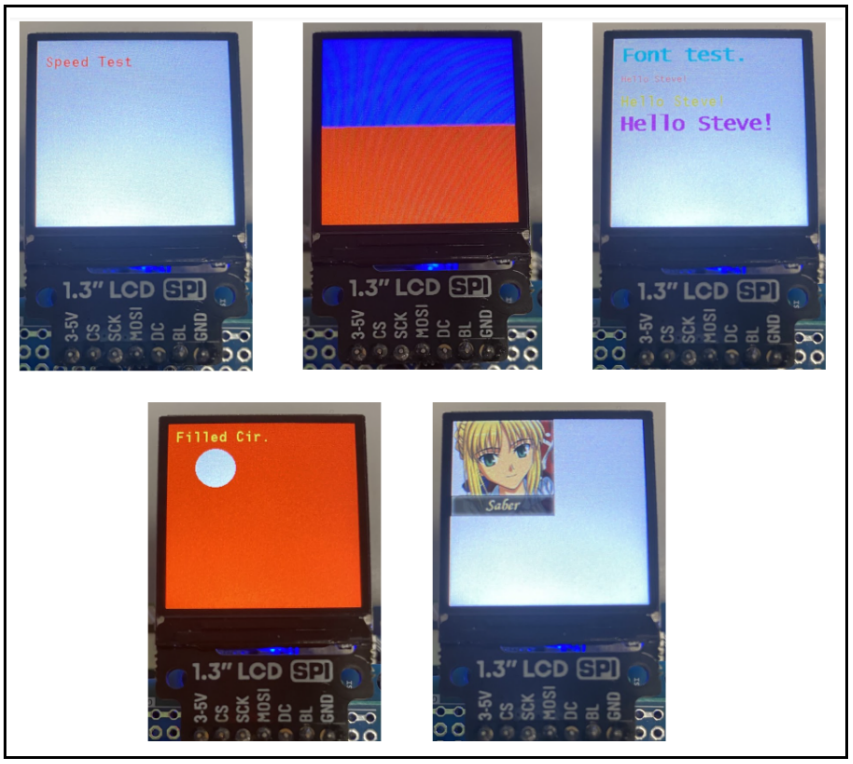


Fig 27: *ST7789_Test* photos

5.3. Display - Integration with Another Interface

The display has been easily integrated with all the other interfaces used by the other members of the group. In fact, using the functions previously illustrated is simple to write datas into the display. This functionality has been used in all the tasks of the puzzle game.

6. Individual Deliverables - Rostislav Sorokin

6.1. Accelerometer Sensor

The B-L475E-IOT01A board has an integrated accelerometer sensor device LSM6DSL which has been addressed to by the I2C communication protocol. The setting up of parameters of the sensor and then the reading of the sensor's three measurement values representing x, y, z axis is commenced via HAL I2C functions. Due to the fact that the ultimate decimal values of acceleration measurements can be negative depending on how the device is tilted, the values which are read from the x,y,z accelerations sensor registers are split into 8 bit MSB and LSB parts. Therefore, specific concatenation and conversion functions are needed in order to get the proper acceleration values currently represented in milli-g format for sensitivity purposes. Once the data for acceleration is available, the GPIO extender device which runs on SPI protocol reacts to these

parameters in a specific way. The metaconcept of the usage of the accelerometer is to provide the user experience of a game when the terminal or LCD display requests to tilt the board by 45 degrees angle and gives 5 seconds to achieve this. If the algorithm senses that a specific side of the board is tilted accordingly and the accelerometer values fall within a specific range, then the user "wins" the game, otherwise the game is lost.

6.1.1. Accelerometer Sensor - Functions

The functions of the accelerometer are addressed from the "main_acc.c" file which directs the compiler to the "functions.c" file (figure x10).

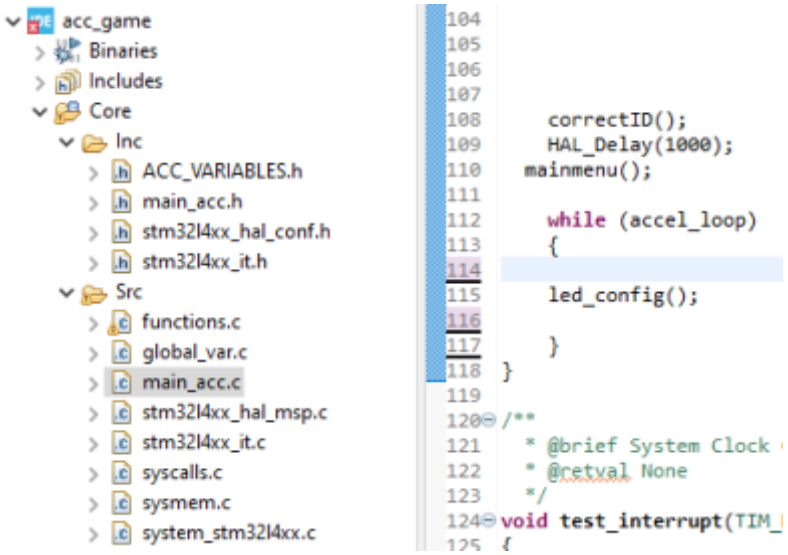


Fig x10: Structure of the Accelerometer Game Files

The most important function which manages the game is the "mainmenu()" function. It has to be noted that all the functions for the accelerometer are situated in the "functions.c" file. The function utilises global variables applicable to this game module only.

The "main_acc.c" source file calls the accelerometer initialisation function as seen in the software code snippet x1 below. The function "correctID()" is called in line 16 and initializes the accelerometer. Then before the main menu is activated, the initialization result stays on the screen as the only text entry for 1 second (line 17 of the software code snippet x2 below).

```
1  int main_acc(void)
2        {
3         HAL_Init();
4         SystemClock_Config();
5         MX_GPIO_Init();
6         MX_DFSDM1_Init();
7         MX_I2C2_Init();
8         MX_QUADSPI_Init();
```

```
9         MX_SPI3_Init();
10        MX_USART1_UART_Init();
11        MX_USART3_UART_Init();
12        MX_USB_OTG_FS_PCD_Init();
13        MX_SPI1_Init();
14        MX_TIM2_Init();
15       HAL_TIM_Base_Start_IT(&htim2);
16       correctID();
17             HAL_Delay(1000);
18        mainmenu();
19             while (accel_loop)
20             {
21              led_config();
22             }
23       }
```

Software code x1: Main Accelerometer Module Function

```
1  void mainmenu(void){
2  int i = rand ()%(2-0+1);// random number representing each side
3  HAL_UART_Transmit(&huart1,"\033[2J\033[1;1H",
4  strlen("\033[2J\033[1;1H"), 100);
5  initialize_extender();//testing function
6  HAL_UART_Transmit(&huart1, msg_acc, strlen(msg_acc), 100);
7  HAL_UART_Transmit(&huart1,"\n\rchecking accelerator. press c to play
8  a game\n\r"
9                "(press t to sTop the current game)\n\n\r",
10 strlen("\n\rchecking accelerator. press c to play a"
11           " game\n\r(press t to sTop the current
12 game)\nn\n\r"),200);
13 while (mainmenuloop){
14 HAL_UART_Receive (&huart1, UART1_rxBuffer, 12, 100);/*scan for a
15 pressed keyboard button*/
16 if (UART1_rxBuffer[0]=='c'){//if keyboard button "c" is pressed, then
17 play
18 mainmenuloop=0;
19 led_config(i);
20 }
21 else if (UART1_rxBuffer[0]=='t'){/*if keyboard button "t" is pressed,
22 then go to main menu of the project*/
23 main();
24 }
25 }
26 }
```

Software code x2: Function for the Accelerometer Game Menu

The comments of the code of the function “mainmenu()” explain each step performed by the function. The code in line 3 clears the terminal screen. Then the function of the SPI device initialisation is called (line 5). The most common practice used in this and other functions is establishing “while” loop in order to keep monitoring keyboard for pressing a particular button, as can be seen in line 13. For this purposes the character buffer and UART receiving function are used.

Second most important function which fully uses the accelerometer usage and game process is called “led_config(i)” (software code snippet 3); an integer parameter “i” is passed to it, as the board has three sides which can be tilted, therefore the compiler randomly chooses which side should be tilted by 45 degrees as can be seen in the above snippet on the line 2. In the below code on line 12 the value of the “mainmenuloop” variable is set back to 1. This is done in order to ensure the “while” loop is activated when the game returns to the main menu of the accelerator game module so that the keyboard entry can be scanned again if a user wants to activate the game again. This function also activates the led lights in a specific flashing order. The function used for activating the LED lights is “sendDataSPI()”, and it uses the specific “GPPU” register of the GPIO Extender device to light up one light at a time. In order for the accelerator to be activated, produce data and this data to be printed in a terminal, a function “play_acc()” is called. Lastly, the function checks whether the letter “t” is pressed anytime, and then it assesses whether the required board side has been tilted at about 45 degrees - it is done within the range of 450-550 milli-g. (g-force, gravitational force).

```
1  void led_config(int i){
2  int t=1;
3  if (i==0){
4        sides_acc=(int)proba1;
5  }
6  else if (i==1){
7        sides_acc=(int)proba2;
8  }
9  else if (i==2){
10       sides_acc=(int)proba3;
11 }
12       mainmenuloop=1;
13       sprintf(msg_acc,"\rlet's play a guessing game\n\rtilt the
14 board's side by 45 degrees\n\ryou have %d ms to do it \n\r",seconds);
15       HAL_UART_Transmit(&huart1,msg_acc,strlen(msg_acc), 100);
16       seconds=seconds-200;
17       sendDataSPI(GPPU, led_counter);
18       led_counter=led_counter/2;
19       if (led_counter<0x01){
20             led_counter=0x80;
21       }
22       play_acc();
23       HAL_UART_Receive (&huart1, UART1_rxBuffer, 12, 100);
24 HAL_UART_Transmit(&huart1,"\033[F\033[F\033[F",strlen("\033[F\033[F\03
25 3[F"),100);
26 if (UART1_rxBuffer[0]=='t'){
27        UART1_rxBuffer[0]="";
28        seconds=5000;
29        main_acc();
30 }
31 else if (seconds==0 && (450>sides_acc || sides_acc>550)){
32              youlost();
33 }
34 else if (seconds==0 && (450<sides_acc && sides_acc<550)){
35 youwon();
36 }
37 }
```

Software code x3: Function for Playing the Game and Configuring Leds

On a side note, the game is given 5000 ms to be completed. Figure x111 shows the text for the main game menu. The cursor is moved three positions down so that the active game part text does not overlap with the first three lines of the game menu text.

Fig x111: The Text for the Main Menu

In order for the accelerometer to produce the required data, the "play_acc()" function is called (Software code snippet 4).

```
1  void play_acc(void) {
2  acc_reg();
3  HAL_UART_Receive( & huart1, UART1_rxBuffer, 12, 100);
4  HAL_UART_Transmit( & huart1, msg_acc, strlen(msg_acc), 100);
```

Software code x4: Main Function for Reading Accelerometer Registers.

This function calls three more functions via "acc_reg()" function. After the registers are activated and read, the signal is transmitted to a terminal. The software code snippet 5 below shows the three functions responsible for the reading of the accelerometer. Firstly, with the help of two variables b_acc= 0x10 (accessing the control register) and the a_acc = 0xB0 (setting low power high performance control parameters) the accelerometer is activated and up and running.

Then the 8 bit MSB and LSB registers for each acceleration axis are read, and since they represent two's complement value, they are joined together and the final value is multiplied by the specific coefficient. Ultimately, the "sprintf" command is used in order to get the decimal numbers of the milli-g value, of the g-force acceleration scalar. Once the function is concluded, the x, y, z measurements results are printed out as it can be seen in the figures x13 and x14.

```
1  void acc_reg(void){
2         HAL_I2C_Mem_Write(&hi2c2, 0xD5, b_acc, 1, &a_acc, 1, 100);
3         readregisters();
4         twoscomplement();
5         writevalue();
6  }
7  void readregisters(void)
8  {
```

```
HAL_I2C_Mem_Read(&hi2c2, 0xD4, x_axis_l, 1, &dataxl, 1, 100);
HAL_I2C_Mem_Read(&hi2c2, 0xD4, x_axis_h, 1, &dataxh, 1, 100);
HAL_I2C_Mem_Read(&hi2c2, 0xD4, y_axis_l, 1, &datayl, 1, 100);
HAL_I2C_Mem_Read(&hi2c2, 0xD4, y_axis_h, 1, &datayh, 1, 100);
HAL_I2C_Mem_Read(&hi2c2, 0xD4, z_axis_l, 1, &datazl, 1, 100);
HAL_I2C_Mem_Read(&hi2c2, 0xD4, z_axis_h, 1, &datazh, 1, 100);
}
void twoscomplement (void){
proba1=(int16_t)(dataxh<<8 | dataxl)*0.061;
proba2=(int16_t)(datayh<<8 | datayl)*0.061;
proba3=(int16_t)(datazh<<8 | datazl)*0.061;
}
void writevalue(void)
{
sprintf(msg_acc, "x, y, z acceleration %d %d %d ", proba1, proba2,
proba3);
}
```

Software Code x5: Functions Responsible for Reading Data from the Accelerometer

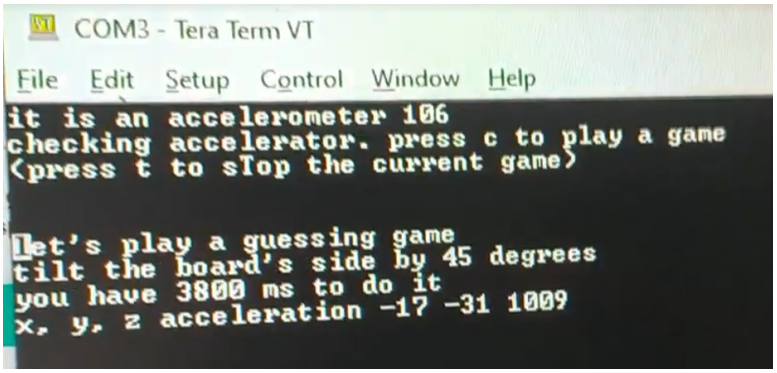


Fig x13: Accelerometer Game in Progress

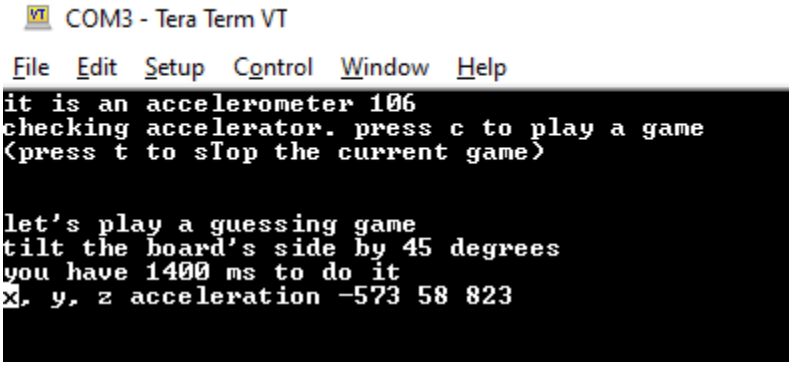


Fig x14: SAccelerometer Game in Progress when the Board is Tilted

6.1.2. Accelerometer Sensor - Global Variables

In order for all the functions to work properly, a header file was created "ACC_VARIABLES.h" and the source code file "global_var.c" was established. They both

work together declaring almost all the variables needed to run the functions for the accelerometer game and providing default values for these variables.

Software code snippet 5a shows these declared variables with relevant comments. Some variables are defined as constants, mainly registers, the rest are declared as externals, since their values will change depending on the data provided by the accelerometer and the sequential data values sent to LED lights.

```
1   #ifndef INC_ACC_VARIABLES_H_
2   #define INC_ACC_VARIABLES_H_
3   extern uint8_t IODIR;//GPIO Expander register
4   #define IPOL 0x01//GPIO Expander register
5   #define GPINTEN 0x02//GPIO Expander register
6   #define DEFVAL 0x03//GPIO Expander register
7   #define INTCON 0x04//GPIO Expander register
8   #define IOCON 0x05//GPIO Expander register
9   extern uint8_t GPPU;/*GPIO Expander register dealing with pull-up
10  internal resistors*/
11  #define INTF 0x07//GPIO Expander register
12  extern uint8_t INTCAP;//GPIO Expander register
13  extern uint8_t GPIO;//GPIO Expander register
14  extern uint8_t OPCODEW;/*operational address for writing into the
15  MCP23S09 GPIO Expander register*/
16  extern uint8_t OPCODER;/*operational address for reading from the
17  MCP23S09 GPIO Expander register*/
18  extern char UART1_rxBuffer[12];/*global character buffer for scanning
19  for pressed keyboard buttons*/
20  #define SPI_TRANSFER_TIMEOUT 1000
21  extern int16_t proba1;/*variable for two's complement function
22  conversion, x;*/
23  extern int16_t proba3;/*variable for two's complement function
24  conversion, z;*/
25  extern int16_t proba2;/*variable for two's complement function
26  conversion, y;*/
27  extern char msg_acc[35];/*global character buffer for registering,
28  storing and displaying text for menu*/
29  extern int accel_loop;/*"while" loop logical variable for running
30  loops on dynamic menu parts*/
31  extern int mainmenuloop;/"while" loop logical variable for
32  extern uint8_t leds_binary;
33  int sides_acc;
34  extern uint8_t templsb;//temp register low optional
35  extern uint8_t tempmsb;//temp register high optional
36  extern uint8_t templ;//buffer for lsb temperature reading optional
37  extern uint8_t temph;//buffer for msb temperature reading optional
38  extern int16_t temp;//temp register value after twos complement
39  extern uint8_t x_axis_l;//out x l axis lsb
40  extern uint8_t x_axis_h;// out x h axis msb
41  extern uint8_t y_axis_l;//out y l axis lsb
42  extern uint8_t y_axis_h;// out y h axis msb
43  extern uint8_t z_axis_l;//out z l axis lsb
44  extern uint8_t z_axis_h;// out z h axis msb
45  extern uint8_t WHO_AM_I;//who am i register
46  extern uint8_t a_acc;/*low power high performance control parameters*/
47  extern uint8_t b_acc;//control register
48  extern uint8_t dataxl;//x axis lsb
49  extern uint8_t dataxh;//x axis msb
50  extern uint8_t datayl;//y axis lsb
```

```
51  extern uint8_t datayh;//y axis msb
52  extern uint8_t datazl;//z axis lsb
53  extern uint8_t datazh;//z axis msb
54  extern uint8_t read_add;//I2C reading command register
55  extern uint8_t write_add;//I2C writing command register
56  #endif /* INC_ACC_VARIABLES_H_ */
57
```

Software code 5a: Global Variables used in Accelerometer and Expander Module

As mentioned before, some variables have values assigned to them in advance in the “global_var.c” file as seen in the software code snippet 5b.

```
1   #include "stdio.h"
2   IODIR = 0x00;
3   GPPU = 0x06;
4   INTCAP = 0x08;
5   GPIO = 0x09;
6   OPCODEW = 0x40;
7   OPCODER = 0x41;
8   accel_loop = 1;
9   mainmenuloop=1;
10  leds_binary=0x00;
11  UART1_rxBuffer[12]={0};
12  msg_acc[35]={0};
13  proba1=0;
14  proba2=0;
15  proba3=0;
16  sides_acc=0;
17  read_add = 0xD5;
18  write_add = 0xD4;
19  a_acc = 0xB0;
20  b_acc= 0x10;
21  x_axis_l = 0x28;//out x l axis
22  x_axis_h = 0x29;// out x h axis
23  y_axis_l = 0x2A;//out y l axis
24  y_axis_h = 0x2B;// out y h axis
25  z_axis_l = 0x2C;//out z l axis
26  z_axis_h = 0x2D;// out z h axis
27  templsb=0x20;
28  tempmsb=0x21;
29  dataxl=0;//x axis lsb
30  dataxh=0;//x axis msb
31  datayl=0;//y axis lsb
32  datayh=0;//y axis msb
33  datazl=0;//z axis lsb
34  datazh=0;//z axis msb
35
36
```

Software code 5b: Source Code File configuring Global Variables.

6.1.3. Accelerometer Sensor - Testing

The accelerometer function “correctID” is simple, but extremely important and efficient, as it shows the contents of any register of the accelerometer. In this example shown in the software code snippet x the “write” and “read” slave addresses are used as from the device’s datasheet (Figure x11). The software code snippet x_ is shown below.

```
1   void correctID (void)
2   {
3   uint8_t WHO_AM_I = 0x0F;
4   uint8_t data3;
5   HAL_UART_Transmit(&huart1,"\033[2J\033[1;1H",
6   strlen("\033[2J\033[1;1H"), 100);
7   HAL_I2C_Master_Transmit(&hi2c2, 0xD5, &WHO_AM_I, 1,100);
8   HAL_I2C_Master_Receive(&hi2c2, 0xD4, &data3,1, 100);
9   if (data3==106)
10  {
11        sprintf(msg_acc,"it is an accelerometer %d",data3);
12  HAL_UART_Transmit(&huart1, msg_acc, strlen(msg_acc), 1000);
13  }
14  else if (data3!=106)
15  {
16        sprintf(msg_acc,"it is NOT an accelerometer %d",3);
17  HAL_UART_Transmit(&huart1, msg_acc, strlen(msg_acc), 1000);
18  }
19  }
```

Software code x_: Function for Identifying Registers Content of the Accelerometer

Command	SAD[6:1]	SAD[0] = SA0	R/W	SAD+R/W
Read	110101	0	1	11010101 (D5h)
Write	110101	0	0	11010100 (D4h)

Table x: Slave Address for Writing and Reading the Accelerometer

In this particular instance a 0x0F register was checked by the functions and result produced written on a terminal (figure x12).

Fig x12: Correctly Identified Accelerometer showing decimal number of it’s ID

The number 106 represents the LSM6DSL accelerometer’s decimal form of ID converted from the binary form 01101010.

Any other registers can be checked or rewritten by simply transmitting/receiving appropriate LSM6DSL register map’s register address via HAL interface functions.

6.1.4. Accelerometer Sensor - Practical Example

There are four functions responsible for providing the outcome for the game after 5000ms (software code snippet 6)

```
void youlost(void){
      int t=1;
      UART1_rxBuffer[0]="";
                  seconds=5000;
                  HAL_UART_Transmit(&huart1,"\033[F"
"╔═══════════════════════╗\n\r"
"║       YOU LOST        ║\n\r"
"╚═══════════════════════╝\n\r",
strlen("\033[F"
"╔═══════════════════════╗\n\r"
"║       YOU LOST        ║\n\r"
"╚═══════════════════════╝\n\r"),200);
while (t==1){
      HAL_UART_Receive (&huart1, UART1_rxBuffer, 12, 100);
      if    (UART1_rxBuffer[0]=='c'){
      UART1_rxBuffer[0]="";
      t=0;
      main_acc();
            }
loss_led();
}
}

void youwon(void){
int t=1;
UART1_rxBuffer[0]="";
seconds=5000;
HAL_UART_Transmit(&huart1,"\033[F"
"╔═══════════════════════╗\n\r"
"║       YOU WON!        ║\n\r"
"╚═══════════════════════╝\n\r",
strlen("\033[F"
"╔═══════════════════════╗\n\r"
"║       YOU WON!        ║\n\r"
"╚═══════════════════════╝\n\r"),100);
      while (t==1){
      HAL_UART_Receive (&huart1, UART1_rxBuffer, 12, 100);
      if    (UART1_rxBuffer[0]=='c'){
      UART1_rxBuffer[0]="";
      t=0;
      main_acc();
            }
      won_led();
      }
}
void won_led(void){
            sendDataSPI(GPPU, 0xFF);
            HAL_Delay(50);
            sendDataSPI(GPPU, 0x00);
            HAL_Delay(50);
}
```

```
54  void loss_led(void){
55        sendDataSPI(GPPU, 0xFF);
56        HAL_Delay(500);
57        sendDataSPI(GPPU, 0x00);
58        HAL_Delay(500);
59  }
```

Software code x6: Functions for winning or losing the game

Both functions “youwon()” and “youlost()” draw the box with the words, scan the keyboard for input (line 15 and line 40) and call for functions responsible for lighting up the led lights allowing them to blink differently depending on the outcome of the game.

Below figures x13 and x14 x15 show successful practical results of the game when it passes, fails and runs.

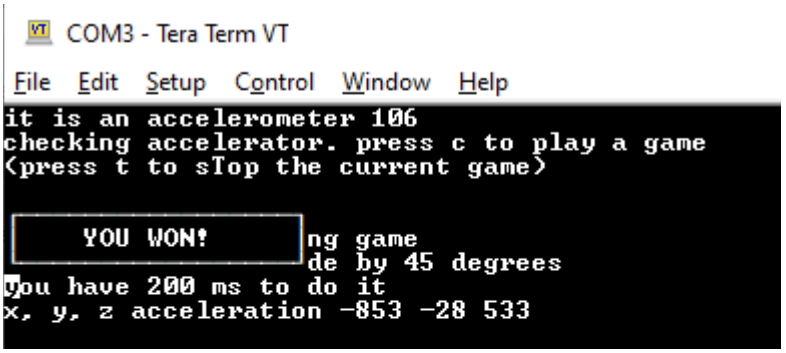


Fig x13: Passed Game Status, Terminal Information

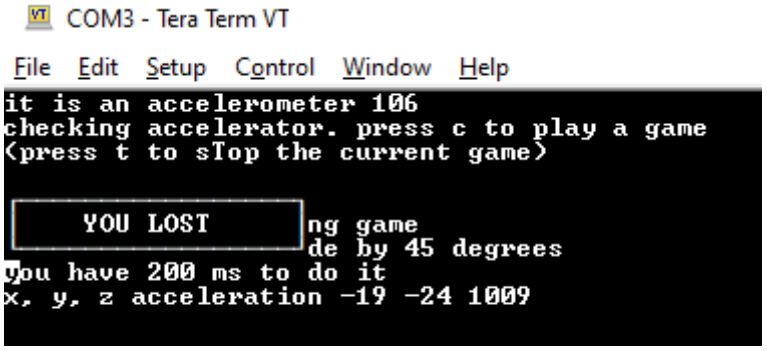


Fig x14: Failed Game Status, Terminal Information

Below are the videos showing how the game actually runs:

- Video 1 - running and winning the game.
- Video 2 - losing the game

6.2. IO Expander Device

The GPIO Expander device MCP23S09 used in the project provides additional 8 I/O pins for the STM322 board accessed via Arduino shield. This device uses SPI communication protocol, therefore HAL SPI functions are used in order to access, configure and read/write to the device. For the accelerometer game purposes all 8 pins are configured as inputs, LED lights are connected to them and functions using these lights are called. This device has been fitted onto the Arduino Proto Shield and connected to the STM32 B-L475-IOT01A board.

