***Internet of Things Course***

**LAB 6**

**IoT Sensors**

**Issue 1.0**

Contents

[1 Introduction 1](#_Toc36655910)

[Lab overview 1](#_Toc36655911)

[2 Requirements 1](#_Toc36655912)

[2.1 Software functions 1](#_Toc36655913)

[3 Application Code 2](#_Toc36655914)

[3.1 Program structure 3](#_Toc36655915)

# Introduction

## Lab overview

In this lab we will learn how to interface with environmental sensors using the DISCO-L475VG-IOT01A board and Mbed API.

The DISCO-L475VG-IOT01A device comes with on-board sensors, including a pressure sensor (LPS22HB), 3-axis accelerometer and 3-axis gyroscope (LSM6DSL), a 3-axis magnetometer (LIS3MDL), a humidity sensor (HTS221), a capacitive digital sensor for relative humidity and temperature (HTS221), a Time-of-Flight and gesture-detection sensor (VL53L0X), and two digital omnidirectional microphones (MP34DT01).

In this lab, we will read data from the sensors, transmit them to the PC via UART, and display the readings on a terminal emulator.

# Requirements

## Software functions

The functions that may be used in this lab are listed below:

Table 1: Software Functions

|  |  |
| --- | --- |
| **Function name** | **Description**  |
| ***UART functions*** |
| Serial (PinName tx, PinName rx, const char \*name=NULL) | Create a Serial port connected to the specified transmit and receive pins |
| void baud (int baudrate) | Set the baud rate of the serial port |
| void format (int bits=8, Parity parity=SerialBase::None, int stop\_bits=1) | Set the transmission format used by the serial port |
| int readable () | Determine if there is a character available to read |
| int writeable () | Determine if there is space available to write a character |
| void attach (void(\*fptr)(void), IrqType type=RxIrq) | Attach a function to call whenever a serial interrupt is generated |
| void send\_break () | Generate a break condition on the serial line |
| void set\_flow\_control (Flow type, PinName flow1=NC, PinName flow2=NC) | Set the flow control type on the serial port |
| int putc(int ch, FILE \*stream ) | Writes the character ch to stream. Function returns the character written, or EOF if an error happens |
| int getc(FILE \*stream ) | Read a character from the stream; an EOF indicates the end of file is reached |
| int printf( const char \*format, ... ) | Prints output both text string and data, according to format and other arguments passed to printf() |
| ***On-board Sensor functions*** |
| **Initialization of Sensors** |
| BSP\_B-L475E-IOT01: | HTS221 | Temperature | BSP\_TSENSOR\_Init() |
| Humidity | BSP\_HSENSOR\_Init() |
| LPS22HB | Pressure | BSP\_PSENSOR\_Init() |
| LIS3MDL | Magnetometer | BSP\_MAGNETO\_Init() |
| LSM6DSL | Accelerometer | BSP\_ACCELERO\_Init() |
| Gyroscope | BSP\_GYRO\_Init() |
| **Reading Data from sensors** |
| HTS221 class | float BSP\_TSENSOR\_ReadTemp(void) | Get the temperature from the HTS221 sensor  |
| float BSP\_HSENSOR\_ReadHumidity(void) | Get the humidity from the HTS221 sensor |
| LPS22HB class | float BSP\_PSENSOR\_ReadPressure(void) | Get the pressure from the sensor LPS25H |
| uint8\_t BSP\_PSENSOR\_ReadID(void) | Get the raw lecture of the pressure sensor |
| LIS3MDL class | void BSP\_MAGNETO\_GetXYZ(int16\_t \*pDataXYZ) | Get the 3-axis values of the magnetometer |
| void BSP\_MAGNETO\_LowPower(uint16\_t status) | Activates low power mode in the magnetometer |
| LSM6DSL | void BSP\_GYRO\_GetXYZ(float\* pfData) | Get the 3-axis values of the gyroscope |
| void BSP\_GYRO\_LowPower(uint16\_t status) | Activates low power mode in the magnetometer |
| void BSP\_ACCELERO\_AccGetXYZ(int16\_t \*pDataXYZ) | Get the 3-axis values of the accelerometer |
| void BSP\_ACCELERO\_LowPower(uint16\_t status) | Activates low power mode in the magnetometer |

# Application Code

In this lab exercise we will write a program that reads the temperature, humidity, pressure, magnetometer, accelerometer, and gyroscope values from the sensors on-board, every 3 seconds. The temperature is measured in degrees Celsius but we will convert it to both Fahrenheit and Kelvins.

The program will then enter sleep mode and wait for interrupts.

At the same time the program will blink a LED every second to show its aliveness.

We will inspect the sensor readings via the serial debug interface.

## Program structure

* Initialization
	+ Create a DigitalOut object for the LED.
	+ Initialize variables.
* Handlers
	+ Toggle the LED and update the measurements.
	+ Raise a flag that indicates that the measurements need to be read and displayed again.
* Main function
	+ Check if the flag is high.
	+ Read from the sensors.
	+ Convert the temperature into Fahrenheit and Kelvins.
	+ Enter sleep mode and wait for interrupts.