Laura-Mihaela LELUȚIU

ANALYSIS AND SIGNAL PROCESSING

Applications with Arduino

Transilvania University Press

2025

EDITURA UNIVERSITĂȚII TRANSILVANIA DIN BRAȘOV

Adresa: Str. Iuliu Maniu nr. 41A
500091 Brașov
Tel.: 0268 476 050
Fax: 0268 476 051
E-mail: editura@unitbv.ro

Editură recunoscută CNCSIS, cod 81

ISBN 978-606-19-1837-9 (e-book)

Lucrarea a fost avizată de Consiliul Departamentului de Inginerie electrică și fizică aplicată, Facultatea de Inginerie electrică și știința calculatoarelor a Universității Transilvania din Brașov.

Content

PREFACE

This book was originally designed to be useful for students taking the course "Signal Analysis and Processing" at Transilvania University of Brașov, Romania Faculty of Electrical Engineering and Computer Science Department of Electrical Engineering and Applied Physics, Research Department - Advanced Electrical Systems.

At the same time, the work is also useful for students in the electronic profile from other faculties and universities, as well as students from related specialties.

The book contains eight practical projects made with the Arduino Uno development board, some of which include several individual applications. The topics covered in the papers include presenting the basic principles in using the Arduino Uno development board, the principles of using analog or digital sensors to measure environmental parameters, distance and proximity, lighting level, voltage and current intensity, as well as how to create a real-time clock, which can be useful in creating complex circuits along with the other applications presented.

Each paper has been designed and structured so as to include all the information necessary to complete it, without the student having to consult previous papers, with the risk of repeating certain elements already presented in them, with the idea of maintaining ease in the implementation of each

application. Each paper contains an introductory part in which fundamental concepts useful in understanding how to use the modules and components used are highlighted, a detailed presentation of the hardware components and, where necessary, how to configure them, an individual explanation of the instructions, functions and operators used in the code sequences, detailing how to make the electronic assembly, as well as a presentation of the logic diagram of the operations and the complete code sequence, with an explanation of the role of each line of code. In addition, each paper contains a set of additional exercises and conclusions at the end that help consolidate the concepts addressed in the applications.

Paper 1

INTRODUCTION
BASIC PRINCIPLES IN USING A DEVELOPMENT BOARD

1. Work Description

1.1. Objectives of the Work

- Understanding the main features of the Arduino Uno board.
- Understanding how to use/program the analog and digital input/output ports of the Arduino Uno board.

1.2. Theoretical Description

Introduction

Arduino is the most popular open-source electronic platform in the business. Arduino boards are used to create projects either in small or bigger projects as it is destinated for teaching but if it used by experts, you can create many interesting and very complex projects. The Arduino boards are able to read digital or analog inputs from sensors and other and through the Arduino IDE the we can choose to do whatever we want and we can send them as output to control a motor, lighting a LED or actioning a radio transceiver module. The Arduino Integrated Development Environment (IDE), is an application used for writing and editing the code that drives the Arduino boards. The IDE code editor also has many features such as syntax highlighting, automatic indenting, text cutting and pasting and it also provides a button for compiling and uploading the code directly to an Arduino board, there is also a text console in

which you can find information if the code was successfully compiled and uploaded to the board but also how much storage space and dynamic memory it uses. It was written in the Java, C and C++ programming language. It is operational on Windows, macOS and Linux. The code written in the application are called sketches. The IDE also offers pre-made sketches that can help for different types of projects, the sketches also vary from easy programs to light up an LED to transforming the Arduino into a AVRISP microcontroller. Different boards have different connections ports, for example Arduino uno uses a USB type B and the Arduino nano uses the USB type B mini. Once connected to the computer in the IDE we must choose the type of board we use and the port of the computer to which it is connected so that the upload can be properly done otherwise we will receive error messages in text console.

To choose the board and port of the connection you must go to TOOLS and there we can choose what Arduino we use and the port to which is connected, if we use the Arduino nano, we might also need to choose a different processor as well because some boards may use older bootloaders. The code for Arduino projects usually consists of 3 main parts [1]:

1. The initialization section in which the pins utilized by the board are initialized
2. The setup section in which the code sets the initial values
3. And the last part is the loop section where the main code is located that is run repeatedly, it is used for control.

The board can be powered either through the USB port or from an external source via the power jack. The positioning and labeling of all the board's ports and pins can be seen in the following figures.

Figure 1. 1 ***Arduino Uno Development Board*** **(from [1])**

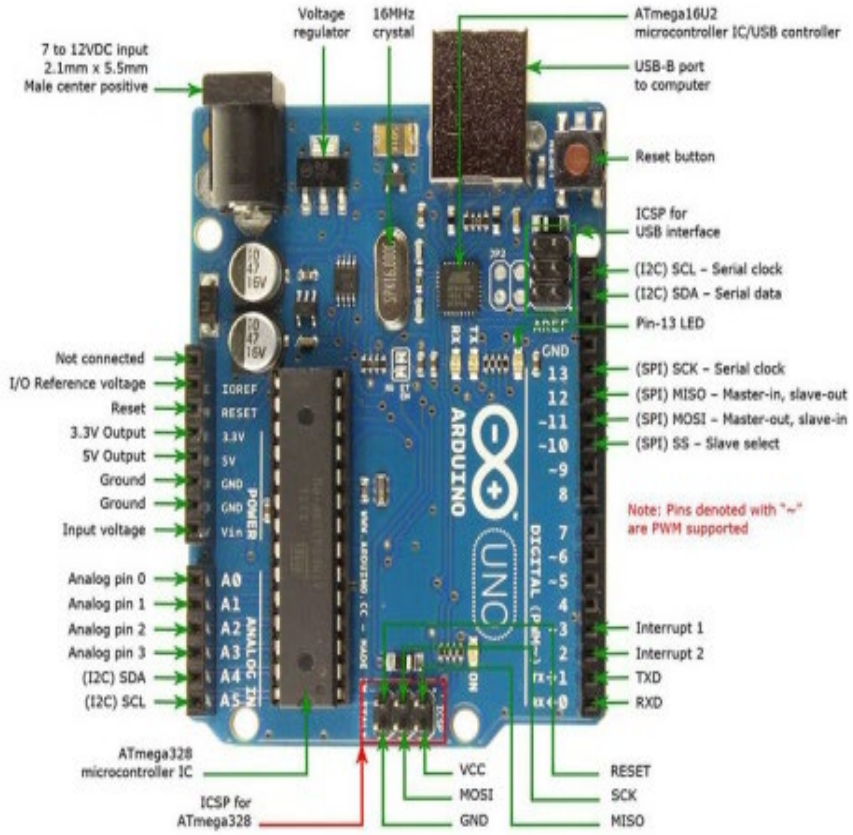


Figure 1.2. ***Ports of the Arduino Uno Development Board*** **(from [1], [6])**

The following ports will be used during the lab work [1]:

- **5 V power pin**: voltage supplied by the board's internal power source (avoid using this pin to power high-current external modules).
- **GND**: ground pin.
- **Analog input pin**
- **Digital input/output pin**: 5 V, maximum 40 mA.
- **PWM digital output pin**: 5 V, maximum 40 mA. PWM (Pulse Width Modulation) involves the controlled variation of the output voltage waveform by rapidly switching the logic level from 1 to 0 (the signal frequency is approximately 490 Hz), depending on a duty cycle (its value can range from 0 to 255).
    - This allows for generating a variable power signal and simulating analog voltages between 0 and 5 V using a digital port [2].

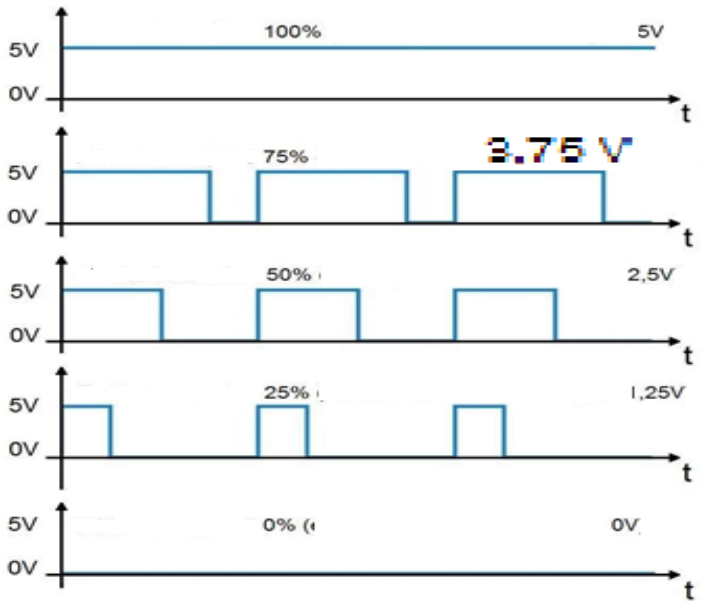


Figure 1.3. ***Voltage Variations at a PWM Output*** **(from [2])**

The Arduino board can be expanded by attaching modules called shields, which can be directly connected to the board's external pins (such as GPS, Wi-Fi, LCD – Figure 1.4, touchscreen, motor control, etc.). These pins can still be accessed afterward, as most shield modules come with their own extended pin headers.

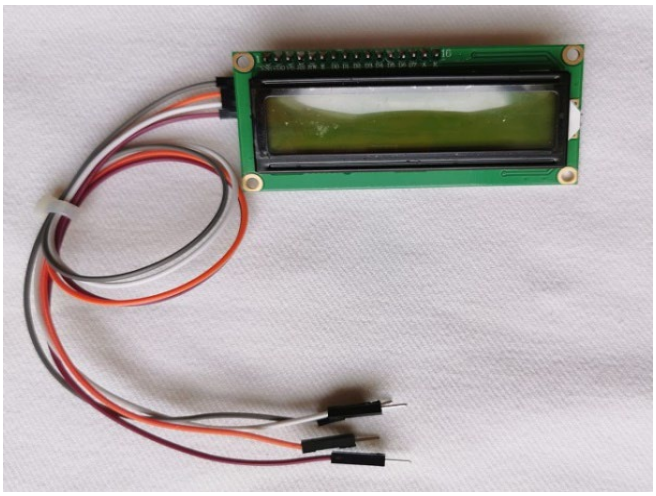


Figure 1.4. ***Using an LCD Shield on an Arduino Uno Board*** **(personal work)**

For building electronic circuits that require external components (and when using components mounted on a printed circuit board is not desired—for example, in prototype scenarios), a testing board called a breadboard can be used (Figure s1. 5 – the right side illustrates the electrical connections between pins).

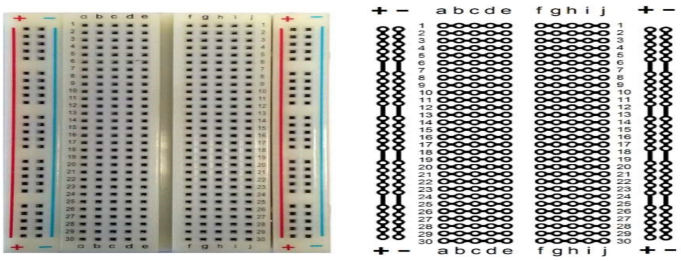


Figure 1.5. ***The Breadboard and Its Internal Connections*** **(from [1])**

The microcontroller is programmed in a language derived from C++. The required program is called Arduino IDE and can be downloaded from the manufacturer's website [6]. After connecting the board and installing and launching the program, make sure that the correct Arduino board and port are selected (in the Tools menu, under the Board and Port submenus) [3, 4, 5].

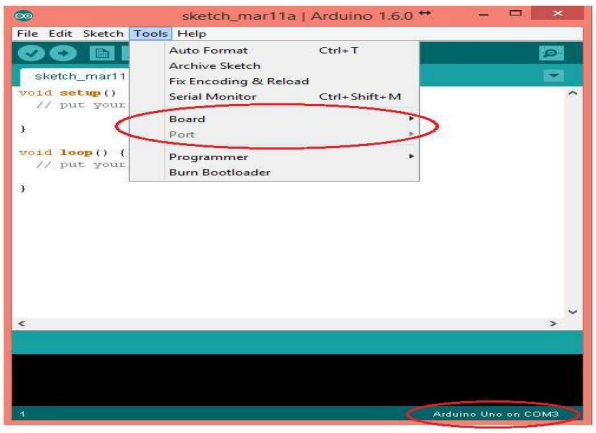


Figure 1.6. ***The Graphical Interface of the Arduino IDE Program*** **(from [1], [6])**

The most commonly used buttons in the software application are the compile button (to check for errors), also the upload button and the Serial Monitor button (to display data sent from the board to the computer).

Figure 1.7. ***Buttons and Menus of the Arduino IDE Program*** **(from [1], [6])**

BIBLIOGRAPHY

1. Iordache, V., Cormoș, A. 2019. S*enzori, traductoare și achiziții de date cu Arduino Uno.* București: Editura Politehnica Press, ISBN: 978-606-515-853-5
2. Leluțiu L.M. 2013. *Measuring, data acquisition and processing systems*. Brașov: Editura Universității „Transilvania" din Brașov, ISBN 978-606-19-0304-7
3. Lertlakkhanakul, J., Choi, W. 2008. *"*Building Data Model and Simulation Platform for Spatial Interaction Management in Smart Home.*" Automation in Construction*, Vol. 17(8): 948-957.
4. Robles, R. J. & Kim, T. 2010. "Applications, Systems and Methods in Smart Home Technology: A Review". *International Journal of Advanced Science and Technology,* Vol. 15: 41-42; 50-58.
5. Yılmaz, E.N. 2006. "Education Set Design for Smart Home Applications." *Computer Applications in Engineering Education*, Vol. 19(4): 631-638.
6. *** . 2016. *Arduino: A Technical Reference* (First Edition). Sebastopol, CA: O'Reilly Media, Inc. „Arduino IDE", http://arduino.cc/en/software/

Paper 2

APPLICATIONS USING *LEDs*

1. Work Description

1.1. Objectives of the Work

Creating and testing some simple applications circuits that use LEDs.

1.2. Theoretical Description

In this lab work, five practical applications will be created.

Application 1. Blinking LED

This application will use a digital output pin to connect an LED. The goal is to turn the LED on and off alternately, continuously, by predefining the duration for which the LED stays on or off.

Application 2. Button-Controlled LED

This application will use a digital input pin to connect a button and a digital output pin to connect an LED. The goal is to change the state of the LED from off to on when the button is pressed.

Application 3. Pulsing LED

This application will use a PWM digital output pin to connect an LED. The goal is to turn the LED on and off continuously, varying its light intensity between 0 and maximum over a predefined period.

Application 4. LED connected to one of the PWM digital outputs

This application will use an analog input pin to connect a potentiometer in a variable voltage divider configuration [3],[7].

The goal is to display the digital values provided by the Arduino board on the screen (in the Serial Monitor), proportional to the voltage at the analog input.

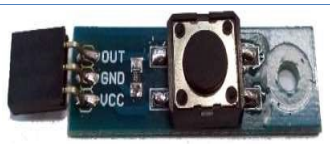
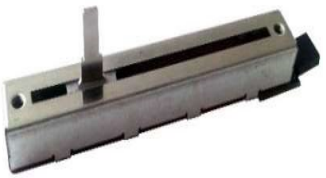
Application 5. Pressure Sensing LED

This application will use an analog input pin to connect a pressure transducer.

The goal is to display on the screen (in the Serial Monitor) the values measured by the transducer, proportional to the pressure applied to it, at predefined time intervals.

2. Hardware Components

The electronic components and modules used in this lab are listed in the following table:

Component	Characteristics	Quantity	Image
Arduino Uno		1	
LED Module	LED + resistor	1	
Jumper Wire	Male-to-Male	5	
Button Module	Button + resistor	1	
Potentiometer	50 kΩ linear	1	

Component	Characteristics	Quantity	Image
Resistive Pressure Sensor Module	FSR 406 + resistor	1	

The LED module contains, in addition to the LED, a correctly sized resistor (the sizing method is presented in Lab 3).

The **Button module** is used to detect a press and control, in this case, the state of an LED. This transducer can also be replaced with any other type of button along with a 10 kΩ resistor, as will be shown in the following paragraph.

The **Potentiometer** is used to create an adjustable voltage divider, with the purpose of applying a variable voltage to the analog input within the range: 0 … VCC. As a result, the value measured by the board will be available as a digital value, ranging from 0 to 1023, proportional to the applied voltage [2].

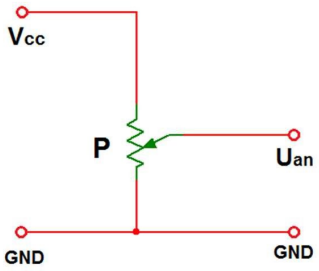


Figure 2.1. ***Adjustable Voltage Divider*** **(from [2, 3])**

The digital value range is obtained by converting the analog voltage into a digital signal, a process carried out by the Arduino board using its built-in 10bit ADC.

The analog voltage value can be measured with a voltmeter or approximated based on the digital value provided by the board, as follows:

U=Val_dig1023×VCC (1)U = \frac{V_{al_dig}} {1023} \times VCC \quad (1)U=1023Val_dig×VCC (1)

The VCC is the supply voltage and is theoretically 5 V [3]. Since the actual value may differ, calibration will be necessary.

Thus, the VCC voltage supplied by the board will be measured using a voltmeter, and the measured value will be written in the program when declaring the VCC variable.

The **resistive pressure transducer module** detects the level of pressure, based on the use of a pressure-sensitive resistor FSR 406 [1]. The value measured by the Arduino board is available as a digital value that ranges between 0 and 1023.

The pressure sensor is made of three substrates (see Figure 2.2), with a very high resistance between the electrodes (>10 MΩ) when no pressure is applied.

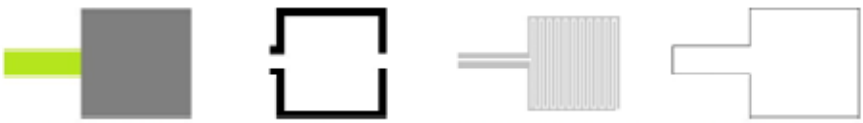


Figure 2.2. ***Internal Construction of the Pressure Sensor*** **(from [2])**

Increasing the pressure applied results in an electrical contact between the conductive substrates, leading to a decrease in the resistance value at its terminals (see Figure2.3). The resistance value depends not only on the applied force but also on the flexibility, dimensions, and shape of the object that applies the pressure [1].

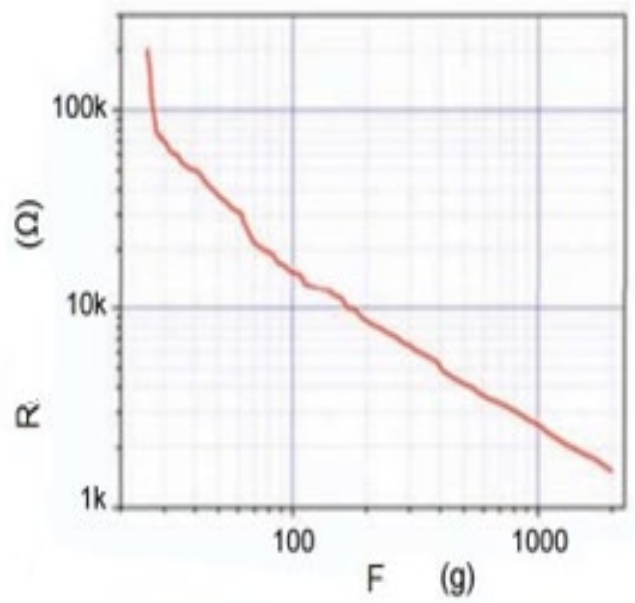


Figure 2.3. ***Variation of the Pressure Sensor Resistance in Relation to the Applied Pressure Force*** **(from [2])**

The transducer will be powered with a voltage of VCC = 5 V. Both the pressure transducer module and the button module contain, in addition to the sensor, a resistor **R** connected between the module's output pin (OUT) and ground (GND). For the button, **R** is called a **pull-down resistor** [6] (see Figure 2.4) and serves the purpose of keeping the logical value at 0 on the module's output when no pressure is applied to the sensor or the button is not pressed. Establishing a safe logical level (0 in this case) prevents the random occurrence of a 0 or 1 value at the digital input of the Arduino board due to possible electrical noise [5], [7].

For the pressure transducer (see Figure 2.4), **R** serves both as a pull-down resistor (in case no force is applied to the sensor, meaning the resistance at its terminals is very high) and as a component of a resistive voltage divider [4] (it

helps generate an analog output voltage proportional to the sensor's resistance when force is applied).

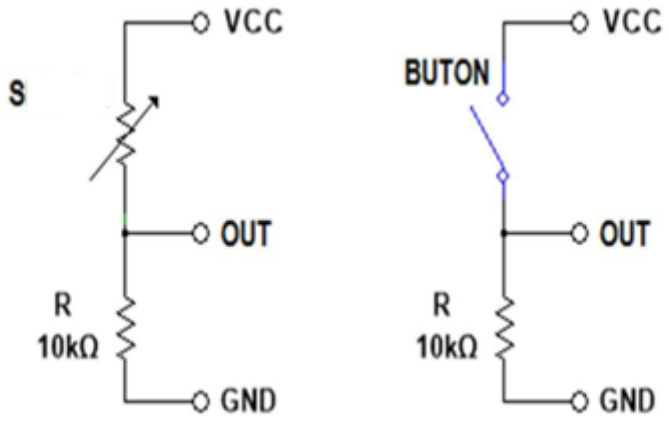


Figure 2.4. ***Use of the Resistor R*** **(from [3, 4])**

3. Software Components

- int variable = value: This sets a value for a 16-bit signed integer variable (ranging from -32,768 to 32,767).
- const: This signifies a constant, modifying the behavior of a variable. The variable becomes **read-only**, meaning its value cannot be changed.
- unsigned int variabilă = valoare: This sets a value for a 16-bit unsigned integer variable (ranging from 0 to 65,535).
- boolean variable = valoare: This sets a value for a logical variable (true or false).
- float variable = valoare: This sets a value for a 32-bit signed floating-point variable (ranging from -3.4028235E+38 to 3.4028235E+38).
- The total number of digits displayed with precision is 6-7 (including all digits, not just those after the decimal point).

Functions and Commands:

- void setup ():

This function (which does not return data and has no parameters) runs only once at the beginning of the program. It is used to set up general initialization (setting pins, activating serial ports, etc.).

- void loop ():

This is the main function of the program (which does not return data and has no parameters) and is executed continuously as long as the board is running and is not reset [7, 8].

- pin Mode (pin, mod):

Configures the specified digital pin as either input or output.

- if(condition) {instruction/i} else {instruction /instructions}: Tests whether a condition is met or not.
- for (initialization, condition, increment) {instruction /instructions}: Repeats a block of instructions until the condition is met.
- digital Write (pin, value): Writes value to the digital pin.
- digital Read (pin):
- analogRead(pin):
- analog Write (pin, value): Writes a value representing the duty cycle for a PWM signal, to the specified PWM digital pin.
- delay(ms): Pauses the program for some milliseconds.
- Serial.begin(speed): Sets the baud rate for the serial port in bits per second.
- Serial. print (value or variable, numbering system): Prints data as ASCII characters using the serial port.

- Serial. print ln (value or variable, numbering system): Prints data as ASCII characters using the serial port, adding a newline after the printed data.
- ==: Means "equal to".

4. Application 1. Blinking LED

4.1. Building the electronic circuit

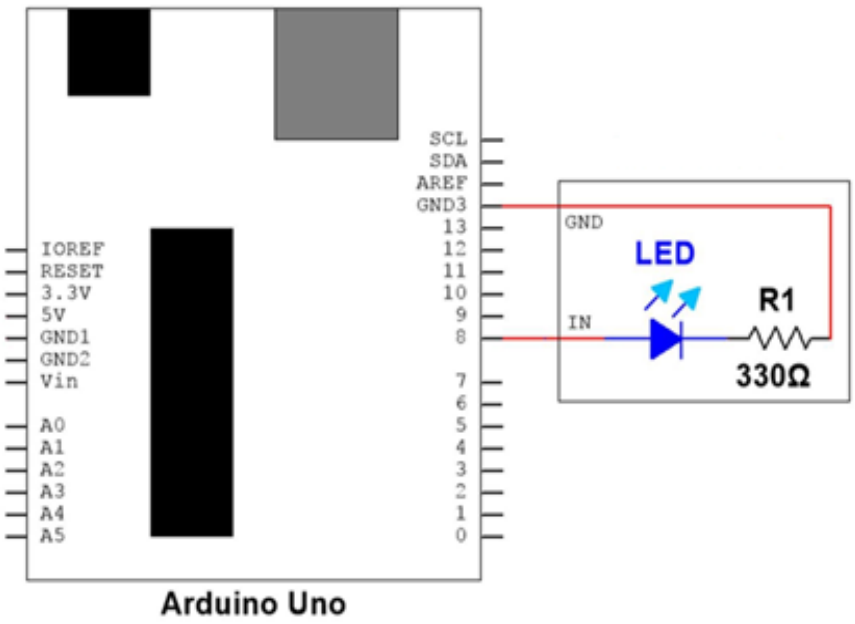


Figure 2.5. ***Schematic diagram for Application 1*****(from [2], [7, 8])**

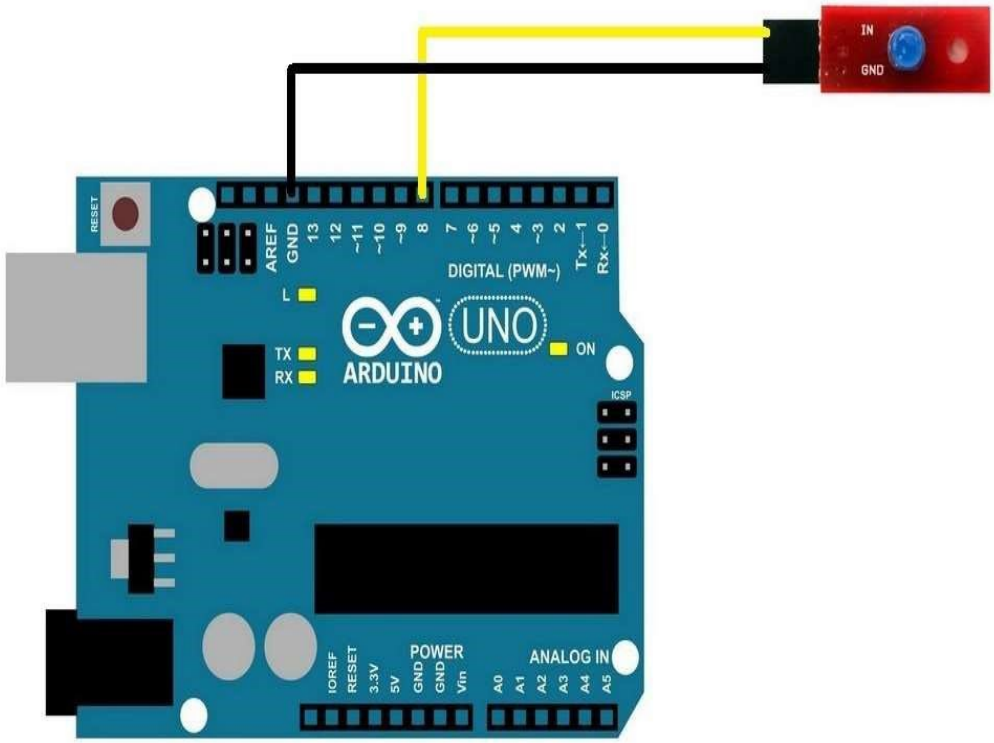
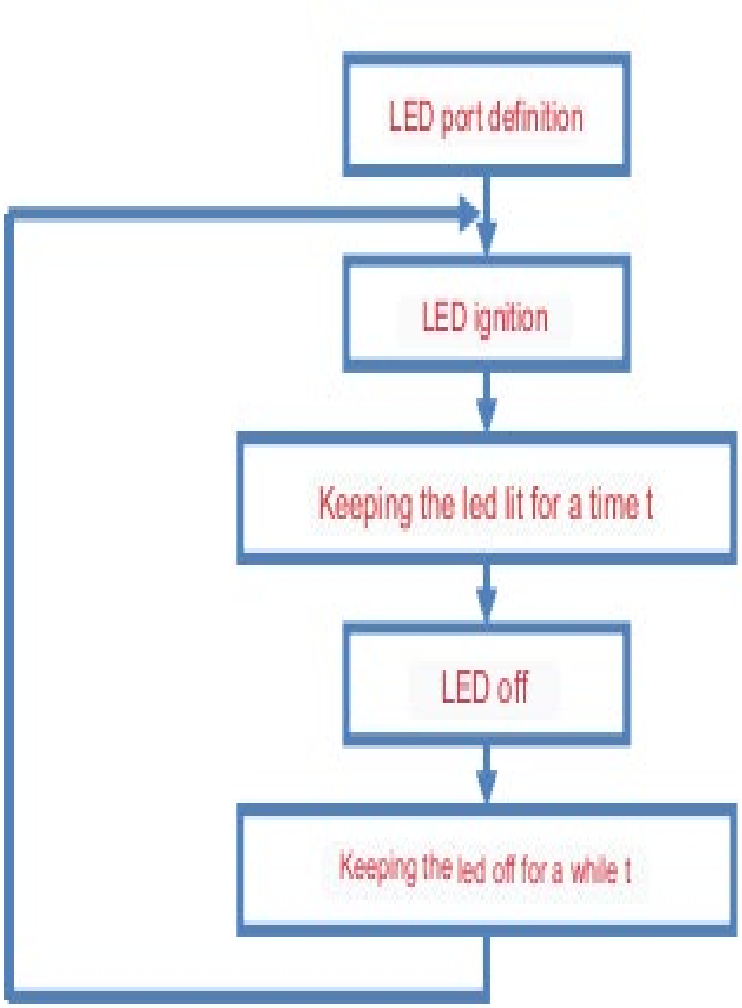


Figure 2.6. ***Electrical connections for Application 1*****(from [7, 8])**

The following connections are made:

- The digital pin 8 on the board is connected via a wire to the IN pin of the LED module;
- The digital GND pin on the board is connected via a wire to the GND pin of the LED module.