Since the device uses SPI, its' pin configuration is set as follows (schematics figure x16):

- Pin #1 - voltage supply (5V) (VDD)
- Pin #3 - Chip Select (PD_14)
- Pin #4 - System Clock (PA_5)
- Pin #5 - Chip MOSI (PA_7)
- Pin #6 - Chip MISO (PA_6)
- Pin #7 - Device reset (RESET)
- Pins #9-#16 - GPIOs for Led lights (8 of them)
- Pin #18 - Ground (GND)

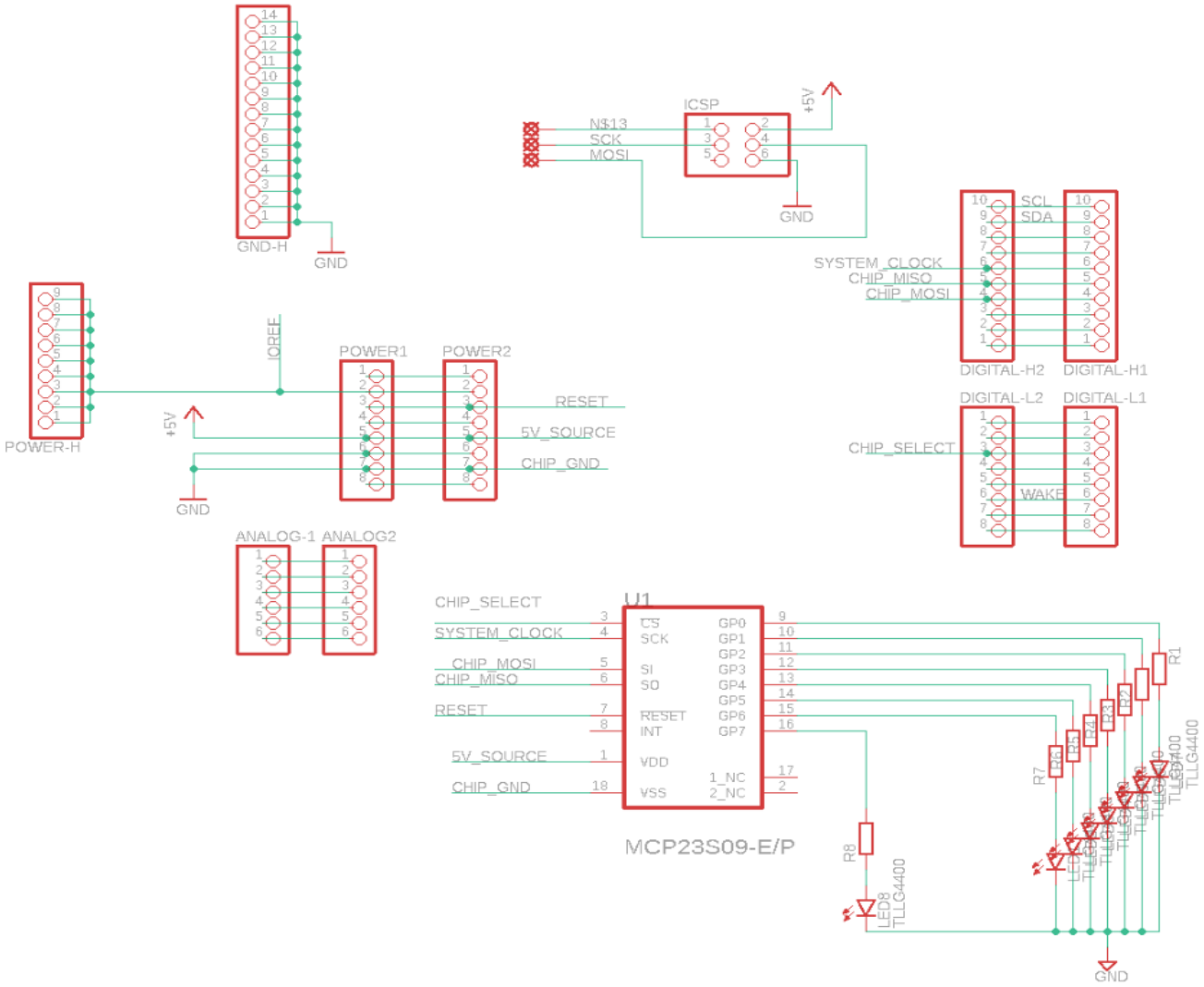


Fig x16: GPIO Expander Schematic

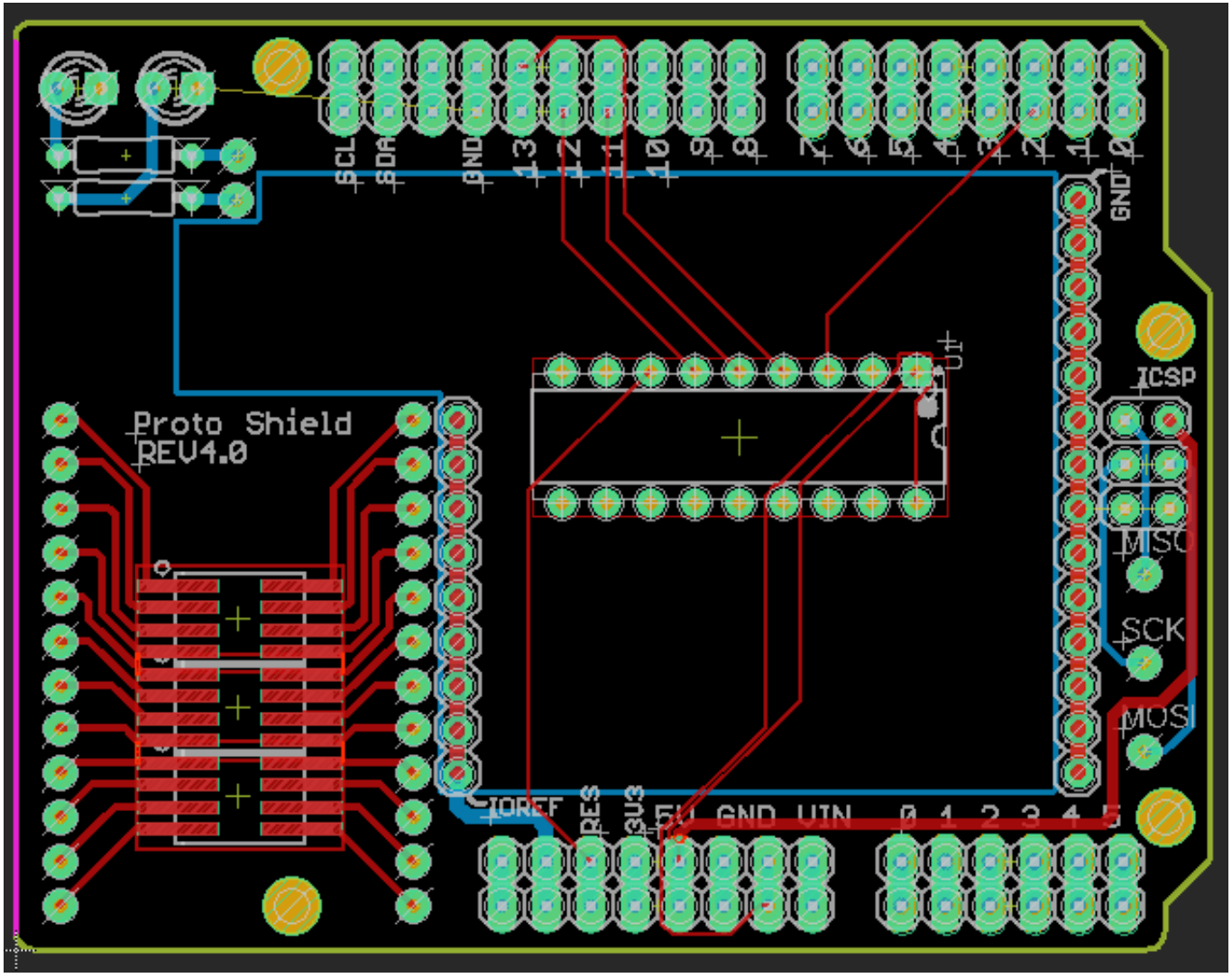


Fig x17: GPIO Expander Board - Arduino Protoshield and GPIO Expander connections

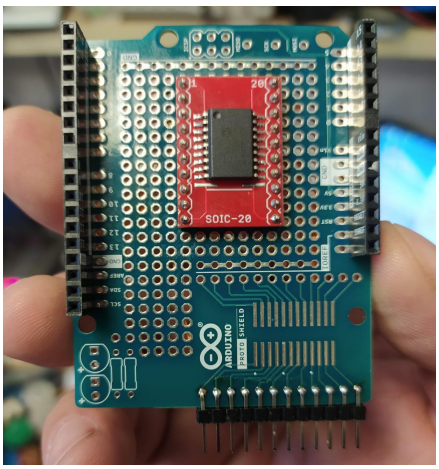


Fig x18: PIO Expander integrated to the Arduino Shield Board

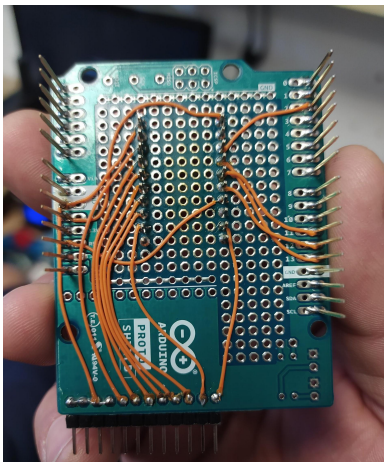


Fig. x19: Soldered GPIO Expander connections to the relevant pins

6.2.1. IO Expander Hardware

Before the GPIO Expander could be soldered onto the shield, a board image with described connections was generated via Eagle PCB software (figure x17). The connections for the LED lights are not included in this image. Finally, the figures x18 and x19 show the soldered shield board based on the provided schematics. As LED lights all need the ground output and only 4 pins on the shield allocated for the ground are not enough, the cathode pins of other 4 LED lights are connected into the same "GROUND" slots via cables. Finally, the figures x19, x20 and x21 present a fully soldered and assembled tandem of the Arduino shield, STM32 Discovery board and the LED lights conveniently connected by cables.

Fig. x19: Fully assembled accelerometer and LEDs module

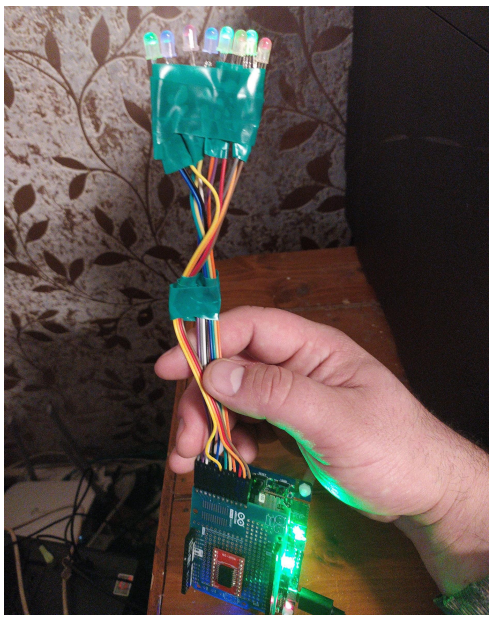


Fig. x20: Working LED lights, full set

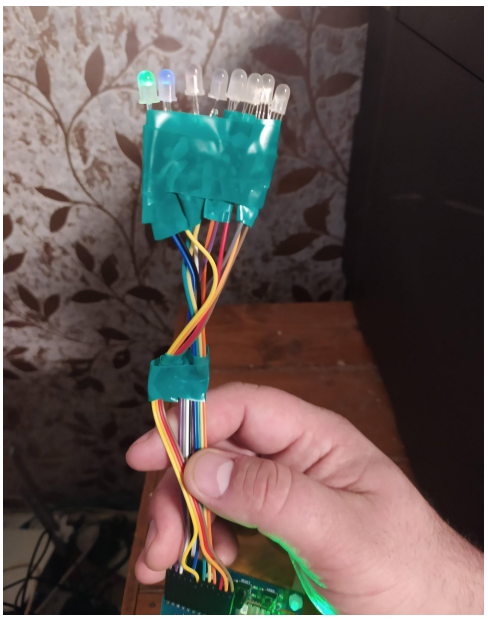


Fig. x21: LED lights activated in sequence

6.2.2. IO Expander Software

As with I2C protocol, Hardware Abstraction Layer functions are used for SPI protocol in order to communicate with the device.

The most important core function responsible for activating LED lights is called "SendDataSPI()" as seen in the software code snippet 7.

```
1   void sendDataSPI(uint8_t reg, uint8_t value){
2   uint8_t buf;
3   HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET); //
4   Chip-select set low
5   HAL_SPI_Transmit(&hspi1,(uint8_t
6   *)&OPCODEW,sizeof(uint8_t),SPI_TRANSFER_TIMEOUT); // Send
7   the MCP23S09 write operation code
8   HAL_SPI_Transmit(&hspi1,(uint8_t
9   *)&reg,sizeof(uint8_t),SPI_TRANSFER_TIMEOUT); // Send the
10  register we want to write
11  HAL_SPI_Transmit(&hspi1,(uint8_t
12  *)&value,sizeof(uint8_t),SPI_TRANSFER_TIMEOUT); // Send the
13  byte
14  HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_SET); // Take
15  slave-select high
16  }
```

Software code x7: Function for Activating the LED lights.

Two values are sent to the function. One value in the form of 8bit variable "reg" provides the address of the particular register and the 8bit "value" variable is responsible for setting up the register which is addressed through "reg" value. For this project the register commonly used for dealing with the Expander is GPPU (0x06); The

function uses three HAL I2C transmitting functions in order to control pull-up internal resistors for the available 8 pins (decided in “value” variable”). Either all pins can have their internal resistors pulled up or one sequentially, depending on the game status (logical “1”). By default when the main menu starts the registers have three-state logic activated. However, it seems when at least one internal resistor is pulled up, then other pins have logical “0” (LED inactive); For testing purposes another function is used “initialize_extender()” (Software Code snippet 8);

```
1     void initialize_extender(void){
2     HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET); //
3     Take chip-select low
4     HAL_SPI_Transmit(&hspi1,(uint8_t
5     *)&OPCODER,sizeof(uint8_t),SPI_TRANSFER_TIMEOUT); // Send
6     the MCP23S09 operational code for reading
7     HAL_SPI_Transmit(&hspi1,(uint8_t
8     *)&GPPU,sizeof(uint8_t),SPI_TRANSFER_TIMEOUT); // Send the
9     MCP23S09 operational code for accessing the control
10    responsible for pull-up registers.
11    HAL_SPI_Receive(&hspi1,(uint8_t
12    *)&leds_binary,sizeof(uint8_t),SPI_TRANSFER_TIMEOUT); //
13    Send the required register
14    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_SET);
15    }
```

Software Code x8: Function Responsible for Reading the Status of GPIO Pins of the Expander

This function is very similar to the “SendDataSPI()” function, because instead of a writing address the reading address is used. Pin D14 is put low as this is the chip select pin for the GPIO Expander. Then the operational read variable is called and the register GPPU responsible for setting up the individual pins high or low is read. Moreover, for testing purposes the aforementioned “led_config()” function can be rewritten with the code which shows how GPIO pins are set up in real time (Software Code snippet 9).

```
1     void led_config(void){
2     mainmenuloop=1;
3     sendDataSPI(GPPU, 0x80);
4     led_colours("10000000 \r");
5     HAL_Delay(200);
6     sendDataSPI(GPPU, 0x40);
7     led_colours("01000000\r");
8     HAL_Delay(200);
9     sendDataSPI(GPPU, 0x20);
10    led_colours("00100000 \r");
11    HAL_Delay(200);
12    sendDataSPI(GPPU, 0x10);
13    led_colours("00010000 \r");
14    HAL_Delay(200);
15    sendDataSPI(GPPU, 0x08);
16    led_colours("00001000\r");
```

```
17  HAL_Delay(200);
18  sendDataSPI(GPPU, 0x4);
19  led_colours("00000100 \r");
20  HAL_Delay(200);
21  sendDataSPI(GPPU, 0x02);
22  led_colours("00000010 \r");
23  HAL_Delay(200);
24  sendDataSPI(GPPU, 0x01);
25  led_colours("00000001\r");
26  HAL_Delay(200);
27  if (UART1_rxBuffer[0]=='t'){
28  mainmenu();
29  }
30  }
```

Software Code x9. Function which Lights Up LED Lights Sequentially.

The 0x80 value for GPPU is responsible for lighting up the very left LED (binary code 10000000), the 0x40 value shifts the pulled-up internal resistor to the left whilst other pins have logical "0" and so on.

The function "led_colours" simply presents the current states of the eight GPIO pins in pseudo-binary format (Software Code snippet x10). The character variable is printed onto the terminal.

```
1  void led_colours(char colour[]){
2  int i=strlen(colour)+strlen("\rrgb test %s\n\r%s
3  \033[F")+strlen(msg_acc);
4  char message;
5  sprintf(message, "\rrgb test %s\n\r%s \033[F",colour, msg_acc);
6  HAL_UART_Transmit(&huart1, message, strlen(message), 200);
7  }
```

Software Code x10: Function for Presenting activated LED Lights in the Pseudo-binary Format

6.3. IO Expander - Integration with Another Interface

There are several ways to achieve the integration with other devices in this project. two functions "won_led()" and "lost_led()" (of the software code snippet 6 (line 48 and line 54 accordingly)) lines allow other game modules with different sensors to blink the led lights if their games pass or fail. Another way to provide integration to the accelerometer game is to send the text of the game progress and man menu of this module to the LCD display via a specific function, for example: ST7789_WriteString(uint16_t x, uint16_t y, const char *str, FontDef font, uint16_t color, uint16_t bgcolor). EXAMPLE: `ST7789_WriteString(10, 50, "\rlet's play a guessing game\n\rtilt the board's side by 45 degrees\n\ryou have %d ms to do it \n\r, WHITE);`

7. Individual Deliverables - Tomáš Krupička

7.1. Humidity Sensor

Humidity sensor is located on a HTS221 chip on the discovery board together with the temperature sensor (see section 4.1). This means that the usage of these sensors is very similar, but can be described separately, because some of the registers used for the sensor are exclusive. The registers from the sensor are accessed using I2C protocol. These registers can be used to configure the sensor, and get humidity from it.

Relative humidity is described in percentages, ranging from 0% to 100%. The task for the game is simple - the user has to get the current measured humidity to a given value (about 10% off the current humidity).

7.1.1. Humidity Sensor - Functions

The humidity sensor library uses functions in 2 different pairs (.c and .h) of files, plus 1 more pair for testing (mentioned later in section 7.1.3). Those files are humidity.h, humidity.c, humidity-task.h and humidity-task.c. Below, you can see all functions in the files described.

7.1.1.1 humidity.c

```
uint8_t readHumidityRegister(uint8_t registerAddress){
	uint8_t out;
	HAL_I2C_Master_Transmit(&hi2c2, HTS221_WRITE, &registerAddress, 1, 1000);
	HAL_I2C_Master_Receive(&hi2c2, HTS221_READ, &out, 1, 1000);
	return out;
}

void writeToHumidityRegister(uint8_t registerAddress, uint8_t value){
	uint8_t in[2];
	in[0] = registerAddress;
	in[1] = value;
	HAL_I2C_Master_Transmit(&hi2c2, HTS221_WRITE, in, 2, 1000);
}
```

Software code x: HTS221 register access

These are basic functions that allow simple reading from and writing to registers. The values *HTS221_WRITE* and *HTS221_READ* are read and write addresses specific to the discovery board we have been using, and they are stored in code as preprocessor defines.

```
void HTS221On(){
	uint8_t value = readHumidityRegister(HTS221_CTRL_REG1);
	writeToHumidityRegister(HTS221_CTRL_REG1, value | (1 << 7));
}

void HTS221Off(){
	uint8_t value = readHumidityRegister(HTS221_CTRL_REG1);
	uint8_t mask = 1 << 7;
	mask = ~mask;
	writeToHumidityRegister(HTS221_CTRL_REG1, value & mask);
}
```

Software code x: HTS221 Off/On

These functions set the sensors to be working - if the sensors are off, values in their registers are not updated. Register values are from a BSP library. On/off state is defined by a single bit, so I make use of masking to edit only that bit.

```
void HTS221valueLock(){
	uint8_t value = readHumidityRegister(HTS221_CTRL_REG1);
	writeToHumidityRegister(HTS221_CTRL_REG1, value | (1 << 2));
}

void HTS221valueUnlock(){
	uint8_t value = readHumidityRegister(HTS221_CTRL_REG1);
	uint8_t mask = 1 << 2;
	mask = ~mask;
	writeToHumidityRegister(HTS221_CTRL_REG1, value & mask);
}
```

Software code x: HTS221 value lock

In these functions, the “value lock” bit is set (again, using masking and BSP library register values). When the bit is set (“value lock” is active), when the library tries to read from value registers, after it reads the first one, the second one won’t be updated because of the humidity change until it is read too.

```
float humidityGet(){
	return HTS221_H_ReadHumidity(HTS221_READ);
}
```

Software code x: humidity value reading

This function is simple, it only wrapps the BSP function used for humidity reading to be used in my library.

```
void HTS221Init(){
	HTS221_H_Init(HTS221_WRITE);
	HTS221valueLock();
}
```

Software code x: HTS221 initialization

As in previous code, this function wrapps BSP function used for the I2C communication initialization. Also, it sets the value lock, so the sensor is ready to be used when turned on.

7.1.1.2 humidity-task.c

```
void humidityShowTaskInfo(){
	#if USE_DISPLAY == 1

	#else
		printf("\rYour task is to get to target humidity!\n");
	#endif
}
```

Software code x: Task info display

This simple function shows general info about the upcoming task. The function (and other humidity task and test functions) doesn't have output to the display, because at the time of writing, the display library was just newly available to be used, so the code doesn't have the display output yet.

```
task_result_t humidityStartTask(float difficulty){
	humidityTaskTargetValue = humidityGet();
	int difference = rand() % 10 + 2;
	difference *= (rand() % 3 == 0) ? -1 : 1;
	humidityTaskTargetValue += difference;
	if(humidityTaskTargetValue > 100){
		humidityTaskTargetValue -= (humidityTaskTargetValue - 100) * 2;
	}
	humidityTaskStartTime = getTimeMs();
	humidityTaskDifficultyMultiplier = difficulty;
	#if USE_DISPLAY == 1

	#else
		printf("\rTarget humidity: %f%%\n", humidityTaskTargetValue);
	#endif
	return TASK_RUNNING;
}
```

Software code x: Task initialization

This function initializes the task - sets up all values (in global variables), randomly generates the target humidity, and shows it to the user. The random generation is modified by setting the range (up 2-12% from the initial value) and setting a higher chance for getting the humidity increase (it is simpler).

```
task_result_t humidityCheckTaskState(){
	float currentHumidity = humidityGet();
	int difference = currentHumidity - humidityTaskTargetValue;
	#if USE_DISPLAY == 1

	#else
		if((getTimeMs() - humidityTaskStartTime) % 1000 == 0){
			printf("\rCurrent humidity: %f%%\n", currentHumidity);
		}
		if(difference < 0.5 && difference > -0.5){
			printf("\rTask passed!\n");
			return TASK_PASSED;
		}
	#endif
	if((getTimeMs() - humidityTaskStartTime) > 10000 / humidityTaskDifficultyMultiplier){
		return TASK_FAILED;
	}
	return TASK_RUNNING;
}
```

Software code x: Task result check

The final task function is used to determine the final result of the task, by calling the function in a loop until the function returns *TASK_PASSED* or *TASK_FAILED*. Failure can be achieved by running out of time (which is 10 seconds, modified by difficulty), success by getting to correct humidity (with 0.5% precision).

7.1.2. Humidity Sensor - Global Variables

The humidity sensor library also uses 4 global variables. One (*hi2c2*) is an external variable - an I2C handler to use the communication protocol. This is used in the humidity.c file and is crucial for the basic library function. The other 3 are used in the humidity-task.c file, namely *humidityTaskTargetValue*, *humidityTaskStartTime* and *humidityTaskDifficultyMultiplier*. These are set when the task is initialized, and allow to carry the information needed for the task to be used in the loop-called task function.

7.1.3. Humidity Sensor - Testing

Testing for the humidity sensor is done in 3 parts. The first one is common for the whole project - successful compilation without any errors or warnings.

7.1.3.1 Humidity auto tests

The second are auto tests - tests that are easily done automatically, and have precise output (PASS/FAIL). These tests check things related to registers in general. Because of their simple nature, these tests can run automatically at the beginning of every run of the program.

```
int humidityRunAutoTests(){
	int result = 0;
	#if defined(INC_DISPLAY_H_) && (USE_DISPLAY == 1)
		//Run tests differently with output
		return result;
	#else
		printf("\r========RUNNING HUMIDITY BASIC TESTS========\n");
		result = humidityRunWhoAmITest();
		if(result == 0){
			result = humidityRunRWRTest();
		}
		if(result == 0){
			printf("\r========HUMIDITY BASIC TESTS PASSED========!\n");
		}
		else{
			printf("\r========HUMIDITY BASIC TESTS FAILED!========\n");
		}
		return result;
	#endif
}
```

Software code x: Run all auto tests

First test function runs all auto tests (2), and returns/prints overall results from the auto tests.

```
int humidityRunWhoAmITest(){
	uint8_t deviceId;
	#if defined(INC_DISPLAY_H_) && (USE_DISPLAY == 1)
		//Run tests differently with output
		return 0;
	#else
		printf("\r====RUNNING WHO_AM_I TEST====\n");
		printf("\rReading WHO_AM_I address of humidity sensor\n");
		deviceId = readHumidityRegister(HTS221_WHO_AM_I_REG);
		printf("\rRead value - %d, correct value - %d\n", deviceId,
HTS221_WHO_AM_I_VAL);
		if(deviceId == HTS221_WHO_AM_I_VAL){
			printf("\r====WHO_AM_I test passed!====\n");
			return 0;
		}
		else{
			printf("\r====WHO_AM_I test failed!====\n");
			return 1;
		}
	#endif
}
```

Software code x: Who am I test

This test checks if the program is able to read from a register, specifically from the WHO_AM_I register. The value there is constant, so it can be easily checked if it is read correctly. Also, this allows checking if the device address used for assessing the sensor is correct (other devices would have different values).

```
int humidityRunRWRTest(){
	uint8_t oldValue;
	uint8_t newValue;
	#if defined(INC_DISPLAY_H_) && (USE_DISPLAY == 1)
		//Run tests differently with output
		return 0;
	#else
		printf("\r====RUNNING RWR TEST====\n");
		oldValue = readHumidityRegister(HTS221_CTRL_REG1);
		uint8_t writtenValue;
		if((oldValue & (1 << 7)) == 0){
			writtenValue = oldValue + (1 << 7);
		}
		else{
			writtenValue = oldValue - (1 << 7);
		}
		printf("\rRead value - %d\n", oldValue);
		writeToHumidityRegister(HTS221_CTRL_REG1, writtenValue);
		printf("\rWrite value - %d\n", writtenValue);
		newValue = readHumidityRegister(HTS221_CTRL_REG1);
		printf("\rRead control of current value - %d\n", newValue);
		writeToHumidityRegister(HTS221_CTRL_REG1, oldValue);
		if(oldValue != newValue && writtenValue == newValue){
			printf("\r====RWR test passed!====\n");
			return 0;
		}
		else{
			printf("\r====RWR test failed!====\n");
			return 1;
		}
	#endif
}
```

Software code x: Read write read test

While the first test checks read ability, the second test checks write ability by changing a value in a register (on/off bit) and comparing read values before and after.

7.1.3.2 Humidity interactive tests

The third are interactive tests - tests that can be done automatically and partially automatically evaluated, but the result can be misleading (for example, the sensor is working, but measuring double the actual humidity). Therefore, these need to be checked by a user and are not expected to be automatically run on start.

```
int humidityRunInteractiveTests(){
	#if defined(INC_DISPLAY_H_) && (USE_DISPLAY == 1)
		//Run tests differently with output
		return 0;
	#else
		printf("\r========RUNNING HUMIDITY INTERACTIVE TESTS========\n");
		int result = humidityRunSensorTest();
		if(result == 0){
			result = humidityRunLockTest();
		}
		if(result == 0){
			printf("\r========HUMIDITY INTERACTIVE TESTS PASSED========!\n");
		}
		else{
			printf("\r========HUMIDITY INTERACTIVE TESTS FAILED!========\n");
		}
		return result;
	#endif
}
```

Software code x: Run all interactive tests

This function, similar to the first one, runs all (2) interactive tests with appropriate result output and aggregation.

```
int humidityRunSensorTest(){
	#if defined(INC_DISPLAY_H_) && (USE_DISPLAY == 1)
		//Run tests differently with output
		return 0;
	#else
		printf("\r====RUNNING SENSOR TEST====\n");
		HTS221On();
		int changed = 0;
		float lastValue = humidityGet();
		printf("\rHUMIDITY: %f\n", lastValue);
		for(int i = 0; i < 10; i++){
			HAL_Delay(1000);
			float newValue = humidityGet();
			float diff = newValue - lastValue;
			if(diff < 0){
				diff = diff * -1;
			}
			if(diff > 0.01){
				changed = 1;
			}
			lastValue = newValue;
			printf("\rHUMIDITY: %f\n", lastValue);
		}
		if(changed == 1){
			printf("\r====HUMIDITY SENSOR TEST PASSED====!\n");
			return 0;
		}
		else{
			printf("\r====HUMIDITY SENSOR TEST FAILED!====\n");
			return 1;
		}
	#endif
}
```

Software code x: Sensor test

This interactive test outputs currently measured humidity values, which a user can check if they are correct. The automatic part counts on small constant small changes in measured values, therefore reassuring the sensor is measuring something (updating its value registers).

```
int humidityRunLockTest(){
	#if defined(INC_DISPLAY_H_) && (USE_DISPLAY == 1)
		//Run tests differently with output
		return 0;
	#else
		printf("\r====RUNNING LOCK TEST====\n");
		HTS221On();
```

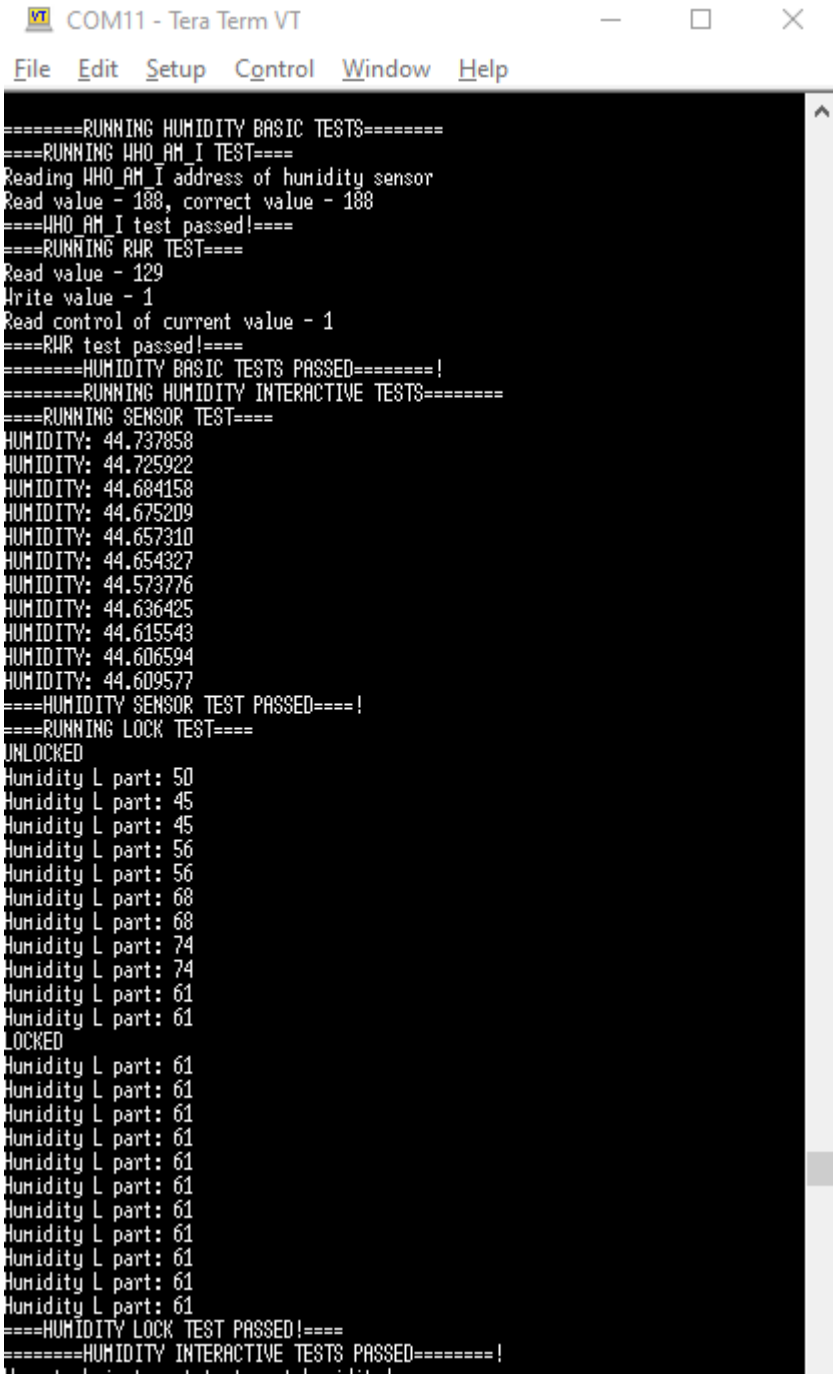
```
		int changed = 0;
		printf("\rUNLOCKED\n");
		HTS221valueUnlock();
		int lastValue = readHumidityRegister(HTS221_HR_OUT_L_REG);
		printf("\rHumidity L part: %d\n", lastValue);
		for(int i = 0; i < 10; i++){
			HAL_Delay(500);
			int newValue = readHumidityRegister(HTS221_HR_OUT_L_REG);
			int diff = newValue - lastValue;
			if(diff != 0){
				changed = 1;
			}
			lastValue = newValue;
			printf("\rHumidity L part: %d\n", lastValue);
		}
		if(changed == 0){
			printf("\r====HUMIDITY LOCK TEST FAILED!====\n");
			return 1;
		}
		printf("\rLOCKED\n");
		HTS221valueLock();
		lastValue = readHumidityRegister(HTS221_HR_OUT_L_REG);
		changed = 0;
		printf("\rHumidity L part: %d\n", lastValue);
		for(int i = 0; i < 10; i++){
			HAL_Delay(500);
			int newValue = readHumidityRegister(HTS221_HR_OUT_L_REG);
			int diff = newValue - lastValue;
			if(diff != 0){
				changed = 1;
			}
			lastValue = newValue;
			printf("\rHumidity L part: %d\n", lastValue);
		}
		if(changed == 1){
			printf("\r====HUMIDITY LOCK TEST FAILED!====\n");
			return 1;
		}
		else{
			printf("\r====HUMIDITY LOCK TEST PASSED!====\n");
			return 0;
		}
	#endif
}
```

Software code x: Lock test

This test checks the ability to lock the read value, when only one half is read by constantly reading only the lower part under locked and unlocked state.

7.1.4. Humidity Sensor - Practical Example

The following screenshot shows a successful run of all tests (auto and interactive) seen in a terminal connected through a serial port.



7.2. SD Card Device

The SD card is another device that is connected to the board and communicated using the SPI protocol. The SD card is controlled by sending commands, and receiving responses. The SD card itself is inserted into a slot on a holder, which is the main part of the hardware design.

Unfortunately, the SD card wasn't working in the end. Because of unknown reason, hardware or software, the library wasn't able to initialize SPI communication, so the card couldn't be used. The hardware and software was however completely finished, so fixing this problem would make the card functions usable.

7.2.1. SD Card Hardware

The SD card holder has 7 pins, 6 of which are used for the SPI interface, and the last pin works as a hardware check for card insertion.

- CD - Card detect - powered on when the card is not inserted. Connected to pin D10
- DO - Data output - output pin for SPI interface. Connected to MISO pin D12
- GND - Ground
- SCK - Serial clock - clock signal for SPI interface. Connected to SCK pin D13
- VCC - Power. Connected to 5V pin
- DI - Data input - input pin for SPI interface. Connected to MOSI pin D11
- CS - Chip select - chip select pin for SPI interface. Connected to this device's CS pin D5

The SD card holder is soldered onto an arduino protoshield and wired to correct pins, similarly to other devices that have been used.

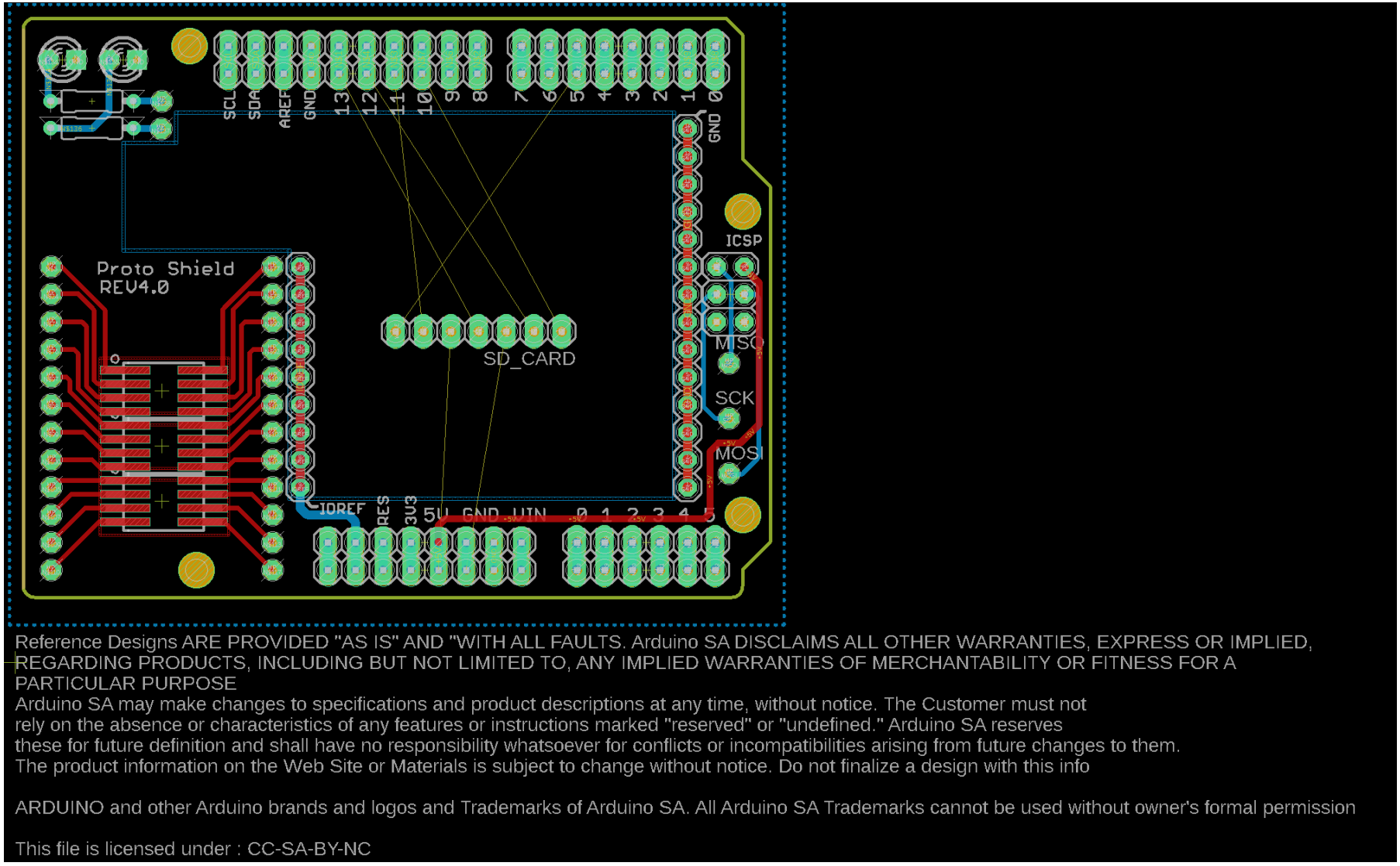


Fig x: SD card shield board

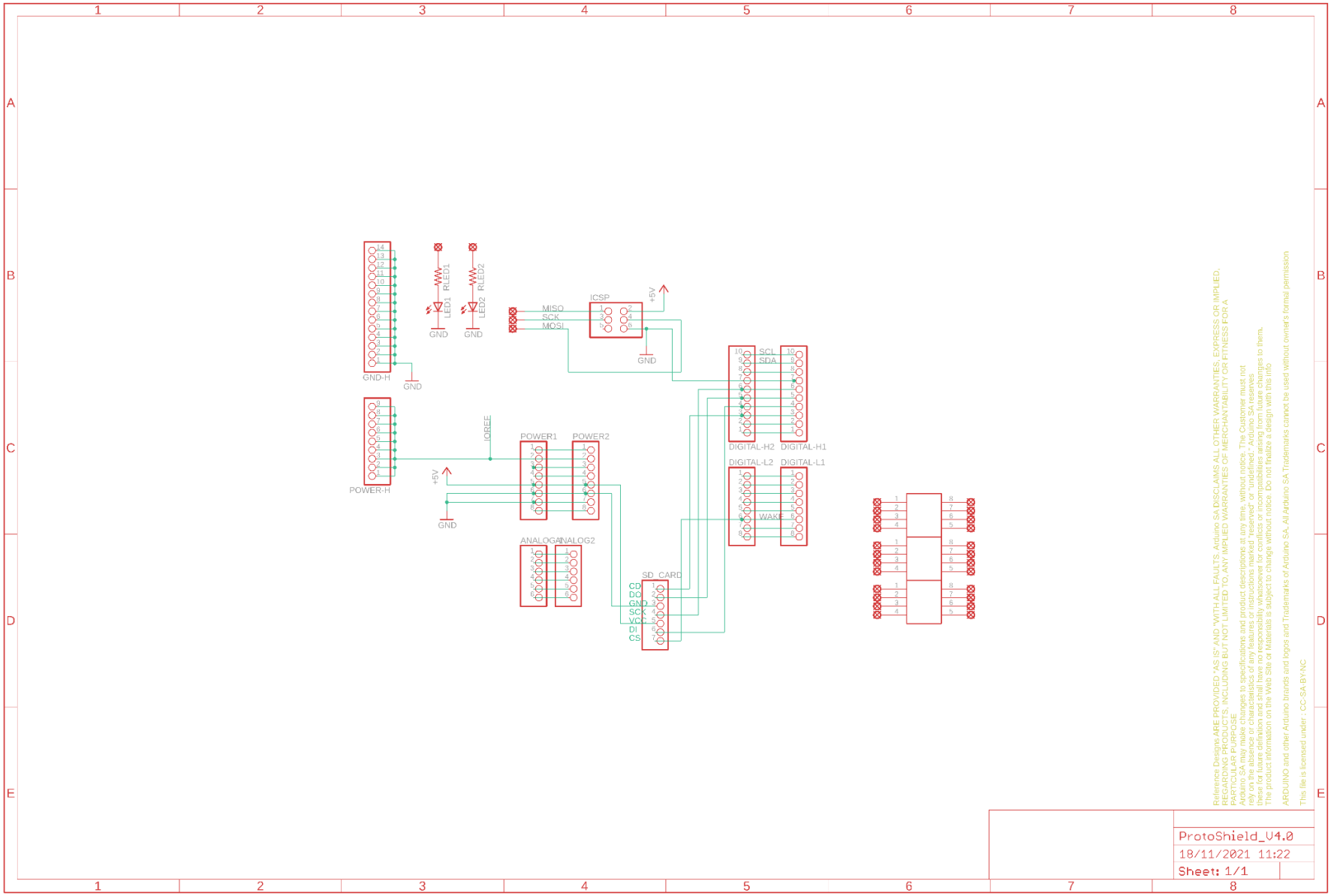


Fig x: SD card shield schematic

Previous images show the EAGLE schematic and board of the protoshield, which has the SD card holder soldered on. The next images show photos of the soldered shield

Fig x: Upper view

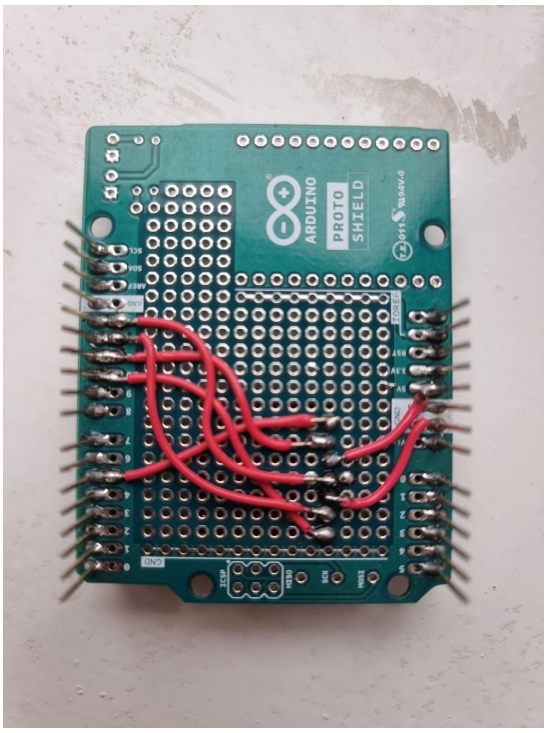


Fig x: Lower view

7.2.2. SD Card Software

The SD card library mostly uses third party FatFS library, and also open source library found here: https://github.com/kiwih/cubeide-sd-card/tree/master/cubeide-sd-card/FATFS/Target

```
/*
 * sd-card.c
 *
 *      Author: tomas
 */

#include "sd-card.h"

FATFS SDFatFs;

bool SDFatFSMounted = false;

bool isCardPresent(){
	return (HAL_GPIO_ReadPin(SD_DETECT_GPIO_Port, SD_DETECT_Pin) ==
GPIO_PIN_SET);
}
```

```
int SDMount(){
	if(SDFatFSMounted || !isCardPresent){
		return -1;
	}
	FRESULT res = f_mount(&SDFatFs, "", 1);
	SDFatFSMounted = true;
	return res;
}

int openFile(FIL * file, char * name, bool write){
	if(!SDFatFSMounted || !isCardPresent){
		return -1;
	}
	return f_open(file, name, write ? FA_WRITE : FA_READ);
}

int closeFile(FIL * file){
	if(!SDFatFSMounted || !isCardPresent){
		return -1;
	}
	return f_close(file);
}

int readLine(FIL * file, char * out, int length){
	if(!SDFatFSMounted || !isCardPresent){
		return -1;
	}
	char c;
	int pos = 0;
	int res = 1;
	res = f_gets(&c, 1, file);
	while(c != '\n' && (pos != length - 1) && res != 0){
		out[pos] = c;
		res = f_gets(&c, 1, file);
		pos ++;
	}
	if(res == 0){
		return pos;
	}
	if(c != '\n'){
		out[pos] = c;
		return pos + 1;
	}
	out[pos] = 0;
	return pos;
}

int writeLine(FIL * file, char * in, int length){
	if(!SDFatFSMounted || !isCardPresent){
		return -1;
	}
```

```
	unsigned_int res;
	unsigned_int bytesWrote;
	res = f_write(file, in, length, &bytesWrote);
	if(bytesWrote != length){
		return -2;
	}
	if(res != FR_OK){
		return res;
	}
	char c = '\n';
	res = f_write(file, &c, 1,  &bytesWrote);
	return res;
}
```

Software code x: sd-card.c

This file uses both libraries to easily read from and write to the SD card. These functions are then used for specific purposes.

7.3. SD Card - Integration with Another Interface

The SD card has 2 uses in the game. Firstly, it reads the configuration file (config.txt), and loads from there the number of lives the player has in one game, and coefficient, which defines the acceleration of the levels throughout the game.

```
bool loadConfig(int * lives, float * diffMult){
	#if USE_SD_CARD == 1
		FIL file;
		int res = openFile(&file, "config.txt", false);
		if(res){
			return false;
		}
		char data[30];
		res = readLine(file, data, 30);
		if(!res){
			return false;
		}
		*lives = atoi(data);
		res = readLine(file, data, 30);
		if(!res){
			return false;
		}
		*diffMult = atof(data);
		closeFile(&file);
	#endif
	return true;
}
```

Software code x: Configuration load

Secondly, it is used to manage a leaderboard of best players that is persistent between turn-ons.

```
int leaderboardScores[LEADERBOARD_POSITIONS];

int saveNewScore(int score){
	int position = -1;
	for(int i = 0; i < LEADERBOARD_POSITIONS; i++){
		if(leaderboardScores[i] < score){
			position = i;
		}
	}
	if(position != -1){
		for(int i = LEADERBOARD_POSITIONS - 1; i > position; i--){
			leaderboardScores[i] = leaderboardScores[i - 1];
		}
		#if USE_SD_CARD == 1
			FIL file;
```

```
                    int res = openFile(&file, "leaderboard.txt", true);
                    if(res){
                          return false;
                    }
                    char line[30];
                    for(int i = 0; i < LEADERBOARD_POSITIONS; i++){
                          sprintf(line, "%d", leaderboardScores[i]);
                          res = writeLine(file, line, strlen(line));
                          if(res){
                                return false;
                          }
                    }
                    res = closeFile(file);
                    if(res){
                          return false;
                    }
             #endif
      }
      return position;
}


bool loadLeaderboard(){
      for(int i = 0; i < LEADERBOARD_POSITIONS; i++){
             leaderboardScores[i] = -1;
      }
      #if USE_SD_CARD == 1
             FIL file;
             int res = openFile(&file, "leaderboard.txt", false);
             if(res){
                    return false;
             }
             char line[30];
             for(int i = 0; i < LEADERBOARD_POSITIONS; i++){
                    res = readLine(file, line, 30);
                    if(res == 0){
                          break;
                    }
                    leaderboardScores[i] = atoi(line);
             }
             res = closeFile(file);
             if(res){
                    return false;
             }
      #endif
}

int getScore(int pos){
      return leaderboardScores[pos];
}
```

Software code x: Leaderboard

8. Individual Deliverables - Turan Hudasoy

8.1. I2C Sensor (Barometer)

8.1.1. Functions

WHO_AM_I Function:

The code below shows how the WHO_AM_I address was verified. If any value other than the WHO_AM_I address was received, the error message would be seen on the terminal.

```
HAL_I2C_Master_Transmit(&hi2c2, pressure_write_address,
&WHO_AM_I, 1, 100);
HAL_I2C_Master_Receive(&hi2c2, pressure_read_address, &Output,
1, 100);
//HAL_UART_Transmit(&huart1, &Output, 1, 100);
printf("\rOutput:\n \r%d\n",Output);

if (Output==177)
{
printf("\rWHO_AM_I address verified\n");
}
else
{
printf("\rERRORRR!\n");
}
```

8.1.2. Testing

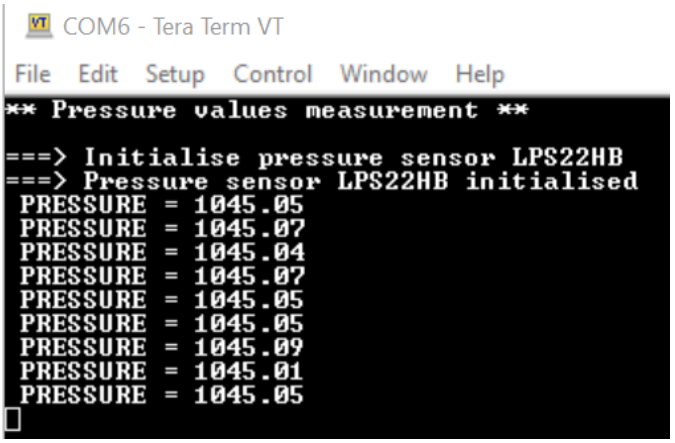
WHO_AM_I Test

The test code was run to test that the WHO_AM_I address could be verified successfully. The figure below shows the output value printed within the Tera Term terminal. The code also tested whether the barometer sensor could be written to and read off of which also proved to be a success. The full code can be seen in the appendices.

Obtaining Pressure Sensor Reading Test

The test code in the appendices was run to test that pressure sensor readings could be taken from the barometer sensor. The figure below shows the successful

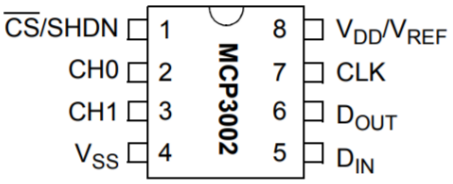
results. The full code, which utilises the board support package (BSP) drivers provided by ST. The full code can be seen in the appendices.

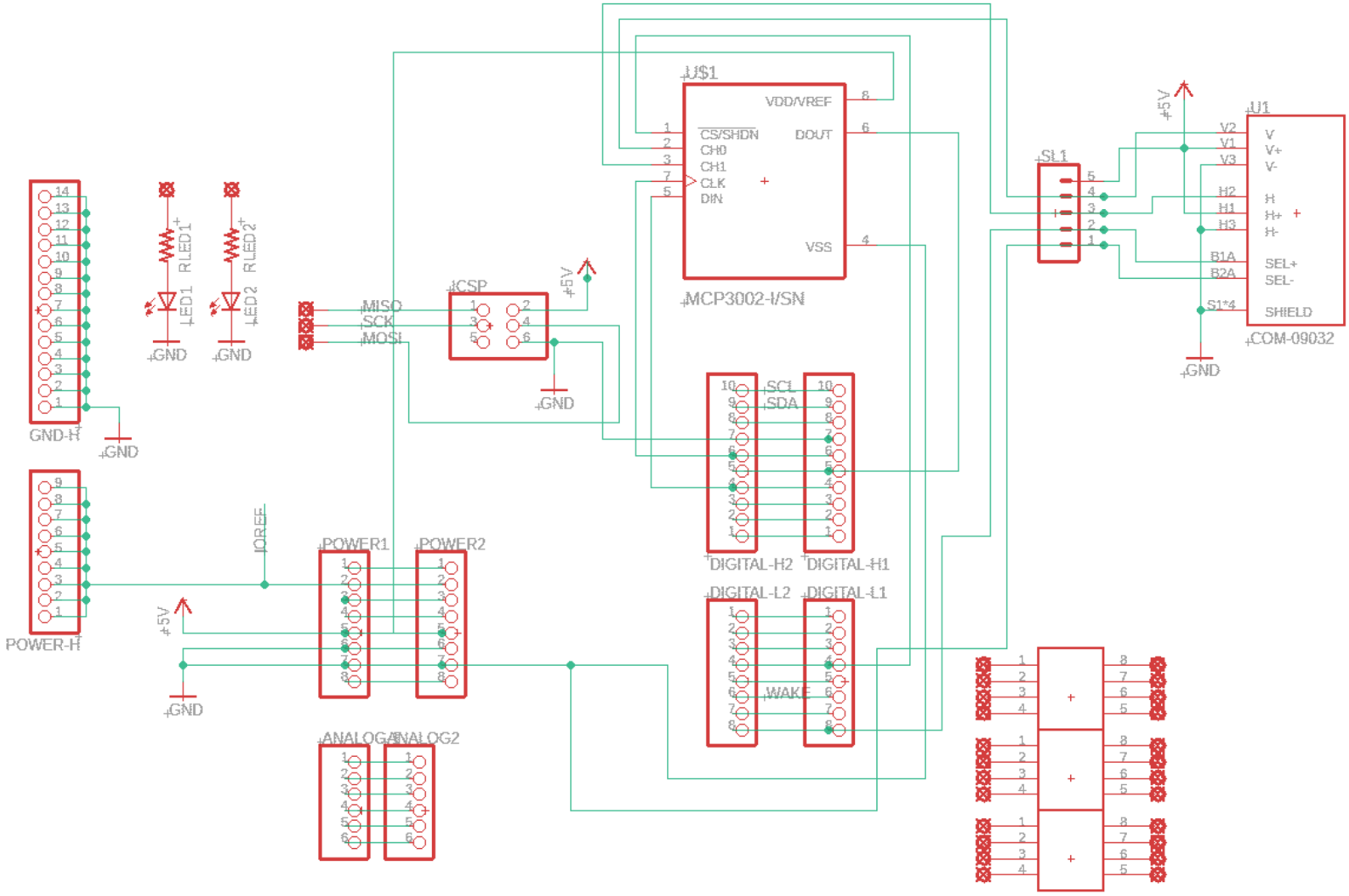


8.2. SPI Device (Thumb Joystick)

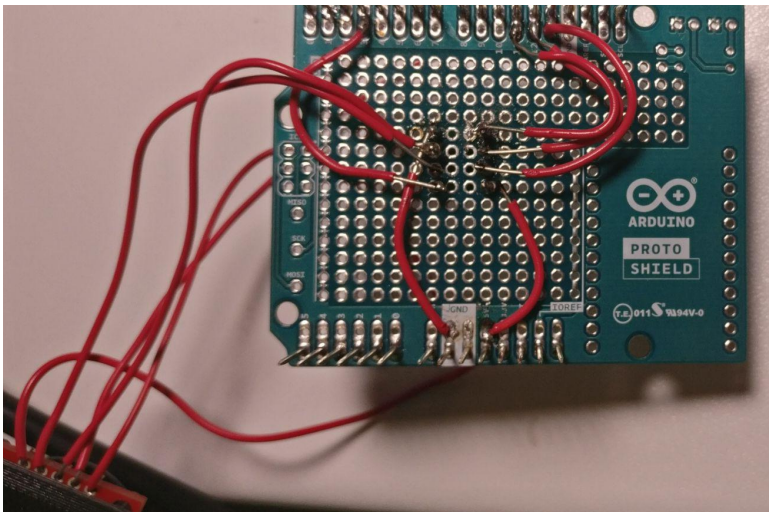
8.2.1. Hardware

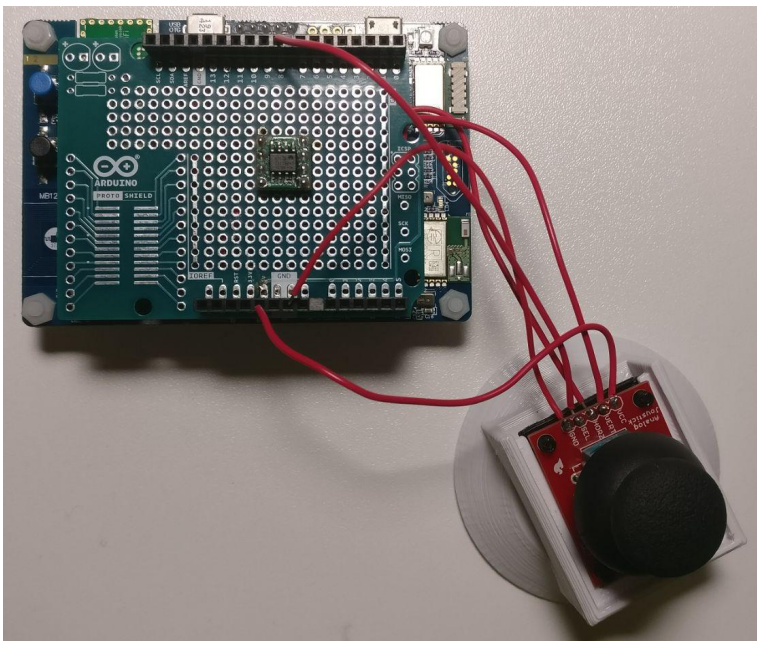
Before any soldering could be done on the Arduino proto shield board, a schematic was produced showing the design of the circuit that utilised the SPI bus as shown in Figure #. The circuit included a thumb joystick, a breakout board and an MCP3002 analog-to-digital converter (ADC). The ADC was connected following the configuration set in the MCP3002 datasheet, (provided by Microchip Technology Inc.), which can be seen below. The chip select pin that was allocated for use by the ADC was GPIO pin 4. Pin 2 (CH0) on the ADC was connected to the vertical movements of the joystick and pin 3 (CH1) was connected to the horizontal. The supply voltage (VDD/VREF) was set to 5V, as opposed to 3.3V, in order for the full range of the joystick's movements to be detected and the least amount of saturation to occur. CLK, DOUT and DIN (CLK, MISO and MOSI) were connected to GPIO pins 5, 6 and 7 which were the pins allocated to be used by all SPI devices.