4.2. Logical diagram



Additional Exercises - Blinking LED

1. Calculate the frequency at which the LED blinks.
2. Modify the code sequence so that the LED blinks at a frequency of 10 Hz.
3. Modify the code sequence.

5. Application 2: LED controlled by a button

5.1. Building the electronic circuit

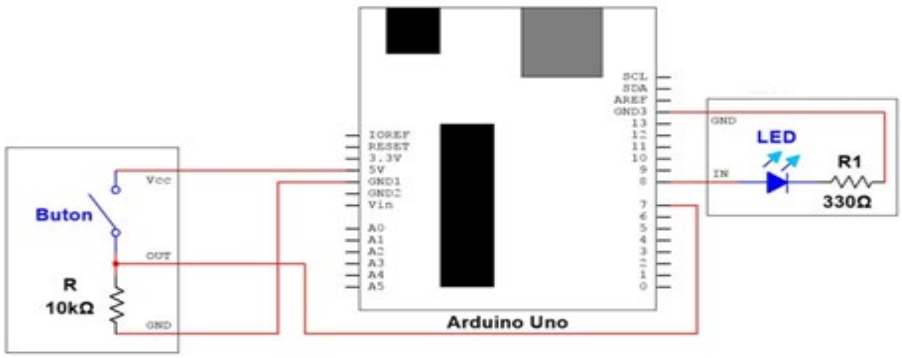


Figure 2.7. *Schematic diagram for Application 2* (from [2], [7])

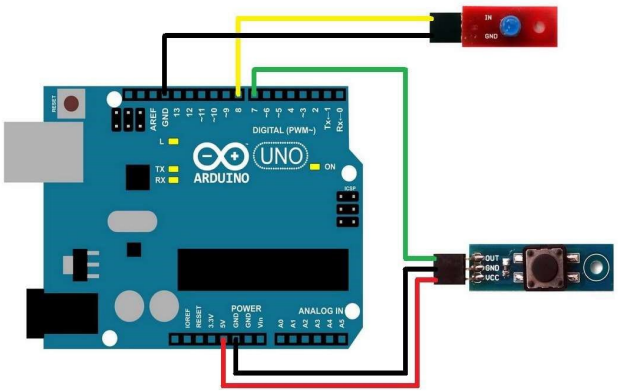


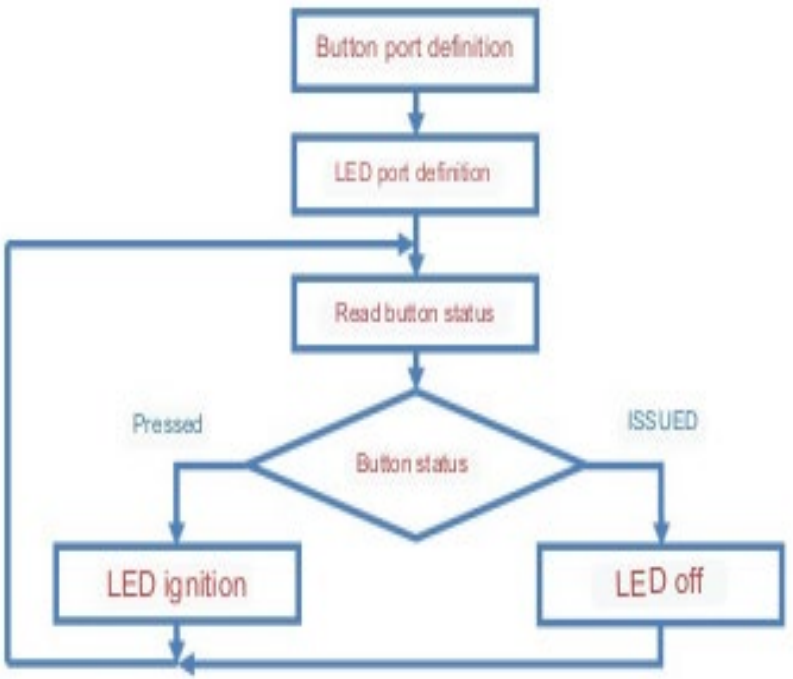
Figure 2.8. ***Making the electrical connections for Application 2*** **(from [7])**

The following connections are made:

- The digital pin 8 on the board is connected with a wire to the IN pin of the LED module;

- The GND Pin (digital) is connected with a wire to the GND pin of the LED module;
- The digital Pin 7 is connected with a wire to the OUT pin of the Button module;
- The GND Pin (power is connected with a wire to the GND Pin of the Button module;
- The 5V Pin (power) is connected with a wire to the VCC pin of the Button module.

5.2. Logical diagram and code sequence




```
const int buton = 7;
// defining the button variable corresponding to the digital port 7 where it will be. connected OUT pin
const int led = 8;
```

```
// defining the led variable corresponding to the digital port 8 where it will be
connected pin IN of the LED module
void setup() {
pinMode(buton, INPUT);
//the button pin is declared as input
pinMode(led, OUTPUT);
//declare the led pin as output
}
void loop() {
boolean stareButon = digitalRead(buton);
//declares the Boolean state Button variable that takes the logical value of
pin condition button
if (stareButon == HIGH) {
//if the condition of the button pin is 1 logic ( button pressed )
digitalWrite(led, HIGH);
//then write the value 1 logically on the led pin ( aprinded )
}
 else {
//otherwise
digitalWrite(led, LOW);
//write the value 0 logically on the led pin ( turn led )
    }
```

Additional Exercises - LED controlled by a button

1. Modify the code sequence.
2. Modify the code sequence so that when the button is pressed, the LED toggles to the opposite state (turn off if it was on, or turn on if it was off).

6. Application 3. Pulsating LED

6.1. Building the electronic circuit

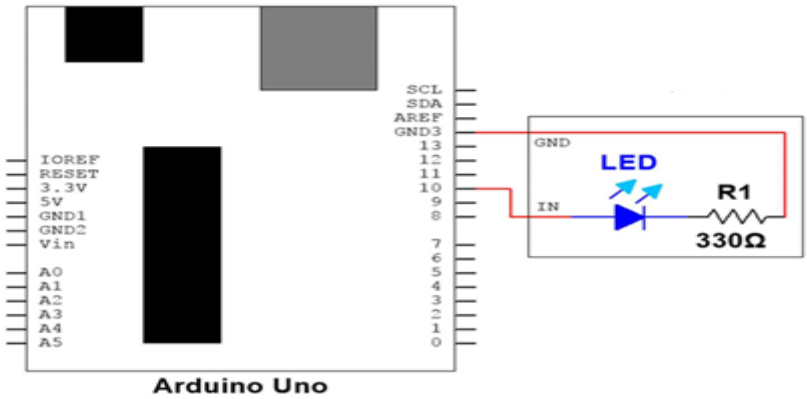


Figure 2.9. ***Schematic diagram for Application 3*** **(from [2], [7, 8])**

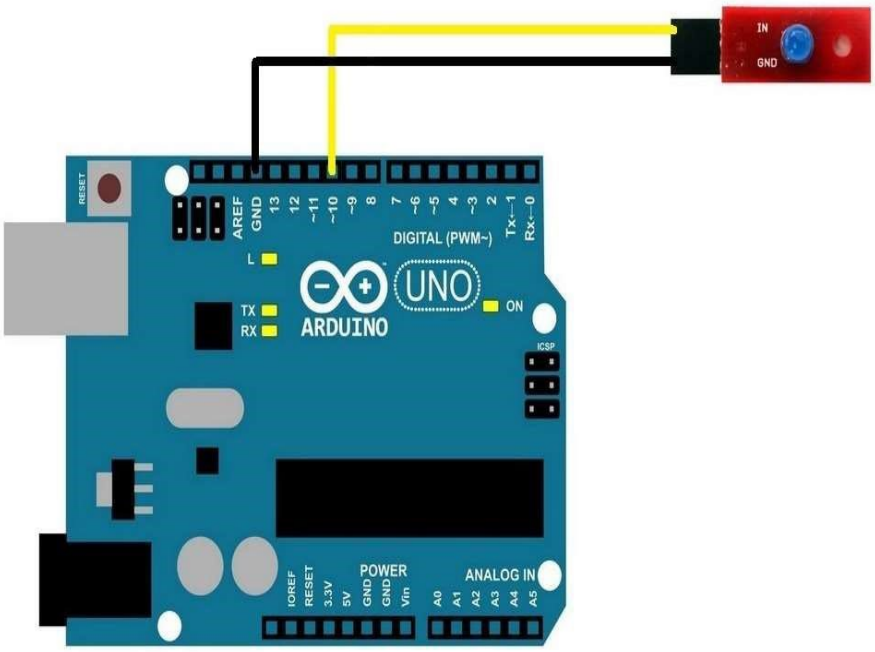
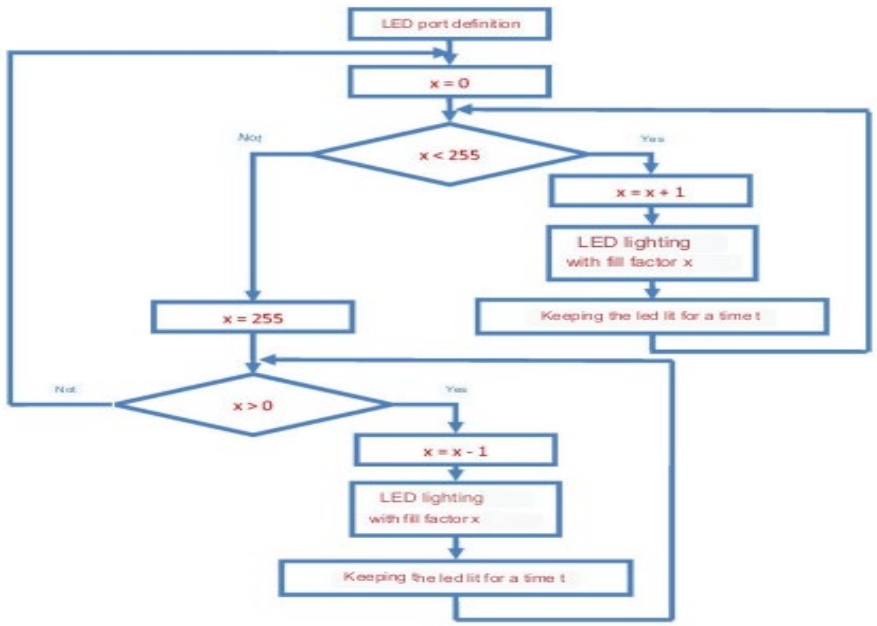


Figure 2.10. ***Electrical connections for Application 3*** **(from [7, 8])**

6.2. Logical Diagram and Code Sequence




```
const int led = 10;
//defining the led variable corresponding to the digital port PWM 10 where the
IN pin of the LED module will be connected
void setup() {
pinMode(led, OUTPUT);
//se declară pinul led ca fiind de ieșire
}
void loop() {
for (int x=0; x<255; x=x+1) {
//varies the value of x ascending from 0 to 254
analogWrite(led, x);
//write the analog value with the filling factor x on the led pin
delay(20);
//delay 20ms
```

```
}
for (int x=255; x>0; x=x-1) {
//varies the value of x decreasing from 255 to 1
analogWrite(led, x);
//write the analog value with the filling factor x on the led pin
delay(20);
//delay 20ms
  }
}
```

Additional Exercises - Pulsating LED

1. Calculate the time period for one cycle of the LED being on/off (the loop).
2. Modify the code sequence so that the time period for the LED on/off cycle is 2.55 seconds.

7. Application 4. LED connected to one of the PWM digital outputs

7.1. Building the Electronic Circuit

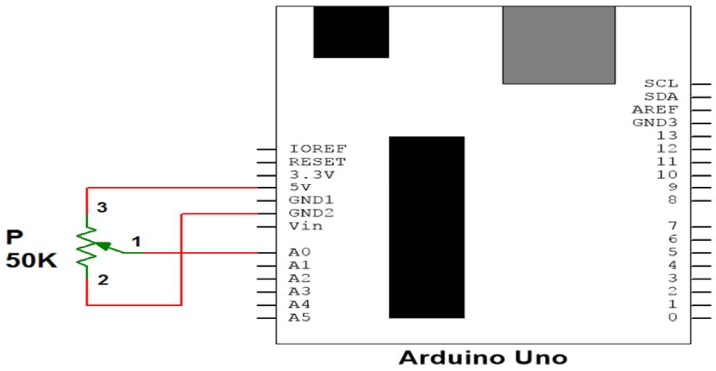


Figure 2.11. ***Schematic Diagram for Application 4*****(from [7])**

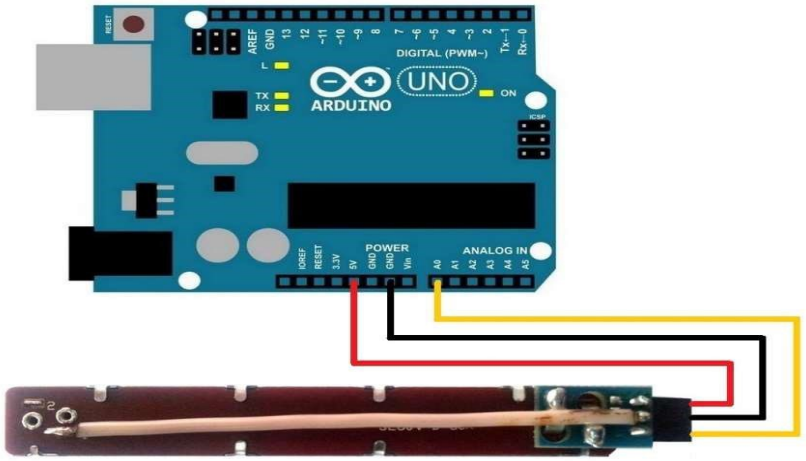


Figure 2.12. ***Wiring for Application 4*****(from [7, 8])**

The following connections are made:

- The analog Pin AO on the board is connected with a wire to pin 1 (the wiper) of the potentiometers.
- The GND Pin (power) on the board is connected with a wire to pin 2 of the potentiometers.
- The 5V Pin (power) on the board is connected with a wire to pin 3 of the potentiometers.

2.5.2. Code sequence

```
float Vcc = 5;
void setup() {
  Serial.begin(9600);
}
 void loop() {
  const int val_dig = analogRead(0);
  float Uan = Vcc*val_dig/1023;
  Serial.print("Value digital: ");
```

```
 Serial.println(val_dig);
  Serial.print("Tensiune analogica: ");
 Serial.print(Uan,3);
  Serial.println(" V");
          Serial.println();
  delay(1000);
}
```

Additional Exercises

1. Display the position of the potentiometer cursor in percentage on the serial monitor.
2. Connect an LED to one of the PWM digital outputs and vary its brightness using the potentiometer.

8. Application 5. Pressure Sensing LED

8.1. Building the electronic assembly

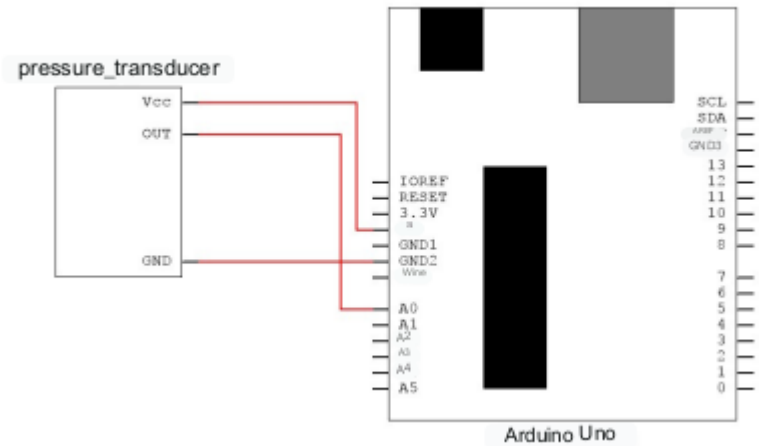


Figure 2. 13. ***Schematic diagram for Application 5*** **(from [2], [7, 8])**

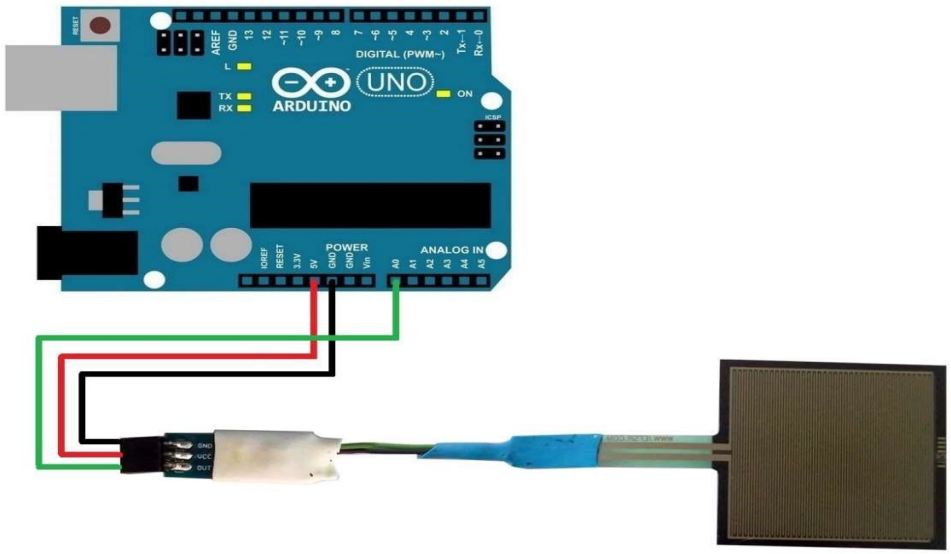
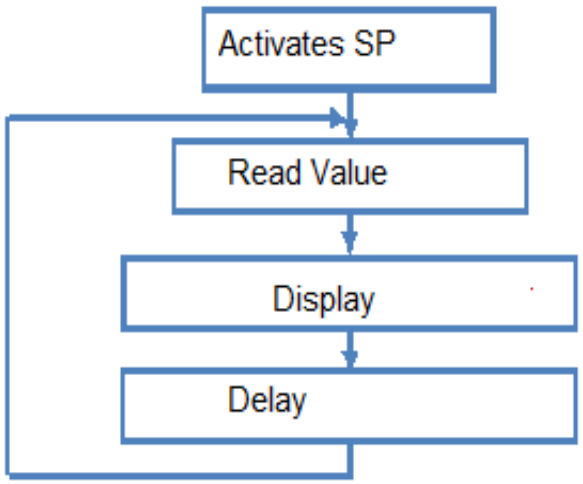


Figure 2. 14. ***Realization of electrical connections for Application 5*** **(from [2], [8])**

8.2. Logical diagram and code sequence




```
float Vcc = 5;
//defining the U variable corresponding to the supply voltage
void setup () {
```

```
Serial.begin (9600);
//activates the output of the serial port with a rate of 9600 baud }
void loop () {
const int val_dig = analogRead(0);
//the constant variable of integer wave_dig type is declared which takes the value read at
float Uan = Vcc*val_dig/1023;
//analog voltage calculation at input A0 – see formula (1 )
Serial.print ( “Digital value:” );
// displays on the serial monitor the text in parentheses
Serial.println(val_dig);
//displays on the serial monitor the read digital value
Serial.print ( “Analytical voltage:” );
//displays on the serial monitor the text in parentheses
Serial.print (Uan,3);
/ /displays on the serial monitor the calculated value of the analog voltage, with three decimal places
Serial.println (“ V”);//displays on the serial monitor the text in parentheses
Serial.println ();//displays a contentless line on the serial monitor
Delay (1000);
//delay next display by 1000 ms}}
```

Additional Exercises - Reading the pressure level

1. Introduce a LED in the application that will turn on when the pressure value is less than 50 and turn off when it is greater than or equal to 50.
2. Introduce a LED in the application that will blink if the pressure value is greater than 100 and turn off when it is less than or equal to 100.

The Arduino development board [5], [8] has digital and analog input/outputs that allow the connection of various external modules that can be controlled or that can generate commands. In this work, the command modules are: the Button Module, the Potentiometer, and the Pressure Transducer Module. These modules serve to transmit commands to the development board and form a human-machine interface with the Arduino system. The commanded module is the LED module, which, through the development board, will receive commands from users. The Button Module will generate digital commands: pressing the button will generate a logical 1 (+5 V), and the opposite state will generate a logical 0 (0 V – ground). The Potentiometer and Pressure Transducer modules will generate analog commands. Each of their outputs will generate a variable voltage. The command signal (or the signal acquired by the Arduino development board) will be converted into a digital signal by the board to be processed.

This conversion is done by an analog-to-digital converter (ADC). The digital or analog signals acquired on one of the digital or analog lines of the Arduino development board will be converted into digital values, and these values will be used as variables in the programs written for the development board. Certain variables that the programs developed for the Arduino board work with will need to be sent to the digital or PWM output lines. These will have a corresponding electrical signal on the output/line to which they were sent.

BIBLIOGRAPHY

1. Bartmann, E. 2015. *Interlink Electronics FSR Series.* O'Reilly Media**,** Inc., https://www.interlinkelectronics.com/fsr-406
2. Iordache, V., Cormoș, A. 2019. S*enzori, traductoare și achiziții de date cu Arduino Uno.* București: Editura Politehnica Press.
3. Leluțiu, L.M. 2013. *Measuring, data acquisition and processing systems*. Brașov: Editura Universității "Transilvania" din Brașov, ISBN 978-606-19-0304-7.
4. Lelutiu, L.M. 2016. *Data acquisition*. Brasov. Editura Universității „Transilvania" din Brasov, ISBN 978-606-19-0866- 0
5. Massimo, B. 2009. *Getting Started with Arduino.* O Reilly Media, Inc., https://www.oreilly.com/library/view/getting-started-with/ 9780596155704/
6. ***. 2015. *Arduino Playground*, http://playground.arduino.cc/Common Topics/PullUpDownResistor.
7. *** . 2016. *Arduino: A Technical Reference* (First Edition). Sebastopol, CA: O'Reilly Media, Inc. „Arduino IDE", http://arduino.cc/en/software/
8. *** . 2016. *Arduino Boards*, https://www.arduino.cc/en/hardware/#boards

Paper 3

MEASUREMENT OF ENVIRONMENTAL PARAMETERS USING ANALOG SENSORS

1. Work Description

1.1. Objectives of the Work

- Designing and testing the medium complexity circuits using sensors and transducers.
- Using the shield-type electronic modules.
- Making a practical application for measuring temperature and relative humidity values, calculating the thermal comfort index and displaying them on an LCD screen.

1.2. Theoretical Description

Introduction

Temperature is a physical quantity that characterizes the thermal state of an environment or a body.

The relationship between them is as follows: $tC[°F] = tF[°C] \times 1,8 + 32$.

Atmospheric humidity is the amount of water vapour in the air. The relative humidity of the air is the proportional relationship between the current humidity at a certain temperature and the maximum possible humidity at the same temperature and is measured as a percentage. It cannot exceed 100% because the excess is removed by condensation

The thermal comfort index [1] is used to describe the apparent temperature felt by the human body and is calculated according to the air temperature and relative humidity, according to the following formula:

$$ICT = (t_C * 1,8 + 32) - (0,55 - 0,0055 * h) * [t_C * (1,8 + 32) - 58] \quad (3.1)$$

where t_C is the temperature (°C) and h is the relative humidity (%).

The value of the thermal comfort index is interpreted as follows:

- ICT ≤ 65, state of comfort;
- 65 < ICT < 80, alert state;
- ICT ≥ 80, state of discomfort.

The purpose of this application is to make an electronic circuit that measures the temperature and relative humidity of the environment using analog sensors, calculates the thermal comfort index and displays them numerically on an LCD screen.

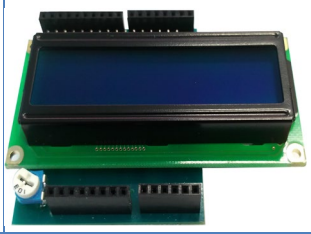
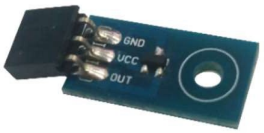
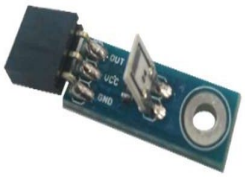
To measure the temperature, a transducer based on a specialized, precision integrated circuit, LM50 [5], will be used. The transducer outputs an analog voltage that will be applied to one of the analog inputs of the Arduino board, having the advantage of a linear characteristic of the variation of the output voltage in relation to temperature. Based on the analog input voltage, the board will provide a corresponding digital value, which will be used to calculate and display the measured temperature.

To measure the humidity, a resistive sensor SYH-2R [3] will be used (the resistance at its terminals varies according to the humidity) together with a fixed resistor, to form a resistive voltage divider to provide the Arduino board [1] with an analog voltage that varies according to measured humidity.

Based on the analog input voltage, the board will provide a corresponding digital value, which will be used to determine and display the measured humidity.

2. Hardware Components

The electronic components and modules used in the work are those in the following table:

Component or module	Characteristics	Number of pieces	Image
Arduino Uno		1	
Breadboard	82x52x10 mm	1	
LCD S	Display on 2 lines of 16 characters each	1	
Connecting wire	Male to Male	8	
Temperature transducer module	LM50	1	
Humidity transducer module	SYH-2R	1S	

Remarks breadboard

In this paper, a breadboard type test board will be used to perform the electronic assembly using external components (Figure 3.1 - on the right side, the electrical connections between the pins are symbolized).

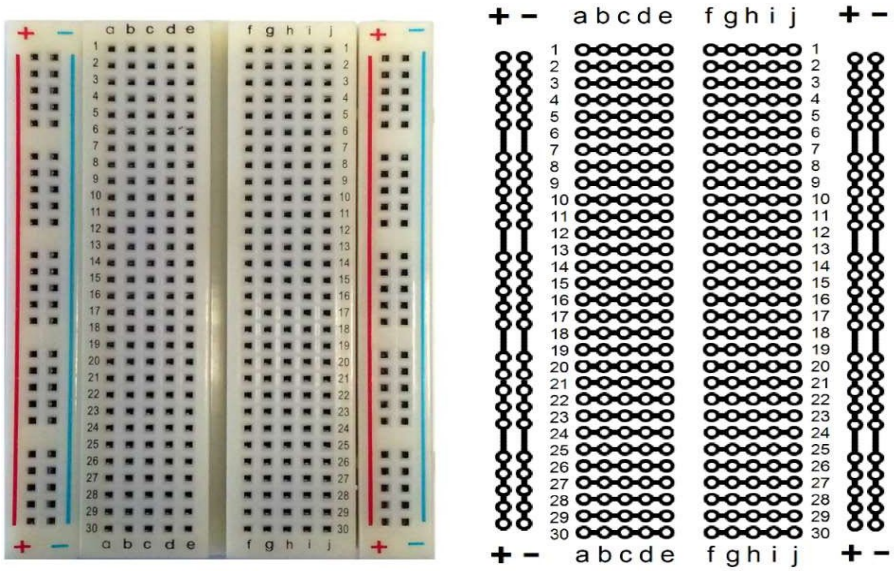


Figure 3.1. ***Breadboard and internal connections*** **(from [3], [7])**

The temperature transducer measures the ambient temperature based on the use of a precision temperature sensor LM50[2]. The sensor can measure temperatures between −40°C and +125°C, the output voltage being proportional to the temperature in degrees Celsius and varying in steps of 10 mV/°C. Considering that the sensor also measures negative temperatures, without the need for a negative voltage source, for the temperature of 0°C the output voltage is NOT 0 V, but has the value of 500 mV. Based on the variation of this voltage (ideally between 0.1 and 1.75 V), applied to one of the analog ports, the Arduino board provides a digital value that varies between 21 and 359 (103 for the temperature of 0°C). The accuracy of the sensor is ±3°C at room temperature and ±4°C, over the entire measurement range.