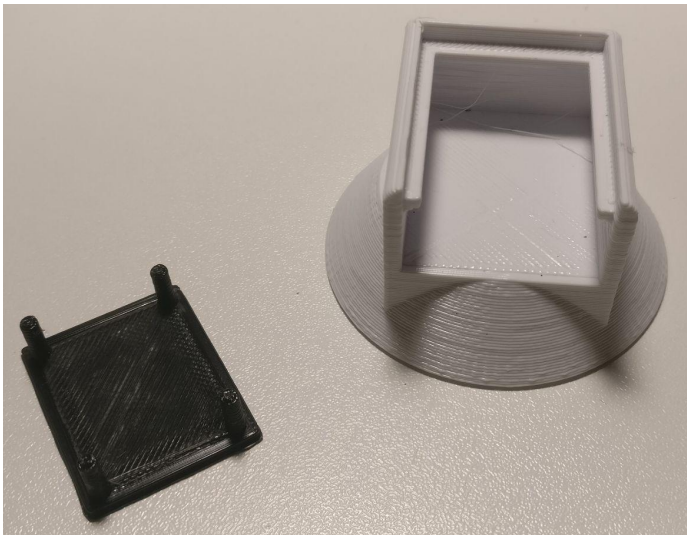



U$1
VDD/VREF
CS/SHDN
CH0
CH1
CLK
DIN
DOUT
VSS
MCP3002-I/SN
U1
V2
V1
V3
V-
H2
H1
H3
H-
B1A
B2A
S1*4
V
V+
H
H+
SEL+
SEL-
SHIELD
COM-09032
SL1
+5V
GND
MISO
SCK
MOSI
ICSP
RLED1
RLED2
LED1
LED2
GND-H
POWER-H
IOREF
POWER1
POWER2
ANALOG1
ANALOG2
DIGITAL-H2
DIGITAL-H1
DIGITAL-L2
DIGITAL-L1
SCL
SDA
WAKE

Once the circuit schematic design was approved, the circuit was built. The ADC chip was soldered onto a breakout board using soldering flux to make the process easier due to the size of the chip. Connectors were then added to the breakout board to attach the chip to the proto shield board. A continuity test was performed to confirm all connections had been properly soldered onto the chip's breakout board before moving onto the rest of the circuit. As soon as the test was successful, the rest of the circuit was connected and soldered. A continuity test was performed at regular intervals to detect mistakes as soon as possible. This was to prevent troubleshooting issues once the hardware was completed. The complete SPI shield circuit can be seen below. A small mount and stand was created for the thumb joystick using a 3D printer to enable the user to operate the device without touching any exposed pins or wiring which can be seen below. The mount was attached underneath the thumb joystick using the 4 mounting holes in the corners of the breakout board. This also allowed the breakout board to slot into the stand. During the soldering process, the soldering iron was set to 400°C by mistake which resulted in partial burns to the proto shield board. This can be seen below.




ARDUINO
PROTO SHIELD



9. Conclusion

In order to create the system development project stated in this report, both C programming abilities, hardware design and circuit building skills were needed. This enabled the use of six I2C sensors and allowed for communication between interfaces. The creation of shield board schematics allowed for the practical build of six different circuits with varying capabilities which were used to enhance the final development project idea. As a result, an almost complete working version of an interactive puzzle game was achieved. The report also outlines problems encountered during the project process and the ways in which those problems were solved.

Knowledge gained from this project included using a discovery kit, learning how to communicate with onboard sensors and devices using I2C and SPI, soldering onto an Arduino proto shield board and extracting information efficiently from sensor and device datasheets. This allowed for successful and safe use of equipment to prevent damage to any provided components. This included proper use of software libraries such as HAL and BSP as well as initialising devices to ensure they operate at optimal conditions.

10. Appendices

10.1 Barometer Code

WHO_AM_I Test Code:

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2021 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */

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

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <stdio.h>
/* 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 */
#ifdef __GNUC__
/* With GCC/RAISONANCE, small msg_info (option LD Linker->Libraries->Small
msg_info
set to 'Yes') calls __io_putchar() */
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#define GETCHAR_PROTOTYPE int __io_getchar(void)
```

```c
#else
#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#define GETCHAR_PROTOTYPE int fgetc(FILE *f)
#endif /* __GNUC__ */
/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
DFSDM_Channel_HandleTypeDef hdfsdm1_channel1;

I2C_HandleTypeDef hi2c2;

QSPI_HandleTypeDef hqspi;

SPI_HandleTypeDef hspi3;

UART_HandleTypeDef huart1;
UART_HandleTypeDef huart3;

PCD_HandleTypeDef hpcd_USB_OTG_FS;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DFSDM1_Init(void);
static void MX_I2C2_Init(void);
static void MX_QUADSPI_Init(void);
static void MX_SPI3_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_USART3_UART_Init(void);
static void MX_USB_OTG_FS_PCD_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
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_DFSDM1_Init();
  MX_I2C2_Init();
  MX_QUADSPI_Init();
  MX_SPI3_Init();
  MX_USART1_UART_Init();
  MX_USART3_UART_Init();
  MX_USB_OTG_FS_PCD_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  uint8_t pressure_read_address = 0xBB;
  uint8_t pressure_write_address = 0xBA;
  uint8_t WHO_AM_I = 0x0F;
  uint8_t Output;

  while (1)
  {
    /* USER CODE END WHILE */
HAL_I2C_Master_Transmit(&hi2c2, pressure_write_address, &WHO_AM_I, 1, 100);
HAL_I2C_Master_Receive(&hi2c2, pressure_read_address, &Output, 1, 100);
//HAL_UART_Transmit(&huart1, &Output, 1, 100);
printf("\rOutput:\n \r%d\n",Output);

if (Output==177)
{
printf("\rWHO_AM_I address verified\n");
}
```

```
else
{
printf("\rERRORRR!\n");
}

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

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** Configure LSE Drive Capability
  */
  HAL_PWR_EnableBkUpAccess();
  __HAL_RCC_LSEDRIVE_CONFIG(RCC_LSEDRIVE_LOW);
  /** Initializes the CPU, AHB and APB busses clocks
  */
  RCC_OscInitStruct.OscillatorType =
RCC_OSCILLATORTYPE_LSE|RCC_OSCILLATORTYPE_MSI;
  RCC_OscInitStruct.LSEState = RCC_LSE_ON;
  RCC_OscInitStruct.MSIState = RCC_MSI_ON;
  RCC_OscInitStruct.MSICalibrationValue = 0;
  RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_6;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_MSI;
  RCC_OscInitStruct.PLL.PLLM = 1;
  RCC_OscInitStruct.PLL.PLLN = 40;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV7;
  RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
  RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB busses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
```

```
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_4) != HAL_OK)
  {
    Error_Handler();
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_USART1|RCC_PERIPHCLK_USART3
                              |RCC_PERIPHCLK_I2C2|RCC_PERIPHCLK_DFSDM1
                              |RCC_PERIPHCLK_USB;
  PeriphClkInit.Usart1ClockSelection = RCC_USART1CLKSOURCE_PCLK2;
  PeriphClkInit.Usart3ClockSelection = RCC_USART3CLKSOURCE_PCLK1;
  PeriphClkInit.I2c2ClockSelection = RCC_I2C2CLKSOURCE_PCLK1;
  PeriphClkInit.Dfsdm1ClockSelection = RCC_DFSDM1CLKSOURCE_PCLK;
  PeriphClkInit.UsbClockSelection = RCC_USBCLKSOURCE_PLLSAI1;
  PeriphClkInit.PLLSAI1.PLLSAI1Source = RCC_PLLSOURCE_MSI;
  PeriphClkInit.PLLSAI1.PLLSAI1M = 1;
  PeriphClkInit.PLLSAI1.PLLSAI1N = 24;
  PeriphClkInit.PLLSAI1.PLLSAI1P = RCC_PLLP_DIV7;
  PeriphClkInit.PLLSAI1.PLLSAI1Q = RCC_PLLQ_DIV2;
  PeriphClkInit.PLLSAI1.PLLSAI1R = RCC_PLLR_DIV2;
  PeriphClkInit.PLLSAI1.PLLSAI1ClockOut = RCC_PLLSAI1_48M2CLK;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure the main internal regulator output voltage
  */
  if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
  {
    Error_Handler();
  }
  /** Enable MSI Auto calibration
  */
  HAL_RCCEx_EnableMSIPLLMode();
}

/**
  * @brief DFSDM1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_DFSDM1_Init(void)
{

  /* USER CODE BEGIN DFSDM1_Init 0 */

  /* USER CODE END DFSDM1_Init 0 */

  /* USER CODE BEGIN DFSDM1_Init 1 */

  /* USER CODE END DFSDM1_Init 1 */
  hdfsdm1_channel1.Instance = DFSDM1_Channel1;
```

```
  hdfsdm1_channel1.Init.OutputClock.Activation = ENABLE;
  hdfsdm1_channel1.Init.OutputClock.Selection =
DFSDM_CHANNEL_OUTPUT_CLOCK_SYSTEM;
  hdfsdm1_channel1.Init.OutputClock.Divider = 2;
  hdfsdm1_channel1.Init.Input.Multiplexer = DFSDM_CHANNEL_EXTERNAL_INPUTS;
  hdfsdm1_channel1.Init.Input.DataPacking = DFSDM_CHANNEL_STANDARD_MODE;
  hdfsdm1_channel1.Init.Input.Pins = DFSDM_CHANNEL_FOLLOWING_CHANNEL_PINS;
  hdfsdm1_channel1.Init.SerialInterface.Type = DFSDM_CHANNEL_SPI_RISING;
  hdfsdm1_channel1.Init.SerialInterface.SpiClock =
DFSDM_CHANNEL_SPI_CLOCK_INTERNAL;
  hdfsdm1_channel1.Init.Awd.FilterOrder = DFSDM_CHANNEL_FASTSINC_ORDER;
  hdfsdm1_channel1.Init.Awd.Oversampling = 1;
  hdfsdm1_channel1.Init.Offset = 0;
  hdfsdm1_channel1.Init.RightBitShift = 0x00;
  if (HAL_DFSDM_ChannelInit(&hdfsdm1_channel1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN DFSDM1_Init 2 */

  /* USER CODE END DFSDM1_Init 2 */

}

/**
  * @brief I2C2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C2_Init(void)
{

  /* USER CODE BEGIN I2C2_Init 0 */

  /* USER CODE END I2C2_Init 0 */

  /* USER CODE BEGIN I2C2_Init 1 */

  /* USER CODE END I2C2_Init 1 */
  hi2c2.Instance = I2C2;
  hi2c2.Init.Timing = 0x00000E14;
  hi2c2.Init.OwnAddress1 = 0;
  hi2c2.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c2.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c2.Init.OwnAddress2 = 0;
  hi2c2.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
  hi2c2.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c2.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c2) != HAL_OK)
  {
    Error_Handler();
```

```
  }
  /** Configure Analogue filter
  */
  if (HAL_I2CEx_ConfigAnalogFilter(&hi2c2, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure Digital filter
  */
  if (HAL_I2CEx_ConfigDigitalFilter(&hi2c2, 0) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C2_Init 2 */

  /* USER CODE END I2C2_Init 2 */

}

/**
  * @brief QUADSPI Initialization Function
  * @param None
  * @retval None
  */
static void MX_QUADSPI_Init(void)
{

  /* USER CODE BEGIN QUADSPI_Init 0 */

  /* USER CODE END QUADSPI_Init 0 */

  /* USER CODE BEGIN QUADSPI_Init 1 */

  /* USER CODE END QUADSPI_Init 1 */
  /* QUADSPI parameter configuration*/
  hqspi.Instance = QUADSPI;
  hqspi.Init.ClockPrescaler = 255;
  hqspi.Init.FifoThreshold = 1;
  hqspi.Init.SampleShifting = QSPI_SAMPLE_SHIFTING_NONE;
  hqspi.Init.FlashSize = 1;
  hqspi.Init.ChipSelectHighTime = QSPI_CS_HIGH_TIME_1_CYCLE;
  hqspi.Init.ClockMode = QSPI_CLOCK_MODE_0;
  if (HAL_QSPI_Init(&hqspi) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN QUADSPI_Init 2 */

  /* USER CODE END QUADSPI_Init 2 */

}
```

```c
/**
  * @brief SPI3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_SPI3_Init(void)
{

  /* USER CODE BEGIN SPI3_Init 0 */

  /* USER CODE END SPI3_Init 0 */

  /* USER CODE BEGIN SPI3_Init 1 */

  /* USER CODE END SPI3_Init 1 */
  /* SPI3 parameter configuration*/
  hspi3.Instance = SPI3;
  hspi3.Init.Mode = SPI_MODE_MASTER;
  hspi3.Init.Direction = SPI_DIRECTION_2LINES;
  hspi3.Init.DataSize = SPI_DATASIZE_4BIT;
  hspi3.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi3.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi3.Init.NSS = SPI_NSS_SOFT;
  hspi3.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
  hspi3.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi3.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi3.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi3.Init.CRCPolynomial = 7;
  hspi3.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
  hspi3.Init.NSSPMode = SPI_NSS_PULSE_ENABLE;
  if (HAL_SPI_Init(&hspi3) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN SPI3_Init 2 */

  /* USER CODE END SPI3_Init 2 */

}

/**
  * @brief USART1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART1_UART_Init(void)
{

  /* USER CODE BEGIN USART1_Init 0 */
```

```c
  /* USER CODE END USART1_Init 0 */

  /* USER CODE BEGIN USART1_Init 1 */

  /* USER CODE END USART1_Init 1 */
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  huart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART1_Init 2 */

  /* USER CODE END USART1_Init 2 */

}

/**
  * @brief USART3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART3_UART_Init(void)
{

  /* USER CODE BEGIN USART3_Init 0 */

  /* USER CODE END USART3_Init 0 */

  /* USER CODE BEGIN USART3_Init 1 */

  /* USER CODE END USART3_Init 1 */
  huart3.Instance = USART3;
  huart3.Init.BaudRate = 115200;
  huart3.Init.WordLength = UART_WORDLENGTH_8B;
  huart3.Init.StopBits = UART_STOPBITS_1;
  huart3.Init.Parity = UART_PARITY_NONE;
  huart3.Init.Mode = UART_MODE_TX_RX;
  huart3.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart3.Init.OverSampling = UART_OVERSAMPLING_16;
  huart3.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart3.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart3) != HAL_OK)
```

```
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART3_Init 2 */

  /* USER CODE END USART3_Init 2 */

}

/**
  * @brief USB_OTG_FS Initialization Function
  * @param None
  * @retval None
  */
static void MX_USB_OTG_FS_PCD_Init(void)
{

  /* USER CODE BEGIN USB_OTG_FS_Init 0 */

  /* USER CODE END USB_OTG_FS_Init 0 */

  /* USER CODE BEGIN USB_OTG_FS_Init 1 */

  /* USER CODE END USB_OTG_FS_Init 1 */
  hpcd_USB_OTG_FS.Instance = USB_OTG_FS;
  hpcd_USB_OTG_FS.Init.dev_endpoints = 6;
  hpcd_USB_OTG_FS.Init.speed = PCD_SPEED_FULL;
  hpcd_USB_OTG_FS.Init.phy_itface = PCD_PHY_EMBEDDED;
  hpcd_USB_OTG_FS.Init.Sof_enable = DISABLE;
  hpcd_USB_OTG_FS.Init.low_power_enable = DISABLE;
  hpcd_USB_OTG_FS.Init.lpm_enable = DISABLE;
  hpcd_USB_OTG_FS.Init.battery_charging_enable = DISABLE;
  hpcd_USB_OTG_FS.Init.use_dedicated_ep1 = DISABLE;
  hpcd_USB_OTG_FS.Init.vbus_sensing_enable = DISABLE;
  if (HAL_PCD_Init(&hpcd_USB_OTG_FS) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USB_OTG_FS_Init 2 */

  /* USER CODE END USB_OTG_FS_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
```

```
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOE_CLK_ENABLE();
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOE,
M24SR64_Y_RF_DISABLE_Pin|M24SR64_Y_GPO_Pin|ISM43362_RST_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, ARD_D10_Pin|SPBTLE_RF_RST_Pin|ARD_D9_Pin,
GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB,
ARD_D8_Pin|ISM43362_BOOT0_Pin|ISM43362_WAKEUP_Pin|LED2_Pin
                          |SPSGRF_915_SDN_Pin|ARD_D5_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOD,
USB_OTG_FS_PWR_EN_Pin|PMOD_RESET_Pin|STSAFE_A100_RESET_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(SPBTLE_RF_SPI3_CSN_GPIO_Port, SPBTLE_RF_SPI3_CSN_Pin,
GPIO_PIN_SET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC, VL53L0X_XSHUT_Pin|LED3_WIFI__LED4_BLE_Pin,
GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(SPSGRF_915_SPI3_CSN_GPIO_Port, SPSGRF_915_SPI3_CSN_Pin,
GPIO_PIN_SET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(ISM43362_SPI3_CSN_GPIO_Port, ISM43362_SPI3_CSN_Pin,
GPIO_PIN_SET);

  /*Configure GPIO pins : M24SR64_Y_RF_DISABLE_Pin M24SR64_Y_GPO_Pin
ISM43362_RST_Pin ISM43362_SPI3_CSN_Pin */
  GPIO_InitStruct.Pin =
M24SR64_Y_RF_DISABLE_Pin|M24SR64_Y_GPO_Pin|ISM43362_RST_Pin|ISM43362_SPI3_CSN_Pi
n;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);
```

```
  /*Configure GPIO pins : USB_OTG_FS_OVRCR_EXTI3_Pin SPSGRF_915_GPIO3_EXTI5_Pin
SPBTLE_RF_IRQ_EXTI6_Pin ISM43362_DRDY_EXTI1_Pin */
  GPIO_InitStruct.Pin =
USB_OTG_FS_OVRCR_EXTI3_Pin|SPSGRF_915_GPIO3_EXTI5_Pin|SPBTLE_RF_IRQ_EXTI6_Pin|IS
M43362_DRDY_EXTI1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);

  /*Configure GPIO pin : BUTTON_EXTI13_Pin */
  GPIO_InitStruct.Pin = BUTTON_EXTI13_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(BUTTON_EXTI13_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : ARD_A5_Pin ARD_A4_Pin ARD_A3_Pin ARD_A2_Pin
                           ARD_A1_Pin ARD_A0_Pin */
  GPIO_InitStruct.Pin = ARD_A5_Pin|ARD_A4_Pin|ARD_A3_Pin|ARD_A2_Pin
                        |ARD_A1_Pin|ARD_A0_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_ANALOG_ADC_CONTROL;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pins : ARD_D1_Pin ARD_D0_Pin */
  GPIO_InitStruct.Pin = ARD_D1_Pin|ARD_D0_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Alternate = GPIO_AF8_UART4;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pins : ARD_D10_Pin SPBTLE_RF_RST_Pin ARD_D9_Pin */
  GPIO_InitStruct.Pin = ARD_D10_Pin|SPBTLE_RF_RST_Pin|ARD_D9_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pin : ARD_D4_Pin */
  GPIO_InitStruct.Pin = ARD_D4_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  GPIO_InitStruct.Alternate = GPIO_AF1_TIM2;
  HAL_GPIO_Init(ARD_D4_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : ARD_D7_Pin */
  GPIO_InitStruct.Pin = ARD_D7_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_ANALOG_ADC_CONTROL;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
```

```
  HAL_GPIO_Init(ARD_D7_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : ARD_D13_Pin ARD_D12_Pin ARD_D11_Pin */
  GPIO_InitStruct.Pin = ARD_D13_Pin|ARD_D12_Pin|ARD_D11_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Alternate = GPIO_AF5_SPI1;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pin : ARD_D3_Pin */
  GPIO_InitStruct.Pin = ARD_D3_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(ARD_D3_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : ARD_D6_Pin */
  GPIO_InitStruct.Pin = ARD_D6_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_ANALOG_ADC_CONTROL;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(ARD_D6_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : ARD_D8_Pin ISM43362_BOOT0_Pin ISM43362_WAKEUP_Pin
LED2_Pin
                           SPSGRF_915_SDN_Pin ARD_D5_Pin SPSGRF_915_SPI3_CSN_Pin
*/
  GPIO_InitStruct.Pin =
ARD_D8_Pin|ISM43362_BOOT0_Pin|ISM43362_WAKEUP_Pin|LED2_Pin

|SPSGRF_915_SDN_Pin|ARD_D5_Pin|SPSGRF_915_SPI3_CSN_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /*Configure GPIO pins : LPS22HB_INT_DRDY_EXTI0_Pin LSM6DSL_INT1_EXTI11_Pin
ARD_D2_Pin HTS221_DRDY_EXTI15_Pin
                           PMOD_IRQ_EXTI12_Pin */
  GPIO_InitStruct.Pin =
LPS22HB_INT_DRDY_EXTI0_Pin|LSM6DSL_INT1_EXTI11_Pin|ARD_D2_Pin|HTS221_DRDY_EXTI15
_Pin
                          |PMOD_IRQ_EXTI12_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  /*Configure GPIO pins : USB_OTG_FS_PWR_EN_Pin SPBTLE_RF_SPI3_CSN_Pin
PMOD_RESET_Pin STSAFE_A100_RESET_Pin */
  GPIO_InitStruct.Pin =
USB_OTG_FS_PWR_EN_Pin|SPBTLE_RF_SPI3_CSN_Pin|PMOD_RESET_Pin|STSAFE_A100_RESET_Pi
n;
```

```
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  /*Configure GPIO pins : VL53L0X_XSHUT_Pin LED3_WIFI__LED4_BLE_Pin */
  GPIO_InitStruct.Pin = VL53L0X_XSHUT_Pin|LED3_WIFI__LED4_BLE_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pins : VL53L0X_GPIO1_EXTI7_Pin LSM3MDL_DRDY_EXTI8_Pin */
  GPIO_InitStruct.Pin = VL53L0X_GPIO1_EXTI7_Pin|LSM3MDL_DRDY_EXTI8_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pin : PMOD_SPI2_SCK_Pin */
  GPIO_InitStruct.Pin = PMOD_SPI2_SCK_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Alternate = GPIO_AF5_SPI2;
  HAL_GPIO_Init(PMOD_SPI2_SCK_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : PMOD_UART2_CTS_Pin PMOD_UART2_RTS_Pin
PMOD_UART2_TX_Pin PMOD_UART2_RX_Pin */
  GPIO_InitStruct.Pin =
PMOD_UART2_CTS_Pin|PMOD_UART2_RTS_Pin|PMOD_UART2_TX_Pin|PMOD_UART2_RX_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Alternate = GPIO_AF7_USART2;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  /*Configure GPIO pins : ARD_D15_Pin ARD_D14_Pin */
  GPIO_InitStruct.Pin = ARD_D15_Pin|ARD_D14_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_OD;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Alternate = GPIO_AF4_I2C1;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);

  HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);
```

```
}

/* USER CODE BEGIN 4 */
PUTCHAR_PROTOTYPE
{
/* Place your implementation of fputc here */
/* e.g. write a character to the serial port and Loop until the end of
transmission */
while (HAL_OK != HAL_UART_Transmit(&huart1, (uint8_t *) &ch, 1, 30000))
{
;
}
return ch;
}



/**
* @brief Retargets the C library scanf function to the USART.
* @param None
* @retval None
*/
GETCHAR_PROTOTYPE
{
/* Place your implementation of fgetc here */
/* e.g. readwrite a character to the USART2 and Loop until the end of
transmission */
uint8_t ch = 0;
while (HAL_OK != HAL_UART_Receive(&huart1, (uint8_t *)&ch, 1, 30000))
{
;
}
return ch;
}
/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */

  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
```

```
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line
number,
     tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
*/
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

Reading Pressure Sensor Values Test

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2021 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */

/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include <stdio.h>

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "stm32l475e_iot01.h"
#include "stm32l475e_iot01_psensor.h"
#include "math.h"
```

```c
/* 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 ---------------------------------------------------------*/
DFSDM_Channel_HandleTypeDef hdfsdm1_channel1;

I2C_HandleTypeDef hi2c2;

QSPI_HandleTypeDef hqspi;

SPI_HandleTypeDef hspi3;

UART_HandleTypeDef huart1;
UART_HandleTypeDef huart3;

PCD_HandleTypeDef hpcd_USB_OTG_FS;

/* USER CODE BEGIN PV */
float pressure_value = 0;
char str_pressure[100] = "";
uint8_t msg1[] = "** Pressure values measurement **\n\n\r";
uint8_t msg2[] = "===> Initialise pressure sensor LPS22HB \r\n";
uint8_t msg3[] = "===> Pressure sensor LPS22HB initialised \r\n";
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DFSDM1_Init(void);
static void MX_I2C2_Init(void);
static void MX_QUADSPI_Init(void);
static void MX_SPI3_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_USART3_UART_Init(void);
static void MX_USB_OTG_FS_PCD_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */
```

```c
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
  uint32_t BSP_PSENSOR_Init(void);
  uint8_t  BSP_PSENSOR_ReadID(void);
  float    BSP_PSENSOR_ReadPressure(void);
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
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_DFSDM1_Init();
  MX_I2C2_Init();
  MX_QUADSPI_Init();
  MX_SPI3_Init();
  MX_USART1_UART_Init();
  MX_USART3_UART_Init();
  MX_USB_OTG_FS_PCD_Init();
  /* USER CODE BEGIN 2 */
  HAL_UART_Transmit(&huart1,msg1,sizeof(msg1),1000);
  HAL_UART_Transmit(&huart1,msg2,sizeof(msg2),1000);
  BSP_PSENSOR_Init();
  HAL_UART_Transmit(&huart1,msg3,sizeof(msg3),1000);
  /* USER CODE END 2 */
```

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

    /* USER CODE BEGIN 3 */
         pressure_value = BSP_PSENSOR_ReadPressure();
         int tmpInt1 = pressure_value;
         float tmpFrac = pressure_value - tmpInt1;
         int tmpInt2 = trunc(tmpFrac * 100);
         snprintf(str_pressure,100," PRESSURE = %d.%02d\n\r", tmpInt1, tmpInt2);
         HAL_UART_Transmit(&huart1,( uint8_t *
)str_pressure,sizeof(str_pressure),1000);
         HAL_Delay(1000);
  }

  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** Configure LSE Drive Capability
  */
  HAL_PWR_EnableBkUpAccess();
  __HAL_RCC_LSEDRIVE_CONFIG(RCC_LSEDRIVE_LOW);
  /** Initializes the CPU, AHB and APB busses clocks
  */
  RCC_OscInitStruct.OscillatorType =
RCC_OSCILLATORTYPE_LSE|RCC_OSCILLATORTYPE_MSI;
  RCC_OscInitStruct.LSEState = RCC_LSE_ON;
  RCC_OscInitStruct.MSIState = RCC_MSI_ON;
  RCC_OscInitStruct.MSICalibrationValue = 0;
  RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_6;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_MSI;
  RCC_OscInitStruct.PLL.PLLM = 1;
  RCC_OscInitStruct.PLL.PLLN = 40;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV7;
  RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
  RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
```

```
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB busses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_4) != HAL_OK)
  {
    Error_Handler();
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_USART1|RCC_PERIPHCLK_USART3
                              |RCC_PERIPHCLK_I2C2|RCC_PERIPHCLK_DFSDM1
                              |RCC_PERIPHCLK_USB;
  PeriphClkInit.Usart1ClockSelection = RCC_USART1CLKSOURCE_PCLK2;
  PeriphClkInit.Usart3ClockSelection = RCC_USART3CLKSOURCE_PCLK1;
  PeriphClkInit.I2c2ClockSelection = RCC_I2C2CLKSOURCE_PCLK1;
  PeriphClkInit.Dfsdm1ClockSelection = RCC_DFSDM1CLKSOURCE_PCLK;
  PeriphClkInit.UsbClockSelection = RCC_USBCLKSOURCE_PLLSAI1;
  PeriphClkInit.PLLSAI1.PLLSAI1Source = RCC_PLLSOURCE_MSI;
  PeriphClkInit.PLLSAI1.PLLSAI1M = 1;
  PeriphClkInit.PLLSAI1.PLLSAI1N = 24;
  PeriphClkInit.PLLSAI1.PLLSAI1P = RCC_PLLP_DIV7;
  PeriphClkInit.PLLSAI1.PLLSAI1Q = RCC_PLLQ_DIV2;
  PeriphClkInit.PLLSAI1.PLLSAI1R = RCC_PLLR_DIV2;
  PeriphClkInit.PLLSAI1.PLLSAI1ClockOut = RCC_PLLSAI1_48M2CLK;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure the main internal regulator output voltage
  */
  if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
  {
    Error_Handler();
  }
  /** Enable MSI Auto calibration
  */
  HAL_RCCEx_EnableMSIPLLMode();
}

/**
  * @brief DFSDM1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_DFSDM1_Init(void)
```

```
{

  /* USER CODE BEGIN DFSDM1_Init 0 */

  /* USER CODE END DFSDM1_Init 0 */

  /* USER CODE BEGIN DFSDM1_Init 1 */

  /* USER CODE END DFSDM1_Init 1 */
  hdfsdm1_channel1.Instance = DFSDM1_Channel1;
  hdfsdm1_channel1.Init.OutputClock.Activation = ENABLE;
  hdfsdm1_channel1.Init.OutputClock.Selection =
DFSDM_CHANNEL_OUTPUT_CLOCK_SYSTEM;
  hdfsdm1_channel1.Init.OutputClock.Divider = 2;
  hdfsdm1_channel1.Init.Input.Multiplexer = DFSDM_CHANNEL_EXTERNAL_INPUTS;
  hdfsdm1_channel1.Init.Input.DataPacking = DFSDM_CHANNEL_STANDARD_MODE;
  hdfsdm1_channel1.Init.Input.Pins = DFSDM_CHANNEL_FOLLOWING_CHANNEL_PINS;
  hdfsdm1_channel1.Init.SerialInterface.Type = DFSDM_CHANNEL_SPI_RISING;
  hdfsdm1_channel1.Init.SerialInterface.SpiClock =
DFSDM_CHANNEL_SPI_CLOCK_INTERNAL;
  hdfsdm1_channel1.Init.Awd.FilterOrder = DFSDM_CHANNEL_FASTSINC_ORDER;
  hdfsdm1_channel1.Init.Awd.Oversampling = 1;
  hdfsdm1_channel1.Init.Offset = 0;
  hdfsdm1_channel1.Init.RightBitShift = 0x00;
  if (HAL_DFSDM_ChannelInit(&hdfsdm1_channel1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN DFSDM1_Init 2 */

  /* USER CODE END DFSDM1_Init 2 */

}

/**
  * @brief I2C2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C2_Init(void)
{

  /* USER CODE BEGIN I2C2_Init 0 */

  /* USER CODE END I2C2_Init 0 */

  /* USER CODE BEGIN I2C2_Init 1 */

  /* USER CODE END I2C2_Init 1 */
  hi2c2.Instance = I2C2;
  hi2c2.Init.Timing = 0x00000E14;
```

```
  hi2c2.Init.OwnAddress1 = 0;
  hi2c2.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c2.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c2.Init.OwnAddress2 = 0;
  hi2c2.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
  hi2c2.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c2.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c2) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure Analogue filter
  */
  if (HAL_I2CEx_ConfigAnalogFilter(&hi2c2, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure Digital filter
  */
  if (HAL_I2CEx_ConfigDigitalFilter(&hi2c2, 0) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C2_Init 2 */

  /* USER CODE END I2C2_Init 2 */

}

/**
  * @brief QUADSPI Initialization Function
  * @param None
  * @retval None
  */
static void MX_QUADSPI_Init(void)
{

  /* USER CODE BEGIN QUADSPI_Init 0 */

  /* USER CODE END QUADSPI_Init 0 */

  /* USER CODE BEGIN QUADSPI_Init 1 */

  /* USER CODE END QUADSPI_Init 1 */
  /* QUADSPI parameter configuration*/
  hqspi.Instance = QUADSPI;
  hqspi.Init.ClockPrescaler = 255;
  hqspi.Init.FifoThreshold = 1;
  hqspi.Init.SampleShifting = QSPI_SAMPLE_SHIFTING_NONE;
  hqspi.Init.FlashSize = 1;
  hqspi.Init.ChipSelectHighTime = QSPI_CS_HIGH_TIME_1_CYCLE;
```

```
  hqspi.Init.ClockMode = QSPI_CLOCK_MODE_0;
  if (HAL_QSPI_Init(&hqspi) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN QUADSPI_Init 2 */

  /* USER CODE END QUADSPI_Init 2 */

}

/**
  * @brief SPI3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_SPI3_Init(void)
{

  /* USER CODE BEGIN SPI3_Init 0 */

  /* USER CODE END SPI3_Init 0 */

  /* USER CODE BEGIN SPI3_Init 1 */

  /* USER CODE END SPI3_Init 1 */
  /* SPI3 parameter configuration*/
  hspi3.Instance = SPI3;
  hspi3.Init.Mode = SPI_MODE_MASTER;
  hspi3.Init.Direction = SPI_DIRECTION_2LINES;
  hspi3.Init.DataSize = SPI_DATASIZE_4BIT;
  hspi3.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi3.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi3.Init.NSS = SPI_NSS_SOFT;
  hspi3.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
  hspi3.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi3.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi3.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi3.Init.CRCPolynomial = 7;
  hspi3.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
  hspi3.Init.NSSPMode = SPI_NSS_PULSE_ENABLE;
  if (HAL_SPI_Init(&hspi3) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN SPI3_Init 2 */

  /* USER CODE END SPI3_Init 2 */

}
```

```c
/**
  * @brief USART1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART1_UART_Init(void)
{

  /* USER CODE BEGIN USART1_Init 0 */

  /* USER CODE END USART1_Init 0 */

  /* USER CODE BEGIN USART1_Init 1 */

  /* USER CODE END USART1_Init 1 */
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  huart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART1_Init 2 */

  /* USER CODE END USART1_Init 2 */

}

/**
  * @brief USART3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART3_UART_Init(void)
{

  /* USER CODE BEGIN USART3_Init 0 */

  /* USER CODE END USART3_Init 0 */

  /* USER CODE BEGIN USART3_Init 1 */

  /* USER CODE END USART3_Init 1 */
  huart3.Instance = USART3;
```

```
  huart3.Init.BaudRate = 115200;
  huart3.Init.WordLength = UART_WORDLENGTH_8B;
  huart3.Init.StopBits = UART_STOPBITS_1;
  huart3.Init.Parity = UART_PARITY_NONE;
  huart3.Init.Mode = UART_MODE_TX_RX;
  huart3.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart3.Init.OverSampling = UART_OVERSAMPLING_16;
  huart3.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart3.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart3) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART3_Init 2 */

  /* USER CODE END USART3_Init 2 */

}

/**
  * @brief USB_OTG_FS Initialization Function
  * @param None
  * @retval None
  */
static void MX_USB_OTG_FS_PCD_Init(void)
{

  /* USER CODE BEGIN USB_OTG_FS_Init 0 */

  /* USER CODE END USB_OTG_FS_Init 0 */

  /* USER CODE BEGIN USB_OTG_FS_Init 1 */

  /* USER CODE END USB_OTG_FS_Init 1 */
  hpcd_USB_OTG_FS.Instance = USB_OTG_FS;
  hpcd_USB_OTG_FS.Init.dev_endpoints = 6;
  hpcd_USB_OTG_FS.Init.speed = PCD_SPEED_FULL;
  hpcd_USB_OTG_FS.Init.phy_itface = PCD_PHY_EMBEDDED;
  hpcd_USB_OTG_FS.Init.Sof_enable = DISABLE;
  hpcd_USB_OTG_FS.Init.low_power_enable = DISABLE;
  hpcd_USB_OTG_FS.Init.lpm_enable = DISABLE;
  hpcd_USB_OTG_FS.Init.battery_charging_enable = DISABLE;
  hpcd_USB_OTG_FS.Init.use_dedicated_ep1 = DISABLE;
  hpcd_USB_OTG_FS.Init.vbus_sensing_enable = DISABLE;
  if (HAL_PCD_Init(&hpcd_USB_OTG_FS) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USB_OTG_FS_Init 2 */

  /* USER CODE END USB_OTG_FS_Init 2 */
```

```
}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOE_CLK_ENABLE();
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOE,
M24SR64_Y_RF_DISABLE_Pin|M24SR64_Y_GPO_Pin|ISM43362_RST_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, ARD_D10_Pin|SPBTLE_RF_RST_Pin|ARD_D9_Pin,
GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB,
ARD_D8_Pin|ISM43362_BOOT0_Pin|ISM43362_WAKEUP_Pin|LED2_Pin
                          |SPSGRF_915_SDN_Pin|ARD_D5_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOD,
USB_OTG_FS_PWR_EN_Pin|PMOD_RESET_Pin|STSAFE_A100_RESET_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(SPBTLE_RF_SPI3_CSN_GPIO_Port, SPBTLE_RF_SPI3_CSN_Pin,
GPIO_PIN_SET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC, VL53L0X_XSHUT_Pin|LED3_WIFI__LED4_BLE_Pin,
GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(SPSGRF_915_SPI3_CSN_GPIO_Port, SPSGRF_915_SPI3_CSN_Pin,
GPIO_PIN_SET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(ISM43362_SPI3_CSN_GPIO_Port, ISM43362_SPI3_CSN_Pin,
GPIO_PIN_SET);
```

```
  /*Configure GPIO pins : M24SR64_Y_RF_DISABLE_Pin M24SR64_Y_GPO_Pin
ISM43362_RST_Pin ISM43362_SPI3_CSN_Pin */
  GPIO_InitStruct.Pin =
M24SR64_Y_RF_DISABLE_Pin|M24SR64_Y_GPO_Pin|ISM43362_RST_Pin|ISM43362_SPI3_CSN_Pi
n;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);

  /*Configure GPIO pins : USB_OTG_FS_OVRCR_EXTI3_Pin SPSGRF_915_GPIO3_EXTI5_Pin
SPBTLE_RF_IRQ_EXTI6_Pin ISM43362_DRDY_EXTI1_Pin */
  GPIO_InitStruct.Pin =
USB_OTG_FS_OVRCR_EXTI3_Pin|SPSGRF_915_GPIO3_EXTI5_Pin|SPBTLE_RF_IRQ_EXTI6_Pin|IS
M43362_DRDY_EXTI1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOE, &GPIO_InitStruct);

  /*Configure GPIO pin : BUTTON_EXTI13_Pin */
  GPIO_InitStruct.Pin = BUTTON_EXTI13_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(BUTTON_EXTI13_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : ARD_A5_Pin ARD_A4_Pin ARD_A3_Pin ARD_A2_Pin
                           ARD_A1_Pin ARD_A0_Pin */
  GPIO_InitStruct.Pin = ARD_A5_Pin|ARD_A4_Pin|ARD_A3_Pin|ARD_A2_Pin
                        |ARD_A1_Pin|ARD_A0_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_ANALOG_ADC_CONTROL;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pins : ARD_D1_Pin ARD_D0_Pin */
  GPIO_InitStruct.Pin = ARD_D1_Pin|ARD_D0_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Alternate = GPIO_AF8_UART4;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pins : ARD_D10_Pin SPBTLE_RF_RST_Pin ARD_D9_Pin */
  GPIO_InitStruct.Pin = ARD_D10_Pin|SPBTLE_RF_RST_Pin|ARD_D9_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pin : ARD_D4_Pin */
  GPIO_InitStruct.Pin = ARD_D4_Pin;
```

```
  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  GPIO_InitStruct.Alternate = GPIO_AF1_TIM2;
  HAL_GPIO_Init(ARD_D4_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : ARD_D7_Pin */
  GPIO_InitStruct.Pin = ARD_D7_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_ANALOG_ADC_CONTROL;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(ARD_D7_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : ARD_D13_Pin ARD_D12_Pin ARD_D11_Pin */
  GPIO_InitStruct.Pin = ARD_D13_Pin|ARD_D12_Pin|ARD_D11_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Alternate = GPIO_AF5_SPI1;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pin : ARD_D3_Pin */
  GPIO_InitStruct.Pin = ARD_D3_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(ARD_D3_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : ARD_D6_Pin */
  GPIO_InitStruct.Pin = ARD_D6_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_ANALOG_ADC_CONTROL;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(ARD_D6_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : ARD_D8_Pin ISM43362_BOOT0_Pin ISM43362_WAKEUP_Pin
LED2_Pin
                           SPSGRF_915_SDN_Pin ARD_D5_Pin SPSGRF_915_SPI3_CSN_Pin
*/
  GPIO_InitStruct.Pin =
ARD_D8_Pin|ISM43362_BOOT0_Pin|ISM43362_WAKEUP_Pin|LED2_Pin

|SPSGRF_915_SDN_Pin|ARD_D5_Pin|SPSGRF_915_SPI3_CSN_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /*Configure GPIO pins : LPS22HB_INT_DRDY_EXTI0_Pin LSM6DSL_INT1_EXTI11_Pin
ARD_D2_Pin HTS221_DRDY_EXTI15_Pin
                           PMOD_IRQ_EXTI12_Pin */
  GPIO_InitStruct.Pin =
LPS22HB_INT_DRDY_EXTI0_Pin|LSM6DSL_INT1_EXTI11_Pin|ARD_D2_Pin|HTS221_DRDY_EXTI15
_Pin
```

```
                          |PMOD_IRQ_EXTI12_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  /*Configure GPIO pins : USB_OTG_FS_PWR_EN_Pin SPBTLE_RF_SPI3_CSN_Pin
PMOD_RESET_Pin STSAFE_A100_RESET_Pin */
  GPIO_InitStruct.Pin =
USB_OTG_FS_PWR_EN_Pin|SPBTLE_RF_SPI3_CSN_Pin|PMOD_RESET_Pin|STSAFE_A100_RESET_Pi
n;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  /*Configure GPIO pins : VL53L0X_XSHUT_Pin LED3_WIFI__LED4_BLE_Pin */
  GPIO_InitStruct.Pin = VL53L0X_XSHUT_Pin|LED3_WIFI__LED4_BLE_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pins : VL53L0X_GPIO1_EXTI7_Pin LSM3MDL_DRDY_EXTI8_Pin */
  GPIO_InitStruct.Pin = VL53L0X_GPIO1_EXTI7_Pin|LSM3MDL_DRDY_EXTI8_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pin : PMOD_SPI2_SCK_Pin */
  GPIO_InitStruct.Pin = PMOD_SPI2_SCK_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Alternate = GPIO_AF5_SPI2;
  HAL_GPIO_Init(PMOD_SPI2_SCK_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : PMOD_UART2_CTS_Pin PMOD_UART2_RTS_Pin
PMOD_UART2_TX_Pin PMOD_UART2_RX_Pin */
  GPIO_InitStruct.Pin =
PMOD_UART2_CTS_Pin|PMOD_UART2_RTS_Pin|PMOD_UART2_TX_Pin|PMOD_UART2_RX_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
  GPIO_InitStruct.Alternate = GPIO_AF7_USART2;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  /*Configure GPIO pins : ARD_D15_Pin ARD_D14_Pin */
  GPIO_InitStruct.Pin = ARD_D15_Pin|ARD_D14_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_OD;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
```

```
  GPIO_InitStruct.Alternate = GPIO_AF4_I2C1;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);

  HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */

  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line
number,
     tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
*/
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

10.2 Main code (current state)

```
/*
 * configuration.h
 *
 *      Author: tomas
 */

#ifndef INC_CONFIGURATION_H_
#define INC_CONFIGURATION_H_

#include <stdbool.h>
#include <stdlib.h>
#include "sd-card.h"

bool loadConfig(int * lives, float * diffMult);

#endif /* INC_CONFIGURATION_H_ */
```

```
/*
 * configuration.c
 *
 *      Author: tomas
 */

#include "configuration.h"

bool loadConfig(int * lives, float * diffMult){
	#if USE_SD_CARD == 1
		FIL file;
		int res = openFile(&file, "config.txt", false);
		if(res){
			return false;
		}
		char data[30];
		res = readLine(file, data, 30);
		if(!res){
			return false;
		}
		*lives = atoi(data);
		res = readLine(file, data, 30);
		if(!res){
			return false;
		}
		*diffMult = atof(data);
		closeFile(&file);
	#endif
	return true;
}
```

```c
/*
 * device-configs.h
 *
 *      Author: tomas
 */

#ifndef INC_DEVICE_CONFIGS_H_
#define INC_DEVICE_CONFIGS_H_

// When you have the display with you, simply change (on your computer) the
value to 1
#define USE_DISPLAY 0

#define USE_JOYSTICK 0

#define USE_SD_CARD 0

#endif /* INC_DEVICE_CONFIGS_H_ */
```

```c
/*
 * device-libs.h
 *
 *      Author: tomas
 */

#ifndef INC_DEVICE_LIBS_H_
#define INC_DEVICE_LIBS_H_

#include "device-configs.h"

#include "humidity.h"
#include "joystick.h"
#include "sd-card.h"

#include "gyroscope.h"
#include "stm32l475e_iot01.h"
// Include all of our custom libraries here

/*
 * Each library have to have a safeguard
 * Humidity has safeguard INC_HUMIDITY_H_
 * Example: display has safeguard INC_DISPLAY_H_
 * See humidity-tests.h and humidity-tests.c for usage
 */


int initAllDevices();

#endif /* INC_DEVICE_LIBS_H_ */
```

```c
/*
 * device-libs.c
 *
 *      Author: tomas
 */

#include "device-libs.h"


int initAllDevices(){
	HTS221Init();
	HTS221On();

	#if USE_SD_CARD == 1
		if(SDMount() != 0){
			return 1;
		}
	#endif
	return 0;
}
```

```c
/*
 * fonts.h
 *
 */
#ifndef __FONT_H
#define __FONT_H

#include "stdint.h"

typedef struct {
    const uint8_t width;
    uint8_t height;
    const uint16_t *data;
} FontDef;

//Font lib.
extern FontDef Font_7x10;
extern FontDef Font_11x18;
extern FontDef Font_16x26;

//16-bit(RGB565) Image lib.
/*****************************************
 *             CAUTION:
 *   If the MCU onchip flash cannot
 *  store such huge image data,please
 *           do not use it.
 * These pics are for test purpose only.
 *****************************************/
```

```
/* 128x128 pixel RGB565 image */
extern const uint16_t saber[][128];

/* 240x240 pixel RGB565 image
extern const uint16_t knky[][240];
extern const uint16_t tek[][240];
extern const uint16_t adi1[][240];
*/
#endif
```

```
/*
 * game.h
 *
 *      Author: tomas
 */

#ifndef INC_GAME_H_
#define INC_GAME_H_

#include "test_functions.h"
#include "task_functions.h"
#include "game-timer.h"
#include "device-libs.h"
#include "leaderboard.h"
#include "task-results.h"
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

#define ILLEGAL_TASK_RESULT_ERROR 123


enum state {
	STARTUP_STATE,
	MENU_STATE,
	GAME_CHOICE_STATE,
	TASK_RUNNING_STATE,
	TASK_START_STATE,
	TASK_RETRY_STATE,
	GAME_END_STATE,
	LEADERBOARD_STATE,
	CREDITS_STATE
};

typedef enum state state_t;

int runGame();
```

```
#endif /* INC_GAME_H_ */
```

```
/*
 * game.c
 *
 *      Author: tomas
 */

#include "game.h"

void showStartupError(int code){
	// print different messages regarding the startup error code from
runAllTests function
	#if USE_DISPLAY == 0
		switch(code){
			default:
				printf("\rGeneral error!\n");
		}
	#else
		switch(code){
			default:
				break;
		}
	#endif
}

void showRunError(int code){
	#if USE_DISPLAY == 0
		switch(code){
			case ILLEGAL_TASK_RESULT_ERROR:
				printf("\rTask return illegal state as its
result!\n");
				break;
			default:
				printf("\rGeneral error!\n");
		}
	#else
		switch(code){
			default:
				break;
		}
	#endif
}

int showMenuPage(int page, state_t state){
	switch(state){
		case MENU_STATE:
			switch(page){
				case 0:
					//new game
```

```
                                #if USE_DISPLAY == 0
                                        printf("\rNew game\n");
                                #endif
                                break;
                        case 1:
                                //leaderboard
                                #if USE_DISPLAY == 0
                                        printf("\rLeaderboard\n");
                                #endif
                                break;
                        case 2:
                                //credits
                                #if USE_DISPLAY == 0
                                        printf("\rCredits\n");
                                #endif
                                break;
                }
                break;
        case GAME_CHOICE_STATE:
                switch(page){
                        case 0:
                                // endless run
                                break;
                        case 1:
                                // return to main menu
                                break;
                }
                break;
        default:
                break;
    }
    return 0;
}

int showStartupScreen(){
    #if USE_DISPLAY == 1

    #else
        printf("\rStartup\n");
    #endif
    return 0;
}

int showLeaderboardScreen(){
    return 0;
}

int showCreditsScreen(){
    return 0;
}
```

```
int showGameEndScreen(int score){
	#if USE_DISPLAY == 1

	#else
		printf("\rGame over! Final score: %d\n", score);
	#endif
	return 0;
}

int showRetryScreen(int lives, bool choice){
	#if USE_DISPLAY == 1

	#else
		printf("\rYou have %d lives left. Do you wish to repeat current
task?\n", lives);
		printf("\r%s\n", choice ? "YES" : "NO");
	#endif
	return 0;
}

bool showTaskInfo(int task){
	switch(task){
		case 0:
			humidityShowTaskInfo();
			return true;
		case 1:
			return false;
		case 2:
			return false;
		case 3:
			return false;
		case 4:
			return false;
		case 5:
			return false;
		default:
			return false;
	}
}

bool navigateThroughMenu(int * currentPage, int pageCount, bool sigLeft, bool
sigRight){
	if(sigLeft){
		if(*currentPage == 0){
			*currentPage = pageCount;
		}
		(*currentPage) --;
		return true;
	}
	if(sigRight){
		(*currentPage) ++;
```

```
		if(*currentPage == pageCount){
			*currentPage = 0;
		}
		return true;
	}
	return false;
}

int chooseRandomTask(){
	return 0;//return rand() % 6;
}

task_result_t startTask(int task, float difficulty){
	// here would be switch with all the tasks, each with number from 0 to n
	// tasks that are blocking (task function has it's own timing and ends with result) should return TASK_FAILED or TASK_PASSED
	// tasks that use main timing should return TASK_RUNNING, and will be afterwards called on loop
	switch(task){
		case 0:
			return humidityStartTask(difficulty);
		case 1:
			return TASK_PASSED;
		case 2:
			return TASK_PASSED;
		case 3:
			return TASK_PASSED;
		case 4:
			return TASK_PASSED;
		case 5:
			return TASK_PASSED;
		default:
			return TASK_PASSED;
	}
}

task_result_t checkRunningTask(int task){
	// Here in-loop running tasks will be called until they return either TASK_FAILED or TASK_PASSED
	switch(task){
		case 0:
			return humidityCheckTaskState();
		case 1:
			return TASK_PASSED;
		case 2:
			return TASK_PASSED;
		case 3:
			return TASK_PASSED;
		case 4:
			return TASK_PASSED;
		case 5:
```

```
                        return TASK_PASSED;
                default:
                        return TASK_PASSED;
        }
}

int runGame(){
        int result = runAllTests();


        if(result != 0){
                showStartupError(result);
                return 1;
        }

        state_t currentState = STARTUP_STATE;
        int currentPage = 0;
        int pageCount = 1;
        bool option = true;
        int JOYSTICK_HOLD_TIME = 600;

        int prevTime = getTimeMs();
        bool rendered = false;

        bool pressed = false;
        bool sigPress = false;

        bool left = false;
        bool sigLeft = false;
        int lastLeft = getTimeMs();

        bool right = false;
        bool sigRight = false;
        int lastRight = getTimeMs();

        int score = 0;
        float currentDifficulty = 1;
        int lives = 3;
        float difficultyMultiplier = 1.2;

        loadConfig(&lives, &difficultyMultiplier);


        int currentTask = -1;
        task_result_t taskResult = TASK_NOT_STARTED;


        while(true){
                result = 0;

                #if USE_JOYSTICK == 1
```

```
                    if(!joystickIsPressed() && pressed){
                          sigPress = true;
                    }
                    else{
                          sigPress = false;
                    }
                    pressed = joystickIsPressed();

                    if(joystickIsLeft() && (!left || (getTimeMs() - lastLeft >
JOYSTICK_HOLD_TIME))){
                          sigLeft = true;
                          lastLeft = getTimeMs();
                    }
                    else{
                          sigLeft = false;
                    }
                    left = joystickIsLeft();

                    if(joystickIsRight() && (!right || (getTimeMs() - lastRight
> JOYSTICK_HOLD_TIME))){
                          sigRight = true;
                          lastRight = getTimeMs();
                    }
                    else{
                          sigRight = false;
                    }
                    right = joystickIsRight();
              #else
                    if((BSP_PB_GetState(BUTTON_USER) == GPIO_PIN_RESET) &&
(!right || (getTimeMs() - lastRight > JOYSTICK_HOLD_TIME))){
                          sigRight = true;
                          lastRight = getTimeMs();
                    }
                    else{
                          sigRight = false;
                          sigPress = false;
                    }
                    right = BSP_PB_GetState(BUTTON_USER) == GPIO_PIN_RESET;
              #endif

              switch(currentState){
                    case STARTUP_STATE:
                          if(!rendered){
                                result = showStartupScreen();
                                rendered = true;
                          }
                          if(getTimeMs() - prevTime > 3000 || sigPress){
                                currentState = MENU_STATE;
                                currentPage = 0;
                                pageCount = 3;
                                rendered = false;
```