Figure 3.2. ***Temperature Transducer*** LM50 **(from [5])**

The transducer will be powered with the voltage VCC = 5 V

Temperature measurement

To calculate the value of the measured temperature, first determine the value of the analog voltage (U_temp) applied to the analog input of the Arduino board, based on the digital value provided by it (val_dig_temp).

$$U_{temp} = \frac{val\_dig\_temp \cdot V_{cc}}{123} \tag{3.2}$$

Taking into account that the temperature of 0°C corresponds to a value of 0.5 V at the output of the transducer and that the output voltage varies by 0.01 V/°C, the temperature value can be calculated with the following formula:

$$temp = \frac{Utemp-0,5}{001} \tag{3.3}$$

The humidity transducer [6] measures the humidity of the environment, based on the use of a resistive humidity sensor. The sensor can measure relative humidity between 10% and 95% and the variation of the output resistance as a function of humidity (measured at a temperature of 25 °C) is as shown in Figure 3.4.

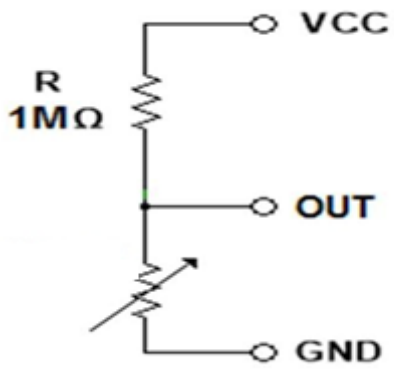


Figure 3.3. ***Resistive voltage divider*** **(from [2])**

The humidity transducer contains, in addition to the sensor, a resistor connected between the output pin of the module (OUT) and VCC. It forms, together with the sensor, a resistive voltage divider (Figure 3).

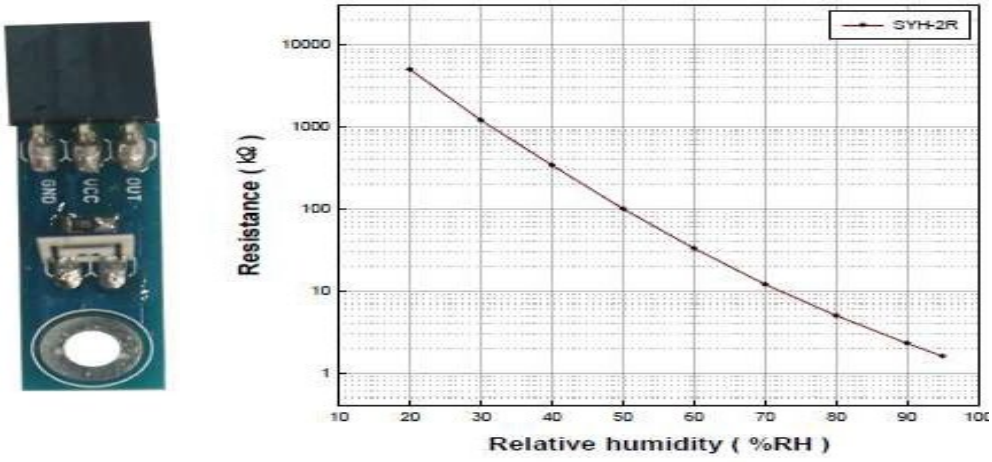


Figure 3.4. ***Humidity transducer and its characteristic*** **(from [1], [6])**

The transducer will be powered with the voltage VCC = 5 V.

Humidity measurement

Since the sensor manufacturer does not provide a calculation formula for humidity as a function of sensor resistance, the determination of the measured humidity value will be done by a less accurate method but with acceptable results for a demonstration laboratory work. The method consists in using a computer program capable of digitizing graphics in image form (in the present case Plot Digitizer [3] was used). Thus, by calibrating the X and Y axes, the values of the digitized feature can be read by clicking with the mouse on each individual point. The values taken from the graph can be found in the table below, in the Resistance and Humidity columns. It was chosen to display the measured humidity in steps of 5% to simplify the code sequence. By using a resistive voltage divider to which the Vcc voltage is applied, a voltage proportional to the variation of the sensor resistance (U_humid) calculated according to the formula is obtained at the output:

$$U_{umid} = U_{OUT} = V_{CC} \cdot \frac{R_{sensor}}{R + R_{sensor}} \tag{3.4}$$

The digital value provided by the Arduino board corresponding to each input voltage level is calculated according to the formula:

$$ValDigUmid = \frac{1023 * U_{Umid}}{V_{CC}} \tag{3.5}$$

Thus, a range of digital values is established to approximate the value of the measured humidity (with a step of 5%).

LCD Shield allows characters to be displayed on a liquid crystal display with LED lighting. It mounts over the Arduino board and has the connectors such that the board pins will still be accessible.

The LCD screen consists of 2 lines of 16 characters each, each character being composed of 5x8 pixels. Column (character) numbering it is done from

0 to 15 (from left to right), and of rows from 0 to 1 (from top to bottom).

3. Software components

LiquidCrystal.h is the library that contains the commands for the LCD shield.

LiquidCrystal lcd (rs, enable, d4, d5, d6, d7) creates an lcd variable specifying the digital pins used to control the LCD shield.

int variable = value sets a value to a 16-bit signed integer variable (from -32,768 to 32,767).

const has the meaning of constant modifying the behavior of a variable. The variable will become Read-only, that is, its value cannot be changed.

variable float = value sets a value to a signed 32-bit floating-point real variable (from -3.4028235E+38 to 3.4028235E+38). The total number of digits displayed accurately is 6 – 7 (includes all digits, not just the ones after the decimal point).

void setup() is a function (which returns no data and has no parameters) that runs only once at the beginning of the program. This is where the general program preparation instructions are set (setting pins, enabling serial ports, etc.).

void loop() is the main function of the program (which returns no data and has no parameters) and is executed continuously as long as the board is working and is not reset.

analogRead(*pin*) reads the value of the specified analog pin.

for(initialization, condition, increment) { statement/ statements } repeats a block of statements until the condition is met.

switch(variable) / case(value/value range): statement / break compares the value of a variable with the values specified in the case conditions and executes the statement when there is a match. The break command exits the switch statement.

lcd.begin(columns, rows) initializes the LCD screen interface and specifies its number of rows and columns.*lcd*.setCursor(*coloană*, *raw*) sets the LCD cursor position.

For the LCD used in this application, the number of columns is from 0 to 15, and the number of rows is from 0 to 1.

lcd.clear()

lcd.print() displays the data (values of some variables)/text between parentheses on the LCD screen. To display a text it is necessary that it be placed between quotation marks ("text"). To display the value of a variable of type char, byte, int, long, or string, write the name of the variable and, optionally, its number base (variable, BIN or DEC or OCT or HEX). To display the value of a float or double type variable, write the name of the variable and after the comma, the number of decimals you want to display (variable, no. of decimals).

delay(*ms)*.

++ is used to increment a variable

Creating custom characters to be displayed on the LCD

variable byte[no. values] = {values} sets a value to an unsigned byte variable. In this application the variable defined does not have a single value but a matrix of values, which has the role of determining which pixels will be on (value 1) and which pixels will be off (value 0) in the composition of a custom character (a character of on LCD it consists of 5x8 pixels).

lcd.createChar(number, variable) creates a custom character that is assigned a number between 0 and 7, having the pixel distribution according to the variable.

lcd.write(number) displays the character at the specified position (number).

In this paper, the custom character will be the one in Figure 3.5, representing the symbol for degree Celsius.

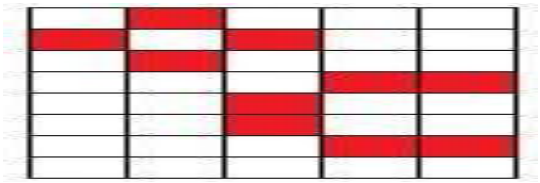


Figure 3.5. *Custom character* (from [4])

To calculate the value of the measured temperature, first determine the value of the analog voltage (U_temp) applied to the analog input of the Arduino board, based on the digital value provided by it (val_dig_temp).

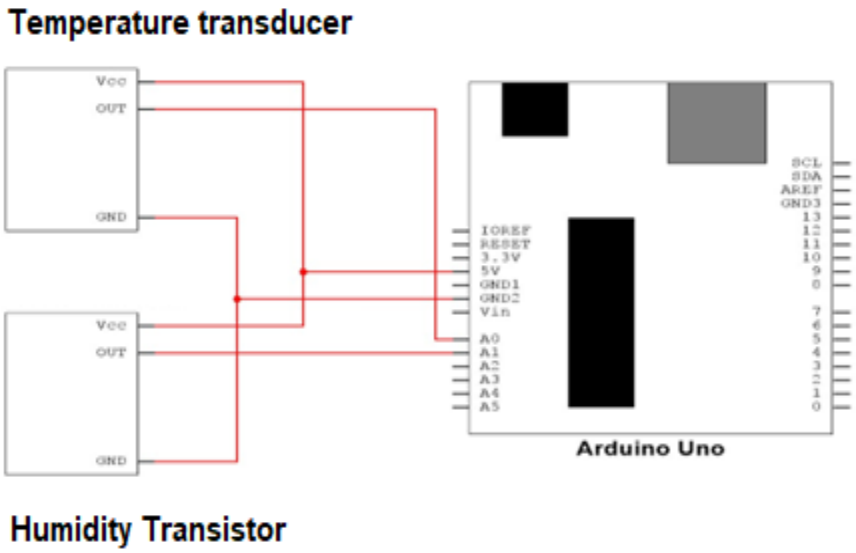


Figure 3.6 *Principle diagram* (from [1], [3], [6])

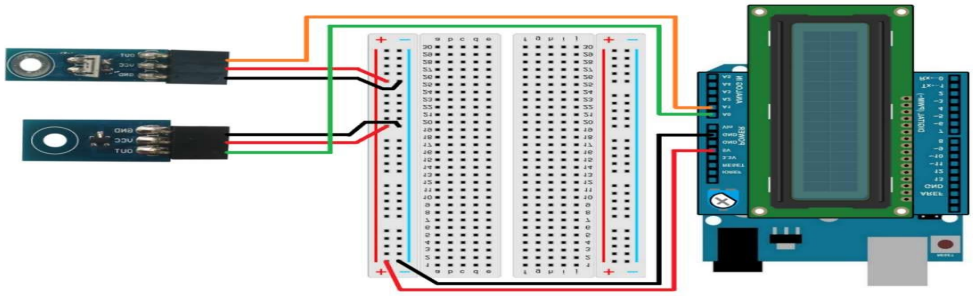
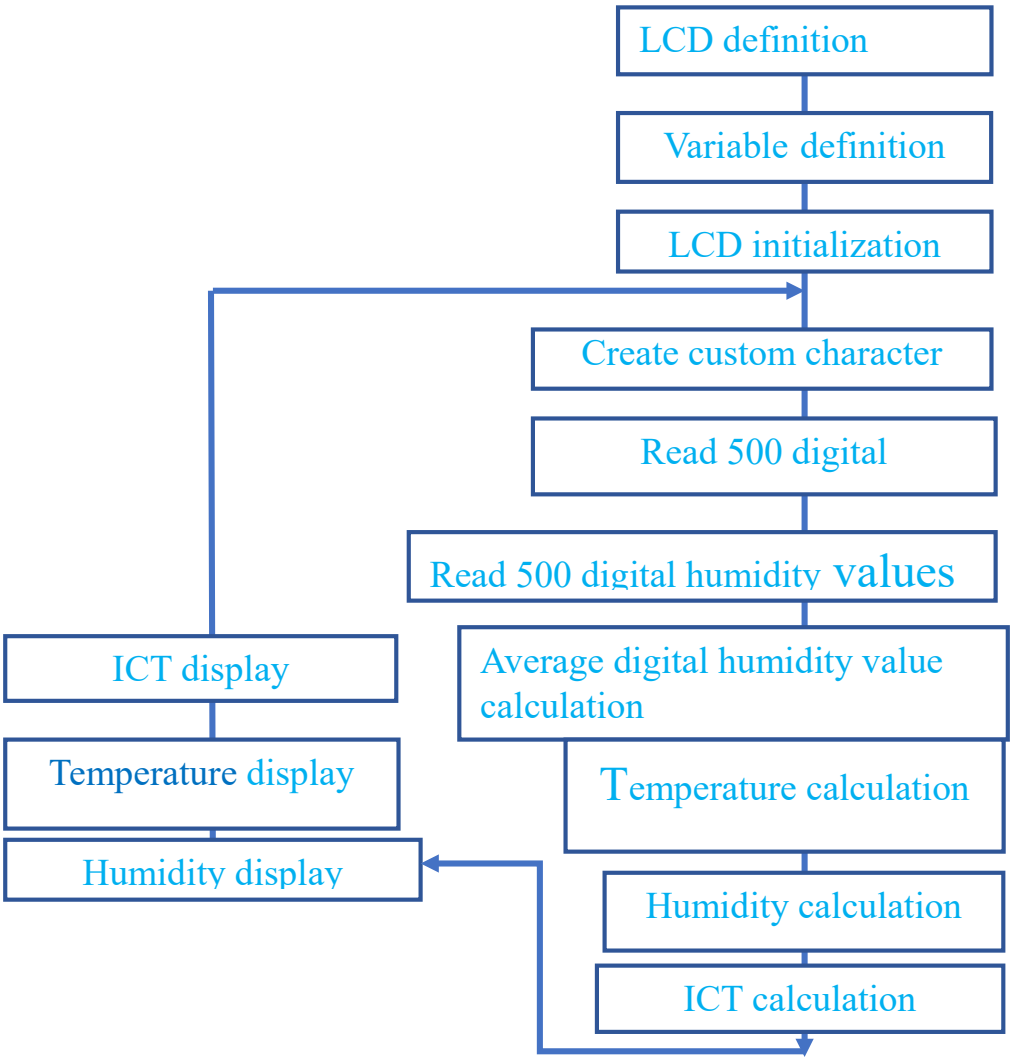


Figure 3.7. ***Electrical connections*** **(from [1], [3])**

The following connections are made:

- The GND (power) is connected with a wire to the minus bar of the;
- The 5V is connected with a wire to the plus;
- A0 on the Arduino board is wired to the OUT pin of the temperature transducer module;
- A1 is wired to the OUT pin of the humidity transducer module;
- The GND pins of the transducers are connected with a wire to the busbar minus of the breadboard;
- The Vcc pins of the transducers are connected with a wire to the plus

4.2. Logic diagram and code sequence



Code sequence

#include < **Liquid Crystal. h** >

//include in the program the command library for LCD

Liquid Crystal.

//defining the variable degree of type byte, as being a matrix with 8 rows that have the values in brackets**;**

```
const  int portTemperature = 0;
//defining the portTemperature variable corresponding to analog port A0 where the OUT pin of the temperature sensor will be connected
const int portHumidity = 1;
//defining the portHumidity variable corresponding to analog port A1 where the OUT pin of the humidity sensor will be connected
float temp = 0.0;
//define the temperature variable
int umid = 0.0;
//defining the humidity variable
//defining the val_dig_temp variable that will have the digital value corresponding to the read temperature
int val_dig_humid = 0.0;
//defining the val_dig_humid variable that will have the digital value corresponding to the read humidity
float U_temp = 0.0;
//define the variable for the analog voltage provided by the temperature transducer
float ICT = 0.0;
//definition of the variable for ICT (thermal comfort index)
 float Vcc = 5.0;
//defining the variable for the Vcc voltage that will have the initial value of 5V
// initialize the   screen interface and specify its number of rows and columns
lcd.createChar(1, degree);
//creating the custom character that will have the contents of the degree array and assigning position 1 }
//read the digital value corresponding to the temperature 500 times and sum all the values
val_dig_humid = val_dig_humid + analogRead(portHumidity);
```

```
//read the digital value corresponding to humidity 500 times and sum all the values
delay(1);
//delay 1 millisecond
}
//calculation of the average digital temperature value
//calculation of average digital value of humidity
U_temp = (val_dig_temp * Vcc)/1023;
//calculation of the analog voltage equivalent to the read digital value –) temperature calculation)
switch (val_dig_humid) {
//determine the humidity value based on the read digital value
//calculation of the Thermal Comfort Index – formula (1)
 lcd.clear();
//clear LCD   screen contents
lcd.print("t=");
//display on the LCD the text between the quotes
lcd.print(temp,1);
//display the value of the temp variable on the LCD with one decimal place
lcd.write(1);
//display on the LCD the custom character having position 1
lcd.print(" h=");
//display the text between the quotes on the LCD screen
lcd.print(humid);
//display on the LCD screen the value of the umid variable
lcd.print("%");
//display on the LCD the text between the quotes
lcd. set Cursor  (0,   1);
lcd. print("ICT=");
```

```
//display the text between the quotes on the LCD screen
lcd.print(ICT,1);
//display on the LCD the value of the variable ICT, with one decimal place
}
```

Since the actual values differ from the theoretical ones, a calibration of the electronic measuring circuit has to be done, by making changes in the software part:

The Vcc voltage has the theoretical value 5 V. The real voltage will be measured with the help of a voltmeter, and the measured value will be written in the program when the Vcc variable is declared.

Additional Exercises and conclusions

1. Modify the code sequence so that the temperature display is in degrees Fahrenheit.
2. Modify the program so that the messages are displayed "Comfort state" for ICT $\leq$ 65, "Alert state" for 65 < ICT < 80 and "discomfort state" for ICT $\geq$ 80.
3. Modify the program to display "Red Code" messages for ICT $\geq$ 80 and temperature greater than 30 °C.
4. Modify the program so that, in addition to displaying the instantaneous temperature and humidity, it calculates and displays the average values of temperature and humidity for different time intervals (eg 30 sec., 1 min, 12 hours, 24 hours, etc.).

The role of transducers is to convert a variation of a physical quantity into an electrical signal. Thus, at the output of the temperature transducer, a voltage will be found whose value is proportional to the measured temperature.

The range of voltage variation at the output of the transducer is recommended to be identical to that of the analog input of the acquisition board, in general, or of the Arduino development board, in this particular case.

This is important to preserve the resolution of the development board, i.e. the minimum voltage variation sensed by the analog input of the development board. The adaptation of the voltage variation range from the output of the transducer to the variation range accepted by the input of the development board is done by means of signal conditioning circuits. This adaptation is only necessary if the resolution of the development board is to be maintained.

A 10-bit resolution for an analog input corresponds to 210 (1024 or 1K) different voltage levels that can be sensed by that input. For a voltage variation between 0 and 10 V, at an analog input characterized by a 10-bit resolution, 1024 distinct levels can be obtained between 0 V and 10 V, which means a minimum detectable variation of 9.765 mV.

All measuring devices and instruments need initial calibration and periodic calibration.

This calibration is performed using a standard measuring device or instrument or by generating the measured quantity in the standard system.

The display of information on LCD devices is limited by the resolution of the display (or the number of pixels per unit of display area), available memory, writing speed and other parameters, depending on the applications in which these LCD displays are used.

BIBLIOGRAPHY

1. Iordache, V., Cormoș, A. 2019. S*enzori, traductoare și achiziții de date cu Arduino Uno.* București: Editura Politehnica Press.
2. Leluțiu, L.M. 2016. *Data acquisition.* Brașov: Editura Universității „Transilvania din Brașov, ISBN 978-606-19-0866- 0
3. McRoberts, Michael. 2013. *Beginning Arduino*, 2nd edition. München, Oldenburg, Apress, https://www.oreilly.com/library/view/beginning-arduino-second/9781430250166/
4. Teodoreanu, E., Bunescu, I. 2007. "Thermal Confort". *Journal Present Environment and Sustainable Development*, 1: 135-142. Iași: University "Alexandru Ioan Cuza".
5. ***. 1999 / 2013. Texas Instruments. *LM50/LM50-Q1 – Datasheet* https://ro.mouser.com/new/texas-instruments/ti-lm50-temperature-sensors/
6. ***. 2025. *Humidity sensor SYH-2R series – specifications*, https://www.shoptronica.com/files/001_SYH-2R.pdf
7. ***. 2025. *Plot Digitize*. *Arduino IDE*, http://plotdigitizer.sourceforge.net/.

Paper 4

MEASUREMENT OF ENVIRONMENTAL PARAMETERS USING DIGITAL SENSORS

1. Work Description

1.1. Objectives of the Work

- Creating and testing medium complexity circuits using sensors and transmitters.
- Using shield-type electronic modules.
- Developing a practical application for measuring temperature, relative humidity, atmospheric pressure values, and displaying them on an LCD screen

1.2. Theoretical Description

Introduction

Temperature is a quantity that characterizes the thermal state of a medium or an object. The relationship between the two is as follows: tC[°F] = tF[°C] × 1,8 + 32. Atmospheric humidity represents the amount of water vapor in the air. Relative humidity is the proportional relationship between the current humidity at a specific temperature and the maximum possible humidity at the same temperature, measured in percentages. It cannot exceed 100% because any excess will condense out. Atmospheric pressure represents the force with which air presses down on a unit area of the Earth's surface. Pressure is measured in Newtons per square meter or Pascal. In the case of atmospheric pressure, the most commonly used units of measurement are the millibar (1 mb = 100 Pa = 100 N/m2) and the millimeter of mercury.

Altitude is measured vertically in relation to a reference level, typically considered to be sea level. Atmospheric pressure decreases with increasing

altitude, and vice versa (approximately 10 mb per 100 m, valid up to a maximum of 3000 m. To calculate altitude based on pressure, the international barometric formula [1] can be used:

$$altitude = 44330 \cdot \left(1 - \left({}^{p}/_{p_0}\right)^{\frac{1}{5{,}255}}\right) \tag{4.1}$$

The standard pressure at sea level is 1013 mb or 760 mm Hg, but it can vary depending on atmospheric conditions.

Description of the applications

The purpose of these applications is to ultimately create a complex electronic circuit that measures the temperature, relative humidity, and atmospheric pressure in the surrounding environment using digital sensors, performs altitude calculations, and displays them numerically on an LCD screen. To create a weather station, the clock described in Paper 8 can be added, and it is recommended to use a larger LCD screen.

Application 1. Measuring humidity and temperature

To measure humidity, a transducer DHT22 [4], containing a capacitive humidity sensor as well as a temperature sensor will be used, due to the need for temperature compensation. The measured relative humidity and temperature values will be transmitted to the Arduino board [2], [5], through serial communication, using one of the digital input/output pins.

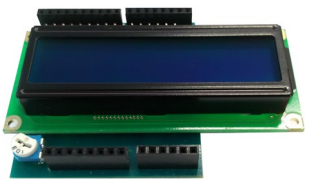
Application 2. Measuring atmospheric pressure and temperature

For measuring atmospheric pressure, a transducer BMP180 [6] containing a piezo-resistive pressure sensor as well as a temperature sensor will be used, due to the need for temperature compensation.

The measured atmospheric pressure and temperature values will be transmitted to the Arduino board through I2C, serial communication, using the available SCL and SDA data pins on the Arduino board. Inter-Integrated Circuit serial communication is a type of multi-master, multi-slave communication invented by Philips Semiconductor specifically for transmitting data between low-speed integrated circuits and processors or microcontrollers. The communication bus consists of two lines: one for transmitting/receiving data and one for transmitting/receiving the CLK signal. It is mandatory to install a pull-up resistor to 1 on each of the two data lines, and each circuit connected to an I2C bus must have its own address.

2. Hardware Components

The electronic components and modules used in the laboratory are those listed in the following table [1, 2], [4,5], [6,7]:

Component or module	Characteristics	Number of pieces	Image
Arduino Uno		1	
Breadboard	82x52x10 mm	1	
LCD S	Display on 2 lines of 16 characters each	1	

Connecting wire	Male to Male	8	
Humidity Transducer	DHT22	1	
Pressure transduce	BMP180	1	

Observations

In this laboratory, to assemble the electronic circuit using external components, a breadboard (Figure 4. 1 - electrical connections between pins are symbolized on the right side) will be used.

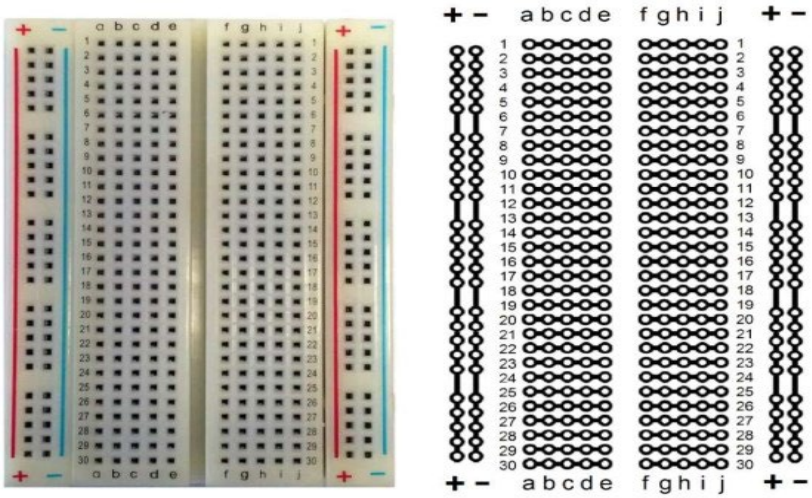


Figure 4.1. ***The breadboard and internal connections*** **(from [2], [5])**

Figure 4.2. *Humidity transducer* (from [4])

The LCD shield allows displaying characters on a liquid crystal display (LCD) screen with LED backlighting. It is mounted on top of the Arduino board, and its connectors are designed so that the board's pins remain accessible.

The LCD screen consists of 2 lines, each with 16 characters, where each character is composed of a 5x8 pixel grid.

The humidity transducer measures both relative humidity and ambient temperature, relying on the use of a calibrated and temperature-compensated precision sensor (DHT22).

The sensor is capacitive and can measure humidity, providing a digital data signal output through a serial connection.

The sensor's accuracy is ±0.5 °C for temperature and ±2% for humidity [2]. The pinout is presented in Figure 4.2 (NC stands for Not Connected).

The transducer requires the use of a 10 kΩ resistor between the data pin and VCC, acting as a pull-up resistor [5] (see Figure 4. 3). Its purpose is to maintain a logic level 1 at the data pin of the transducer when switching between input or output modes, or when there is no signal on this pin.

Establishing a stable logic level (1 in this case) prevents the occurrence of random 0 or 1 values at the Arduino board's digital input due to potential electrical noise [2].

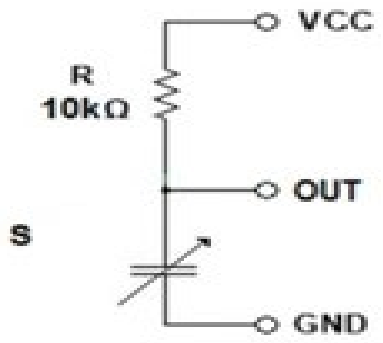


Figure 4.3. ***Utilization of pull-up resistor to*** **(from [1], [2], [5])**

The transducer will be powered with a voltage of VCC = 5V.

Measuring humidity and temperature

As it is a digital transducer, the measurement and determination of humidity and temperature values are done automatically by it. For displaying them, it is necessary to read the data signal without the need for other calculation formulas. The method of reading this signal can be implemented in the code sequence using instructions from the transducer's datasheet [4], but this requires more time and advanced programming knowledge.

However, there is a faster way to obtain the values for humidity and temperature, taking advantage of the fact that it is a widely used sensor, by using a library developed specifically for this type of sensor [6], called DHT.h (two libraries must be downloaded from the Internet - Adafruit_Sensor and DHT-sensor-library.

The advantage is the implementation in the code sequence with only a few steps:

- Setting the sensor type.
- Setting the digital pin where the data pin is connected.
- Defining the sensor.
- Initializing the sensor.
- Reading the humidity value with the command
- sensor_name.readHumidity() and assigning it to a variable.
- Reading the temperature value with the command sensor_name.readTemperature() and assigning it to a variable.

NOTE!

DHT22 is a "slow" sensor, meaning it will not react instantaneously to sudden changes in temperature or humidity. Reading from it may take up to 2 seconds or more [4]. The pressure transducer measures atmospheric pressure and ambient temperature, relying on the use of a high-precision and linear sensor (BMP180). The sensor is piezo-resistive and can measure pressure between 300-1100 mb and temperature between 0-65 °C, providing a digital data signal output through an I2C serial connection. The typical absolute accuracy of the sensor is ±1 °C for temperature and ±1 mb for pressure. The pressure transducer also contains two 4.7 kΩ resistors connected between each of the SCL and SDA data pins and VCC, acting as pull-up resistors (see Figure 4). When an I2C bus is shared by multiple modules, each having its own set of pull-up resistors, only one set will be retained, for example, by removing the solder from the SJ1 jumper, circled in green in Figure 4.4.