```
                        }
                        break;
                case MENU_STATE:
                        if(!rendered){
                                result = showMenuPage(currentPage,
currentState);
                                wait(250);
                                rendered = true;
                        }
                        if(navigateThroughMenu(&currentPage, pageCount,
sigLeft, sigRight)){
                                rendered = false;
                                break;
                        }
                        if(sigPress){
                                switch(currentPage){
                                        case 0:
                                                currentState = GAME_CHOICE_STATE;
                                                currentPage = 0;
                                                pageCount = 2;
                                                break;
                                        case 1:
                                                currentState = LEADERBOARD_STATE;
                                                break;
                                        case 2:
                                                currentState = CREDITS_STATE;
                                                break;
                                }
                                rendered = false;
                        }
                        break;
                case GAME_CHOICE_STATE:
                        if(!rendered){
                                result = showMenuPage(currentPage,
currentState);
                                wait(250);
                                rendered = true;
                        }
                        if(navigateThroughMenu(&currentPage, pageCount,
sigLeft, sigRight)){
                                rendered = false;
                                break;
                        }
                        if(sigPress){
                                switch(currentPage){
                                        case 0:
                                                currentState = TASK_START_STATE;
                                                taskResult = TASK_NOT_STARTED;
                                                currentTask = -1;
                                                currentDifficulty = 1;
                                                rendered = false;
```

```
                                        break;
                                case 1:
                                        currentState = MENU_STATE;
                                        currentPage = 0;
                                        pageCount = 3;
                                        rendered = false;
                                        break;
                                }
                        }
                        break;
                case TASK_START_STATE:
                        if(currentTask == -1){
                                currentTask = chooseRandomTask();
                                if(!showTaskInfo(currentTask)){
                                        taskResult = startTask(currentTask,
currentDifficulty);
                                }
                                prevTime = getTimeMs();
                                wait(250);
                        }
                        if(taskResult == TASK_NOT_STARTED && (sigPress ||
(getTimeMs() - prevTime > 3000))){
                                taskResult = startTask(currentTask,
currentDifficulty);
                        }
                        if(taskResult != TASK_NOT_STARTED){
                                if(taskResult == TASK_RUNNING){
                                        currentState = TASK_RUNNING_STATE;
                                }
                                else if(taskResult == TASK_PASSED){
                                        score += 1;
                                        currentTask = -1;
                                        taskResult = TASK_NOT_STARTED;
                                        currentDifficulty *=
difficultyMultiplier;
                                }
                                else if(taskResult == TASK_FAILED){
                                        lives -= 1;
                                        if(lives == 0){
                                                currentState = GAME_END_STATE;
                                                rendered = false;
                                        }
                                        else{
                                                currentState = TASK_RETRY_STATE;
                                                option = true;
                                                rendered = false;
                                        }
                                }
                                else{
                                        result = ILLEGAL_TASK_RESULT_ERROR;
                                }
```

```
                        break;
                }
                break;
        case TASK_RUNNING_STATE:
                taskResult = checkRunningTask(currentTask);
                if(taskResult == TASK_PASSED){
                        currentState = TASK_START_STATE;
                        score += 1;
                        currentTask = -1;
                        taskResult = TASK_NOT_STARTED;
                        currentDifficulty *= difficultyMultiplier;
                }
                else if(taskResult == TASK_FAILED){
                        lives -= 1;
                        if(lives == 0){
                                currentState = GAME_END_STATE;
                                rendered = false;
                        }
                        else{
                                currentState = TASK_RETRY_STATE;
                                option = true;
                                rendered = false;
                        }
                }
                else if(taskResult == TASK_NOT_STARTED){
                        result = ILLEGAL_TASK_RESULT_ERROR;
                }
                break;
        case TASK_RETRY_STATE:
                if(!rendered){
                        result = showRetryScreen(lives, option);
                        wait(250);
                        rendered = true;
                }
                if(sigLeft || sigRight){
                        option = !option;
                        rendered = false;
                        break;
                }
                if(sigPress){
                        if(!option){
                                currentTask = -1;
                        }
                        taskResult = TASK_NOT_STARTED;
                        currentState = TASK_START_STATE;
                }
                break;
        case GAME_END_STATE:
                if(!rendered){
                        result = saveNewScore(score);
                        if(result != 0){
```

```
                                        break;
                                }
                                result = showGameEndScreen(score);
                                wait(250);
                                rendered = true;
                        }
                        if(sigPress){
                                currentState = MENU_STATE;
                                currentPage = 0;
                                pageCount = 3;
                                rendered = false;
                        }
                        break;
                case LEADERBOARD_STATE:
                        if(!rendered){
                                result = showLeaderboardScreen();
                                wait(250);
                                rendered = true;
                        }
                        if(sigPress){
                                currentState = MENU_STATE;
                                currentPage = 1;
                                pageCount = 3;
                                rendered = false;
                        }
                        break;
                case CREDITS_STATE:
                        if(!rendered){
                                result = showCreditsScreen();
                                wait(250);
                                rendered = true;
                        }
                        if(sigPress){
                                currentState = MENU_STATE;
                                currentPage = 2;
                                pageCount = 3;
                                rendered = false;
                        }
                        break;
        }

        if(result != 0){
                showRunError(result);
                return 2;
        }
        wait_D();
    }
}
```

```c
/*
 * game-timer.h
 *
 *      Author: tomas
 */

#ifndef INC_GAME_TIMER_H_
#define INC_GAME_TIMER_H_

#include "stm32l475e_iot01.h"

#define GAME_TIMER_DELAY_MS 10

void updateTimer_D();

void updateTimer(int time);

int getTimeMs();

void wait_D();

void wait(int time);

#endif /* INC_GAME_TIMER_H_ */
```

```c
/*
 * game-timer.c
 *
 *      Author: tomas
 */

#include "game-timer.h"

int gameTimer = 0;

void updateTimer(int time){
	gameTimer += time;
}

void updateTimer_D(){
	updateTimer(GAME_TIMER_DELAY_MS);
}

void wait(int time){
	HAL_Delay(time);
	updateTimer(time);
}

void wait_D(){
	wait(GAME_TIMER_DELAY_MS);
```

```
}

int getTimeMs(){
	return gameTimer;
}
```

```
/*
 * gyroscope.h
 *
 *      Author: kevinmancini
 */

#include <stdlib.h>
#include <stdio.h>

#ifndef INC_GYROSCOPE_H_
#define INC_GYROSCOPE_H_


//  Start the Gyroscope task
//  Return: TaskPassed ( 0 ) or TaskFailed ( 1 )
uint8_t GYRO_Task(void);




#endif /* INC_GYROSCOPE_H_ */
```

```
/*
 * gyroscope.c
 *
 *      Author: kevinmancini
 */

#define GYRO_NUMBER_OF_ATTEMPTS 10
#define GYRO_NUMBER_OF_TASKS 6
#define GYRO_WHO_AM_I_ADDRESS 0x0F
#define GYRO_OUT_TEMP_L_ADDRESS 0x20
#define GYRO_OUT_TEMP_H_ADDRESS 0x21
#define GYRO_MAX_TEMP 85
#define GYRO_MIN_TEMP -30
#define GYRO_OUT_TEMP_H_ADDRESS 0x21
#define GYRO_WHO_AM_I_VALUE 0x6A
#define GYRO_WRITE_ADDRESS 0xD5
#define GYRO_READ_ADDRESS 0xD4
#define GYRO_TEST_OK 0
#define GYRO_TEST_ERROR 1
#define GYRO_X_MAX 40000
```

```c
#define GYRO_X_MIN -40000
#define GYRO_Y_MAX 40000
#define GYRO_Y_MIN -40000
#define GYRO_Z_MAX 40000
#define GYRO_Z_MIN -40000



#include "gyroscope.h"
#include "lsm6dsl.h"
#include "st7789.h"
#include "fonts.h"
#include "stm32l475e_iot01_gyro.h"


uint8_t GYRO_WhoAmI_Address = 0x0F;
uint8_t GYRO_OutTempL_Address = 0x20;
uint8_t GYRO_OutTempH_Address = 0x21;
I2C_HandleTypeDef GYRO_hi2c2;
UART_HandleTypeDef huart1;
int16_t GYRO_current_temp;

enum state {
	TEST_STATE,
	OUT_OF_SERVICE_STATE,
	IDLE_STATE,
	TASK_STATE };
typedef enum state state_t;

/**
  * @brief  Compare the user's values  with the criteria of the task
  * @param  n indicates which criteria should be used, pfData user's values
  * @retval 1 CORRECT
  */
uint8_t GYRO_Compare_Values(uint8_t n, float *pfData){
	if (n==0){
		// X check
		return ( pfData[0]<GYRO_X_MIN || pfData[0]>GYRO_X_MAX ) &&
pfData[1]>GYRO_Y_MIN && pfData[1]<GYRO_X_MAX && pfData[2]>GYRO_X_MIN &&
pfData[2]<GYRO_X_MAX;
	} else if (n==1){
		// Y check
		return pfData[0]>GYRO_X_MIN && pfData[0]<GYRO_X_MAX && (
pfData[1]<GYRO_Y_MIN || pfData[1]>GYRO_X_MAX ) && pfData[2]>GYRO_X_MIN &&
pfData[2]<GYRO_X_MAX;
	} else if (n==2){
		// Z check
		return pfData[0]>GYRO_X_MIN && pfData[0]<GYRO_X_MAX &&
pfData[1]>GYRO_Y_MIN && pfData[1]<GYRO_X_MAX && ( pfData[2]<GYRO_X_MIN ||
pfData[2]>GYRO_X_MAX );
	} else {
```

```
            return 1;
      }
}

/**
  * @brief  Get the value of the WHO_AM_I register
  * @param  NONE
  * @retval WHO_AM_I register's value
  */
uint16_t GYRO_Get_Temp(void){
      uint8_t cur_temp_l;
      uint8_t cur_temp_h;

      HAL_I2C_Master_Transmit(&GYRO_hi2c2, GYRO_WRITE_ADDRESS,
&GYRO_OutTempL_Address, 1, 100);
      HAL_I2C_Master_Receive(&GYRO_hi2c2, GYRO_READ_ADDRESS, &cur_temp_l, 1,
100);

      HAL_I2C_Master_Transmit(&GYRO_hi2c2, GYRO_WRITE_ADDRESS,
&GYRO_OutTempL_Address, 1, 100);
      HAL_I2C_Master_Receive(&GYRO_hi2c2, GYRO_READ_ADDRESS, &cur_temp_h, 1,
100);

      return ((cur_temp_h << 8) | cur_temp_l);
}

/**
  * @brief  Check if the current temperature is between the admitted boundaries
  * @retval 1 if correct temperature, otherwise 0
  */
uint8_t GYRO_Check_Temp(void){
      GYRO_current_temp = GYRO_Get_Temp();
      return GYRO_current_temp>GYRO_MIN_TEMP &&
GYRO_current_temp<GYRO_MAX_TEMP;
}

/**
  * @brief  Print on the terminal messages, used for debug only
  * @param  char array of the message
  */
void GYRO_PrintTerminal(char* msg){
      HAL_UART_Transmit(&huart1,( uint8_t * )msg,sizeof(msg),1000);
}

/**
  * @brief  Run the gyroscope task
  * @retval 0 if PASSED, otherwise 1 FAILED
  */
uint8_t GYRO_Task(void){
      // START TASK
      char points0[] = "X X X X X ";
```

```
	char points1[] = "O X X X X ";
	char points2[] = "O O X X X ";
	char points3[] = "O O O X X ";
	char points4[] = "O O O O X ";
	char points5[] = "O O O O O ";
	// Init
	HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_RESET);
	ST7789_Init();
	uint8_t GYRO_init = BSP_GYRO_Init();
	float *pfData = malloc(3*sizeof(float));
	float *gyro_value = malloc(3*sizeof(float));
	HAL_Delay(1000);
	BSP_GYRO_GetXYZ(gyro_value);
	int p = 0;
	ST7789_Fill_Color(WHITE);
	if (GYRO_init == 0)
	while(1){
		// MINI-TASK
		for(int n = 0; n<3; n++){
			if(n==0)
				ST7789_WriteString(10, 50, "Tilt left and right
quickly for 5 seconds", Font_16x26, 100, WHITE);
			else if(n==1)
				ST7789_WriteString(10, 50, "Tilt front and back
quickly for 5 seconds", Font_16x26, 100, WHITE);
			else if(n==2)
				ST7789_WriteString(10, 50, "Tilt clockwise and
opposite quickly for 5 seconds", Font_16x26, 100, WHITE);
			while(p<5){
				if (p==0)
					ST7789_WriteString(10, 190, points0,
Font_11x18, 100, WHITE);
				if (p==1)
					ST7789_WriteString(10, 190, points1,
Font_11x18, 100, WHITE);
				if (p==2)
					ST7789_WriteString(10, 190, points2,
Font_11x18, 100, WHITE);
				if (p==3)
					ST7789_WriteString(10, 190, points3,
Font_11x18, 100, WHITE);
				if (p==4)
					ST7789_WriteString(10, 190, points4,
Font_11x18, 100, WHITE);
				if (p==5)
					ST7789_WriteString(10, 190, points5,
Font_11x18, 100, WHITE);
				HAL_Delay(1000);
				//if (GYRO_Check_Temp())
					BSP_GYRO_GetXYZ(pfData);
				if (GYRO_Compare_Values(n,pfData))
```

```
                                p++;
                    }
                    ST7789_Fill_Color(WHITE);
                    ST7789_WriteString(10, 160, "DONE", Font_16x26, 100, WHITE);
                    HAL_Delay(1000);
                    ST7789_Fill_Color(WHITE);
                    p = 0;
             }
             ST7789_Fill_Color(WHITE);
             ST7789_WriteString(10, 160, "COMPLIMENTS", Font_16x26, RED, WHITE);
             HAL_Delay(1000);
             ST7789_Fill_Color(WHITE);
             return 0;
      }
      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_0, GPIO_PIN_SET);
      return 1;
}


/**
  * @brief  Get the value of the WHO_AM_I register
  * @param  NONE
  * @retval WHO_AM_I register's value
  */


/**
  * @brief  Get the value of the WHO_AM_I register of the gyroscope sensor
  * @retval WHO_AM_I register's value
  */
uint8_t GYRO_Get_WHOAMI(void){
      uint8_t ID;
      HAL_I2C_Master_Transmit(&GYRO_hi2c2, GYRO_WRITE_ADDRESS,
&GYRO_WhoAmI_Address, 1, 100);
      HAL_I2C_Master_Receive(&GYRO_hi2c2, GYRO_READ_ADDRESS, &ID, 1, 100);
      return ID;
}



/* TEST: Gyroscope WHOAMI check
 * RETURN: TEST_OK ( 0 ) or TEST_ERROR ( 1 )
 * */
uint8_t WHO_AM_I_TEST(void){
      return
GYRO_Get_WHOAMI()==GYRO_WHO_AM_I_VALUE?GYRO_TEST_OK:GYRO_TEST_ERROR;
}

/* TEST: Gyroscope Initialization test
 * RETURN: TEST_OK ( 0 ) or TEST_ERROR ( 1 )
 * */
```

```
uint8_t GYRO_INIT_TEST(void){
	return BSP_GYRO_Init();
}

/* TEST: READ-WRITE registers test
 * RETURN: TEST_OK ( 0 ) or TEST_ERROR ( 1 )
 * */
//uint8_t READ_WRITE_TEST(void){
//	uint8_t res1,res2,res3;
//
//	HAL_I2C_Master_Transmit(&hi2c2, GYRO_WRITE_ADDRESS, &Test_Address1, 1,
100);
//	HAL_I2C_Master_Receive(&hi2c2, GYRO_READ_ADDRESS, &res1, 1, 100);
//
//	HAL_I2C_Master_Transmit(&hi2c2, GYRO_WRITE_ADDRESS, &Test_Address2, 1,
100);
//	HAL_I2C_Master_Receive(&hi2c2, GYRO_READ_ADDRESS, &res2, 1, 100);
//
//	HAL_I2C_Master_Transmit(&hi2c2, GYRO_WRITE_ADDRESS, &Test_Address3, 1,
100);
//	HAL_I2C_Master_Receive(&hi2c2, GYRO_READ_ADDRESS, &res3, 1, 100);
//	return ( res1==GYRO_TEST_EXPECTED_1 && res2==GYRO_TEST_EXPECTED_2 &&
res3==GYRO_TEST_EXPECTED_3 )?TEST_OK:TEST_ERROR;
//}

/* TEST: Task test
 * RETURN: TEST_OK ( 0 ) or TEST_ERROR ( 1 )
 * */
uint8_t GYRO_TASK_TEST(void){
	uint8_t res = 0;

	float test_pfData_1[] = {0.0,0.0,0.0};
	float test_pfData_2[] = {40001.0,0.0,0.0};
	float test_pfData_3[] = {0.0,40001.0,0.0};
	float test_pfData_4[] = {0.0,0.0,40001.0};
	float test_pfData_5[] = {40001.0,40001.0,0.0};
	float test_pfData_6[] = {0.0,40001.0,40001.0};
	float test_pfData_7[] = {40001.0,0.0,40001.0};
	float test_pfData_8[] = {40001.0,40001.0,40001.0};
	float test_pfData_9[] = {-40001.0,0.0,0.0};
	float test_pfData_10[] = {0.0,-40001.0,0.0};
	float test_pfData_11[] = {0.0,0.0,-40001.0};
	float test_pfData_12[] = {-40001.0,-40001.0,0.0};
	float test_pfData_13[] = {0.0,-40001.0,-40001.0};
	float test_pfData_14[] = {-40001.0,0.0,-40001.0};
	float test_pfData_15[] = {-40001.0,-40001.0,-40001.0};

	res += GYRO_Compare_Values(1,test_pfData_1);
	res += !GYRO_Compare_Values(1,test_pfData_2);
	res += GYRO_Compare_Values(1,test_pfData_3);
	res += GYRO_Compare_Values(1,test_pfData_4);
```

```
	res += GYRO_Compare_Values(1,test_pfData_5);
	res += GYRO_Compare_Values(1,test_pfData_6);
	res += GYRO_Compare_Values(1,test_pfData_7);
	res += GYRO_Compare_Values(1,test_pfData_8);
	res += !GYRO_Compare_Values(1,test_pfData_9);
	res += GYRO_Compare_Values(1,test_pfData_10);
	res += GYRO_Compare_Values(1,test_pfData_11);
	res += GYRO_Compare_Values(1,test_pfData_12);
	res += GYRO_Compare_Values(1,test_pfData_13);
	res += GYRO_Compare_Values(1,test_pfData_14);
	res += GYRO_Compare_Values(1,test_pfData_15);

	res += GYRO_Compare_Values(2,test_pfData_1);
	res += GYRO_Compare_Values(2,test_pfData_2);
	res += !GYRO_Compare_Values(2,test_pfData_3);
	res += GYRO_Compare_Values(2,test_pfData_4);
	res += GYRO_Compare_Values(2,test_pfData_5);
	res += GYRO_Compare_Values(2,test_pfData_6);
	res += GYRO_Compare_Values(2,test_pfData_7);
	res += GYRO_Compare_Values(2,test_pfData_8);
	res += GYRO_Compare_Values(2,test_pfData_9);
	res += !GYRO_Compare_Values(2,test_pfData_10);
	res += GYRO_Compare_Values(2,test_pfData_11);
	res += GYRO_Compare_Values(2,test_pfData_12);
	res += GYRO_Compare_Values(2,test_pfData_13);
	res += GYRO_Compare_Values(2,test_pfData_14);
	res += GYRO_Compare_Values(2,test_pfData_15);

	res += GYRO_Compare_Values(3,test_pfData_1);
	res += GYRO_Compare_Values(3,test_pfData_2);
	res += GYRO_Compare_Values(3,test_pfData_3);
	res += !GYRO_Compare_Values(3,test_pfData_4);
	res += GYRO_Compare_Values(3,test_pfData_5);
	res += GYRO_Compare_Values(3,test_pfData_6);
	res += GYRO_Compare_Values(3,test_pfData_7);
	res += GYRO_Compare_Values(3,test_pfData_8);
	res += GYRO_Compare_Values(3,test_pfData_9);
	res += GYRO_Compare_Values(3,test_pfData_10);
	res += GYRO_Compare_Values(3,test_pfData_11);
	res += GYRO_Compare_Values(3,test_pfData_12);
	res += GYRO_Compare_Values(3,test_pfData_13);
	res += GYRO_Compare_Values(3,test_pfData_14);
	res += GYRO_Compare_Values(3,test_pfData_15);

	return res;
}
```

```
/*
 * humidity.h
 *
```

```
 *      Author: tomas
 */

#ifndef INC_HUMIDITY_H_
#define INC_HUMIDITY_H_

#include "stm32l4xx_hal.h"

uint8_t readHumidityRegister(uint8_t registerAddress);
void writeToHumidityRegister(uint8_t registerAddress, uint8_t value);

void HTS221On();

void HTS221Off();

void HTS221valueLock();

void HTS221valueUnlock();

float humidityGet();

void HTS221Init();


#endif /* INC_HUMIDITY_H_ */
```

```
/*
 * humidity.c
 *      Author: tomas
 */

#include "humidity.h"
#include "hts221.h"

#define HTS221_READ 0xBF
#define HTS221_WRITE 0xBE

extern I2C_HandleTypeDef hi2c2;


uint8_t readHumidityRegister(uint8_t registerAddress){
	uint8_t out;
	HAL_I2C_Master_Transmit(&hi2c2, HTS221_WRITE, &registerAddress, 1, 1000);
	HAL_I2C_Master_Receive(&hi2c2, HTS221_READ, &out, 1, 1000);
	return out;
}

void writeToHumidityRegister(uint8_t registerAddress, uint8_t value){
	uint8_t in[2];
	in[0] = registerAddress;
```

```
	in[1] = value;
	HAL_I2C_Master_Transmit(&hi2c2, HTS221_WRITE, in, 2, 1000);
}

void HTS221Init(){
	HTS221_H_Init(HTS221_WRITE);
	HTS221valueLock();
}

void HTS221On(){
	uint8_t value = readHumidityRegister(HTS221_CTRL_REG1);
	writeToHumidityRegister(HTS221_CTRL_REG1, value | (1 << 7));
}

void HTS221Off(){
	uint8_t value = readHumidityRegister(HTS221_CTRL_REG1);
	uint8_t mask = 1 << 7;
	mask = ~mask;
	writeToHumidityRegister(HTS221_CTRL_REG1, value & mask);
}

void HTS221valueLock(){
	uint8_t value = readHumidityRegister(HTS221_CTRL_REG1);
	writeToHumidityRegister(HTS221_CTRL_REG1, value | (1 << 2));
}

void HTS221valueUnlock(){
	uint8_t value = readHumidityRegister(HTS221_CTRL_REG1);
	uint8_t mask = 1 << 2;
	mask = ~mask;
	writeToHumidityRegister(HTS221_CTRL_REG1, value & mask);
}

float humidityGet(){
	return HTS221_H_ReadHumidity(HTS221_READ);
}
```

```
/*
 * humidity-task.h
 *
 *      Author: tomas
 */

#ifndef INC_HUMIDITY_TASK_H_
#define INC_HUMIDITY_TASK_H_

#include "game-timer.h"
#include "task-results.h"
#include "humidity.h"
#include <stdio.h>
```

```h
#include <stdlib.h>


void humidityShowTaskInfo();
task_result_t humidityStartTask(float difficulty);
task_result_t humidityCheckTaskState();

#endif /* INC_HUMIDITY_TASK_H_ */
```

```c
/*
 * humidity-task.c
 *
 *      Author: tomas
 */

#include "humidity-task.h"


float humidityTaskTargetValue;
int humidityTaskStartTime;
float humidityTaskDifficultyMultiplier;

void humidityShowTaskInfo(){
	#if USE_DISPLAY == 1

	#else
		printf("\rYour task is to get to target humidity!\n");
	#endif
}

task_result_t humidityStartTask(float difficulty){
	humidityTaskTargetValue = humidityGet();
	int difference = rand() % 10 + 2;
	difference *= (rand() % 3 == 0) ? -1 : 1;
	humidityTaskTargetValue += difference;
	if(humidityTaskTargetValue > 100){
		humidityTaskTargetValue -= (humidityTaskTargetValue - 100) * 2;
	}
	humidityTaskStartTime = getTimeMs();
	humidityTaskDifficultyMultiplier = difficulty;
	#if USE_DISPLAY == 1

	#else
		printf("\rTarget humidity: %f%%\n", humidityTaskTargetValue);
	#endif
	return TASK_RUNNING;
}

task_result_t humidityCheckTaskState(){
	float currentHumidity = humidityGet();
```

```
	int difference = currentHumidity - humidityTaskTargetValue;
	#if USE_DISPLAY == 1

	#else
		if((getTimeMs() - humidityTaskStartTime) % 1000 == 0){
			printf("\rCurrent humidity: %f%%\n", currentHumidity);
		}
		if(difference < 0.5 && difference > -0.5){
			printf("\rTask passed!\n");
			return TASK_PASSED;
		}
	#endif
	if((getTimeMs() - humidityTaskStartTime) > 10000 /
humidityTaskDifficultyMultiplier){
		return TASK_FAILED;
	}
	return TASK_RUNNING;
}
```

```
/*
 * humidity-tests.h
 *
 *      Author: tomas
 */

#ifndef INC_HUMIDITY_TESTS_H_
#define INC_HUMIDITY_TESTS_H_


#include <device-libs.h>
#include <stdio.h>


int humidityRunAutoTests();
int humidityRunWhoAmITest();
int humidityRunRWRTest();

int humidityRunInteractiveTests();
int humidityRunSensorTest();
int humidityRunLockTest();

#endif /* INC_HUMIDITY_TESTS_H_ */
```

```
/*
 * humidity-tests.c
 *
 *      Author: tomas
 */
```

```c
#include "humidity-tests.h"
#include "hts221.h"


int humidityRunAutoTests(){
	int result = 0;
	#if defined(INC_DISPLAY_H_) && (USE_DISPLAY == 1)
		//Run tests differently with output
		return result;
	#else
		printf("\r========RUNNING HUMIDITY BASIC TESTS========\n");
		result = humidityRunWhoAmITest();
		if(result == 0){
			result = humidityRunRWRTest();
		}
		if(result == 0){
			printf("\r========HUMIDITY BASIC TESTS PASSED========!\n");
		}
		else{
			printf("\r========HUMIDITY BASIC TESTS FAILED!========\n");
		}
		return result;
	#endif
}
int humidityRunWhoAmITest(){
	uint8_t deviceId;
	#if defined(INC_DISPLAY_H_) && (USE_DISPLAY == 1)
		//Run tests differently with output
		return 0;
	#else
		printf("\r====RUNNING WHO_AM_I TEST====\n");
		printf("\rReading WHO_AM_I address of humidity sensor\n");
		deviceId = readHumidityRegister(HTS221_WHO_AM_I_REG);
		printf("\rRead value - %d, correct value - %d\n", deviceId,
HTS221_WHO_AM_I_VAL);
		if(deviceId == HTS221_WHO_AM_I_VAL){
			printf("\r====WHO_AM_I test passed!====\n");
			return 0;
		}
		else{
			printf("\r====WHO_AM_I test failed!====\n");
			return 1;
		}
	#endif
}

int humidityRunRWRTest(){
	uint8_t oldValue;
	uint8_t newValue;
	#if defined(INC_DISPLAY_H_) && (USE_DISPLAY == 1)
		//Run tests differently with output
```

```
		return 0;
	#else
		printf("\r====RUNNING RWR TEST====\n");
		oldValue = readHumidityRegister(HTS221_CTRL_REG1);
		uint8_t writtenValue;
		if((oldValue & (1 << 7)) == 0){
			writtenValue = oldValue + (1 << 7);
		}
		else{
			writtenValue = oldValue - (1 << 7);
		}
		printf("\rRead value - %d\n", oldValue);
		writeToHumidityRegister(HTS221_CTRL_REG1, writtenValue);
		printf("\rWrite value - %d\n", writtenValue);
		newValue = readHumidityRegister(HTS221_CTRL_REG1);
		printf("\rRead control of current value - %d\n", newValue);
		writeToHumidityRegister(HTS221_CTRL_REG1, oldValue);
		if(oldValue != newValue && writtenValue == newValue){
			printf("\r====RWR test passed!====\n");
			return 0;
		}
		else{
			printf("\r====RWR test failed!====\n");
			return 1;
		}
	#endif
}

int humidityRunInteractiveTests(){
	#if defined(INC_DISPLAY_H_) && (USE_DISPLAY == 1)
		//Run tests differently with output
		return 0;
	#else
		printf("\r========RUNNING HUMIDITY INTERACTIVE TESTS========\n");
		int result = humidityRunSensorTest();
		if(result == 0){
			result = humidityRunLockTest();
		}
		if(result == 0){
			printf("\r========HUMIDITY INTERACTIVE TESTS
PASSED========!\n");
		}
		else{
			printf("\r========HUMIDITY INTERACTIVE TESTS
FAILED!========\n");
		}
		return result;
	#endif
}

int humidityRunSensorTest(){
```

```
	#if defined(INC_DISPLAY_H_) && (USE_DISPLAY == 1)
		//Run tests differently with output
		return 0;
	#else
		printf("\r====RUNNING SENSOR TEST====\n");
		HTS221On();
		int changed = 0;
		float lastValue = humidityGet();
		printf("\rHUMIDITY: %f\n", lastValue);
		for(int i = 0; i < 10; i++){
			HAL_Delay(1000);
			float newValue = humidityGet();
			float diff = newValue - lastValue;
			if(diff < 0){
				diff = diff * -1;
			}
			if(diff > 0.01){
				changed = 1;
			}
			lastValue = newValue;
			printf("\rHUMIDITY: %f\n", lastValue);
		}
		if(changed == 1){
			printf("\r====HUMIDITY SENSOR TEST PASSED====!\n");
			return 0;
		}
		else{
			printf("\r====HUMIDITY SENSOR TEST FAILED!====\n");
			return 1;
		}
	#endif
}

int humidityRunLockTest(){
	#if defined(INC_DISPLAY_H_) && (USE_DISPLAY == 1)
		//Run tests differently with output
		return 0;
	#else
		printf("\r====RUNNING LOCK TEST====\n");
		HTS221On();
		int changed = 0;
		printf("\rUNLOCKED\n");
		HTS221valueUnlock();
		int lastValue = readHumidityRegister(HTS221_HR_OUT_L_REG);
		printf("\rHumidity L part: %d\n", lastValue);
		for(int i = 0; i < 10; i++){
			HAL_Delay(500);
			int newValue = readHumidityRegister(HTS221_HR_OUT_L_REG);
			int diff = newValue - lastValue;
			if(diff != 0){
				changed = 1;
```

```
                }
                lastValue = newValue;
                printf("\rHumidity L part: %d\n", lastValue);
        }
        if(changed == 0){
                printf("\r====HUMIDITY LOCK TEST FAILED!====\n");
                return 1;
        }
        printf("\rLOCKED\n");
        HTS221valueLock();
        lastValue = readHumidityRegister(HTS221_HR_OUT_L_REG);
        changed = 0;
        printf("\rHumidity L part: %d\n", lastValue);
        for(int i = 0; i < 10; i++){
                HAL_Delay(500);
                int newValue = readHumidityRegister(HTS221_HR_OUT_L_REG);
                int diff = newValue - lastValue;
                if(diff != 0){
                        changed = 1;
                }
                lastValue = newValue;
                printf("\rHumidity L part: %d\n", lastValue);
        }
        if(changed == 1){
                printf("\r====HUMIDITY LOCK TEST FAILED!====\n");
                return 1;
        }
        else{
                printf("\r====HUMIDITY LOCK TEST PASSED!====\n");
                return 0;
        }
    #endif
}
```

```
/*
 * joystick.h
 *
 *      Author: 695126
 */

#ifndef INC_JOYSTICK_H_
#define INC_JOYSTICK_H_

#include <stdbool.h>
#include "stm32l475e_iot01.h"
#include "device-configs.h"

bool joystickIsPressed();

bool joystickIsUp();
```

```
bool joystickIsDown();
bool joystickIsLeft();
bool joystickIsRight();



#endif /* INC_JOYSTICK_H_ */
```

```
/*
 * joystick.c
 *
 *      Author: tomas
 */

#include "joystick.h"

bool joystickIsPressed(){
	return false;
}


bool joystickIsUp(){
	return false;
}

bool joystickIsDown(){
	return false;
}

bool joystickIsLeft(){
	return false;
}

bool joystickIsRight(){
	#if USE_JOYSTICK == 0
	#else
		return false;
	#endif
}
```

```
/*
 * leaderboard.h
 *
 *      Author: tomas
 */

#ifndef INC_LEADERBOARD_H_
#define INC_LEADERBOARD_H_
```

```h
#include "device-libs.h"
#include <stdbool.h>

#define LEADERBOARD_POSITIONS 10


int saveNewScore(int score);

bool loadLeaderboard();
int getScore(int pos);

#endif /* INC_LEADERBOARD_H_ */
```

```c
/*
 * leaderboard.h
 *
 *      Author: tomas
 */


#include "leaderboard.h"

int leaderboardScores[LEADERBOARD_POSITIONS];

int saveNewScore(int score){
	int position = -1;
	for(int i = 0; i < LEADERBOARD_POSITIONS; i++){
		if(leaderboardScores[i] < score){
			position = i;
		}
	}
	if(position != -1){
		for(int i = LEADERBOARD_POSITIONS - 1; i > position; i--){
			leaderboardScores[i] = leaderboardScores[i - 1];
		}
		#if USE_SD_CARD == 1
			FIL file;
			int res = openFile(&file, "leaderboard.txt", true);
			if(res){
				return false;
			}
			char line[30];
			for(int i = 0; i < LEADERBOARD_POSITIONS; i++){
				sprintf(line, "%d", leaderboardScores[i]);
				res = writeLine(file, line, strlen(line));
				if(res){
					return false;
				}
			}
			res = closeFile(file);
```

```
                    if(res){
                        return false;
                    }
            #endif
        }
        return position;
}


bool loadLeaderboard(){
        for(int i = 0; i < LEADERBOARD_POSITIONS; i++){
            leaderboardScores[i] = -1;
        }
        #if USE_SD_CARD == 1
            FIL file;
            int res = openFile(&file, "leaderboard.txt", false);
            if(res){
                return false;
            }
            char line[30];
            for(int i = 0; i < LEADERBOARD_POSITIONS; i++){
                res = readLine(file, line, 30);
                if(res == 0){
                    break;
                }
                leaderboardScores[i] = atoi(line);
            }
            res = closeFile(file);
            if(res){
                return false;
            }
        #endif
}

int getScore(int pos){
        return leaderboardScores[pos];
}
```

```
/*
 * sd-card.h
 *
 *      Author: tomas
 */

#ifndef INC_SD_CARD_H_
#define INC_SD_CARD_H_

#include "fatfs.h"
#include "stm32l475e_iot01.h"
#include <stdbool.h>
```

```
bool isCardPresent();

int SDMount();

int openFile(FIL * file, char * name, bool write);

int closeFile(FIL * file);

int readLine(FIL * file, char * out, int length);

int writeLine(FIL * file, char * in, int length);


#endif /* INC_SD_CARD_H_ */
```

```
/*
 * sd-card.c
 *
 *      Author: tomas
 */

#include "sd-card.h"

FATFS SDFatFs;

bool SDFatFSMounted = false;

bool isCardPresent(){
	return (HAL_GPIO_ReadPin(SD_DETECT_GPIO_Port, SD_DETECT_Pin) ==
GPIO_PIN_SET);
}

int SDMount(){
	if(SDFatFSMounted || !isCardPresent){
		return -1;
	}
	FRESULT res = f_mount(&SDFatFs, "", 1);
	SDFatFSMounted = true;
	return res;
}

int openFile(FIL * file, char * name, bool write){
	if(!SDFatFSMounted || !isCardPresent){
		return -1;
	}
	return f_open(file, name, write ? FA_WRITE : FA_READ);
}

int closeFile(FIL * file){
```

```
	if(!SDFatFSMounted || !isCardPresent){
		return -1;
	}
	return f_close(file);
}

int readLine(FIL * file, char * out, int length){
	if(!SDFatFSMounted || !isCardPresent){
		return -1;
	}
	char c;
	int pos = 0;
	int res = 1;
	res = f_gets(&c, 1, file);
	while(c != '\n' && (pos != length - 1) && res != 0){
		out[pos] = c;
		res = f_gets(&c, 1, file);
		pos ++;
	}
	if(res == 0){
		return pos;
	}
	if(c != '\n'){
		out[pos] = c;
		return pos + 1;
	}
	out[pos] = 0;
	return pos;
}

int writeLine(FIL * file, char * in, int length){
	if(!SDFatFSMounted || !isCardPresent){
		return -1;
	}
	unsigned_int res;
	unsigned_int bytesWrote;
	res = f_write(file, in, length, &bytesWrote);
	if(bytesWrote != length){
		return -2;
	}
	if(res != FR_OK){
		return res;
	}
	char c = '\n';
	res = f_write(file, &c, 1,  &bytesWrote);
	return res;
}
```