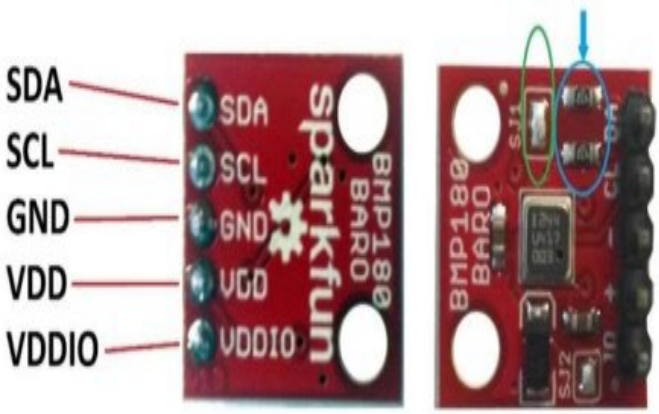


Figure 4.4. ***Utilization of pull-up resistors to 1*****(from [1], [2], [5])**

The VDDIO pin is only used when connecting the transducer to microcontrollers that operate at voltages lower than 3.3V. Since the I2C bus is used, the Wire.h library should be included in the code sequence, which is already available in the package of pre-installed libraries in the Arduino IDE. The transducer will be powered with a voltage of VCC = 3.3V.

Measuring atmospheric pressure and temperature

Similar to the humidity transducer, the measurement and determination of atmospheric pressure and temperature values are done automatically by it, and for displaying them, it is necessary to read the data signal

And for this transducer, a specially developed library called SFE_BMP180.h [7], can be used in the code sequence (the library needs to be downloaded from the Internet and imported into the Arduino IDE using the Sketch -> Import Library… -> Add Library… tab). It should be noted that temperature is measured first, followed by pressure, in order to compensate for temperature. Pressure measurement can be done in four modes (n = 0, 1, 2, 3), depending on the desired accuracy, by taking 1, 2, 4, or 8 samples, with conversion time ranging from 4.5 to 25.5 ms.

The steps required to obtain pressure and temperature value are:

- Define the sensor.
- Initialize the sensor.
- Start temperature measurement using the command sensor_name.startTemperature().
- Read the temperature value using the command sensor_name.getTemperature(temp_variable).
- Start pressure measurement using the command sensor_name.startPressure(n).
- Read the pressure value using the command sensor_name.getPressure(pressure_variable, temp_variable).

Altitude calculation

Altitude is automatically calculated when using the SFE_BMP180.h library. The altitude value can be read using the command sensor_name.altitude(pressure_variable, p0).

It is important to note that for sea level pressure (p0), the standard value of 1013 mb can be used. However, it is recommended to use the actual value correlated with atmospheric conditions (a value known by meteorological institutes and sometimes available online: see [1] for Bucharest.

3. Software Components

SFE_BMP180.h is the library containing the pressure sensor commands.

*LiquidCrystal lcdn*creates a lcd variable

specifying the digital pins used to control the LCD shield.

const means constant changing the behavior of a variable. The variable will become Read-only, that is, its value cannot be changed.

int variable = value sets a value for a signed 16-bit integer variable (from -32,768 to 32,767).

variable float = value sets a value for a signed 32-bit floating-point real variable (from -3.4028235E+38 to 3.4028235E+38). The total number of digits displayed accurately is 6 – 7 (includes all digits, not just the ones after the decimal point).

double variable = value sets a value for a floating-point real variable with double precision of the float variable.

char variable = value sets a value for a character variable

void setup() is a function (which returns no data and has no parameters) that runs only once at the beginning of the program. This is where the general program preparation instructions are set (setting pins, enabling serial ports, etc.).

void loop() is the main function of the program (which returns no data and has no parameters) and is executed continuously as long as the board is working and is not reset.

if(condition) {statement/s} else {statement/statements} tests whether a condition is met or not.

!= has the meaning different from.

humidity_sensor.begin() is a function that initializes the humidity sensor.

humidity_sensor.readHumidity() is a function that reads and returns the measured humidity value.

sensor_humidity.readTemperature() is a function that reads and returns the value of the measured temperature.

wire.begin() is a function that initializes the I2C bus.

pressure_sensor.begin() is a function that initializes the pressure sensor.

pressure_sensor.startTemperature() is a function that commands the start of the temperature measurement and returns the time required to perform this measurement.

sensor_pressure.getTemperature(variable) is a function that reads from the transducer, and gives the value of the measured temperature to the variable.

pressure_sensor.startPressure(n) is a function that commands the start of the pressure measurement and returns the time required for this measurement. n can take values between 0 and 3, signifying the measurement mode that determines the number of samples.

pressure_sensor.getPressure(pres_variable, temp_variable) is a function that reads and gives the measured pressure value to the variable, compensated with the temperature specified by temp_variable.

pressure_sensor.altitude(pres_variable, p0_variable) is a function that reads the altitude value calculated according to the two variables.

lcd.begin(columns, rows) initializes the LCD interface and specifies its number of rows and columns.

lcd.setCursor(column, row) sets the position of the LCD cursor. For the LCD used in this application, the number of columns is from 0 to 15, and the number of rows is from 0 to 1.

lcd.clear()

lcd.print() displays the data (values of some variables)/text between parentheses on the LCD screen.

To display a text it is necessary that it be placed between quotation marks (“text”).

To display the value of a variable of type char, byte, int, long, or string, write the name of the variable and, optionally, its number base (variable, BIN or DEC or OCT or HEX).

To display the value of a float or double type variable, write the name of the variable and after the comma, the number of decimals you want to display (variable, no. of decimals).

delay(ms) pauses

3.1. Functions, Commands, and Symbols Used

Creating custom characters to be displayed on the LCD

byte variable [number of values] = {values} sets a value for an unsigned byte variable. In this application, the defined variable has not a single value but an array of values, which determines which pixels will be turned on (value 1) and which pixels will be turned off (value 0) in the composition of a custom character (a character on the LCD is composed of 5x8 pixels).

lcd.createChar(number, variable) creates a custom character with an allocated number between 0 and 7, with pixel distribution according to the variable.

lcd.write(number) displays the character at the specified position (number).

In this laboratory, the custom character will be the one from Figure 4.5, representing the symbol for degrees Celsius.

4. Application 1. Measuring humidity and temperature

4.1. Building the Electronic Setup

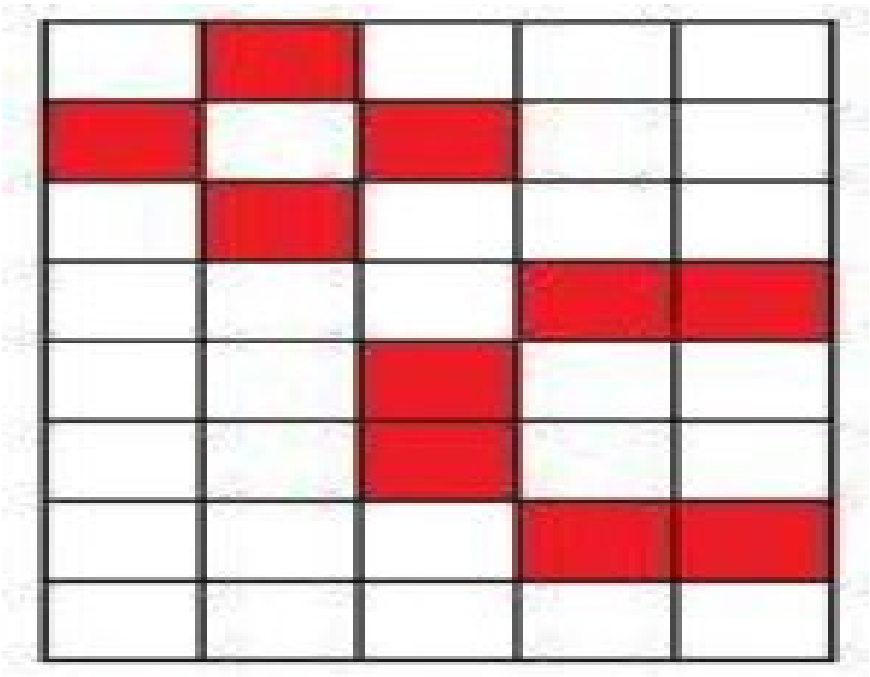


Figure 4.5. ***Custom character*** **(from [2], [5])**

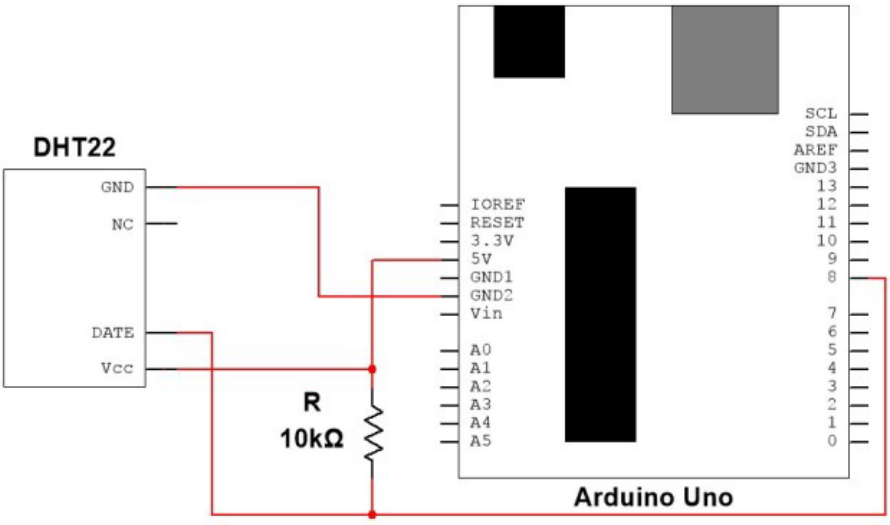


Figure 4.6. ***Block diagram for application 1*****(from [2], [4, 5])**

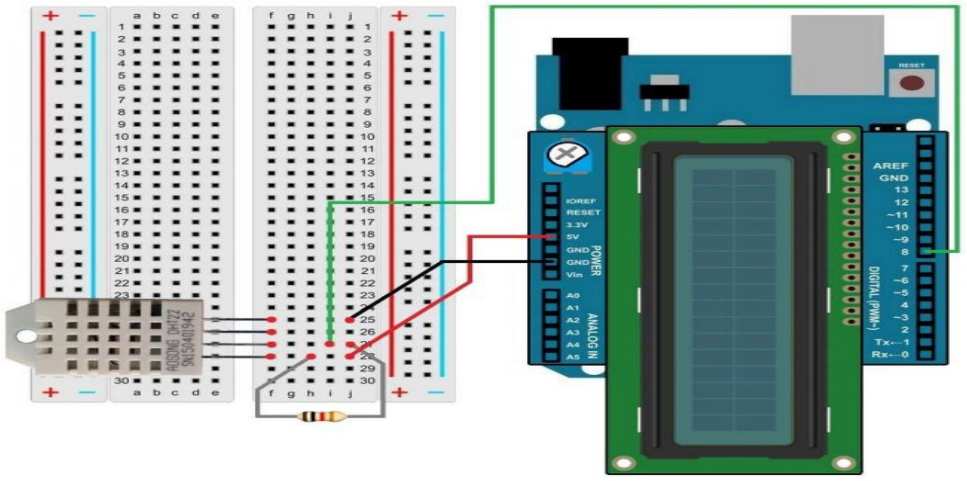
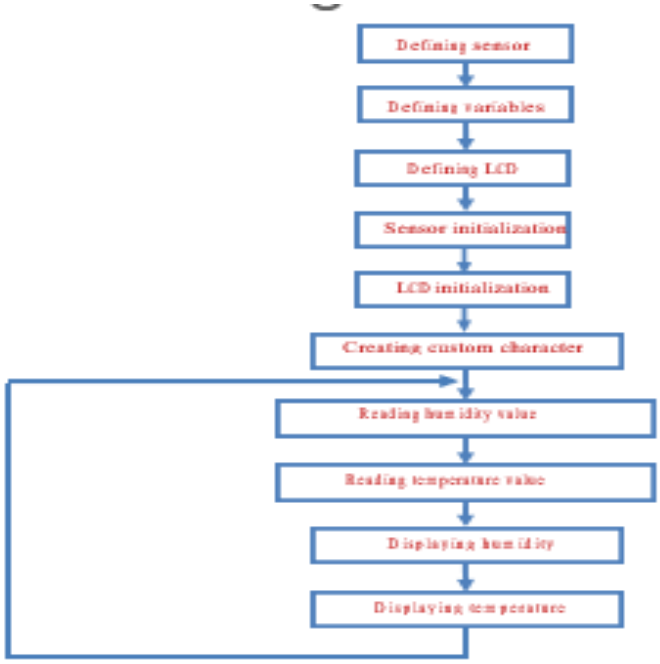


Figure 4.7. ***Electrical connections setup for application 1*****(from [2], [5])**

The following connections are made:

- The GND pin (power) on the Arduino board is connected via a wire to the GND pin of the DHT22 humidity sensor.
- The 5V pin (power) on the Arduino board is connected via a wire to the VCC pin of the DHT22 humidity sensor.
- Digital pin 8 on the Arduino board is connected via a wire to the DATA pin of the DHT22 humidity sensor.
- A resistor, with a value of 10 kΩ, is connected between the VCC and DATA pins of the DHT22 humidity sensor, serving as a pull-up resistor.

4.2. Logical Diagram and Code Sequence




```
#include <DHT.h>
// Including the library commands for the humidity sensor
char sensor_type = DHT22;
// Defining the sensor type
const int sensor_pin = 8;
// Defining the variable sensor_pin corresponding to digital port 8 where the
humidity sensor data output will be connected
// Definition of the humidity sensor
        float humidity;
// Definition of the humidity variable
float temperature;
}; …….
void setup(){
 humidity_sensor.begin();
// Initializing the humidity sensor
```

```
 lcd.begin(16, 2);
// Initializing the interface with the LCD screen and specifying the number of rows and columns
 lcd.createChar(1, degree);
// Creating the custom character that will have the content of the degree matrix and allocating position 1
}
void loop(){
 humidity = humidity_sensor.readHumidity();
// Reading the humidity value
 temperature = humidity_sensor.readTemperature();
// Reading the temperature value
 lcd.clear();
// Clearing the LCD screen
 lcd.print("Humidity = ");
// Printing the text between quotation marks on the LCD screen
 lcd.print(humidity,1);
// Printing the humidity variable value on the LCD screen with one decimal place
 lcd.print("% ");
// Printing the text between quotation marks on the LCD screen
 lcd.setCursor(0, 1);
// Moving the cursor to column 1, row 2
 lcd.print("Temp = ");
// Printing the text
// Printing the custom character on the LCD screen at position 1
 delay(1000);
// Delaying for 1 second
}
```

5. Application 2. Measuring atmospheric pressure and temperature

5.1. Implementing the electronic assembly

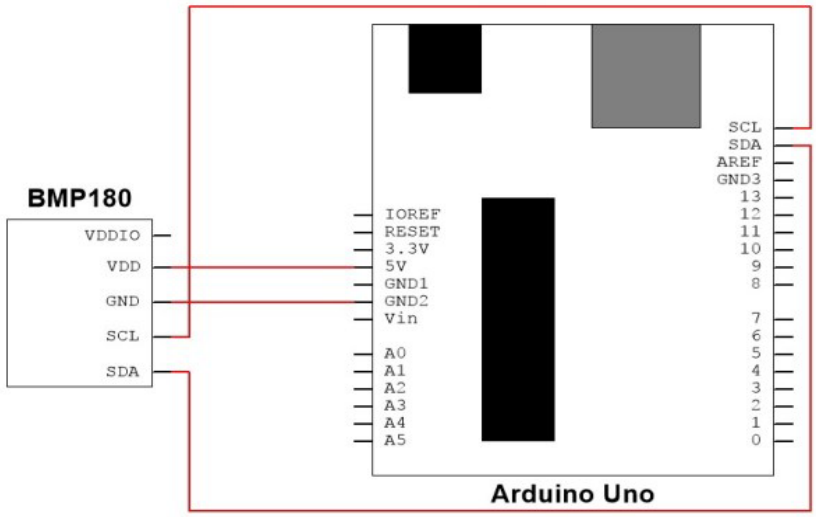


Figure 4.7. ***Block diagram for application 2*****(from [2], [5])**

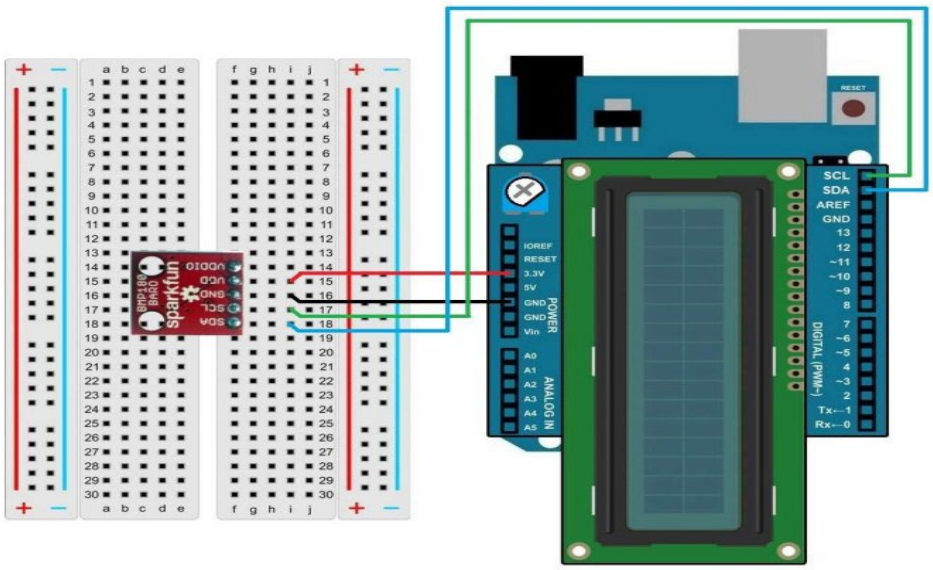
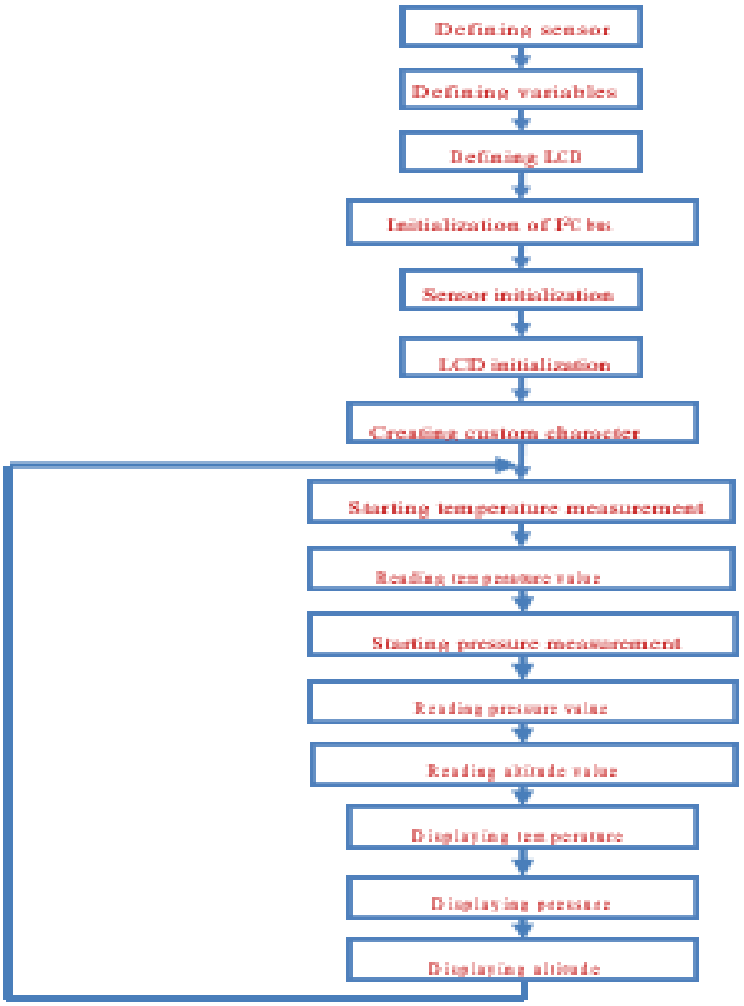


Figure 4.8. ***Electrical connections setup for application 2*****(from [2], [5])**

5.2. Logical diagram and code sequence




```
#include <Wire.h>
// including the library for I2C bus commands
#include <SFE_BMP180.h>
// including the library for pressure sensor commands
SFE_BMP180 pressure_sensor;
```

```
// defining the pressure sensor
double pres, temp, alt;
// defining variables for pressure, temperature, and altitude
double p0 = 1013;
// defining the variable p0, sea-level pressure, see section 2 of the document
void setup (){
 Wire.begin();
// initializing the I2C bus
 pressure_sensor.begin();
// initializing the pressure sensor
 lcd.begin(16, 2);
// initializing the LCD interface and specifying the number of rows and columns
 lcd.createChar(1, grad);
// creating a custom character with the content of the grad array and allocating it to position 1
}
void loop(){
int status;
// defining the status variable as an integer
status = pressure_sensor.startTemperature();
// starting the temperature measurement, the function returns the required time
if (status != 0) {
// if the required measurement time is not zero
delay(status);
// delay for the required time
pressure_sensor.getTemperature(temp);
// assigning the measured temperature value to the temp variable
}
```

```
status = pressure_sensor.startPressure(3);
// starting the pressure measurement (specifying the desired number of samples), the function returns the required time
if (status != 0) {
// if the required measurement time is not zero
delay(status);
// delay for the required time
pressure_sensor.getPressure(pres,temp);
// assigning the measured pressure value to the pres variable, compensated with the temperature temp
}
alt = pressure_sensor.altitude(pres,p0);
// assigning the calculated altitude value to the alt variable based on the measured pressure and p0
lcd.clear();
// clearing the LCD screen
lcd.print(temp,1);
// displaying the value of the temp variable on the LCD screen, with one decimal place
lcd.write(1);
// displaying the custom character on the LCD screen at position 1
lcd.print(" ");
// printing the text between the quotation marks on the LCD screen
lcd.print(pres,1);
// displaying the value of the pres variable on the LCD screen, with one decimal place
lcd.print("mb");
lcd.setCursor(0, 1);
// moving the cursor
```

```
lcd.print("Alt=");
// printing the text between the quotation marks on the LCD screen
lcd.print(alt,1);
// display the value of the alt variable on the LCD screen, with one decimal place
lcd.print("m");
// display the text between the quotation marks on the LCD screen
delay(1000);
// delay for 1 second
}
```

Additional Exercises and Conclusions

Modify the code sequence to display a "Red alert" warning when the temperature exceeds 30°C and humidity exceeds 80%. Remove the automatic altitude reading function from the code sequence and replace it with the calculation formula presented in the introduction (formula (4.1)).

Group the two applications into a single one, selecting some data to be displayed on the LCD screen and others on the serial monitor. For most programming languages, it is very useful to build libraries of functions or programs aimed at simplifying the writing of software applications by using pre-defined functions (especially those commonly used) [1]. A float or single-precision variable is characterized by the allocation of 4 bytes (32 bits) which will be used as follows: the first bit will be the sign bit, the next 8 bits will be necessary for encoding the exponent, and the last 23 bits will be used for encoding the fraction. Representing data and information in different environments, as well as abstracting and encoding certain states, require the use of different data types. The data type is chosen based on criteria regarding the optimization of written programs. Double type variables require a larger volume of stored data than float type variables and more processing time.

BIBLIOGRAPHY

1. Lelutiu, L.M. 2016. *Data acquisition*. Brașov: Editura Universitatii "Transilvania" din Brașov, ISBN 978-606-19-0866- 0
2. McRoberts, Michael. 2013. *Beginning Arduino*, 2nd edition. München, Oldenburg: Apress, https://www.oreilly.com/library/view/beginning-arduino-second/9781430250166/
3. Teodoreanu, E., Bunescu, I. 2007. "Thermal Confort". *Journal Present Environment and Sustainable Development*, 1: 135-142. Iași: University "Alexandru Ioan Cuza".
4. ***. 2025. Aosong Electronics Co., Ltd. *Digital-output relative humidity & temperature sensor/module* - DHT22. https://sigmanortec.ro/en/ temperature-and-humidity-sensor-dht22-am2302-original-module
5. ***. 2015. *Arduino Playground,* http://playground.arduino.cc/ Common Topics/ PullUpDownResistor.
6. *** . 2013. Bosch Sensortec. *BMP180 Digital pressure sensor – Datasheet,* https://github.com/adafruit/DHT-sensor-library.
7. ***. 2013. SparkFun. *BMP180 Breakout Arduino Library*". Bosch Sensortec. *BMP180 Digital pressure sensor – Datasheet,* https://github. com/sparkfun/ BMP180_Breakout_Arduino_Library

Paper 5
MEASUREMENT OF THE LIGHT LEVELS

1. Work Description

1.1. Objectives of the Work

- Designing and testing circuits of medium complexity which use sensors and transducers.
- Creating a practical application for measuring the illumination levels using digital and analogue sensors and showing the result on the monitor.

1.2. Theoretical description:

Introduction:

The oscillations of magnetic and electrical fields which are perpendicularly placed of one another and which are generated reciprocally are called electromagnetic waves. Like any other oscillation, a measuring unit which is defining them is the period (measuring unit: [s]) and its inverse is the frequency (measuring unit: [Hz]).

The propagation speed of electromagnetic waves in vacuum depends on the medium which they cover, their speed may be slower. Knowing the propagation speed and the frequency, the wavelength can be computed.

$$\lambda = \frac{v}{f}$$ (measuring unit: [m])

where:

λ - wavelength,

v - light speed in the covered medium,

f - frequency of electromagnetic waves.

Electromagnetic waves can be classified after their frequency or wavelength:

- Electromagnetic waves of radio-frequency: from some Hz to GHz; for example, VHF (Very High Frequency) has frequencies between 10MHz - 300MHz and wavelengths between 10m - 1m.
- Microwaves: has frequencies between 1GHz - 100GHz and wavelengths between 300 - 3mm.
- Terahertz radiations
- Infrared: has frequencies between 300GHz - THz and wavelengths between 1mm - 700nm.
- Visible Spectrum (of humans): has frequencies between 430THz - 790THz and wavelengths between 700nm - 380nm.
- Ultraviolet: has frequencies between 790THz - 30PHz and wavelengths between 10nm - 380nm.
- X-Rays: has frequencies between 30PHz - 30 EHz and wavelengths between 10pm - 10nm.
- Gamma Rays: has frequencies bigger than 30EHz and wavelengths smaller than 10pm.

Light, perceived like the stimulus of the human eyes, is only the part of visible spectrum of electromagnetic waves and it is characterized by:

- **Color** - is given by the frequency (measured in Hz) or the wavelength of radiation (measured in nanometers). Color, from the technical point of view, it is not the same as the color perceived by the human eyes, making a synthesis of three elementary colors: red, green and blue (RGB). The visible spectrum begins with the red color (610 - 780 nm) and it ends with the violet color (380 - 424 nm), as seen in the following figure 5.1:

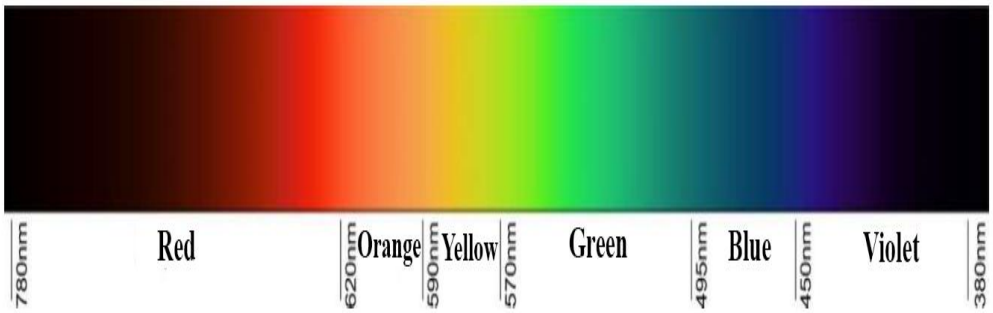


Figure 5.1. ***The visible spectrum*** **(from [2])**

- **The luminous intensity** - it is the power emitted on a given direction or the power transported by the radiation (measured in "cd" - candela).
- **The luminous flux** - it is measure in "lm" - lumen and represents the total quantity of radiation emitted by a source.

  Illumination - it is the luminous intensity distributed on a surface (measured in "lx" - lux).
- **Polarization** - electromagnetic waves oscillations plans.
- **Coherence** - oscillations phase.

Relations between measurement units presented above are the following:

$$1lx = \frac{1lm}{m^2} = \frac{1cd \cdot 1sr}{m^2} = \frac{\frac{1W}{683sr} \cdot 1sr}{m^2} = \frac{1}{683}\frac{W}{m^2}$$

Conditions of illumination [2] typical for different environments are presented in the following table:

Condition of illumination	Luminous intensity
Full moon	1 lx
Street lighting	10 lx
House lighting	30...300 lx
Desk lighting	100...1000 lx
Medical operations lighting	10000 lx
Direct sun light	100000 lx

For the detection or measurement of the presented characteristics, there can be used light sensible devices, like: photoresistors, photodiodes, photovoltaic cells or phototransistors.

Photoresistor - it is a passive electronic component for which its electrical resistance is modified depending on the luminous flux.

The sensitivity of the photoresistor is measured in mA/lx at a constant voltage, linear for big domains of illumination, however it depends a lot on the color of the light (also depending on the material used in the construction of the photoresistor).

The direction of applied voltage does not matter.

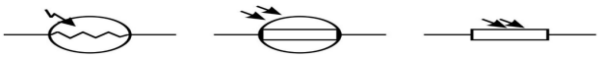


Figure 5.2. ***Symbols used for photoresistors*** **(from [2])**

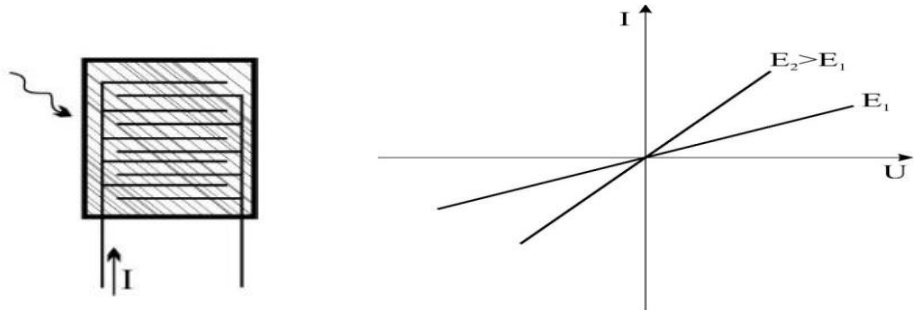


Figure 5.3. ***Construction & U-I characteristic of the*** **photoresistors (from [2])**

Photodiode - it is a pn junction and it's based on the photovoltaic effect. By illuminating the active surface, at the diode's terminals, an electrical voltage will appear from the anode to the cathode. A photodiode is used reversed polarized, the reversed current being represented by the illumination current (for zero illumination, we call it dark current). It has a better sensibility and response time than the photoresistor.

Figure 5.4. ***Symbol used for photodiode***

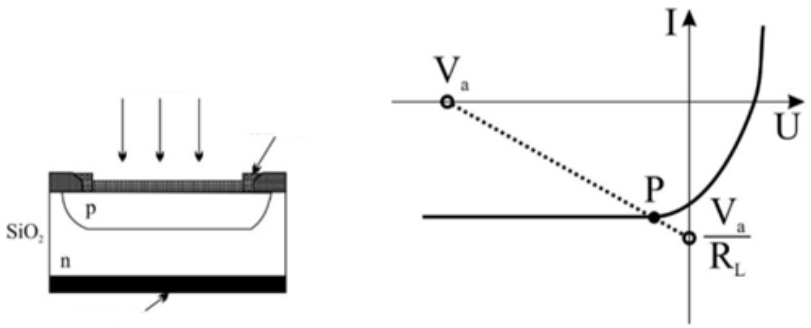


Figure 5.5. ***Construction & U-I characteristic of the photodiode*** **(from [2])**

Phototransistor - it is a combination of two pn junctions (npn or pnp), like on the ordinary bipolar transistors, for which it is illuminated at the base-collector region. The light which falls on the phototransistor generates a base voltage required for its polarization and causes the apparition of a current collector.

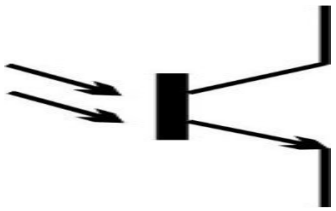


Figure 5.6. ***Symbol used for phototransistor*** **(from [2])**

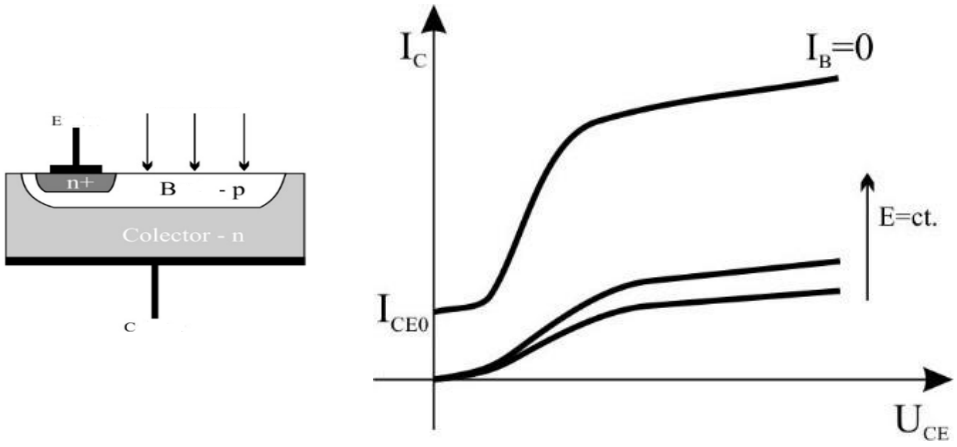


Figure 5.7. ***Construction & U-I characteristic of the phototransistor*** **(from [2])**

DESCRIPTION OF APPLICATIONS

The goal of these applications is to design electronic circuits which can measure the level of light by using different methods and display values on an LCD display or a serial monitor.

1st Application. ***Measuring the level of light with an analogue sensor***

For the measurement of the light level, there will be used a transducer based on a sensor (PT15-21C/TR8) of phototransistor type. The transducer provides at the output an analogue voltage which will be applied at one of the analogue inputs of the Arduino board. On the base of the analogue from the input, the board will provide a corresponding digital value, which will be used to display the measured light level on the serial monitor.

2nd Application. ***Measuring the level of light with a digital sensor***

For light measurement there will be used a transducer (TSL235R) based on a photodiode type of sensor. The transducer provides at the output a digital rectangular signal having its frequency proportional with the measured light level by the sensor. The Arduino board will read this frequency and will display on the serial monitor the corresponding value.

3rd Application. ***Measuring the level of light with a digital RGB sensor***

For the measurement of the RGB light there will be used a transducer based on a ISL29125 sensor. This sensor contains a matrix of photodiodes which decomposes the light in specters of Red, Green and Blue and measures the light level for all them.

The measured values will be transmitted to the Arduino board, by the medium of a series communication of I^2C type, using the available SCL and SDA data pins on the board and after they will be displayed on the serial monitor. Series communication I^2C (Inter Integrated Circuit) is a type of multi-master, multi-slave communication invented by Philips Semiconductor for the transmission of data between slow speed integrated circuits, processors or microcontrollers. The communication bus is formed by two lines, one for the transmission/ reception of data, SDA (Serial Data Line) and one for

transmission/ reception of clock signals, SCL (Serial Clock Line). It is mandatory to mount a lifting resistor on both data lines and every connected circuit to a I^2C bus must have its own address.

2. Hardware Components

The components and electronic modules used in this practical work are presented in the following table:

Component or module	Characteristics	Number of pieces	Image
Arduino Uno	—	1	
Breadboard	82x52x10 mm	1	
Connection Wire	Father-Father	10	
Light Transducer	PT15- 21C/TR8	1	
Digital Light Transducer	TSL235R	1	
Digital RGB Light Transducer	ISL29125	1	
Logic Level Converter	BSS138	1	

In this practical work, a test bench of breadboard type will be used for the external components to be used to make the required electric circuit. (Figure 5.8 – in the right side are the electrical connections between the pins shown).

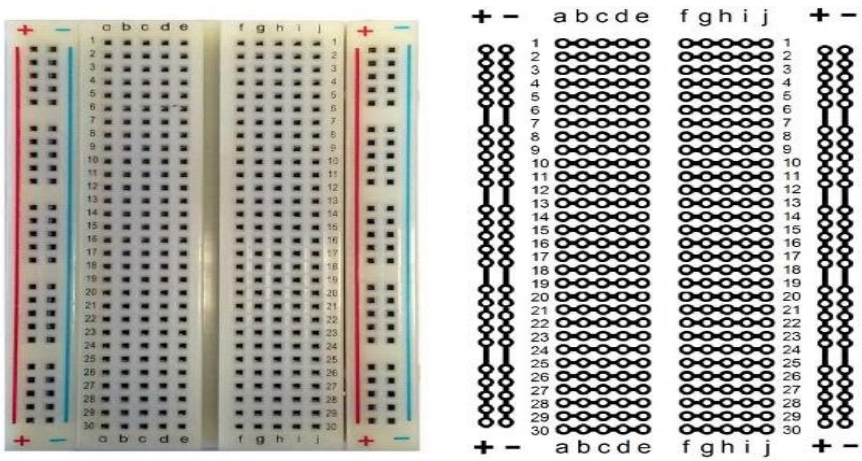


Figure 5.8. ***The breadboard and the internal connections*** **(from [1, 2], [7])**

The analogue light transducer is used to measure the light level, its operation being based on a phototransistor NPN of PT15-21C/TR8 type, having small response time and high sensitivity.

Figure 5.9. ***Analogic Light Transducer*** **(from [1, 2], [7])**

The transducer contains alongside the sensor a resistor connected between the output pin of the module (OUT) and ground (GND). This is called

a pull-down resistor [2] (see Figure 9) with the role of maintaining logical value 0 at the output module when light is not present. Keeping a stable logic level (0 in this case) stops the random occurrences of the values of 0 or 1 at the digital input of the Arduino board due to the possible electric noise [1].

The bandwidth of the light spectrum that can be detected by the transducer is between 400 and 1100 nm [4], with the highest sensitivity being around the value of 940 nm.

The characteristic of the transducer is a linear one, as it can be seen in Figure 10, though it is not standardized. The transducer will be supplied with voltage VCC = 5 V.

2.1. The Measurement of the Light Level

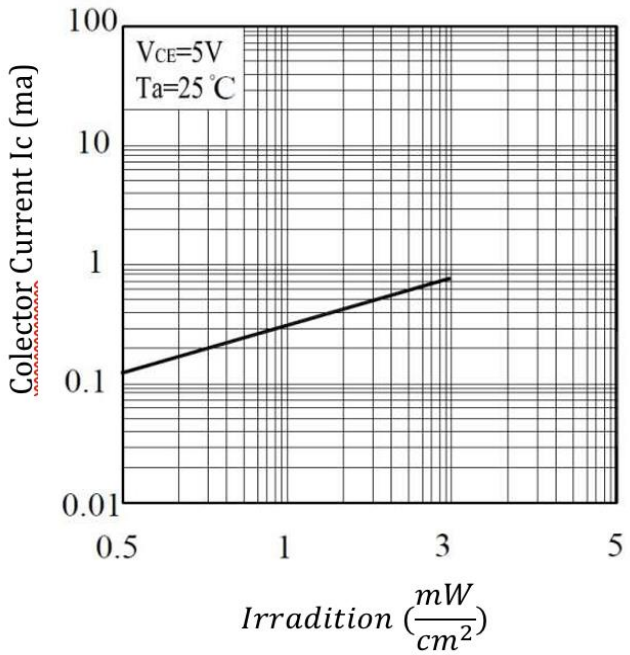
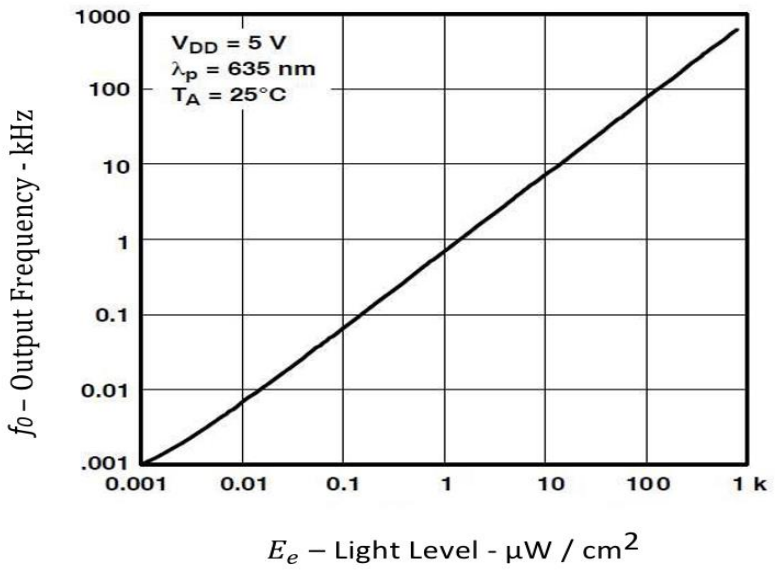


Figure 5.10. ***The Characteristic of the Light Transducer*** **(from [2], [5])**

Depending on the amount of light that falls down on the sensor, the Arduino board will provide a digital value corresponding between 0 (no light) and 1023.

The digital light transducer measures the light level by making use a TSL235R transducer type [5]. This outputs a rectangular signal, having the frequency directly proportional with the light intensity that falls down on the photodiode. The bandwidth of the light spectrum that can be detected by the transducer is between 320 and 1050 nm.

Figure 5.11. ***The Digital Light Transducer*** **(from [2],[5])**



The transducer contains, alongside the capacitor connected between the VCC pin (3.3 V) and ground (GND), used for decoupling. The transducer has an approximately linear characteristic, as it can be seen in Figure 5.12.

The transducer will be supplied with voltage VCC = 3.3 V.

Taking into consideration that for a light level of 430 μW / cm^2 the typical frequency of the output signal is 250 kHz and considering the linear characteristic of the transducer [5] the following calculus formula will result:

$$E_e = \frac{450}{250} \cdot f_o = 1.72 \cdot f_o[\mu W/cm^2],$$

where f_o is measured in kHz

For measuring the frequency of the output signal of the transducer with the Arduino board the use of interrupts has been chosen with the help of the attach Interrupt function.

The function monitors one of the digital input pins (in the case of the Arduino Uno board only 2-3 pins can be used) and activates a special function (named ISR – Interrupt Service Routine) when a specific condition is met (in this case: the passing from 0 to 1 of the logical level applied at the monitored pin). The calculation of the frequency presumes the counting of the passes from 0 to 1 logic of the transducer's signal that appear in a 1 second period. For this, the ISR function will contain a simple counter that will be incremented at each read pulse.

To be remembered! The ISR function [10] cannot have parameters or return a result and functions such as millis (), micros () or delay () cannot be used due to them being based on interrupts.

The type ISL29125 digital RGB light transducer measures the light level for each three light spectrums, red, green, blue, using a sensor made out of a photo-diode matrix. The output data are available to be read through a type I2C bus, the transducer being a slave type device.

The transducer RGB [6] has two domains of optical sensitivity, with the resolution of 12 and 16 bits, selectable through programming. The infrared waves are ignored, as well as the noise made by the 50 and 60 Hz frequencies of the artificial light sources [3].

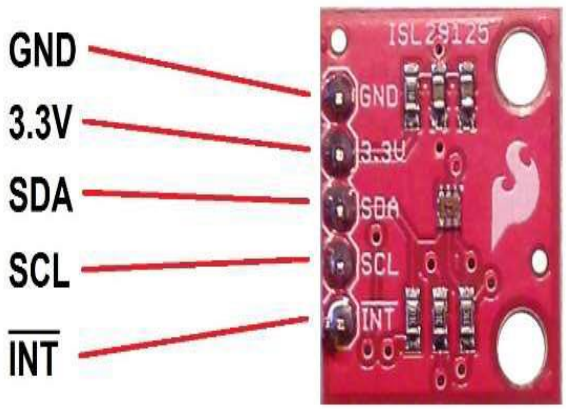


Figure 5.13. ***The digital RGB light transducer*** **(from [6])**

The pins of the transducer have the following roles:

- INT pin is used for the triggering of an interrupt
- SDA and SCL pins are used for the connections to the bus I2C
- GND pin is used for the connection to the ground
- 3.3 V pin is used for the supply of the transducer

The transducer contains two resistors of 10kΩ value each connected between the SCL and SDA data pins and VCC with step up to 1 (Figure 5.14). By eliminating the solder around the jumper circled with green from Figure 5.14 we remove the two resistors, useful when a I2C bus is divided into multiple modules, each having a set of step-ups to 1 resistor, when external or the

included resistors (and automatic activated) in the ATmega328 microcontroller from the Arduino board [3].

Figure 5.14. ***The use of the step up to 1 resistor*** **(from [1],[7])**

The transducer will be supplied with VCC = 3.3 V.

The RGB light transducer is supplied with 3.3 V voltage, the I2C bus will also be supplied with 3.3 V. Because the Arduino board requires 5 V voltage for the I2C bus, a logic level converter is required. The logic level converter has the role to divide two circuits which use logic levels with different voltages, 3.3 V (LV) and 5 V (HV). The converter takes the logical levels (0 or 1) and sends to the output only the value of the signal's voltage. For this practical work a bidirectional logic level converter [8] with four channels which can transfer the logical values between the two circuits in both ways (Figure 5.15).

Figure 5.15. ***The bidirectional logic level convertor*** **(from [1, 2], [7])**

Two of the data channels will be used, one for the SCL connection and one for the SDA one, for the I2C bus.

Measurement of the light level for each RGB component

Because the data readout is made through the I2C bus, the Wire h library will have to be added to the preinstalled libraries in the Arduino IDE. The measuring of the light levels for the RGB components with be made automatically, the data signal will have to be read for them to be displayed. For this transducer, a special library [9] can be used called SparkFunISL29125.h (the library will have to be downloaded and imported into Arduino IDE using Skecth -> Import Library… -> Add Library…).

The steps required for obtaining the values of the light levels are:

- The defining of the sensor.
- The initialization of the sensor.
- The reading of the light level for the red color using the function transducer_RGB.readRed().
- The reading of the light level for the green color using the function transducer_RGB.readGreen().

- The reading of the light level for the blue color using the function transducer_RGB.readBlue().

Depending on the resolution of the analog/digital converter of the transducer, the domain of values that results from the data readout that is provided by the transducer can be:

- 0 … 212 - 1 = 4.095 for the 12 bits resolution.
- 0 … 216 - 1 = 65.535 for the 16 bits resolution.

These values do not describe the irradiance in a known format. Two of the known formats, as well as the mode of transforming them are:

- In standard mode, the domain of values that describes the irradiance is 0 – 1°. To display this, the resulting value is divided by the maximum value of the measuring domain (4.095 or 65.535).
- Another value domain commonly used is 0 – 255 (e.g. R = 147, G = 244, B = 84). To display the irradiance, the resulting value is divided by 4095/255 and 65535/255(=257) respectively.

To be remembered! The library SparkFunISL29125.h has predefined the 16 bits resolution.

The calculation of the general light level

Outside the RGB format, which creates a specific color based on the three fundamental colors and which differs from one system to another depending on the accuracy with which the three colors are produced, the XYZ format is used. This is considered a general format with the help of which any color that is visible to the human eye can be defined.

The conversion between the two formats is made with the help of a matrix that contains the transformation coefficients.

The value Y of a color described through the XYZ format represents its irradiance.

This way, the irradiance measured in lux can be calculated with the following formula:

$$Y = E_v = (C_{YR}\frac{red}{2^n - 1}) \cdot (C_{YG}\frac{green}{2^n - 1}) \cdot (C_{YG}\frac{blue}{2^n - 1}) \cdot d[lux]$$

where:

- The C coefficients are the transformation coefficients from the RGB format into the XYZ format.
- The value of the irradiance for each color will be in the 0 – 1 domain, as presented previously (for this reason the division by 2n – 1 is used, n being the resolution of the analog/digital converter of the transducer).
- The measuring domain for the irradiance can be 375 lux or 10000 lux, depending on the configuration mode of the transducer.

To be remembered! The library SparkFunISL29125.h has the 16 bits resolution and the 10000-lux domain predefined.

There exist many defined RGB domains, depending on the purpose in which these are used, the most known one being the sRGB. The transformation matrix for it is the following:

$$\begin{bmatrix} 0.4125 & 0.3576 & 0.1804 \\ 0.2127 & 0.7152 & 0.0722 \\ 0.0193 & 0.1192 & 0.9503 \end{bmatrix}$$

As a result, the three coefficients used in the previous formula will have the following values: CYR = 0,2127, CYG = 0,7152, CYB = 0,0722.

3. Software Components

Wire.H is the library that contains the commands for the I2C bus.

volatile

const represents a constant. This variable will be of Read-only type, its value not being able to be changed.

int variable = value defines a value for 16 bits signed integer type variable (from -32.768 until 32.767).

unsigned int variable = value defines a value for a 16 bits unsigned integer type variable (from 0 until 65.535).

unsigned long variable = value defines a value for a 32 bits unsigned integer type variable (from 0 until 4.294.967.295).

float variable = value defines a value for 32 bits

The total number of digits shown with precision is 6 – 7 (including all digits, not only the ones after the comma).

void setup() is a function (which returns data and has no parameters) that runs one time at the start of the program. Here, the general instructions for the program are defined (setting up the pins, the trigger of the serial ports, etc).

void loop() is the principal function of the program (it does not return data and has no parameters) which is executed continuously as long as the board is on and functioning and has not been reset.

attachInterrupt(digitalPinToInterrupt(pin), name_function_ISR, mode) allows the use of interrupts for a digital pin. In the case of the Arduino board, the use of interrupts can use only the 2 and 3 pins. The modes of activation of interrupts can be: LOW (when the pin is logic 0), CHANGE (when the pin is changing its logic value), RISING (when the pin is changing its value from logic 0 to logic 1), FALLING (when the pin changes from logic 1 to logic 0).

Serial.begin(speed) defines the transfer rate of data for the serial port in bits/second (BAUD).

Serial.print(value or variable, numbering system) prints out data under the form of ASCII characters using the serial port.

Serial.println(value or variable, numbering system) prints out data under the form of ASCII characters using the serial port, moving to a new line after the data is shown.

analogRead(pin) read the value of the specified digital pin.

If(condition) {instruction/instructions} tests the condition and executes a piece of code depending on the result.

delay(ms) sets a delay in the program for a period of time which is specified in milliseconds.

millis() is a function that returns as a value the number of milliseconds the passed from the beginning of the execution of the code.

/ is a division operator which only displays the integer part of a division.

++ is used to increment a variable.

Application 1. Measuring the level of light with an analogue sensor

Electronic assembly

The following connections will be made:

- Pin GND (power) from the Arduino board to the transducer′s GND pin
- Pin 5V(power) from the Arduino board to the transducer′s VCC pin
- Pin A0 from the Arduino board to the transducer′s OUT pin

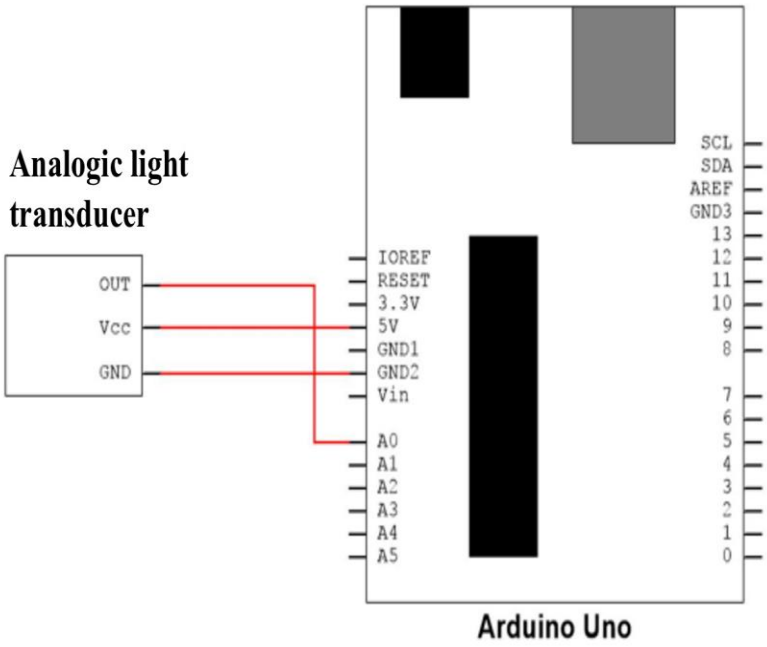


Figure 5.16. ***Diagram for application 1*****(from [1],[7])**

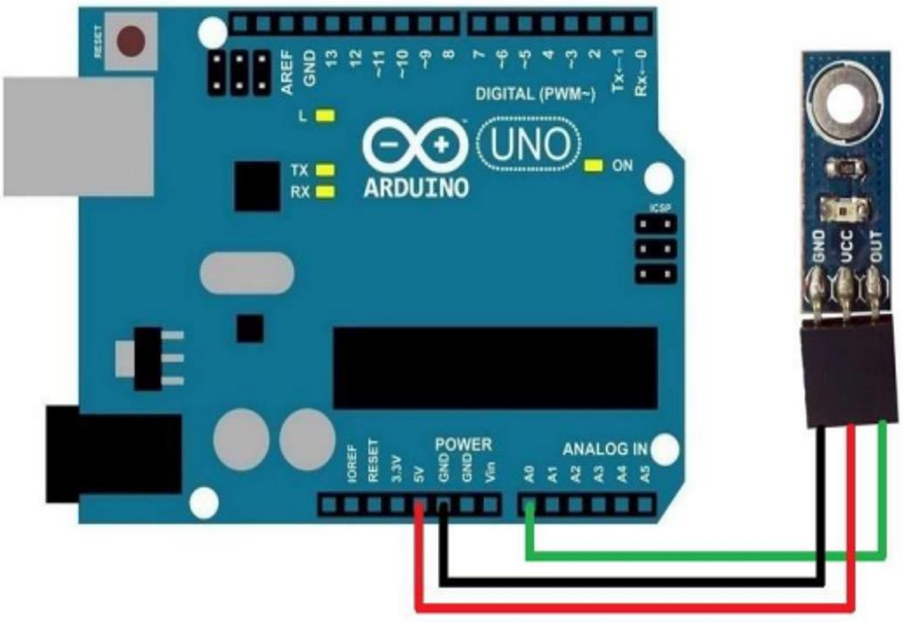
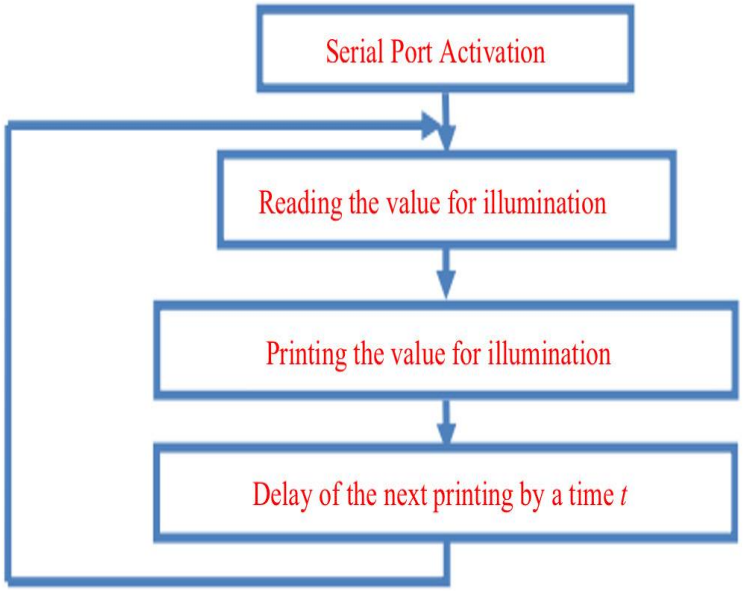


Figure 5.17. ***Electrical connections for application 1*****(from [1, 2])**

Logic scheme and code sequence




```
void setup() {
  Serial.begin(9600);
        //activates the output of the serial port
}

void loop() {
  const int Ilum value = analogRead(0);
        //declaration of a constant integer variable Ilum value, which takes the
value of the analogic input A0
  Serial.print("Ilum value: ");
        //prints the text in parentheses on the serial display
  Serial.println(valueIlum, DEC);
        // displays the recorded value in decimal form
  delay(1000);
        //delays the next display with 1000 ms
```