```
#ifndef __ST7789_H
#define __ST7789_H
```

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

/* choose a Hardware SPI port to use. */
#define ST7789_SPI_PORT hspi1
extern SPI_HandleTypeDef ST7789_SPI_PORT;

/* Pin connection*/
#define ST7789_RST_PORT GPIOA
#define ST7789_RST_PIN GPIO_PIN_1
#define ST7789_DC_PORT GPIOA
#define ST7789_DC_PIN GPIO_PIN_4
#define ST7789_CS_PORT GPIOA
#define ST7789_CS_PIN GPIO_PIN_0

// Use if need backlight control *****
//#define BLK_PORT GPIOA
//#define BLK_PIN GPIO_PIN_0

/**
 * Comment one to use another one.
 * two parameters can be choosed
 * 135x240(0.96 inch) and 240x240(1.3inch)
 * X_SHIFT&Y_SHIFT are used to correct different display's resolution
 */

/* Choose a type you are using */
//#define USING_135X240
#define USING_240X240

/* Choose a display rotation you want to use: (0-3) */
//#define ST7789_ROTATION 0
//#define ST7789_ROTATION 1
#define ST7789_ROTATION 2                    //  use Normally on 240x240
//#define ST7789_ROTATION 3

#ifdef USING_135X240

    #if ST7789_ROTATION == 0
        #define ST7789_WIDTH 135
        #define ST7789_HEIGHT 240
        #define X_SHIFT 53
        #define Y_SHIFT 40
    #endif

    #if ST7789_ROTATION == 1
        #define ST7789_WIDTH 240
        #define ST7789_HEIGHT 135
        #define X_SHIFT 40
        #define Y_SHIFT 52
```

```
    #endif

    #if ST7789_ROTATION == 2
        #define ST7789_WIDTH 135
        #define ST7789_HEIGHT 240
        #define X_SHIFT 52
        #define Y_SHIFT 40
    #endif

    #if ST7789_ROTATION == 3
        #define ST7789_WIDTH 240
        #define ST7789_HEIGHT 135
        #define X_SHIFT 40
        #define Y_SHIFT 53
    #endif

#endif

#ifdef USING_240X240

    #define ST7789_WIDTH 240
    #define ST7789_HEIGHT 240

              #if ST7789_ROTATION == 0
                     #define X_SHIFT 0
                     #define Y_SHIFT 80
              #elif ST7789_ROTATION == 1
                     #define X_SHIFT 80
                     #define Y_SHIFT 0
              #elif ST7789_ROTATION == 2
                     #define X_SHIFT 0
                     #define Y_SHIFT 0
              #elif ST7789_ROTATION == 3
                     #define X_SHIFT 0
                     #define Y_SHIFT 0
              #endif

#endif

/**
 *Color of pen
 *If you want to use another color, you can choose one in RGB565 format.
 */

#define WHITE 0xFFFF
#define BLACK 0x0000
#define BLUE 0x001F
#define RED 0xF800
#define MAGENTA 0xF81F
#define GREEN 0x07E0
#define CYAN 0x7FFF
```

```
#define YELLOW 0xFFE0
#define GRAY 0X8430
#define BRED 0XF81F
#define GRED 0XFFE0
#define GBLUE 0X07FF
#define BROWN 0XBC40
#define BRRED 0XFC07
#define DARKBLUE 0X01CF
#define LIGHTBLUE 0X7D7C
#define GRAYBLUE 0X5458

#define LIGHTGREEN 0X841F
#define LGRAY 0XC618
#define LGRAYBLUE 0XA651
#define LBBLUE 0X2B12

/* Control Registers and constant codes */
#define ST7789_NOP     0x00
#define ST7789_SWRESET 0x01
#define ST7789_RDDID   0x04
#define ST7789_RDDST   0x09

#define ST7789_SLPIN   0x10
#define ST7789_SLPOUT  0x11
#define ST7789_PTLON   0x12
#define ST7789_NORON   0x13

#define ST7789_INVOFF  0x20
#define ST7789_INVON   0x21
#define ST7789_DISPOFF 0x28
#define ST7789_DISPON  0x29
#define ST7789_CASET   0x2A
#define ST7789_RASET   0x2B
#define ST7789_RAMWR   0x2C
#define ST7789_RAMRD   0x2E

#define ST7789_PTLAR   0x30
#define ST7789_COLMOD  0x3A
#define ST7789_MADCTL  0x36

// ADDED REGISTERS

#define ST7789_FRMCTR1 0xB1
#define ST7789_FRMCTR2 0xB2
#define ST7789_FRMCTR3 0xB3
#define ST7789_INVCTR 0xB4
#define ST7789_DISSET5 0xB6
#define ST7789_GCTRL 0xB7
#define ST7789_GTADJ 0xB8
#define ST7789_VCOMS 0xBB
#define ST7789_LCMCTRL 0xC0
```

```
#define ST7789_IDSET 0xC1
#define ST7789_VDVVRHEN 0xC2
#define ST7789_VRHS 0xC3
#define ST7789_VDVS 0xC4
#define ST7789_VMCTR1 0xC5
#define ST7789_FRCTRL2 0xC6
#define ST7789_CABCCTRL 0xC7
#define ST7789_RDID1 0xDA
#define ST7789_RDID2 0xDB
#define ST7789_RDID3 0xDC
#define ST7789_RDID4 0xDD
#define ST7789_GMCTRP1 0xE0
#define ST7789_GMCTRN1 0xE1
#define ST7789_PWCTR6 0xFC



/**
 * Memory Data Access Control Register (0x36H)
 * MAP:     D7  D6  D5  D4  D3  D2  D1  D0
 * param:   MY  MX  MV  ML  RGB MH  -   -
 *
 */

/* Page Address Order ('0': Top to Bottom, '1': the opposite) */
#define ST7789_MADCTL_MY  0x80
/* Column Address Order ('0': Left to Right, '1': the opposite) */
#define ST7789_MADCTL_MX  0x40
/* Page/Column Order ('0' = Normal Mode, '1' = Reverse Mode) */
#define ST7789_MADCTL_MV  0x20
/* Line Address Order ('0' = LCD Refresh Top to Bottom, '1' = the opposite) */
#define ST7789_MADCTL_ML  0x10
/* RGB/BGR Order ('0' = RGB, '1' = BGR) */
#define ST7789_MADCTL_RGB 0x00

#define ST7789_RDID1   0xDA
#define ST7789_RDID2   0xDB
#define ST7789_RDID3   0xDC
#define ST7789_RDID4   0xDD

/* Advanced options */
/**
 * Caution: Do not operate these settings
 * You know what you are doing
 */

#define ST7789_COLOR_MODE_16bit 0x55    //  RGB565 (16bit)
#define ST7789_COLOR_MODE_18bit 0x66    //  RGB666 (18bit)

/* Basic operations */
#define ST7789_RST_Clr() HAL_GPIO_WritePin(ST7789_RST_PORT, ST7789_RST_PIN,
```

```
GPIO_PIN_RESET)
#define ST7789_RST_Set() HAL_GPIO_WritePin(ST7789_RST_PORT, ST7789_RST_PIN,
GPIO_PIN_SET)

#define ST7789_DC_Clr() HAL_GPIO_WritePin(ST7789_DC_PORT, ST7789_DC_PIN,
GPIO_PIN_RESET)
#define ST7789_DC_Set() HAL_GPIO_WritePin(ST7789_DC_PORT, ST7789_DC_PIN,
GPIO_PIN_SET)

#define ST7789_Select() HAL_GPIO_WritePin(ST7789_CS_PORT, ST7789_CS_PIN,
GPIO_PIN_RESET)
#define ST7789_UnSelect() HAL_GPIO_WritePin(ST7789_CS_PORT, ST7789_CS_PIN,
GPIO_PIN_SET)

#define ABS(x) ((x) > 0 ? (x) : -(x))

/* Basic functions. */
void ST7789_Init(void);
void ST7789_SetRotation(uint8_t m);
void ST7789_Fill_Color(uint16_t color);
void ST7789_DrawPixel(uint16_t x, uint16_t y, uint16_t color);
void ST7789_Fill(uint16_t xSta, uint16_t ySta, uint16_t xEnd, uint16_t yEnd,
uint16_t color);
void ST7789_DrawPixel_4px(uint16_t x, uint16_t y, uint16_t color);

/* Graphical functions. */
void ST7789_DrawLine(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2,
uint16_t color);
void ST7789_DrawRectangle(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2,
uint16_t color);
void ST7789_DrawCircle(uint16_t x0, uint16_t y0, uint8_t r, uint16_t color);
void ST7789_DrawImage(uint16_t x, uint16_t y, uint16_t w, uint16_t h, const
uint16_t *data);
void ST7789_InvertColors(uint8_t invert);

/* Text functions. */
void ST7789_WriteChar(uint16_t x, uint16_t y, char ch, FontDef font, uint16_t
color, uint16_t bgcolor);
void ST7789_WriteString(uint16_t x, uint16_t y, const char *str, FontDef font,
uint16_t color, uint16_t bgcolor);

/* Extented Graphical functions. */
void ST7789_DrawFilledRectangle(uint16_t x, uint16_t y, uint16_t w, uint16_t h,
uint16_t color);
void ST7789_DrawTriangle(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2,
uint16_t x3, uint16_t y3, uint16_t color);
void ST7789_DrawFilledTriangle(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t
y2, uint16_t x3, uint16_t y3, uint16_t color);
void ST7789_DrawFilledCircle(int16_t x0, int16_t y0, int16_t r, uint16_t color);

/* Command functions */
```

```
void ST7789_TearEffect(uint8_t tear);

/* Simple test function. */
void ST7789_Test(void);

#if !defined(USING_240X240)
       #if !defined(USING_135X240)
           #error   You should at least choose one display resolution!
       #endif
#endif

#if !defined(USING_135X240)
       #if !defined(USING_240X240)
           #error   You should at least choose one display resolution!
       #endif
#endif


#ifndef ST7789_ROTATION
    #error You should at least choose a display rotation!
#endif

#endif
```

```
#include "st7789.h"

/**
 * @brief Write command to ST7789 controller
 * @param cmd -> command to write
 * @return none
 */
static void ST7789_WriteCommand(uint8_t cmd)
{
       ST7789_Select();
       ST7789_DC_Clr();
       HAL_SPI_Transmit(&ST7789_SPI_PORT, &cmd, sizeof(cmd), HAL_MAX_DELAY);
       ST7789_UnSelect();
}

/**
 * @brief Write data to ST7789 controller
 * @param buff -> pointer of data buffer
 * @param buff_size -> size of the data buffer
 * @return none
 */
static void ST7789_WriteData(uint8_t *buff, size_t buff_size)
{
       ST7789_Select();
       ST7789_DC_Set();
```

```
	// split data in small chunks because HAL can't send more than 64K at
once

	while (buff_size > 0) {
		uint16_t chunk_size = buff_size > 65535 ? 65535 : buff_size;
		HAL_SPI_Transmit(&ST7789_SPI_PORT, buff, chunk_size,
HAL_MAX_DELAY);
		buff += chunk_size;
		buff_size -= chunk_size;
	}

	ST7789_UnSelect();
}
/**
 * @brief Write data to ST7789 controller, simplify for 8bit data.
 * data -> data to write
 * @return none
 */
static void ST7789_WriteSmallData(uint8_t data)
{
	ST7789_Select();
	ST7789_DC_Set();
	HAL_SPI_Transmit(&ST7789_SPI_PORT, &data, sizeof(data), HAL_MAX_DELAY);
	ST7789_UnSelect();
}

/**
 * @brief Set the rotation direction of the display
 * @param m -> rotation parameter(please refer it in st7789.h)
 * @return none
 */
void ST7789_SetRotation(uint8_t m)
{
	ST7789_WriteCommand(ST7789_MADCTL);	// MADCTL
	switch (m) {
	case 0:
		ST7789_WriteSmallData(ST7789_MADCTL_MX | ST7789_MADCTL_MY |
ST7789_MADCTL_RGB);
		break;
	case 1:
		ST7789_WriteSmallData(ST7789_MADCTL_MY | ST7789_MADCTL_MV |
ST7789_MADCTL_RGB);
		break;
	case 2:
		ST7789_WriteSmallData(ST7789_MADCTL_RGB);
		break;
	case 3:
		ST7789_WriteSmallData(ST7789_MADCTL_MX | ST7789_MADCTL_MV |
ST7789_MADCTL_RGB);
		break;
	default:
```

```
            break;
        }
}

/**
 * @brief Set address of DisplayWindow
 * @param xi&yi -> coordinates of window
 * @return none
 */
static void ST7789_SetAddressWindow(uint16_t x0, uint16_t y0, uint16_t x1,
uint16_t y1)
{
        ST7789_Select();
        uint16_t x_start = x0 + X_SHIFT, x_end = x1 + X_SHIFT;
        uint16_t y_start = y0 + Y_SHIFT, y_end = y1 + Y_SHIFT;

        /* Column Address set */
        ST7789_WriteCommand(ST7789_CASET);
        {
                uint8_t data[] = {x_start >> 8, x_start & 0xFF, x_end >> 8, x_end &
0xFF};
                ST7789_WriteData(data, sizeof(data));
        }

        /* Row Address set */
        ST7789_WriteCommand(ST7789_RASET);
        {
                uint8_t data[] = {y_start >> 8, y_start & 0xFF, y_end >> 8, y_end &
0xFF};
                ST7789_WriteData(data, sizeof(data));
        }
        /* Write to RAM */
        ST7789_WriteCommand(ST7789_RAMWR);
        ST7789_UnSelect();
}

/**
 * @brief Initialize ST7789 controller
 * @param none
 * @return none
 */
void ST7789_Init(void)
{
        HAL_Delay(25);
    ST7789_RST_Clr();
    HAL_Delay(25);
    ST7789_RST_Set();
    HAL_Delay(50);

    ST7789_WriteCommand(ST7789_COLMOD);        //     Set color mode
    ST7789_WriteSmallData(ST7789_COLOR_MODE_16bit);
```

```
      ST7789_WriteCommand(0xB2);                    //     Porch control
      {
            uint8_t data[] = {0x0C, 0x0C, 0x00, 0x33, 0x33};
            ST7789_WriteData(data, sizeof(data));
      }
      ST7789_SetRotation(ST7789_ROTATION);    //     MADCTL (Display Rotation)

      /* Internal LCD Voltage generator settings */
    ST7789_WriteCommand(0XB7);                      //     Gate Control
    ST7789_WriteSmallData(0x35);              //     Default value
    ST7789_WriteCommand(0xBB);                      //     VCOM setting
    ST7789_WriteSmallData(0x19);              //     0.725v (default 0.75v for
0x20)
    ST7789_WriteCommand(0xC0);                      //     LCMCTRL
    ST7789_WriteSmallData (0x2C);                   //     Default value
    ST7789_WriteCommand (0xC2);                     //     VDV and VRH command
Enable
    ST7789_WriteSmallData (0x01);                   //     Default value
    ST7789_WriteCommand (0xC3);                     //     VRH set
    ST7789_WriteSmallData (0x12);                   //     +-4.45v (defalut
+-4.1v for 0x0B)
    ST7789_WriteCommand (0xC4);                     //     VDV set
    ST7789_WriteSmallData (0x20);                   //     Default value
    ST7789_WriteCommand (0xC6);                     //     Frame rate control in
normal mode
    ST7789_WriteSmallData (0x0F);                   //     Default value (60HZ)
    ST7789_WriteCommand (0xD0);                     //     Power control
    ST7789_WriteSmallData (0xA4);                   //     Default value
    ST7789_WriteSmallData (0xA1);                   //     Default value
      /**************** Division line ****************/

      ST7789_WriteCommand(0xE0);
      {
            uint8_t data[] = {0xD0, 0x04, 0x0D, 0x11, 0x13, 0x2B, 0x3F, 0x54,
0x4C, 0x18, 0x0D, 0x0B, 0x1F, 0x23};
            ST7789_WriteData(data, sizeof(data));
      }

    ST7789_WriteCommand(0xE1);
      {
            uint8_t data[] = {0xD0, 0x04, 0x0C, 0x11, 0x13, 0x2C, 0x3F, 0x44,
0x51, 0x2F, 0x1F, 0x1F, 0x20, 0x23};
            ST7789_WriteData(data, sizeof(data));
      }
    ST7789_WriteCommand (ST7789_INVON);       //     Inversion ON
      ST7789_WriteCommand (ST7789_SLPOUT);    //     Out of sleep mode
      ST7789_WriteCommand (ST7789_NORON);           //     Normal Display on
      ST7789_WriteCommand (ST7789_DISPON);    //     Main screen turned on

      HAL_Delay(50);
      ST7789_Fill_Color(BLACK);                     //     Fill with Black.
```

```
}

/**
 * @brief Fill the DisplayWindow with single color
 * @param color -> color to Fill with
 * @return none
 */
void ST7789_Fill_Color(uint16_t color)
{
	uint16_t i, j;
	ST7789_SetAddressWindow(0, 0, ST7789_WIDTH - 1, ST7789_HEIGHT - 1);
	ST7789_Select();
	for (i = 0; i < ST7789_WIDTH; i++)
		for (j = 0; j < ST7789_HEIGHT; j++) {
			uint8_t data[] = {color >> 8, color & 0xFF};
			ST7789_WriteData(data, sizeof(data));
		}
	ST7789_UnSelect();
}

/**
 * @brief Draw a Pixel
 * @param x&y -> coordinate to Draw
 * @param color -> color of the Pixel
 * @return none
 */
void ST7789_DrawPixel(uint16_t x, uint16_t y, uint16_t color)
{
	if ((x < 0) || (x >= ST7789_WIDTH) ||
		(y < 0) || (y >= ST7789_HEIGHT))	return;

	ST7789_SetAddressWindow(x, y, x, y);
	uint8_t data[] = {color >> 8, color & 0xFF};
	ST7789_Select();
	ST7789_WriteData(data, sizeof(data));
	ST7789_UnSelect();
}

/**
 * @brief Fill an Area with single color
 * @param xSta&ySta -> coordinate of the start point
 * @param xEnd&yEnd -> coordinate of the end point
 * @param color -> color to Fill with
 * @return none
 */
void ST7789_Fill(uint16_t xSta, uint16_t ySta, uint16_t xEnd, uint16_t yEnd,
uint16_t color)
{
	if ((xEnd < 0) || (xEnd >= ST7789_WIDTH) ||
		(yEnd < 0) || (yEnd >= ST7789_HEIGHT))return;
	ST7789_Select();
```

```
	uint16_t i, j;
	ST7789_SetAddressWindow(xSta, ySta, xEnd, yEnd);
	for (i = ySta; i <= yEnd; i++)
		for (j = xSta; j <= xEnd; j++) {
			uint8_t data[] = {color >> 8, color & 0xFF};
			ST7789_WriteData(data, sizeof(data));
		}
	ST7789_UnSelect();
}

/**
 * @brief Draw a big Pixel at a point
 * @param x&y -> coordinate of the point
 * @param color -> color of the Pixel
 * @return none
 */
void ST7789_DrawPixel_4px(uint16_t x, uint16_t y, uint16_t color)
{
	if ((x <= 0) || (x > ST7789_WIDTH) ||
		(y <= 0) || (y > ST7789_HEIGHT))	return;
	ST7789_Select();
	ST7789_Fill(x - 1, y - 1, x + 1, y + 1, color);
	ST7789_UnSelect();
}

/**
 * @brief Draw a line with single color
 * @param x1&y1 -> coordinate of the start point
 * @param x2&y2 -> coordinate of the end point
 * @param color -> color of the line to Draw
 * @return none
 */
void ST7789_DrawLine(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1,
		uint16_t color) {
	uint16_t swap;
    uint16_t steep = ABS(y1 - y0) > ABS(x1 - x0);
    if (steep) {
		swap = x0;
		x0 = y0;
		y0 = swap;

		swap = x1;
		x1 = y1;
		y1 = swap;
        //_swap_int16_t(x0, y0);
        //_swap_int16_t(x1, y1);
    }

    if (x0 > x1) {
		swap = x0;
		x0 = x1;
```

```
            x1 = swap;

            swap = y0;
            y0 = y1;
            y1 = swap;
        //_swap_int16_t(x0, x1);
        //_swap_int16_t(y0, y1);
    }

    int16_t dx, dy;
    dx = x1 - x0;
    dy = ABS(y1 - y0);

    int16_t err = dx / 2;
    int16_t ystep;

    if (y0 < y1) {
        ystep = 1;
    } else {
        ystep = -1;
    }

    for (; x0<=x1; x0++) {
        if (steep) {
            ST7789_DrawPixel(y0, x0, color);
        } else {
            ST7789_DrawPixel(x0, y0, color);
        }
        err -= dy;
        if (err < 0) {
            y0 += ystep;
            err += dx;
        }
    }
}

/**
 * @brief Draw a Rectangle with single color
 * @param xi&yi -> 2 coordinates of 2 top points.
 * @param color -> color of the Rectangle line
 * @return none
 */
void ST7789_DrawRectangle(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2,
uint16_t color)
{
      ST7789_Select();
      ST7789_DrawLine(x1, y1, x2, y1, color);
      ST7789_DrawLine(x1, y1, x1, y2, color);
      ST7789_DrawLine(x1, y2, x2, y2, color);
      ST7789_DrawLine(x2, y1, x2, y2, color);
      ST7789_UnSelect();
```

```
}

/**
 * @brief Draw a circle with single color
 * @param x0&y0 -> coordinate of circle center
 * @param r -> radius of circle
 * @param color -> color of circle line
 * @return  none
 */
void ST7789_DrawCircle(uint16_t x0, uint16_t y0, uint8_t r, uint16_t color)
{
	int16_t f = 1 - r;
	int16_t ddF_x = 1;
	int16_t ddF_y = -2 * r;
	int16_t x = 0;
	int16_t y = r;

	ST7789_Select();
	ST7789_DrawPixel(x0, y0 + r, color);
	ST7789_DrawPixel(x0, y0 - r, color);
	ST7789_DrawPixel(x0 + r, y0, color);
	ST7789_DrawPixel(x0 - r, y0, color);

	while (x < y) {
		if (f >= 0) {
			y--;
			ddF_y += 2;
			f += ddF_y;
		}
		x++;
		ddF_x += 2;
		f += ddF_x;

		ST7789_DrawPixel(x0 + x, y0 + y, color);
		ST7789_DrawPixel(x0 - x, y0 + y, color);
		ST7789_DrawPixel(x0 + x, y0 - y, color);
		ST7789_DrawPixel(x0 - x, y0 - y, color);

		ST7789_DrawPixel(x0 + y, y0 + x, color);
		ST7789_DrawPixel(x0 - y, y0 + x, color);
		ST7789_DrawPixel(x0 + y, y0 - x, color);
		ST7789_DrawPixel(x0 - y, y0 - x, color);
	}
	ST7789_UnSelect();
}

/**
 * @brief Draw an Image on the screen
 * @param x&y -> start point of the Image
 * @param w&h -> width & height of the Image to Draw
 * @param data -> pointer of the Image array
```

```
 * @return none
 */
void ST7789_DrawImage(uint16_t x, uint16_t y, uint16_t w, uint16_t h, const
uint16_t *data)
{
	if ((x >= ST7789_WIDTH) || (y >= ST7789_HEIGHT))
		return;
	if ((x + w - 1) >= ST7789_WIDTH)
		return;
	if ((y + h - 1) >= ST7789_HEIGHT)
		return;

	ST7789_Select();
	ST7789_SetAddressWindow(x, y, x + w - 1, y + h - 1);
	ST7789_WriteData((uint8_t *)data, sizeof(uint16_t) * w * h);
	ST7789_UnSelect();
}

/**
 * @brief Invert Fullscreen color
 * @param invert -> Whether to invert
 * @return none
 */
void ST7789_InvertColors(uint8_t invert)
{
	ST7789_Select();
	ST7789_WriteCommand(invert ? 0x21 /* INVON */ : 0x20 /* INVOFF */);
	ST7789_UnSelect();
}

/**
 * @brief Write a char
 * @param  x&y -> cursor of the start point.
 * @param ch -> char to write
 * @param font -> fontstyle of the string
 * @param color -> color of the char
 * @param bgcolor -> background color of the char
 * @return  none
 */
void ST7789_WriteChar(uint16_t x, uint16_t y, char ch, FontDef font, uint16_t
color, uint16_t bgcolor)
{
	uint32_t i, b, j;
	ST7789_Select();
	ST7789_SetAddressWindow(x, y, x + font.width - 1, y + font.height - 1);

	for (i = 0; i < font.height; i++) {
		b = font.data[(ch - 32) * font.height + i];
		for (j = 0; j < font.width; j++) {
			if ((b << j) & 0x8000) {
				uint8_t data[] = {color >> 8, color & 0xFF};
```

```
                        ST7789_WriteData(data, sizeof(data));
                }
                else {
                        uint8_t data[] = {bgcolor >> 8, bgcolor & 0xFF};
                        ST7789_WriteData(data, sizeof(data));
                }
        }
    }
    ST7789_UnSelect();
}

/**
 * @brief Write a string
 * @param  x&y -> cursor of the start point.
 * @param str -> string to write
 * @param font -> fontstyle of the string
 * @param color -> color of the string
 * @param bgcolor -> background color of the string
 * @return  none
 */
void ST7789_WriteString(uint16_t x, uint16_t y, const char *str, FontDef font,
uint16_t color, uint16_t bgcolor)
{
    ST7789_Select();
    while (*str) {
        if (x + font.width >= ST7789_WIDTH) {
            x = 0;
            y += font.height;
            if (y + font.height >= ST7789_HEIGHT) {
                break;
            }

            if (*str == ' ') {
                // skip spaces in the beginning of the new line
                str++;
                continue;
            }
        }
        ST7789_WriteChar(x, y, *str, font, color, bgcolor);
        x += font.width;
        str++;
    }
    ST7789_UnSelect();
}

/**
 * @brief Draw a filled Rectangle with single color
 * @param  x&y -> coordinates of the starting point
 * @param w&h -> width & height of the Rectangle
 * @param color -> color of the Rectangle
 * @return  none
```

```
 */
void ST7789_DrawFilledRectangle(uint16_t x, uint16_t y, uint16_t w, uint16_t h,
uint16_t color)
{
	ST7789_Select();
	uint8_t i;

	/* Check input parameters */
	if (x >= ST7789_WIDTH ||
		y >= ST7789_HEIGHT) {
		/* Return error */
		return;
	}

	/* Check width and height */
	if ((x + w) >= ST7789_WIDTH) {
		w = ST7789_WIDTH - x;
	}
	if ((y + h) >= ST7789_HEIGHT) {
		h = ST7789_HEIGHT - y;
	}

	/* Draw lines */
	for (i = 0; i <= h; i++) {
		/* Draw lines */
		ST7789_DrawLine(x, y + i, x + w, y + i, color);
	}
	ST7789_UnSelect();
}

/**
 * @brief Draw a Triangle with single color
 * @param  xi&yi -> 3 coordinates of 3 top points.
 * @param color ->color of the lines
 * @return  none
 */
void ST7789_DrawTriangle(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2,
uint16_t x3, uint16_t y3, uint16_t color)
{
	ST7789_Select();
	/* Draw lines */
	ST7789_DrawLine(x1, y1, x2, y2, color);
	ST7789_DrawLine(x2, y2, x3, y3, color);
	ST7789_DrawLine(x3, y3, x1, y1, color);
	ST7789_UnSelect();
}

/**
 * @brief Draw a filled Triangle with single color
 * @param  xi&yi -> 3 coordinates of 3 top points.
 * @param color ->color of the triangle
```

```
 * @return  none
 */
void ST7789_DrawFilledTriangle(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t
y2, uint16_t x3, uint16_t y3, uint16_t color)
{
	ST7789_Select();
	int16_t deltax = 0, deltay = 0, x = 0, y = 0, xinc1 = 0, xinc2 = 0,
			yinc1 = 0, yinc2 = 0, den = 0, num = 0, numadd = 0,
numpixels = 0,
			curpixel = 0;

	deltax = ABS(x2 - x1);
	deltay = ABS(y2 - y1);
	x = x1;
	y = y1;

	if (x2 >= x1) {
		xinc1 = 1;
		xinc2 = 1;
	}
	else {
		xinc1 = -1;
		xinc2 = -1;
	}

	if (y2 >= y1) {
		yinc1 = 1;
		yinc2 = 1;
	}
	else {
		yinc1 = -1;
		yinc2 = -1;
	}

	if (deltax >= deltay) {
		xinc1 = 0;
		yinc2 = 0;
		den = deltax;
		num = deltax / 2;
		numadd = deltay;
		numpixels = deltax;
	}
	else {
		xinc2 = 0;
		yinc1 = 0;
		den = deltay;
		num = deltay / 2;
		numadd = deltax;
		numpixels = deltay;
	}
```

```
	for (curpixel = 0; curpixel <= numpixels; curpixel++) {
		ST7789_DrawLine(x, y, x3, y3, color);

		num += numadd;
		if (num >= den) {
			num -= den;
			x += xinc1;
			y += yinc1;
		}
		x += xinc2;
		y += yinc2;
	}
	ST7789_UnSelect();
}

/**
 * @brief Draw a Filled circle with single color
 * @param x0&y0 -> coordinate of circle center
 * @param r -> radius of circle
 * @param color -> color of circle
 * @return  none
 */
void ST7789_DrawFilledCircle(int16_t x0, int16_t y0, int16_t r, uint16_t color)
{
	ST7789_Select();
	int16_t f = 1 - r;
	int16_t ddF_x = 1;
	int16_t ddF_y = -2 * r;
	int16_t x = 0;
	int16_t y = r;

	ST7789_DrawPixel(x0, y0 + r, color);
	ST7789_DrawPixel(x0, y0 - r, color);
	ST7789_DrawPixel(x0 + r, y0, color);
	ST7789_DrawPixel(x0 - r, y0, color);
	ST7789_DrawLine(x0 - r, y0, x0 + r, y0, color);

	while (x < y) {
		if (f >= 0) {
			y--;
			ddF_y += 2;
			f += ddF_y;
		}
		x++;
		ddF_x += 2;
		f += ddF_x;

		ST7789_DrawLine(x0 - x, y0 + y, x0 + x, y0 + y, color);
		ST7789_DrawLine(x0 + x, y0 - y, x0 - x, y0 - y, color);

		ST7789_DrawLine(x0 + y, y0 + x, x0 - y, y0 + x, color);
```

```
            ST7789_DrawLine(x0 + y, y0 - x, x0 - y, y0 - x, color);
        }
        ST7789_UnSelect();
}


/**
 * @brief Open/Close tearing effect line
 * @param tear -> Whether to tear
 * @return none
 */
void ST7789_TearEffect(uint8_t tear)
{
        ST7789_Select();
        ST7789_WriteCommand(tear ? 0x35 /* TEON */ : 0x34 /* TEOFF */);
        ST7789_UnSelect();
}


/**
 * @brief A Simple test function for ST7789
 * @param  none
 * @return  none
 */
void ST7789_Test(void)
{
        ST7789_Fill_Color(WHITE);
        HAL_Delay(1000);
        ST7789_WriteString(10, 20, "Speed Test", Font_11x18, RED, WHITE);
        HAL_Delay(1000);
        ST7789_Fill_Color(CYAN);
    HAL_Delay(500);
        ST7789_Fill_Color(RED);
    HAL_Delay(500);
        ST7789_Fill_Color(BLUE);
    HAL_Delay(500);
        ST7789_Fill_Color(GREEN);
    HAL_Delay(500);
        ST7789_Fill_Color(YELLOW);
    HAL_Delay(500);
        ST7789_Fill_Color(BROWN);
    HAL_Delay(500);
        ST7789_Fill_Color(DARKBLUE);
    HAL_Delay(500);
        ST7789_Fill_Color(MAGENTA);
    HAL_Delay(500);
        ST7789_Fill_Color(LIGHTGREEN);
    HAL_Delay(500);
        ST7789_Fill_Color(LGRAY);
    HAL_Delay(500);
        ST7789_Fill_Color(LBBLUE);
```

```
    HAL_Delay(500);
      ST7789_Fill_Color(WHITE);
      HAL_Delay(500);

      ST7789_WriteString(10, 10, "Font test.", Font_16x26, GBLUE, WHITE);
      ST7789_WriteString(10, 50, "Hello Steve!", Font_7x10, RED, WHITE);
      ST7789_WriteString(10, 75, "Hello Steve!", Font_11x18, YELLOW, WHITE);
      ST7789_WriteString(10, 100, "Hello Steve!", Font_16x26, MAGENTA, WHITE);
      HAL_Delay(1000);

      ST7789_Fill_Color(RED);
      ST7789_WriteString(10, 10, "Rect./Line.", Font_11x18, YELLOW, RED);
      ST7789_DrawRectangle(30, 30, 100, 100, WHITE);
      HAL_Delay(1000);

      ST7789_Fill_Color(RED);
      ST7789_WriteString(10, 10, "Filled Rect.", Font_11x18, YELLOW, RED);
      ST7789_DrawFilledRectangle(30, 30, 50, 50, WHITE);
      HAL_Delay(1000);


      ST7789_Fill_Color(RED);
      ST7789_WriteString(10, 10, "Circle.", Font_11x18, YELLOW, RED);
      ST7789_DrawCircle(60, 60, 25, WHITE);
      HAL_Delay(1000);

      ST7789_Fill_Color(RED);
      ST7789_WriteString(10, 10, "Filled Cir.", Font_11x18, YELLOW, RED);
      ST7789_DrawFilledCircle(60, 60, 25, WHITE);
      HAL_Delay(1000);

      ST7789_Fill_Color(RED);
      ST7789_WriteString(10, 10, "Triangle", Font_11x18, YELLOW, RED);
      ST7789_DrawTriangle(30, 30, 30, 70, 60, 40, WHITE);
      HAL_Delay(1000);

      ST7789_Fill_Color(RED);
      ST7789_WriteString(10, 10, "Filled Tri", Font_11x18, YELLOW, RED);
      ST7789_DrawFilledTriangle(30, 30, 30, 70, 60, 40, WHITE);
      HAL_Delay(1000);

      //     If FLASH cannot storage anymore datas, please delete codes below.
      ST7789_Fill_Color(WHITE);
      ST7789_DrawImage(0, 0, 128, 128, (uint16_t *)saber);
      HAL_Delay(3000);
}
```

```
/*
 * task_functions.h
 *
```

```
 *      Author: kevinmancini
 */

#ifndef SRC_TASK_FUNCTIONS_H_
#define SRC_TASK_FUNCTIONS_H_

// import file with task logic here

#include "humidity-task.h"

#endif /* SRC_TASK_FUNCTIONS_H_ */
```

```
/*
 * task-results.h
 *
 *      Author: tomas
 */

#ifndef INC_TASK_RESULTS_H_
#define INC_TASK_RESULTS_H_


enum task_result{
       TASK_RUNNING,
       TASK_FAILED,
       TASK_PASSED,
       TASK_NOT_STARTED
};

typedef enum task_result task_result_t;

#endif /* INC_TASK_RESULTS_H_ */
```

```
/*
 * temp_sensor.h
 *
 *      Author: Inês
 */

#include <stdlib.h>
#include <stdio.h>

 uint8_t TSensor_Task(void);  //Temperature Sensor: Task (temperature level)
 uint8_t ReadADC_MCP3002(void);  //MCP3002 ADC: Task (record voice)

 void TSensor_Who_Am_I_Test(void);  //Temperature Sensor: Test 1
 void TSensor_Initiation_Test(void);  //Temperature Sensor: Test 2
 void TSensor_Admitted_Current_Temperature_Value_Test(void);  //Temperature
Sensor: Test 3
```

```c
/*
 * temp_sensor.c
 *
 *      Author: Inês
 */

#include "stm32l475e_iot01.h"
#include "stm32l475e_iot01_tsensor.h"
#include "stm32l4xx_hal_uart.h"
#include "stm32l4xx_hal_i2c.h"
#include <math.h>

#include <stdlib.h>
#include <stdio.h>
#include "temp_sensor.h"


/* Private variables ---------------------------------------------------------*/
I2C_HandleTypeDef hi2c2;

SPI_HandleTypeDef hspi1;

UART_HandleTypeDef huart1;


/* Global Variables */

uint8_t WhoAmI_Address = 0x0F;  // Who_Am_I Address
uint8_t TN1218_Temperature_WriteAddress = 0xBE; // Write Address
uint8_t TN1218_Temperature_ReadAddress = 0xBF;  // Read Address

uint8_t value_temperature_level = 23;  // Value the player must reach on the
temperature level
uint8_t seconds_temperature_level = 10;  // Seconds the player has to reach the
temperature value

float temperature_value = 0;   // Aux variable to the measured temperature value
float current_temperature_value = 0;   // Aux variable to the measured current
temperature value


/**
  * @brief  Read Who_Am_I register.
  * @retval Temperature Sensor Device Identifier
  */
 uint8_t TSensor_Who_Am_I(void){

      uint8_t Device_ID;

      HAL_I2C_Master_Transmit(&hi2c2, TN1218_Temperature_WriteAddress,
```

```
&WhoAmI_Address, 1, 100);
	HAL_I2C_Master_Receive(&hi2c2, TN1218_Temperature_ReadAddress,
&Device_ID, 1, 100);
	HAL_UART_Transmit(&huart1, &Device_ID, 1, 100);

	return Device_ID;
 }

 /**
   * @brief  Periodically read the temperature value from the register, checking
if the temperature level was passed or failed.
   * @retval 0 if the level was passed
   */
 uint8_t TSensor_Task(void){

		int i;  // Aux variable for the cycles
		uint8_t temperature1;  // Integer part of the temperature value
		uint8_t temperature2;  // Fractional part of the temperature value
		float temperature_fraction;  // Aux variable for the preparation of
the fractional part of the temperature value

		//Formatted Messages
		char str_temperature[100] = ""; // Formatted message to display the
temperature value

		//Messages
		uint8_t message3[] = "***** Temperature Level Initialized *****
\r\n\n ";
		uint8_t message4[] = " Level Completed! :) \r\n ";
		uint8_t message5[] = "Press the Blue Button to Play the Same Level
Again \r\n\n ";
		uint8_t message6[] = " Level Failed! :( \r\n ";
		uint8_t message7[] = "Press the Blue Button to Try Again \r\n\n ";

	while(1){
	 for(i=0; i<10; i++) {  // For cycle to count how many times the
temperature value is read. This can be translated in seconds (an approximation)

		temperature_value = BSP_TSENSOR_ReadTemp();  // Get the value
from the register
		temperature1 = temperature_value;  // Get the integer part of the
temperature value
		temperature_fraction = temperature_value - temperature1;  //
Prepare the fraction part
		temperature2 = trunc(temperature_fraction * 100);  // Get the
fraction part of the temperature value

		snprintf(str_temperature,100," Temperature = %d.%02d\n\r",
temperature1, temperature2);  // Print the temperature value
		HAL_UART_Transmit(&huart1,( uint8_t *
)str_temperature,sizeof(str_temperature),1000);
```

```
                HAL_Delay(1000);  // Wait 1 second

                if(i == seconds_temperature_level - 1) { // If the player has not
reached the temperature value during the given time

                HAL_UART_Transmit(&huart1,message6,sizeof(message6),1000);  //
Message: level failed

                HAL_UART_Transmit(&huart1,message7,sizeof(message7),1000);  //
Message: press the blue button to try again
                while(BSP_PB_GetState(BUTTON_USER) == GPIO_PIN_SET);  // Function
to press the set button
                while(BSP_PB_GetState(BUTTON_USER) == GPIO_PIN_RESET);  //
Function to press the reset button (good practice)
                }

                if(temperature1 == value_temperature_level) {  // If the player
has reached the temperature value in time
                        i = seconds_temperature_level - 1 ;

                        HAL_UART_Transmit(&huart1,message4,sizeof(message4),1000);
// Message: level passed

                        return 0;
                }
              }
       }
 }

 /**
   * @brief Read the ADC voltage during approximately 10 seconds, for a future
playback of the recorded voice.
   * @retval 0 when the recording is over
   */
uint8_t ReadADC_MCP3002(){

       //Hardware Variables
       uint8_t voltage_ref_ADC = 5;

       //Sound In
       uint8_t k;  // Aux varibled for the cycle
       uint8_t TxData_ADC[2];
       uint8_t RxData_ADC[2];
       uint16_t RxData_ADC_shifted;
       char buf[30];
       float voltage_ADC;

        // Message
    uint8_t message15[] = " \r\n ***** MCP3002 ADC Voltage  ***** \r\n ";
        uint8_t message16[] = "Press the Button to Record Voice \r\n\n ";
        uint8_t message17[] = "Recording is Over \r\n\n ";
```

```
      // Construct the message according to the Sound In Hardware and the
MCP3002 datasheet
        TxData_ADC[0] = 0b11;
        TxData_ADC[0] = ((TxData_ADC[0] << 1) + 0) << 5;
        TxData_ADC[1] = 0;

         HAL_UART_Transmit(&huart1,message15,sizeof(message15),1000);  //
Message: name of the task

         HAL_UART_Transmit(&huart1,message16,sizeof(message16),1000);  //
Message: press the blue button to start recording
         while(BSP_PB_GetState(BUTTON_USER) == GPIO_PIN_SET);  // Function to
press the set button
         while(BSP_PB_GetState(BUTTON_USER) == GPIO_PIN_RESET);  // Function to
press the reset button (good practice)

        for(k=0; k<50; k++) { // For cycle to read the ADC voltage 50 times at
200 ms intervals, i.e. for approximately 10 seconds

        //Get the reply
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_RESET);  //Chip Selector
(CS): 6 Digital Pin
        HAL_SPI_TransmitReceive(&hspi1, TxData_ADC, RxData_ADC, 2, 1000);  //Get
the data
        HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_SET);  //Chip Selector
(CS): 6 Digital Pin

        // Combining 2 bytes into 1 byte - Get ADC value by shifting right once
and masking out all but last 10 bits
        RxData_ADC_shifted = ((uint16_t)RxData_ADC[0] << 8) + RxData_ADC[1];
        RxData_ADC_shifted >>= 1;
        RxData_ADC_shifted &= 0b0000001111111111;

        // Calculate the ADC voltage
        voltage_ADC = (5 * RxData_ADC_shifted) / 1024; //Multiply by 5 because
it is the reference voltage and Divided by 1024 because MCP3002 is a 10-bit ADC

        /* Convert voltage to integer and decimal format */
        voltage_ADC = voltage_ADC * 100;
        sprintf(buf, "Voltage = %u.%02u V\r\n",((unsigned int)voltage_ADC /
100),((unsigned int)voltage_ADC % 100)); //Recast the float into two parts (2
integers) and print decimal figure
        HAL_UART_Transmit(&huart1, (uint8_t *)buf, strlen(buf), 1000);

        HAL_Delay(200);  // Wait 200 ms
        }

        HAL_UART_Transmit(&huart1,message17,sizeof(message17),1000);  //
Message: recording is over
        return 0;
```

```
}

/**
  * @brief  Who_Am_I Test
  * @retval -
  */
void TSensor_Who_Am_I_Test(){

	uint8_t Device_ID_aux;

	//Formatted Messages
	char str[100] = ""; // Formatted message to display

	//Debug Messages
	uint8_t message8[] = "\n\n\n\n\n ***** Who_Am_I TEST ***** \r\n ";
	uint8_t message9[] = " ***** Who_Am_I TEST Successfully Performed *****
\r\n ";
	uint8_t message10[] = " ***** ERROR - Who_Am_I TEST ***** \r\n ";


	  /* Who_Am_I TEST*/
	  HAL_UART_Transmit(&huart1,message8,sizeof(message8),1000);  // Message:
name of the test
	  Device_ID_aux = TSensor_Who_Am_I();   // Get the Device Identifier

	  if(Device_ID_aux == 188){  //If the device identifier is what it's
supposed to be (188)
		snprintf(str,100,"Device ID = %d\n\r", Device_ID_aux);  //Print
the device identifier obtained from the register
		HAL_UART_Transmit(&huart1,( uint8_t * )str,sizeof(str),1000);
		HAL_UART_Transmit(&huart1,message9,sizeof(message9),1000);  //
Message: test was successful
	  }

	  else{   // If the device identifier is different from 188
		snprintf(str,100,"Device ID = %d\n\r", Device_ID_aux);  // Print
the number obtained from the register
		HAL_UART_Transmit(&huart1,message10,sizeof(message10),1000);  //
Message: test failed
	  }
}

/**
  * @brief  Sensor Initiation and Test
  * @retval -
  */
void TSensor_Initiation_Test(){

	uint8_t Sensor_Initiation_aux;

	//Debug Messages
```

```
	uint8_t message1[] = "\n***** Initialize Temperature Sensor and TEST
***** \r\n ";
	uint8_t message2[] = "***** Temperature Sensor Initialized ***** \r\n\n
";
	uint8_t message11[] = "***** ERROR - Sensor Initiation TEST ***** \r\n\n
";

	  /* Sensor Initiation and TEST */
	  HAL_UART_Transmit(&huart1,message1,sizeof(message1),1000);  // Message:
name of the test
	  Sensor_Initiation_aux = BSP_TSENSOR_Init();  // Sensor initialisation

	  if(Sensor_Initiation_aux == 0) {  // If the variable returned by the
initialization function is 0
	  HAL_UART_Transmit(&huart1,message2,sizeof(message2),1000);  // Message:
test was successful
	  }

	  if(Sensor_Initiation_aux == 1) { // If the variable returned by the
initialization function is 1
		HAL_UART_Transmit(&huart1,message11,sizeof(message11),1000);  //
Message: test failed
	  }
}

/**
  * @brief  Admitted Current Temperature Value Test
  * @retval -
  */
void TSensor_Admitted_Current_Temperature_Value_Test(){ //Admitted Current
Temperature Value TEST

	uint8_t current_temperature1;
	uint8_t current_temperature2;
	float current_temperature_fraction;

	//Formatted Messages
	char str_current_temperature[100] = ""; // Formatted message to display
the current temperature value

	//Debug Messages
	uint8_t message12[] = "***** Admitted Current Temperature Value TEST
***** \r\n ";
	uint8_t message13[] = " ***** Admitted Current Temperature Value TEST
Successfully Performed ***** \r\n\n ";
	uint8_t message14[] = " ***** ERROR - Admitted Current Temperature Value
TEST ***** \r\n\n ";

	  HAL_UART_Transmit(&huart1,message12,sizeof(message12),1000);  //
Message: name of the test
```

```
        current_temperature_value = BSP_TSENSOR_ReadTemp();  // Get the value
from the register
        current_temperature1 = current_temperature_value;  // Get the integer
part of the temperature value
        current_temperature_fraction = current_temperature_value -
current_temperature1;   // Prepare the fraction part
        current_temperature2 = trunc(current_temperature_fraction * 100);    //
Get the fraction part of the temperature value

        snprintf(str_current_temperature,100," Current Temperature =
%d.%02d\n\r", current_temperature1, current_temperature2);  // Print the current
temperature value
        HAL_UART_Transmit(&huart1,( uint8_t *
)str_current_temperature,sizeof(str_current_temperature),1000);

        if(current_temperature1>-40 && current_temperature1<120){  // If the
temperature value is within an admissible range
              HAL_UART_Transmit(&huart1,message13,sizeof(message13),1000);  //
Message: test was successful
        }

        else{  // If the temperature value is not within an admissible range
              HAL_UART_Transmit(&huart1,message14,sizeof(message14),1000);  //
Message: test failed
        }

}
```

```
/*
 * test_functions.h
 *
 *      Author: kevinmancini
 */

#ifndef SRC_TEST_FUNCTIONS_H_
#define SRC_TEST_FUNCTIONS_H_

#include "humidity-tests.h"


int runAllTests();

#endif /* SRC_TEST_FUNCTIONS_H_ */
```

```
/*
 * test_functions.c
 *
 *      Author: tomas
 */
```

```c
#include "test_functions.h"


int runAllTests(){
	// Add more tests in order with specific return codes
	if (humidityRunAutoTests() == 1){
		return 1;
	}
	return 0;
}
```