Application 2. Measuring the level of illumination with a digital sensor

Electronic Assembly

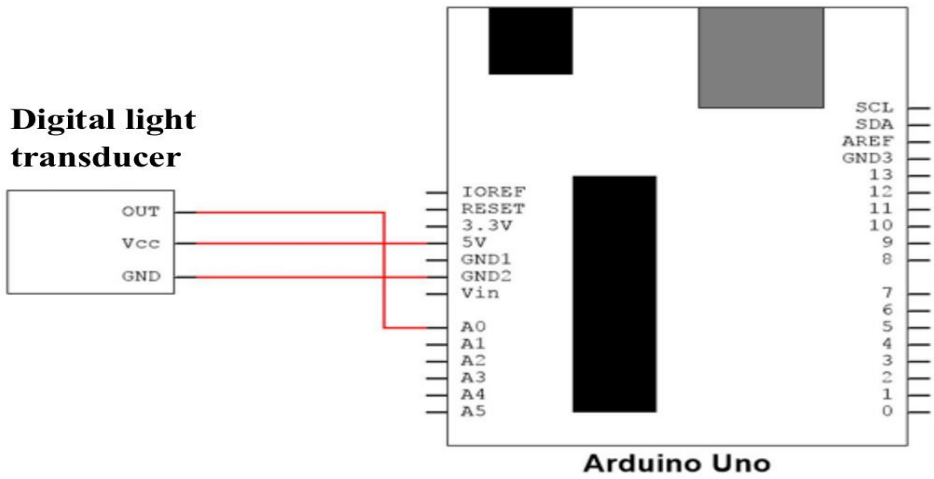


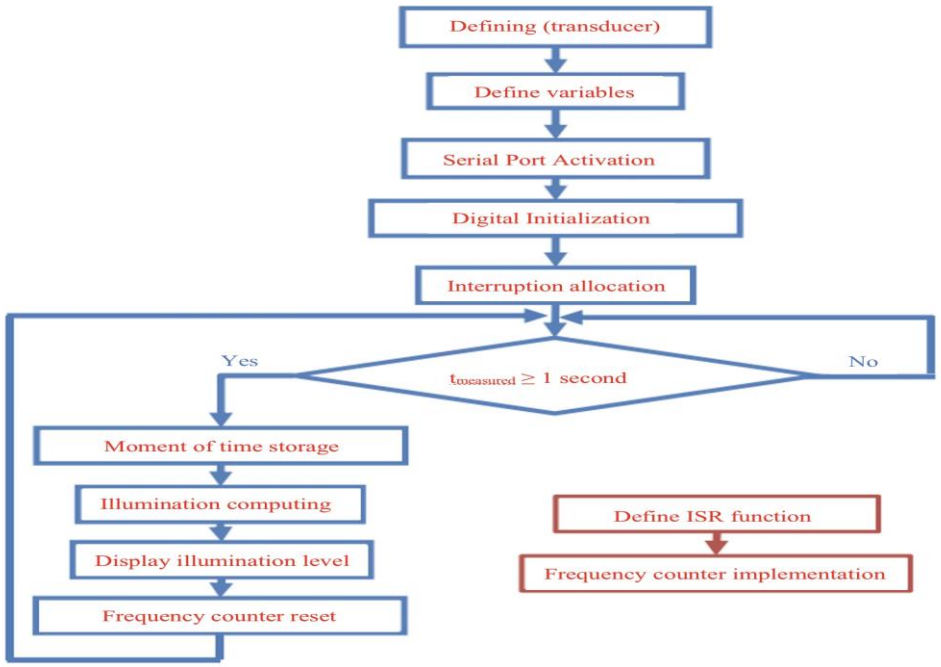
Figure 5.18. ***Diagram for application 2*** **(from [1, 2])**

Figure 5.19. ***Electrical connections*****(from [2])**

Electric fitting

The following connections will be made:

- Pin GND (power) from the Arduino board to the transducer's GND pin
- Pin 3.3 V(power) from the board to a transducer's 3.3 V pin
- Digital pin 2 from the Arduino board to the transducer's OUT pin



Logic scheme and code sequence

```
const int traductor = 2;
//defining the variable corresponding to the second digital port where the transducer′s output OUT is connected
volatile unsigned long frecv = 0;
//variable corresponding to the frequency of the signal received by the transducer
unsigned long millis_vechi = 0;
//variable corresponding to the moment of time from which the impulse counting began
void setup() {
 Serial.begin(115200);
//activates the serial port output with a rate of 115200 baud
```

```
pinMode(traductor, INPUT);
//declares the transducer′s pin as the input
  digitalWrite(traductor, HIGH);
//writes the logic high on the transducer′s pin in order to begin the counting from the first impulse
  attachInterrupt(digitalPinToInterrupt(traductor), irq, RISING);
//it interrupts the transducer′s pin and the irq function executes when a passing from logical 0 to 1 occurs
}
 void loop() {
  if (millis() - millis_vechi >= 1000)
//condition that determines the moment of passing a period of 1 second
  {
   millis_vechi = millis();
//storage of a new moment of time from which the impulses counting begins
    Serial.print("Nivel lumina = ");
//displays the text in the parentheses
    Serial.print(1.72*(frecv/1000));
//displays the value of the level of illumination
    Serial.println(" uW/cm2");
//displays the text in the parentheses
  frecv = 0;
//frequency counter reset
  void irq() {
//ISR function
  frecv++;
//Frequency counter implementation
}
```

Application 3. Light level measurement using an RGB digital sensor

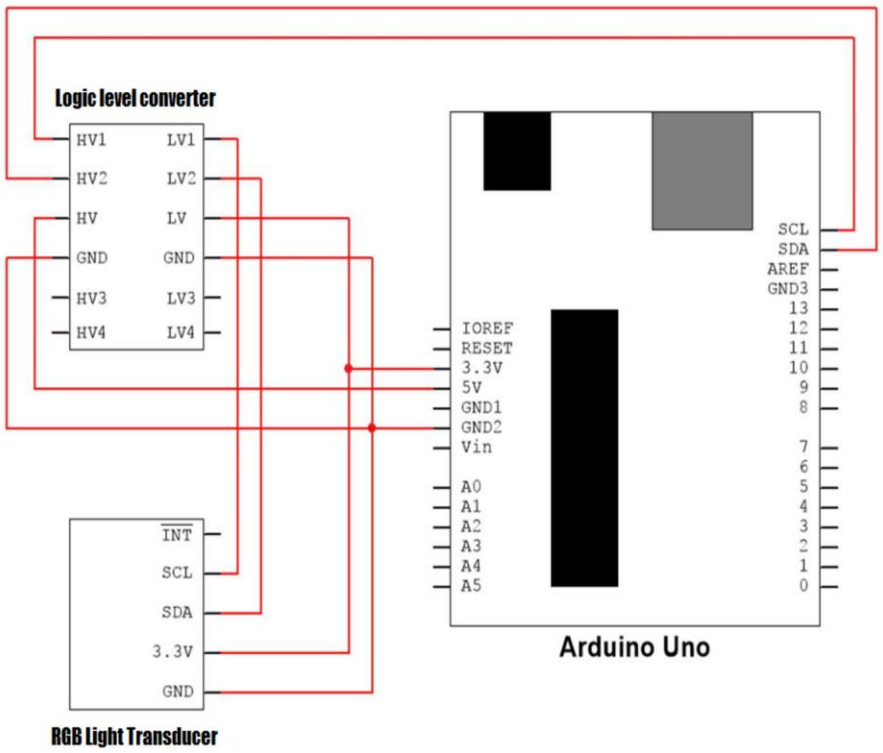


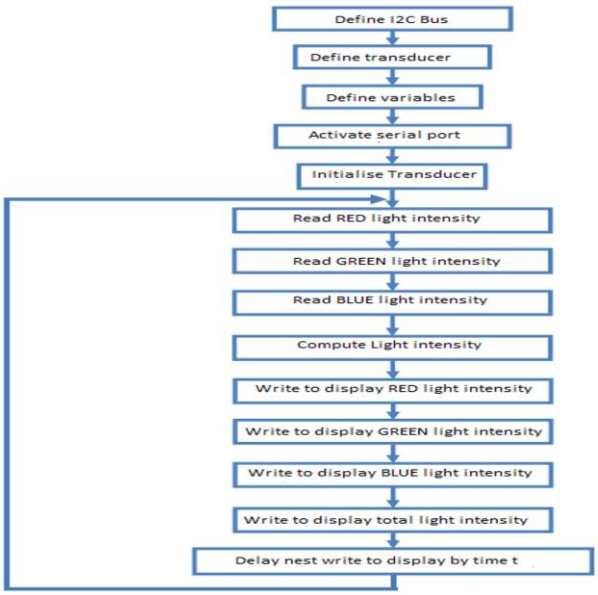
Figure 5.20. ***Principle diagram for application 3*** **(from [2])**

The following connections will be made:

- GND Pin (power) of the board to the GND Pin (LV)
- GND Pin (HV) of the logic level converter to the GND pin of the RGB Light Transducer (it is sufficient, as the GND pins of the converter are already connected to each other)
- 5V Pin (power) of the Arduino to the HV Pin of the converter
- 3.3V Pin (power) of the Arduino to the 3.3V Pin of the transducer
- 3.3V Pin of the transducer to the LV Pin of the converter

- SDA pin of the transducer to the LV2 pin of the converter
- SCL Pin of the Arduino to the HV1 pin of the converter
- SDA pin of the Arduino to the HV2 pin of the converter

Logic scheme and code sequence:




```
#include<Wire.H>
        //importing the I2C bus command library
#include "SparkFunISL29125.h"
        //importing the RGB transducer command library
SFE_ISL29125 RGB_transducer;
        //defining the transducer as ISL29125 type
float Cyr=0.2127
        //defining the Cyr coefficient variable
float Cyg=0.7152
```

```
        //defining the Cyg coefficient variable
float Cyb=0.0722
        //defining the Cyg coefficient variable
float Ev;
//defining the light intensity variable

void setup(){
Serial.begin(115200);
//activate serial gate port with a rate of 115200 baud
RGB_transducer.init();
        //RGB transducer initialization
}
void loop(){
//computing light intensity
Serial.print("Red");
Serial.print(red/257);
/printing the red light intensity value (0…255 domain)
Serial.print("Blue");
Serial.print(blue/257);
//printing the blue light intensity value (0…255 domain)
Serial.print("Green");
Serial.print(green/257);
//printing the green light intensity value (0…255 domain)
Serial.print("Light intensity:");
Serial.print(Ev,0);
Serial.println("lux");
//printing the total light intensity value
Serial.println();
delay(2000);
//2000 ms delay
```

}

//Attention!

Make sure that in the down right corner of the serial display the 115200 baud rate is selected

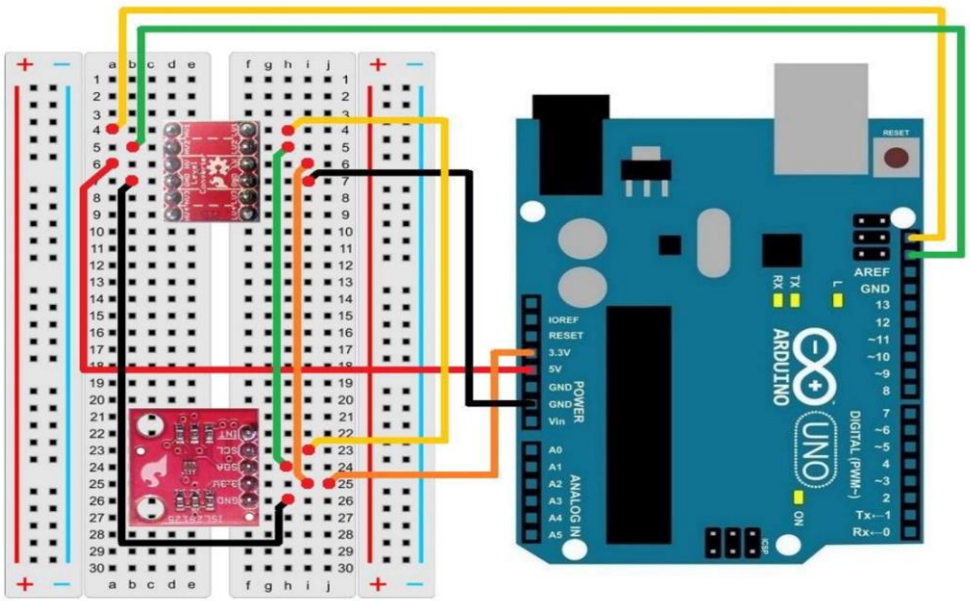


Figure 5.21. ***Electrical connections for application 3*** **(from [2])**

Additional exercises and conclusions

1. Modify the code for application 1 so that it reads the light intensity 10 times in a second and print their average.
2. Modify the code for application 2 so that the light intensity is measures in W/cm*cm
3. Modify the code for application 3 so that the light intensity for each color is printed in the 0…1 domain.
4. Make a single electronic montage that combines the 3 applications and compare the transducer results.

The output characteristic of the transducers is very important for the user, giving information about the dependency between input and output. This characteristic is preferably linear and without medium parameter fluctuations (such as temperature).

To obtain this linearity, compensation actions are necessary. For cases in which the output changes depending on the temperature, thermal compensations are done (components that react inversely with temp are added compared to the uncompensated transducer).

The 3 color RGB base set is used to obtain any colored light in the visible specter. This is done by having 3 light sources, each emitting one base color at different intensities. From a distance the union of these 3 sources will have a specific color.

Another important parameter of color illuminated surfaces is the contrast, which is the difference in color and intensity that makes an object stand out from other objects in the same frame.

BIBLIOGRAPHY

1. Iordache, V., Cormoș, A. 2019. S*enzori, traductoare și achiziții de date cu Arduino Uno*. București: Editura Politehnica Press.
2. Leluțiu, L.M. 2013. *Measuring, data acquisition and processing systems*. Brașov: Editura Universității „Transilvania" din Brașov, ISBN 978-606-19-0304-7
3. Truchsess, W. 2010. „Effects of Varying I2C Pull-Up Resistors." http://dsscircuits.com/articles/effects-of-varying-i2c-pull-up-resistors.html
4. ***. 2003. Everlight Electronics Co., Ltd., „PT15-21C/TR8 - Technical Data Sheet", https://www.tme.eu/ro/details/elpt15-21c/fototranzistori/everlight/pt15-21c-tr8/
5. ***. 2007. Texas Advanced Optoelectronic Solutions Inc., „TSL235R – Light to-frequency converter". https://www.farnell.com/datasheets/323585.pdf
6. ***. 2014. Intersil Americas, „ISL29125 - Datasheet," https://www.renesas.com/en/products/isl29125
7. ***. 2015. „Arduino Playground", http://playground.arduino.cc/CommonTopics/PullUpDownResistor
8. ***. 2015. SparkFun Electronics, „Bi-Directional Logic Level Converter Hookup Guide," https://learn.sparkfun.com/tutorials/bi-directionallogic-level-converter-hookup-guide
9. ***. 2015. SparkFun Electronics, „ISL29125 RGB Light Sensor Hookup Guide", https://learn.sparkfun.com/tutorials/isl29125-rgb-light-sensor-hookupguide
10. ***. 2016. *Arduino Boards,* https://www.arduino.cc/en/hardware/#boards

Paper 6
TRAFFIC LIGHT SYSTEM FOR PEDESTRIAN CROSSING

1. Work Description

1.1. Objectives of the Work

- Creation and testing of the circuit of medium complexity that uses sensors and transducers.
- Development of a practical application that implements a traffic light system for pedestrian crossing.

1.2. Theoretical Description

Introduction

The purpose of the traffic light systems is to give right of way to a specific category of traffic participants. The most vulnerable among them are pedestrians, and the installation of traffic lights at pedestrian crossings is necessary not only at interactions but also in areas with heavy road traffic or where a large number of pedestrians frequently cross the street (shopping centres, schools, etc.). Each traffic light system is characterised by a sequence of phases that, when combined, form a traffic light cycle (any complete sequence of the traffic light signals or the time interval from the display of the green light for one phase until the display of the green light for the next phase). The allocation of time for the traffic light phase must allow pedestrians to cross the street in a single phase, without having to stop or turn back.

The sequence of traffic light phases and the allocation of time for each phase vary depending on the country, the type of pedestrian crossing, its length,

specific local characteristics, etc. An example similar to the traffic light system used in Romania can be found in the following table [1]:

Phase	Vehicles traffic light	Pedestrians traffic light	Duration
A	Green	Red	20-60 seconds
B	Yellow	Red	3 seconds
C	Red	Red	1-3 seconds
D	Red	Green	4-7 seconds (+2 seconds)
E	Red	Green flashing	0-2 seconds
F	Yellow flashing	Green flashing	6-18 seconds
G	Yellow flashing	Red	1-2 seconds

One method to improve safety and traffic flow in the area of pedestrian crossings (by adjusting crossing times, reducing waiting times) is the implementation of traffic light systems where the green signal for pedestrians appears only upon request, thus eliminating pedestrian. The most commonly used solution for receiving a pedestrian request is the use of a push-button. However, various types of detection sensors that do not require any action from pedestrians can also be used, considering that certain categories of pedestrians (such as children, the elderly, or people with disabilities) might have difficulty using push-buttons.

Moreover, based on data provided by the sensors, it is possible to determine when all pedestrians have crossed, allowing for a reduction in the green light duration for pedestrians. These types of sensors can include:

- Pressure sensors installed on the sidewalk before the pedestrian crossing (they can have any shape or color, can distinguish between pedestrians and

other static weights like ice or snow, and use technologies based on inductive loops, easily connectable to traffic management systems), such as SMARTPED and PEDXPAD [4], [7,8].

- Video cameras and pedestrian detection software (such as C-Walk / Safe Walk [4], [7]) that monitor predefined areas and are capable of distinguishing between pedestrians who are crossing, waiting to cross, or approaching the crossing.

Description of applications

The purpose of this application is to create an electronic circuit that controls a traffic light system for a pedestrian crossing, which will include a traffic light for vehicles, a traffic light for pedestrians, and a method for requesting the change of the traffic lights' indications at the request of pedestrians. For this, either a push-button that pedestrians must press or a pressure sensor that detects their presence will be used. The application will also allow for the easy adjustment of the traffic light timing. The sequence of operations can be observed in Figure 6.1.

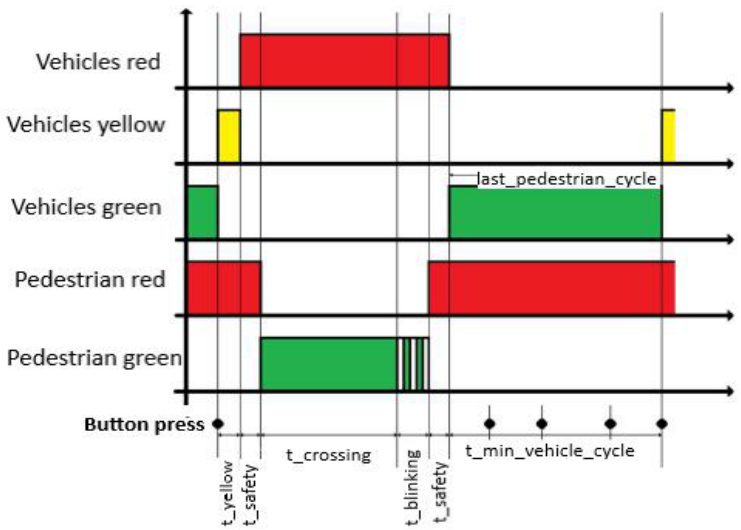


Figure 6.1. ***Succession of traffic light colours*** **(from [1, 2])**

Thus, the traffic light will continuously display green for vehicles and red for pedestrians until a request is made by a pedestrian. At that moment, the green light for vehicles will switch to yellow and then to red.

After some safety time, which is needed so cars can free up the traffic junction, the green pedestrian traffic light will turn on. After the set time passes, the pedestrian traffic light will start blinking for a period of time, and after that it will turn off and the red light will turn on.

After some safety time, which is needed so pedestrians can free up the traffic junction, the green traffic light will turn on.

It is also necessary to implement an imposed minimum traffic signal cycle time for vehicles, so that pedestrian requests are not serviced too often one after the other, leaving too little time for vehicular traffic.

LEDs will be used for the lab application, but the use of relays may allow the control of more powerful light elements powered at higher voltages.

The LEDs and button will be connected to digital ports on the Arduino board, and the pressure sensor will use one of the analog ports.

2. Hardware Components

The electronic components and modules used in the work are those in the following table [1, 2, 3]:

Component or Module	**Characteristics**	**Nr. of parts**	**Image**
Arduino Uno		1	
Breadboard	82×52×10 mm	1	
LED	5 mm, red	2	
LED	5 mm, yellow	1	
LED	5 mm, green	2	
LED	5 mm, white	1	
Resistor	Values need to be calculated	6	
Connecting Wire	Father-Father	1	
Button Module	Button + resistor	1	
Resistive Pressure Transducer Module	FSR 406 +resistor	1	

In this lab work, a breadboard test board will be used to realize the electronic assembly using external components (Figure 6.2 - on the right side are symbolized the electrical connections between pins).

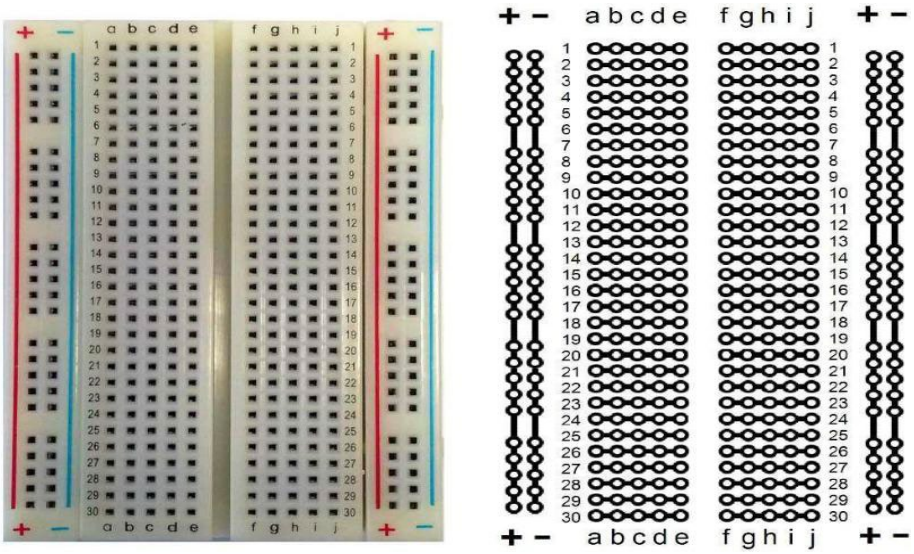


Figure 6.2. ***Breadboard and internal connections*** **(from [1,2])**

The resistor will be mounted in series with the LED, as shown in Figure 6.3, and will be used to limit the current through the LED, as the LED operates at a lower voltage (typically 1.5 - 3 V) than that provided by the Arduino board's digital output port (5 V).

Figure 6.3. ***Using LED current limiting resistor*** **(from [1], [3])**

The formula for calculating the resistance value (applying Ohm's law) is as follows:

$$= (VS – VF) / IF$$

where:

VS = 5 V (voltage provided by the digital output port of the Arduino Uno module)

VF (the voltage on the LED diode in conduction) can be found in the LED diode technical specifications (catalog sheet).

IF (current through the LED diode in conduction) can be found in the LED diode technical specifications (catalog sheet).

If you don't know the manufacturer of an LED diode, you can use a potentiometer instead of a resistor and adjust it until the LED produces the desired illumination, then measure the resistance value and replace the potentiometer with a fixed resistor (Warning! The current through the LED should also be measured in order not to exceed the maximum value that can be provided by the output port of the Arduino board, i.e. 40 mA). Resistive Pressure Transducer Module senses the degree of pressure, relying on the use of a pressure-sensitive resistor [5] FSR 406, the value measured by the Arduino board is available as a digital value ranging from 0 to 1023. It contains a sensor and a pull-down resistor to 0. The pressure sensor is made of three substrates (see Figure 6.4), having a very high resistance between the electrodes (> 10 MΩ) when no pressure is exerted. Increasing the pressure applied to the sensor results in electrical contact between the conductive substrates and thus decreases the resistance value at the sensor terminals (see Figure 6. 5). The value of the resistance depends not only on the applied force but also on the flexibility, dimensions and shape of the object applying the pressure [6].

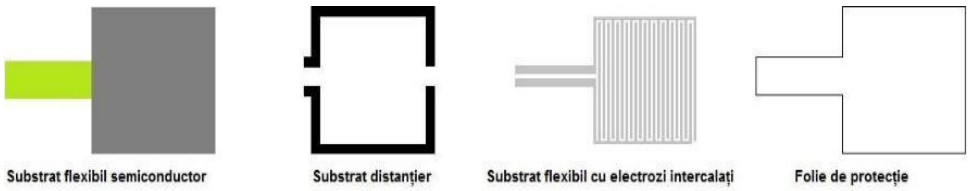


Figure 6.4. ***Internal construction of a pressure sensor*** **(from [1], [5])**

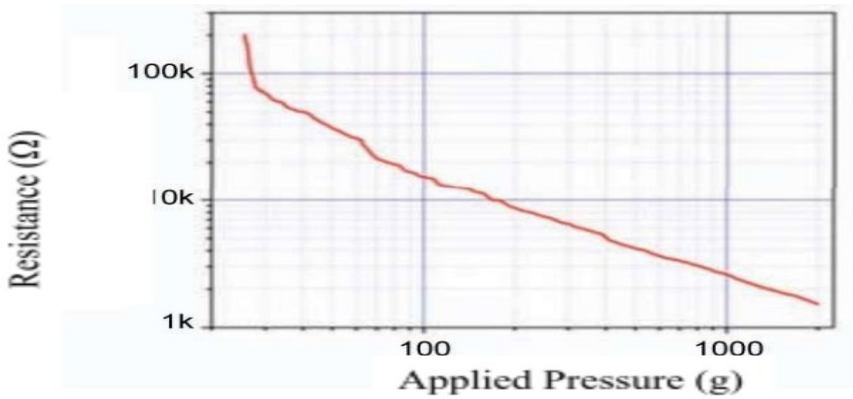


Figure 6.5. ***The Variation of the pressure sensor's resistance in relation to the applied force*** **(from [1])**

The transducer will be supplied with the voltage VCC = 5 V.

The Button Module is used to detect pressing and, in this case, to control the change of the pedestrian traffic light's color. This transducer can also be replaced with any other type of button along with a 10 kΩ resistor, as mentioned in the following paragraph.

Both the resistive pressure transducer module and the button module contain, in addition to the sensor, a resistor connected between the module's output pin (OUT) and ground (GND). This is called a pull-down resistor down [2,7], as you can see in Figure 6.6 and serves to maintain the logical value 0 at

the module's output when no pressure is applied to the sensor or when the button is not pressed.

Establishing a secure logic level (0 in this case) prevents the random appearance of a 0 or 1 value at the Arduino board's digital input due to possible electrical noise [8].

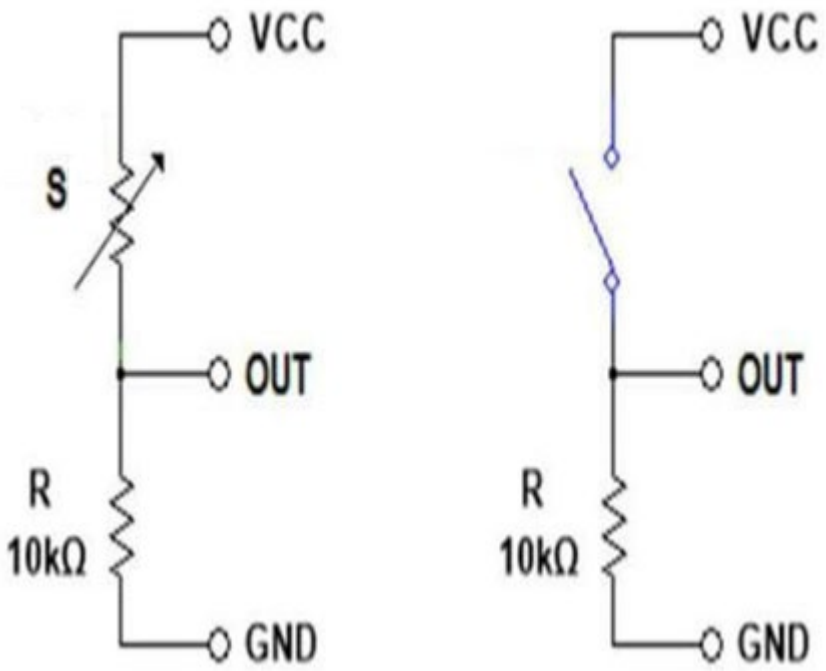


Figure 6.6. ***Use of the pull-down resistor*** **(from [3])**

3. Software Components

int variable = value assigns a value to a signed 16-bit integer variable (ranging from - 32,768 to 32,767).

const indicates a constant, modifying the behavior of a variable. The variable will become Read-only, meaning its value cannot be changed.

unsigned int var

boolean var

long var

void setup() is a function (which does not return data and has no parameters) that runs once at the start of the program. Here, general program preparation instructions are established (pin setup, activation of serial ports, etc.).

void loop() is the main function of the program (which does not return data and has no parameters) and is executed continuously as long as the board is functioning and not reset.

pinMod

output.

millis()

S.begin(baudRate) sets the data transfer rate for the serial port in bits per second (BAUD).

S.print(value or variable, numeral system) prints data as ASCII characters using the serial port.

Serial.println(value or variable, numeral system) prints data as ASCII characters using the serial port, adding a newline after the displayed data.

4. Application 1. Traffic lights system for pedestrian crossings

4.1. Electronic Assembly

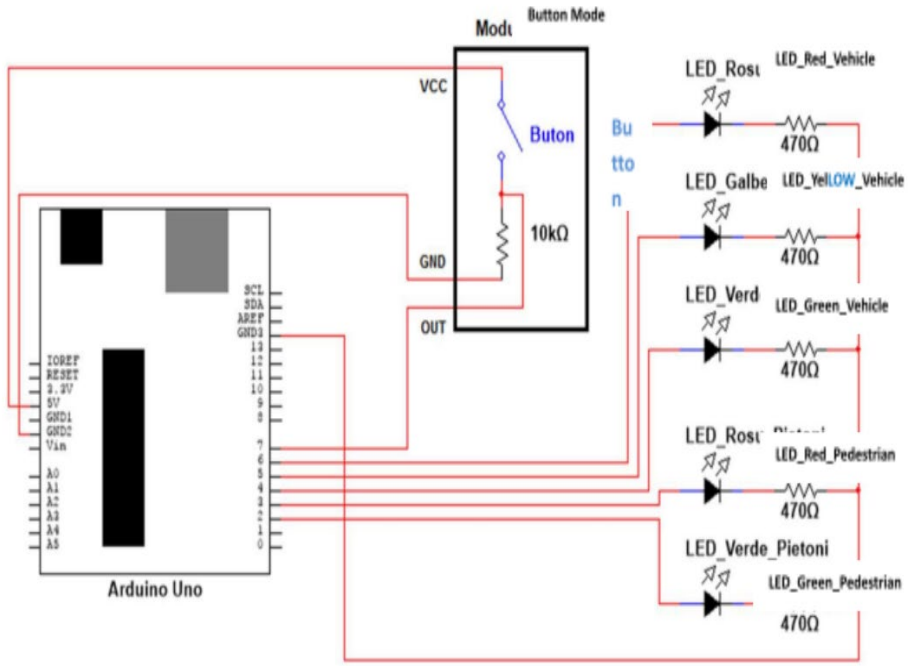


Figure 6.7. ***Diagram for application 1*****(from [1], [7,8])**

The following connections are made:

- Place the LEDs on the breadboard by connecting the anode to the corresponding output of the Arduino board, the cathode to one pin of the current-limiting resistor and its other pin to the GND column marked with "-";
- Digital pin 2 on the board it is connect with a wire to the green LED anode of the pedestrian traffic light;

- Digital pin 3 on the board it is connect with a wire to the anode of the red LED of the pedestrian traffic light;

Application 2. Traffic light system for pedestrian crossing [7, 8]

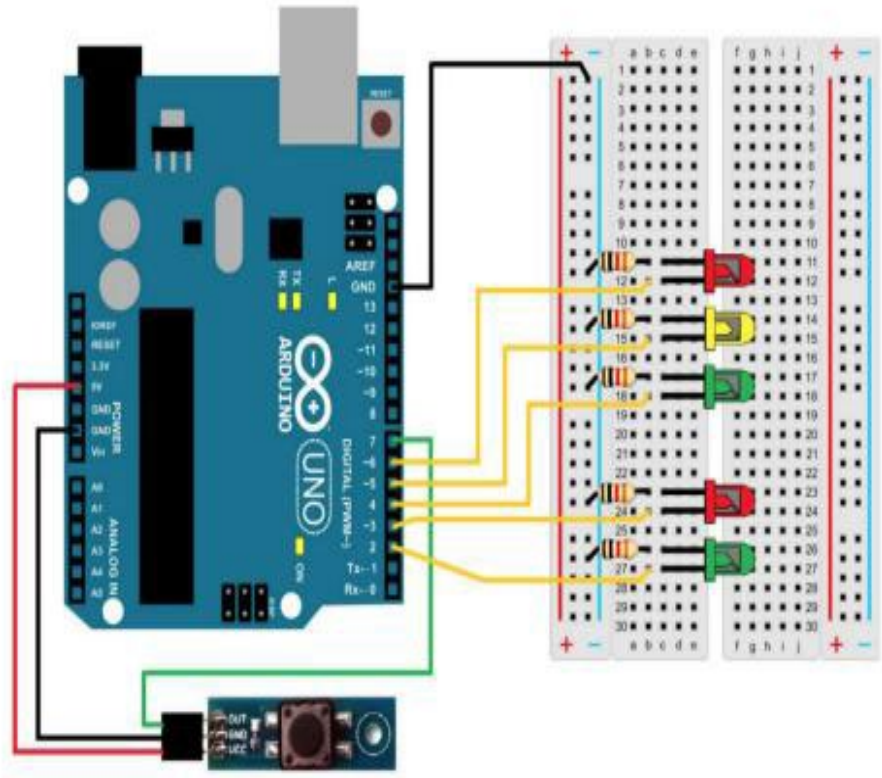
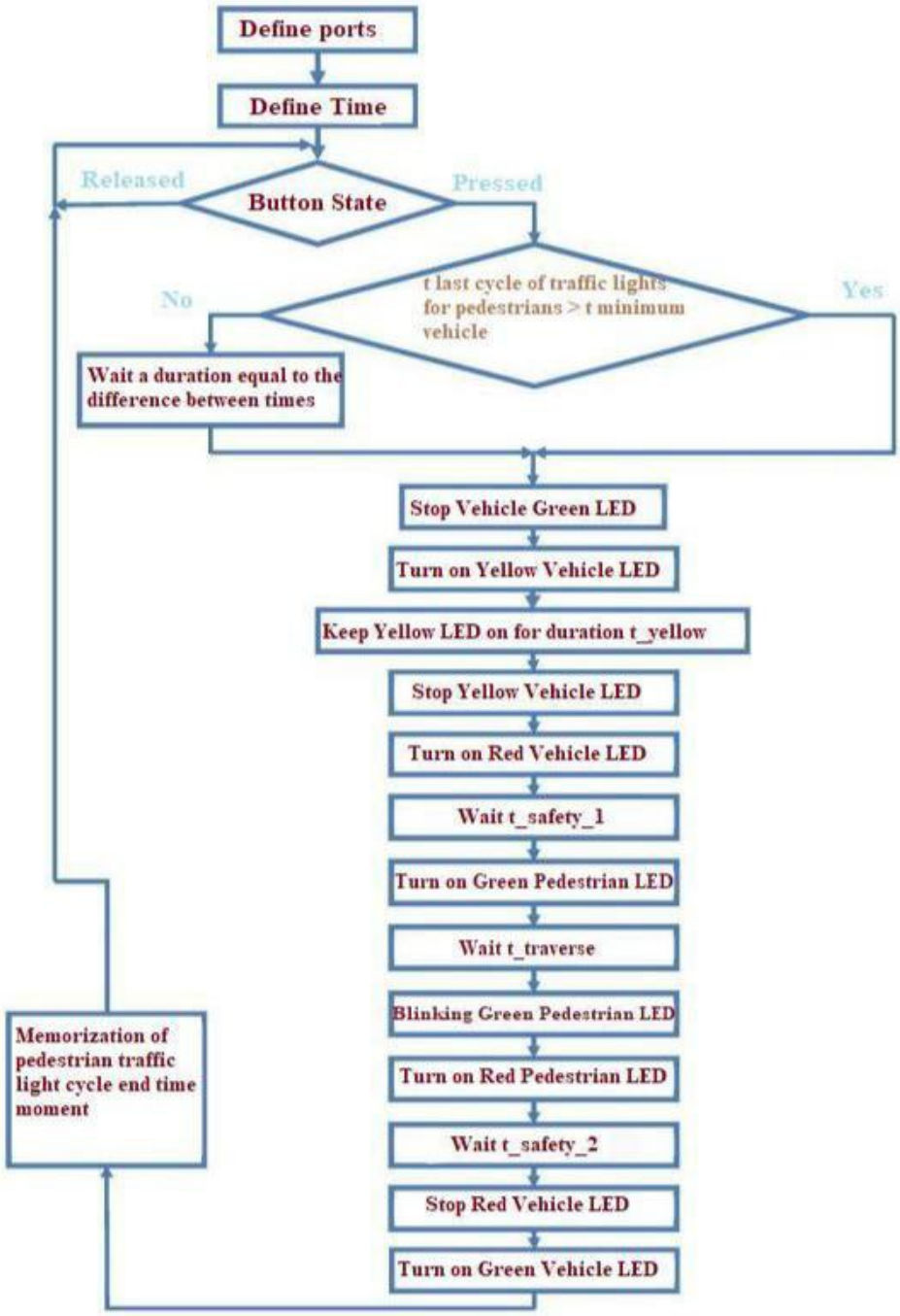


Figure 6.8. ***Electrical connections*** **(from [2, 3])**

4.2. Logical Scheme and code sequence


Define ports
Define Time
Released
Button State
Pressed
t last cycle of traffic lights for pedestrians > t minimum vehicle
No
Yes
Wait a duration equal to the difference between times
Stop Vehicle Green LED
Turn on Yellow Vehicle LED
Keep Yellow LED on for duration t_yellow
Stop Yellow Vehicle LED
Turn on Red Vehicle LED
Wait t_safety_1
Turn on Green Pedestrian LED
Wait t_traverse
Blinking Green Pedestrian LED
Turn on Red Pedestrian LED
Wait t_safety_2
Stop Red Vehicle LED
Turn on Green Vehicle LED
Memorization of pedestrian traffic light cycle end time moment

```
const int led_green_pedestrian = 2;

// define the variable led_green_pedestrian corresponding to digital port 2 where the green pedestrian LED anode will be connected

const int led_red_pedestrian = 3;

// defining the variable led_red_pedestrian corresponding to digital port 3 where the red pedestrian LED anode will be connected

const int led_green_vehicle = 4;

// define the variable led_green_vehicle corresponding to digital port 4 where the anode of the green LED for vehicles will be connected

const int led_yellow_vehicle = 5;

// definition of the variable led_yellow_vehicle corresponding to digital port 5 where the anode of the yellow LED for vehicles will be connected

const int led_red_vehicle = 6;
```

```
// defining the variable led_red_vehicle corresponding to digital port 6 where the anode of the red LED for vehicles will be connected

const int button = 7;

// defining the button variable corresponding to digital port 7 where the OUT pin of the button mode will be connected
const int t_yellow = 2000;
// definition of the yellow time which will have a value of 2 s

const int t_safety_1 = 2000;

// define safety time 1 to be 2 seconds
const int t_safety_2 = 2000;

 // definition of safety time 2 which will have a value of 2 seconds const int t_traverse = 4000;
    // defining the crossing time to be 4 seconds



const int t_blinking = 400;

 // defining the crossing time to be 4 seconds unsigned long t_min_cycle_vehicle = 10000;
  //defining the minimum time imposed for a traffic light cycle for vehicles

unsigned long t_last_cycle_pedestrians = 0;
```

```
 definition of the initial value corresponding to the time moment of the last
pedestrian signal cycle


      // the button pin is declared as input

      digitalWrite(led_green_vehicle, HIGH);

     // the led_green_vehicle lights up (initial state)

     digitalWrite(led_red_pedestrian, HIGH);

     //the led_red_pedestrian lights up (initial state)

}

void loop()

{
boolean state_button = digitalRead(button);

 declare that the boolean variable statusButton takes the logical value of the
button pin

if (state_button == HIGH && millis() - t_last_cycle_pedestrians <
t_min_cycle_vehicle)
```

```
// if the button status is logical 1 (button pressed) AND the elapsed time since the end of the last pedestrian traffic light cycle is less than the minimum time required for a vehicle traffic light cycle
    delay(t_min_cycle_vehicle - (millis() - t_last_cycle_pedestrians)

);

// then waits for a time equal to the difference between the two times initialize_cycle_peetons(); //and then executes the function initialize_cycle_peetons}

if (state_button == HIGH && millis() - t_last_cycle_pedestrians > t_min_cycle_vehicle)

{
  //if the button status is logic 1 (button pressed) AND the elapsed time since the end of the last pedestrian traffic signal cycle is greater than the minimum time required for a vehicle traffic signal cycle

pedestrian_cycle_initiator();

// then execute the function pedestrian_cycle_initiator
}

void pedestrian_cycle_initiator()
{

// definition of the pedestrian_cycle_initiator function
```

```
digitalWrite(led_green_vehicle, LOW);

// write logic 0 to the led_vehicle_green_led pin (turn off led_vehicle_green_led)



digitalWrite(led_yellow_vehicle, HIGH);

//                    write logic 1 to the led_yellow_vehicle pin (turns on led_yellow_vehicle)

delay(t_yellow);

// keeps led_yellow_auto on for a period equal to the yellow time

digitalWrite(led_yellow_vehicle, LOW);
//                    rite logic 0 to the led_vehicle_yellow_led pin (turn off led_vehicle_yellow_led)

digitalWrite(led_red_vehicle, HIGH);

  write logic value 1 to the led_ros_vehicle pin (turns  on led_ros_vehicle)

delay(t_safety_1);

waits for safety time 1, until the green_pedal_LED turns on
```

```
digitalWrite(led_red_pedestrian, LOW);

// write logic 0 to the led_ros_off pin (turns off led_ros_off)

digitalWrite(led_green_pedestrian, HIGH);

write the logic 1 value to the led_green_light pin (turns on the led_green_light)

delay(t_traverse);

keeps the green_pedestrian_light on for a period equal to the crossing time

for ( )

execute the contents of the for loop 5 times
{
digitalWrite(led_green_pedestrian, LOW);

// write the logic 0 value to the led_green_pedestrian pin (turn off led_green_pedestrian)

delay(t_blinking);
// keeps the green_pedestrian_LED on for a time t_blinking

digitalWrite(led_green_pedestrian, HIGH);
//                    write the logic 1 value to the led_green_light pin (turns on the led_green_light)

delay(t_blinking);
```

```
// keeps the green_pedestrian_LED off for a time t_blinking

}
digitalWrite(led_green_pedestrian, LOW);

write the logic 0 value to the led_green_pedestrian pin (turn off
led_green_pedestrian)

digitalWrite(led_red_pedestrian, HIGH);
write the logic 1 value to the led_red_pedestrian pin (turns on
led_red_pedestrian)

delay(t_safety_2);
waits for safety time 1, until the green_auto_led_green_light
turns on

digitalWrite(led_red_vehicle, LOW);

write logic 0 to the led_red_vehicle pin (turn off led_red_vehicle)

digitalWrite(led_green_vehicle, HIGH);

//writes logic value 1 to the led_green_vehicle pin (turns on led_green_vehicle)
t_last_cycle_pedestrians = millis();
// the time of the end of the last pedestrian traffic light cycle is memorized

}
```

Additional exercises and conclusions

The logic or Boolean value is of type 0 or 1 (False or True) and it is transmitted between the various electronic components and equipment by means of digital electrical signals (voltage level 0 V means logic 0 and voltage level +5 V means logic 1). It is very important to understand how to represent different variables in different environments.

Thus, a Boolean variable can be represented on the screen with the characters 0 or 1 or with the strings TRUE or FALSE, while for the same variables transmitted to logic gates or digital inputs an electrical signal with amplitude level will be used variable according to the logic value transmitted.

Another particularly important aspect for the design and realization of circuits or equipment that operate within systems that are directly related to traffic safety (traffic lights, for example) is their design to achieve a certain degree of reliability and, in case of failure, to ensure the transition from false response of the circuit or equipment to erroneous response (three states that a circuit can have will be considered:

- normal operating state - when the circuit response is as designed;
- faulty condition with false response - when the faulty circuit allows commands to be made that lead to accidents - for example, green antagonist at traffic lights;
- and the faulty condition with erroneous response - when the faulty circuit leads to traffic congestion and increased waiting times - e.g. flashing yellow at traffic lights).

The analog inputs of the Arduino development board take the signal from the modules connected to it and transmit them to the analog-to-digital converter, further the signals acquired by the development board are digitally processed by its components.

When pedestrians press the button, a request must be sent to allow pedestrians to pass. The request must be analysed by the system in the context of ensuring a normal flow for both vehicles and pedestrians.

The pedestrian traffic light will not go into the permissive state as soon as the pedestrian presses the button but, depending on the traffic light phases at that moment, after a time resulting from the execution of the request handling algorithm so that the two flows, vehicles and pedestrians, are running under normal conditions.

Another important aspect is system calibration, i.e. the determination and adaptation of thresholds and levels of electrical signals to define certain states of the system or its components. For example, the electrical parameters of the pressure transducer module may change over time, requiring the system to be calibrated to the new values.

BIBLIOGRAPHY

1. Iordache, V., Cormoș, A. 2019. *Senzori, traductoare și achiziții de date cu Arduino Uno.* București: Editura Politehnica Press.
2. McRoberts, Michael. 2013. *Beginning Arduino*, 2nd edition. München, Oldenburg: Apress, https://www.oreilly.com/library/ view/beginning-arduino-second/9781430250166/
3. ***. 2015. "Arduino Playground. *Pull-Up and Pull-Down Resistors*", http://playground.arduino.cc/CommonTopics/PullUpDownResistor
4. ***. 2015. "FLIR Systems", http://www.flir.fr/
5. ***. 2015. "Interlink Electronics. FSR 400 Series", http://www.interlinkelectronics.com/datasheets/Datasheet_FSR.pdf
6. ***. 2015. Newparts. *Caracteristicile intersecțiilor*, http://www.newparts.info/2013/05/caracteristicile-intersesctiilor.html
7. ***. 2015. SmartPed. *Pedestrian Detection,* https://smartcitystreets. com/pedestrian-safety/
8. ***. 2015. "Traffic Safety Corp. *Pedestrian Crossings*", https://xwalk.com/product-categories/all-signs/pedestrian-signs/ts40-pedestrian-flashing-led-edge-lit-sign/

Paper 7

MEASUREMENT OF VOLTAGE AND CURRENT INTENSITY

1. Work Description

1.1 . Objectives of the Work

- Creating and testing circuits of medium complexity using sensors and transducers.
- Using shield-type electronic modules.
- Developing a practical application for measuring voltage and current values (in a DC circuit), calculating power and consumed energy, and displaying them on an LCD screen.

1.2. Theoretical Description

Introduction

Electric current is the directed movement of electric charges that occurs in an electric circuit consisting of an electric generator, conductors, and loads.

Electric current is characterized by several physical quantities, such as:

- Electric voltage, denoted by U, with the unit of measurement volt (V), represents the potential difference between two points in an electric circuit and is proportional to the mechanical work performed by the electric generator to move an electric charge from one point to another.
- Electric current intensity, denoted by I, with the unit of measurement ampere (A), measures the electric charge passing through a conductor's cross-section per unit time.
- Electric power, denoted by P, with the unit of measurement watt (W),

represents the energy supplied by an electric generator in a unit of time and is calculated by the formula $P = W/t = U \cdot I$.

- Electric energy, denote by E and measured in watt-hours (Wh), represents the mechanical work required to transport an electric charge q through a section of a circuit over a period of time, and it is calculated by the formula $E = U \cdot I \cdot t = P \cdot t$.

Application Description

The purpose of this application is to create an electronic circuit that measures the voltage applied to a load (an incandescent bulb in the presented application) as well as the current intensity consumed by it, perform the calculation of power and consumed energy, and display them numerically on an LCD screen.

The measurement of electrical voltage using the Arduino Uno board will be done by applying the voltage to be measured to one of the analog inputs since the board contains the analog/digital converter necessary for converting the analog physical quantity into a digital one. For a variation between 0 V and 5 V applied to the analog input, the Arduino board provides a digital value between 0 and 1023. When the analog voltage can exceed the value of 5 V (as will be the case in this work), a voltage divider is necessary to prevent damage to the microcontroller. The measurement of electric current intensity using the Arduino Uno board will be done with the help of a dedicated transducer for measuring it, based on the use of a Hall magnetic sensor.

The Hall effect (Figure 7.1) consists of the appearance of a transverse electric field and a potential difference in a semiconductor traversed by an electric current when it is introduced into a magnetic field perpendicular to the current direction. The electric current flowing through the semiconductor material is influenced by the magnetic field, therefore, the

output voltage of the sensor will be directly proportional to the intensity of the magnetic field.

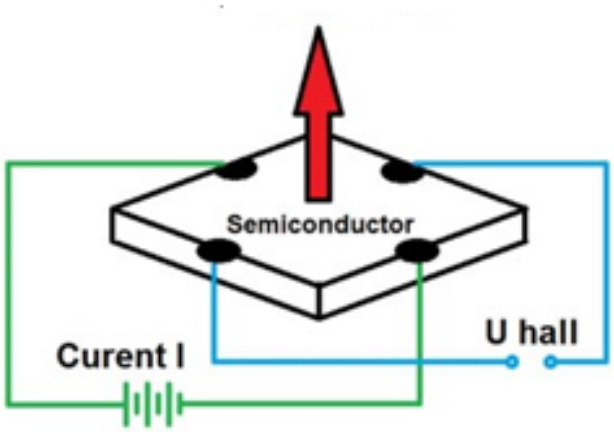


Figure 7.1. ***Hall Effect*** **(from [1,2])**

On the surface of the transducer, there is a copper conductor through which the electric current to be measured passes, generating a magnetic field sensed by the sensor and transformed into a proportional electric voltage (voltage provided at the transducer output) [1, 2]. Based on the variation of the voltage at the transducer output (between 0 and 5 V), applied to one of the analog ports, the Arduino board provides a digital value ranging from 0 to 1023.

2. Hardware Components

The electronic components and modules used in the project are listed in the following table: [1, 2, 3].

Semi-adjustable Resistor	2 kΩ	1	
Adjustable Resistor		1	
Bulb	12V, 35W	1	
Connecting wire	With and without connectors	4	
Connecting thread	Male-Male	6	
Current transducer	ACS711	1	
Power supply	35V, 5A	1	

In this work, for assembling the electronic circuit using external components, a breadboard will be used (Figure 7.2 - the electrical connections between pins are symbolized on the right side).

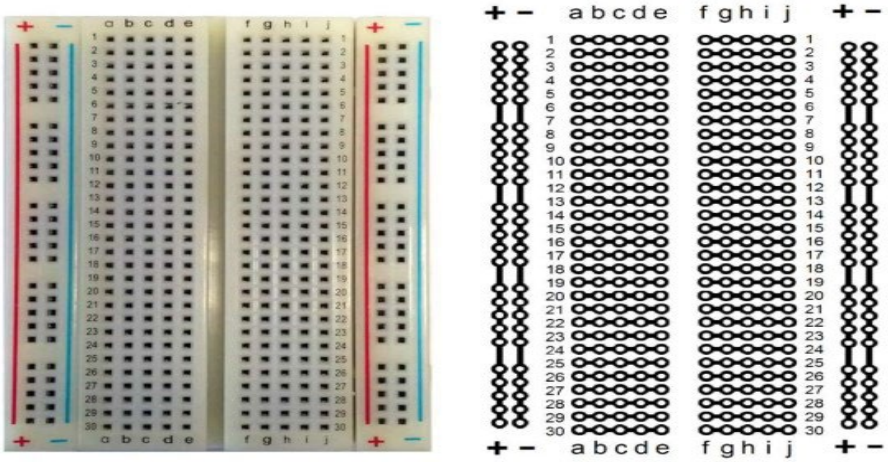


Figure 7.2. ***Breadboard and internal connections*** **(from [1]s)**

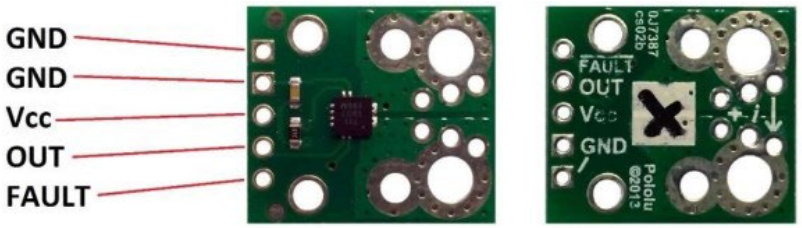


Figure 7.3. ***Current transducer*** **(from [1,2])**

The current transducer [3] measures the intensity of the current absorbed by the load, based on the use of a linear Hall magnetic sensor (ACS711EX). The transducer can measure electric currents with intensities between -15.5 A and 15.5 A, having an internal resistance of approximately 0.6 mΩ and electrical insulation for voltages up to 100 V. The output voltage at rest (the current is 0 A) is Vcc/2, i.e., 2.5 V. However, this value can vary depending on the actual value of the applied Vcc and the thermal drift of the sensor (±5%), see Figure 7.4.

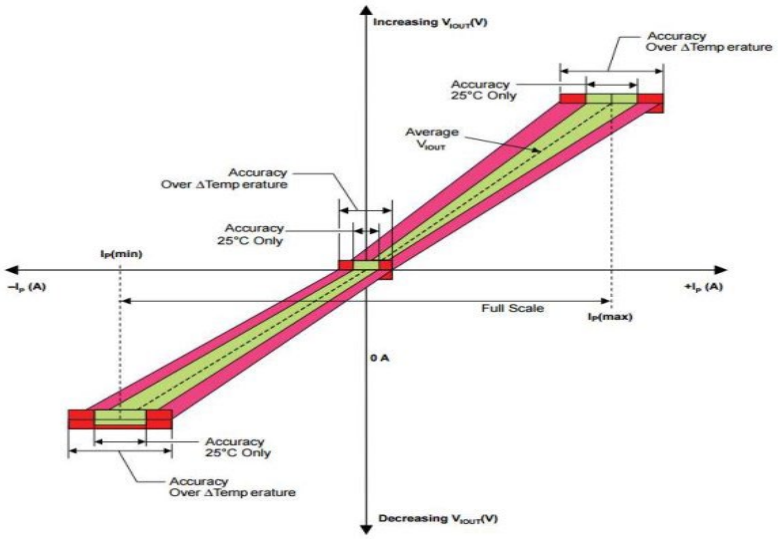


Figure 7.4. ***Output voltage variation depending on current for the ACS711EX sensor*** **(from [2])**

The electronic schematic of the current transducer can be seen in Figure 5.

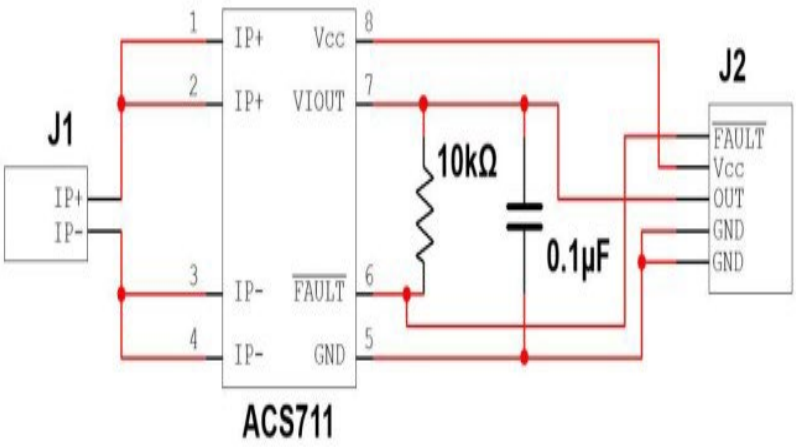


Figure 7.5. ***Electronic schematic of the transducer*** **(from [1, 2])**

The transducer will be powered with the voltage VCC = 5

Measuring current using the transducer

For a current flowing through the transducer between -15.5 A and 15.5 A, it will provide at the output (at the OUT terminal) a voltage between 0 and Vcc (5V). The voltage is read by the analog port of the Arduino board, and it provides a digital value (denoted val_dig_I) in the range of 0 – 1023, corresponding to a current intensity range between -15.5 A and 15.5 A.

Therefore, when the current intensity is 0, val_dig_I will have the value 512 (we will denote this value as val_dig_I0). Thus, for positive electric currents, the range of values will be 512 ÷ 1023 (val_dig_I0 ÷ val_dig_Imax). It follows that the measurement resolution will be 15.5 A / 512 values, approximately 0.03 A/value.

To calculate the current intensity, we will refer to the value corresponding to a current of 1 A, val_dig_1A (to be able to later perform calibration). The LCD shield allows displaying characters on a liquid crystal display with LED backlighting. It is mounted over the Arduino board and has connectors in such a way that the board's pins remain accessible. The LCD screen consists of 2 lines of 16 characters, each character being composed of 5x8 pixels. The columns (characters) are numbered from 0 to 15 (from left to right), and the rows are numbered from 0 to 1 (from top to bottom). To function, the shield uses the digital pins of the Arduino board from 2 to 7 as follows: pin 2 - d7, pin 3 - d6, pin 4 - d5, pin 5 - d4, pin 6 - enable, pin 7 - rs.

The voltage divider (Figura 7.6) is used to measure the electrical voltage and is necessary because both the voltage supplied by the laboratory power supply (35 V) and the maximum operating voltage of the bulb (12 V) exceed the maximum voltage supported by the analog inputs (5 V) of the Arduino Uno board.

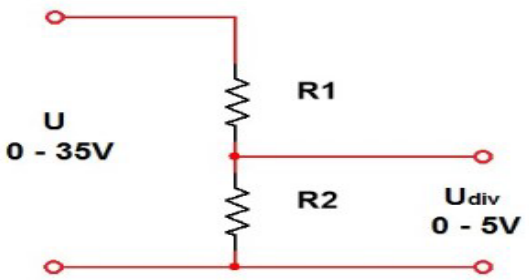


Figure 7.6. ***Voltage divider schematic*** **(from [1])**

The power supply will be adjusted so that it cannot provide more than 12 V (by limiting the current supplied to approximately 3 A, necessary for the bulb to operate at maximum intensity). However, for the protection of the Arduino board against accidental power supply disturbances, the voltage divider will be calculated to provide a maximum voltage of 5 V (Vcc) at the output for a voltage applied to the input terminals of 35 V.

The current absorbed by the analog input is negligible, therefore we can consider that the currents through R1 and R2 are equal. A small value is chosen, of the order of mA. For the following calculations, Idiv = 3 mA was chosen.

$$R_1 + R_2 = \frac{U}{I_{div}} = \frac{35\ V}{3 \cdot 10^{-3}\ A} = 11{,}67\ k\Omega \tag{7.1}$$

Standard resistors R1 = 10 kΩ and R2 = 1.8 kΩ are chosen, with a tolerance of 5%. Due to the tolerance range, the actual values of the resistors can vary as follows: R1 between 9.5 kΩ and 10.5 kΩ, R2 between 1.71 kΩ and 1.89 kΩ. To ensure that the divider will not supply a voltage greater than Vcc on resistor R2, we recalculate the resistance of R1 for the highest value of R2, which is 1.89 kΩ.

$$U_{div-max} = V_{cc} = U\frac{R_2}{R_1 + R_2} \Longrightarrow R_1 = \\ = \frac{(35V \cdot 1{,}89k\Omega - (5V \cdot 1{,}98k\Omega)}{5V} = 11{,}34\ k\Omega \tag{7.2}$$

Considering the situation where the chosen R_1 has the lowest value, i.e., 9.5 kΩ, it is necessary to add an additional resistor to compensate for the difference of 11.34 kΩ - 9.5 kΩ = 1.84 kΩ. Therefore, the solution is to insert a semi-adjustable resistor R with a standard value of 2 kΩ in series with resistor R1.

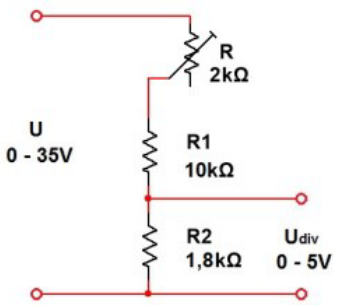


Figure 7.7. ***Final schematic of the voltage divider*** **(from [1])**

Measuring voltage using the voltage divider

For a voltage applied to the divider between 0 and 35 V (power supply), it will provide an output voltage (across resistor R2) between 0 and Vcc (5 V). This voltage is read by the analog port of the Arduino board, which provides a value (denoted as val_dig_U) in the range of 0 – 1023.

Thus, the measurement resolution will be 35V / 1024 values = 34.18 mV/value.

The voltage provided by the voltage divider is determined as follows:

$$\frac{V_{cc}}{1023} = \frac{U_{div}}{val\_dig\_U} \Rightarrow U_{div} = \frac{V_{cc} \cdot val\_dig\_U}{1023} \quad (7.3)$$

The voltage applied to the divider (which is desired to be measured) can also be determined as follows:

$$\frac{U}{U_{div}} = \frac{35}{V_{cc}} \Rightarrow U = U_{div}\frac{35}{V_{cc}} \quad (7.4)$$

Increasing the measurement resolution of voltage

Each analog input port uses an analog-to-digital converter to transform the analog voltage read into a digital value. The resolution of the converter depends on the number of bits used to describe the digital value. Thus, the Arduino board uses 10 bits, so for a voltage input between 0 – 5 V, it will provide 1024 digital values (between 0 – 1023), with a resolution of approximately 5 V / 1024 = 4.88 mV/value. To determine the value of the analog input voltage, it is compared by the converter with a reference voltage. The value of the reference voltage is equal to V_{cc}, i.e., 5 V. In the case of the presented application, the maximum voltage that can be measured is 35 V, but the actual maximum voltage measured cannot exceed 12 V due to the load. Instead of using the entire range of digital values 0 – 1023 for voltages between 0 – 5 V, we can use it to describe the range 0 – 1.71 V. This can be achieved by providing an external reference voltage, which has a value of 1.71 V. Thus, the measurement resolution becomes 1.71 V / 1024 = 1.67 mV/value. An external reference voltage can be applied to the Arduino board on pin AREF (*Analog REFerence*). The voltage can come from an external power source or from an internal source (3.3 V or 5 V) using a voltage divider to lower it. Do not use external reference voltages lower than 0 V or higher than 5 V on pin AREF!

In the case of the presented application, a semi-adjustable resistor (which can be defined as an adjustable resistive divider) will be used instead of two fixed resistors to accurately set the reference voltage value using a measuring device, as shown in Figure 7. 10.

void setup() is a function (that does not return data and has no parameters) that runs only once at the beginning of the program. Here, general program setup instructions are established (setting pins, activating serial ports, etc.).
void loop() is the main function of the program (that does not return data and has no parameters) and is executed continuously as long as the board is operating and not reset.
pinMode(*pin*, *mode*) configures the specified digital pin as input or output.
lcd.begin(columns, rows) initializes the interface with the LCD screen and specifies the number of rows and columns.
Serial.begin(*baudrate*) sets the data transfer rate for the serial port in bits per second (BAUD).
if(*condition*) {*instruction(s)*} else {*instruction(s)*} tests whether a condition is true or not.
for(*initialization, condition, increment*) {*instruction(s)*} repeats a block of instructions until the condition is met.
analogRead(*pin*) reads the value of the specified analog pin.
delay(*ms*) pauses the program for a specified duration in milliseconds.
millis() is a function that returns the number of milliseconds elapsed since the program started execution.
Serial.println(*value or variable, number system*) prints data as ASCII characters using the serial port.
lcd.setCursor(*column, row*) sets the position of the LCD cursor. For the LCD used in this application, the number of columns ranges from 0 to 15, and the number of rows ranges from 0 to 1.

lcd.clear() clears the LCD screen and positions the cursor in the upper-left corner.

lcd.print() displays data (values of variables)/text on the LCD screen within parentheses. To display text, it must be placed between quotation marks ("text"). To display the value of a variable of type char, byte, int, long, or string, write the variable name and, optionally, its numbering base (variable, BIN or DEC or OCT or HEX). To display the value of a variable of type float or double, write the variable name followed by the number of decimal places to be displayed after the comma (variable, number of decimals).

analogReference(EXTERNAL) sets the reference voltage used for analog inputs to be the one provided by an external source.

++ is used to increment a variable

4. Application 1. Measuring Voltage and Current Intensity

4.1. Electronic Assembly

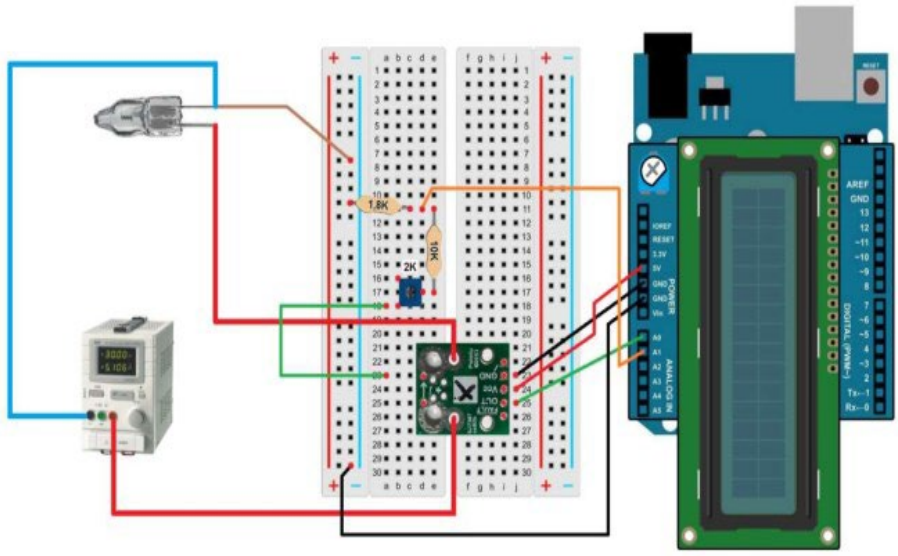


Figure 7.8. ***Schematic diagram for application 1*** **(from [1, 2, 3])**

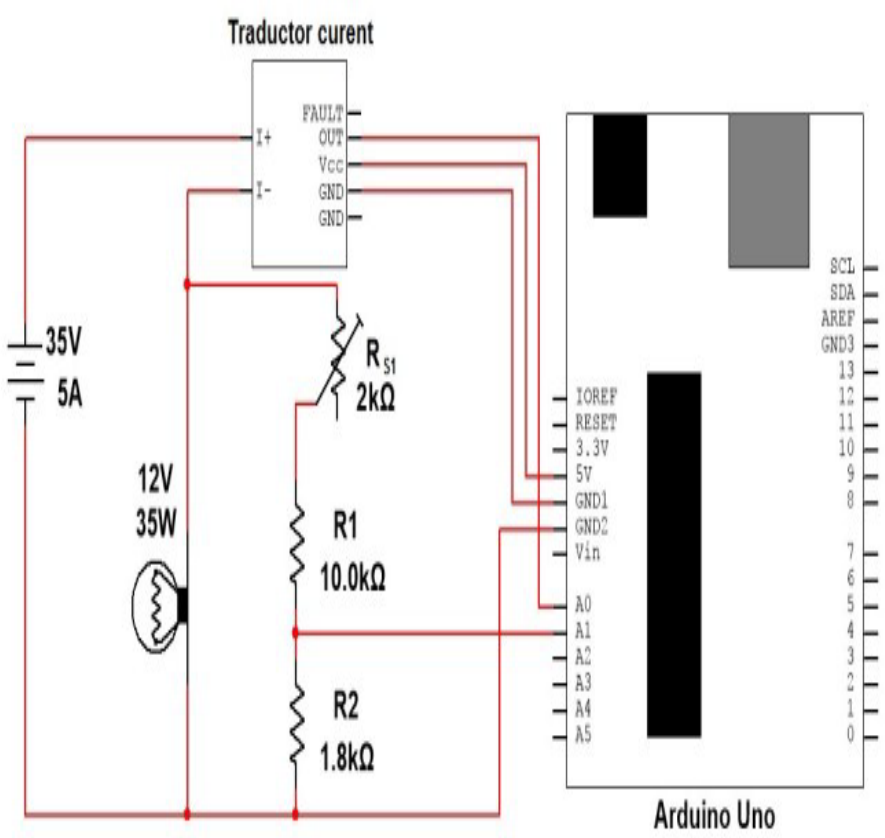


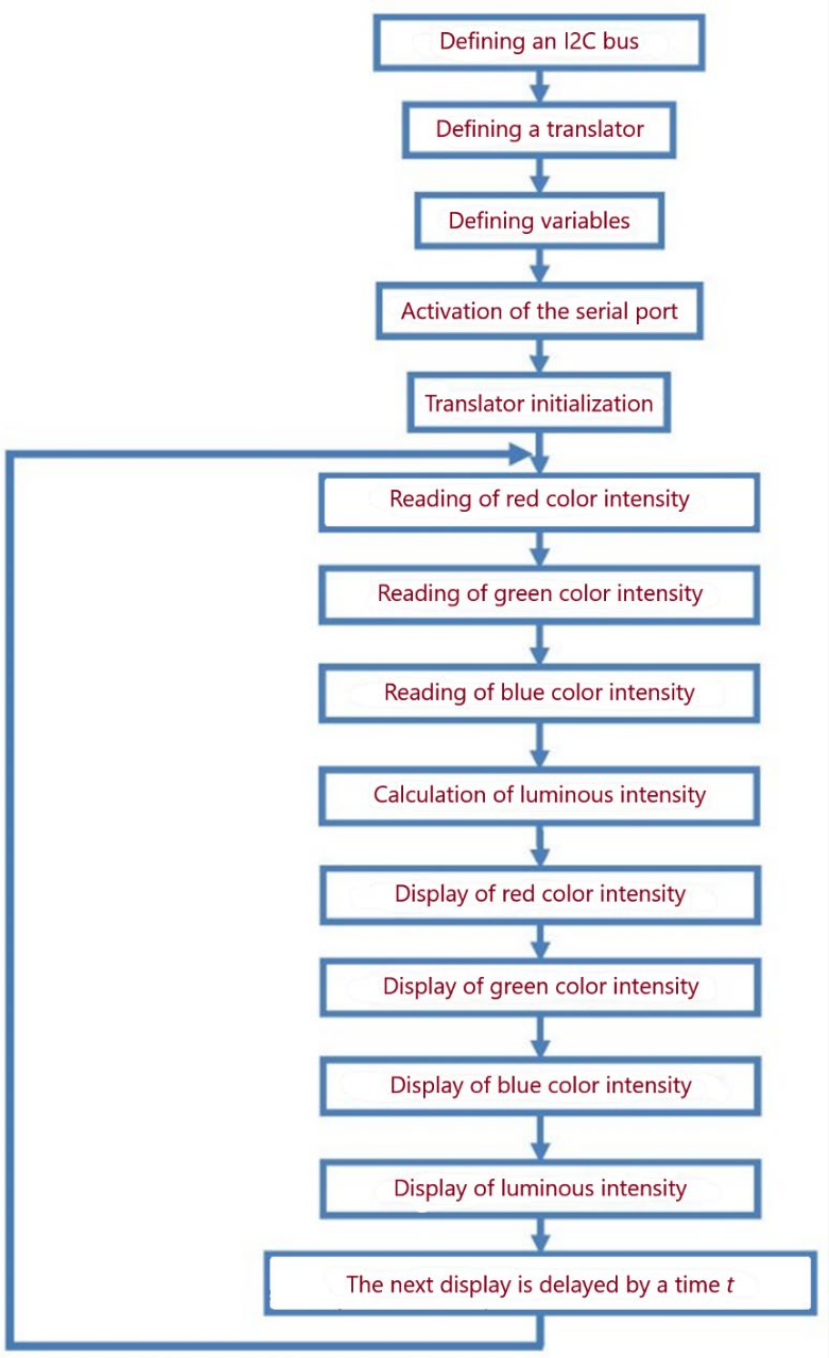
Figure 7.9. ***Electrical connections for application 1*** **(from [1])**

Schematic diagram for application 1

The following connections are made:

- Place the current sensor on the breadboard by connecting the input pins for the current to be measured on column d and the output and power pins on column i;
- Place the resistors on the breadboard according to the schematic diagram;
- Connect the negative pin of the bulb with a wire to the "-" rail of the breadboard;
- Connect analog pin A1 on the Arduino board with a wire to resistors R1 and R2;
- Connect analog pin A0 on the Arduino board with a wire to the OUT pin of the current sensor;
- Connect the GND (power) pin on the Arduino board with a wire to the GND pin of the current sensor;
- Connect the 5V (power) pin on the Arduino board with a wire to the Vcc pin of the current sensor;
- Connect the GND (power) pin on the Arduino board with a wire to the “-“ rail of the breadboard..Verify the correct and secure connection of the 1.8 kΩ resistor and the wire (brown in the above diagram) between the negative side of the bulb and the “-“ rail of the breadboard. Any error in this regard may result in a voltage higher than 5V on pin A1 and may damage the microcontroller.

4.2. Logical Scheme and Code Sequence



```
#include <LiquidCrystal.h>
//Including the LCD shield command library
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);
//Initializing the library and the lcd variable with the pin numbers used by the LCD shield
const int currentInput = 0;
//Defining the currentInput variable corresponding to the analog port A0 where the OUT pin of the current sensor will be connected
const int voltageInput = 1;
//Defining the voltageInput variable corresponding to the analog port A1 where the output of the resistive divider will be connected
float Udiv = 0.0;
//Defining the voltage supplied by the resistive divider
float I = 0.0;
//Defining the electric current variable float U = 0.0;
//Defining the electric voltage variable float E = 0.0;
//Defining the electric energy variable float Etot = 0.0;
//Defining the total electric energy variable float P = 0.0;
//Defining the electric power variable
unsigned long val_dig_I = 0;
//Defining the val_dig_I variable that will contain the value read from the analog port
unsigned long val_dig_U = 0;
//Defining the val_dig_U variable that will contain the value read from the analog port
unsigned long timp = 0;
//Defining the time variable with the initial value 0
float Vcc = 5.0;
//Defining the Vcc variable with the initial value 5V
```

```
        float Uref = 5.0;
         //Defining the Vref variable with the initial value 5V
        float val_dig_I0 = 512;
//Defining the val_dig_I0 variable corresponding to the initial median value provided by the current sensor
const int val_dig_1A = 545;
//Defining the val_dig_1A variable corresponding to the initial value provided by the current sensor when a current of 1A flows
void setu(){ lcd.begin(16, 2);

//Initializing the interface with the LCD screen and specifying the number of rows and columns
Serial.begin(9600)
 //Activate serial port output with a baud rate of 9600 baud
  void loop(){
val_dig_U = analogRead(voltageInput);
            //The variable val_dig_U takes the value read from analog port 1
        /*for (int i=0;i<500;i++) {
val_dig_U = val_dig_U + analogRead(voltageInput); val_dig_I = val_dig_I + analogRead(currentInput); delay(1);
}
val_dig_U = val_dig_U / 500; val_dig_I = val_dig_I / 500;*/
Udiv = (val_dig_U * Uref) / 1023.0;
//Calculate the voltage collected from the terminals of the voltage divider, depending on the value val_dig_U read from analog port 1 – formula (6)
U = Udiv * (35 / Vcc);
//Calculate the voltage applied to the terminals of the voltage divider (voltage applied to the bulb) – formula (7)
val_dig_I = analogRead(currentInput)
```

```
//The variable val_I takes the value read from analog port 0
I = (val_dig_I - val_dig_I0) / (val_dig_1A - val_dig_I0);
//Calculate the current in A, depending on the value val_dig_1A corresponding to a current of 1A – formula (2)
P = U * I;
//Calculate the power in W
E = (P * (millis() - timp)) / 3600000;
//Calculate the energy consumed during the last loop
timp = millis();
//Update the time reference Etot = Etot + E;
//Calculate the total consumed energy Serial.println(val_dig_I);
//Print the value read from analog port 0 on the serial monitor (necessary for calibration)
lcd.clear();
//Clear the LCD screen and position the cursor in the top-left corner
lcd.print("U=");
//Display the text between the quotation marks on the LCD screen
lcd.print(U,1);
//Display the value of U on the LCD screen with one decimal place
lcd.print("V");
  //Display the text between the quotation marks on the LCD screen
lcd.print(" I=");
 //Display the text between the quotation marks on the LCD
lcd.print(I,2);
//Display the value of the variable I on the LCD screen with 2 decimal places
lcd.print"A");
//Display the text between the quotation marks on the LCD s
lcd.setCursor(0,1);
//Move the cursor to column 1, row 2 lcd.print("P="); //Display the text
```

```
between the quotation marks on the LCD screen
lcd.print(P,1);
//Display the value of the variable P on the LCD screen with one decimal place
lcd.print("W");
 //Display the text between the quotation marks on the LCD screen
lcd.print(" E=");
  //Display the text between the quotation marks on the LCD screen
lcd.print(Etot,2);
//Display the value of the variable Etot on the LCD screen with 2 decimal places
lcd.print("Wh");
  //Display the text between the quotation marks on the LCD screen
delay(500);
 //Delay for 500ms
}
```

4.3. Operating Mode and Calibration

Step 1

Write the code sequence. Make the electrical connections according to the diagram in Figure 7.13 and upload the code sequence.

Since the actual values differ from the theoretical ones, it is necessary to calibrate the electronic measurement circuit, both through modifications in the software part and in the hardware part:

- The voltages Vcc and Uref have a theoretical value of 5 V. The real voltage will be measured using a voltmeter, and the measured value will be written in the program when declaring the variables Vcc and Uref.

- val_dig_I0 (the digital value corresponding to the current measurement value of 0) has a theoretical value of 512. With the power supply turned off (no current through the load), the val_dig_I variable is displayed on the serial monitor, and the displayed value will be written in the program when declaring the val_I0 variable.
- val_dig_1A (the digital value corresponding to the current measurement value of 1A) has a theoretical value of 545. With the power supply turned on, gradually increase the power supply voltage until the current through the load is 1 A. Display the val_dig_I variable on the serial monitor, and the displayed value will be written in the program when declaring the val_dig_1A variable.
- Rotate the potentiometer RS1 until the voltage displayed on the LCD matches the voltage displayed by the power supply.

Step 2

Remove the /* and */ characters that mark the following lines as comments:

```
for (int i=0;i<500;i++) {
val_dig_U = val_dig_U + analogRead(voltageInput); val_dig_I = val_dig_I + analogRead(currentInput); delay(1);
}
val_dig_U = val_dig_U / 500; val_dig_I = val_dig_I / 500;
```

and mark the following lines as comments:

```
//val_dig_U = analogRead(voltageInput); //val_dig_I = analogRead(currentInput); //delay(500);
```

5. Aplication 2. Using an external reference voltage

5.1. Building the electronic setup

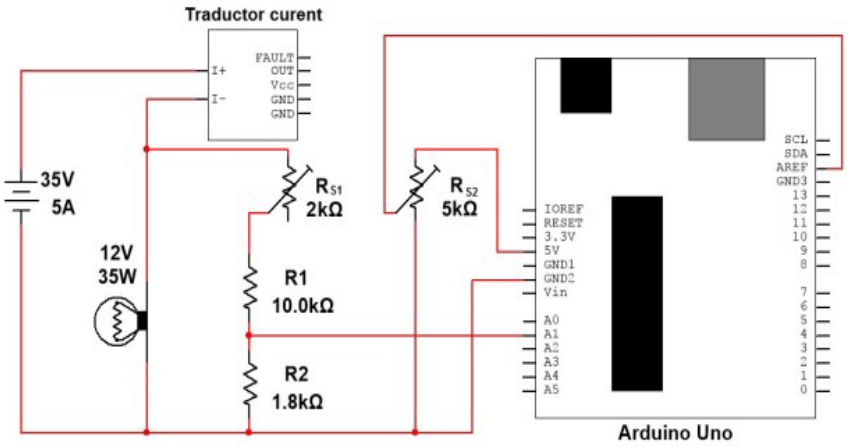


Figure 7.10. ***Schematic diagram for application 2*** **(from [1])**

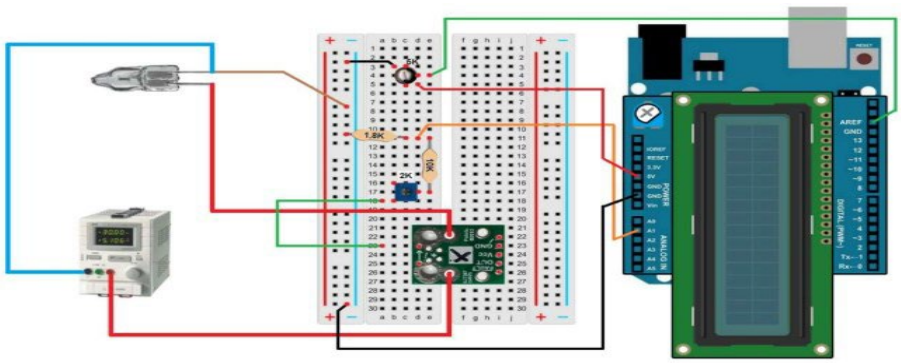


Figure 7.11. ***Making the electrical connections for application 2*** **(from [1])**

The following connections are made:

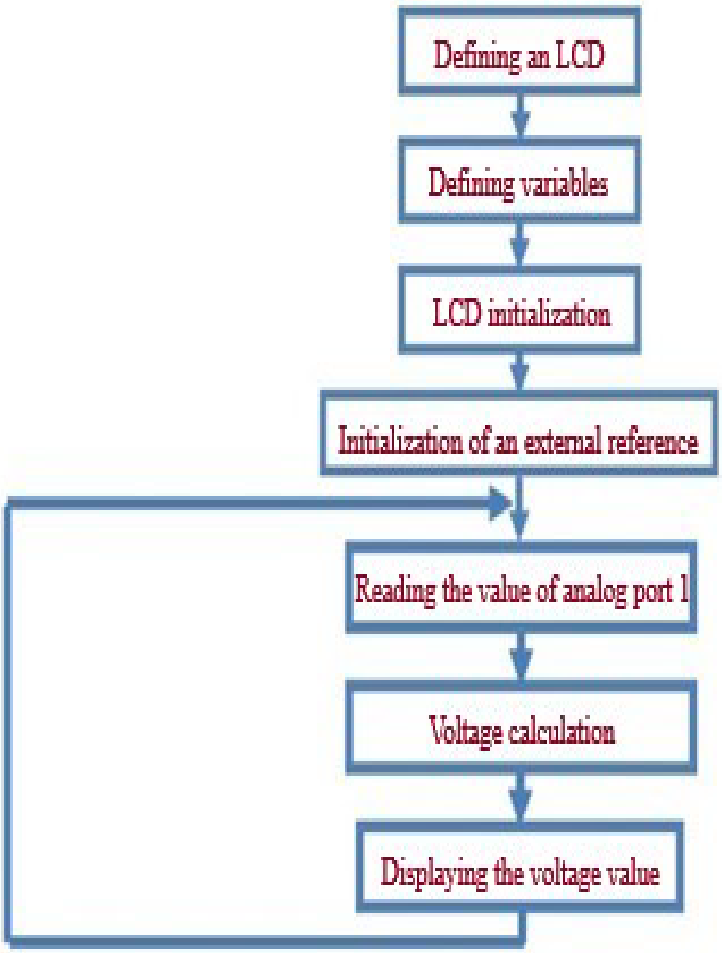
- The current sensor is placed on the breadboard by connecting the input pins for the measured current on column d and the output and power pins on column i;
- Resistors are placed on the breadboard according to the schematic diagram;
- The negative pin of the bulb is connected with a wire to the "-" bar of the breadboard;
- Analog pin A1 on the Arduino board is connected with a wire to resistors R1 and R2;
- The AREF pin on the Arduino board is connected with a wire to the cursor of the 5 KΩ potentiometer.
- The other two pins of the potentiometer are connected with wires to GND and the 5V pin on the Arduino board.
- The GND (power) pin on the Arduino board is connected with a wire to the “-“ bar of the breadboard.

In this application, the current sensor cannot be used because it requires a reference voltage equal to Vcc.

Verify the correct and secure connection of the 1.8 KΩ resistor and the wire (brown in the above diagram) between the negative side of the bulb and the “-“ bar of the breadboard.

Any error in this regard may cause a voltage higher than 5 V to appear on pin A1 and may damage the microcontroller.

5.2. Logical scheme and code sequence




```
#include <LiquidCrystal.h>
        //Including the library for LCD commands into the program
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);
//Initialization of the library and the variable "lcd" with the names of the pins used by the LCD shield
const int voltageInput = 1;
//Defining the variable voltageInput corresponding to analog port A1 where the resistor divider output will be connected
```

```
float Udiv = 0.0;
          //Defining the voltage supplied by the resistor divider
float U = 0.0;
          //Defining the electric voltage variable  unsigned long val_dig_U =
0;
//Defining the variable val_dig_U which will contain the value read from the
analog port
float Vcc = 5.0;
          //Defining the initial value of Vcc variable as 5V
          `   float Uref = 1.71;
            //Defining the initial value of Uref variable as 1.71V

void setup(){ lcd.begin(16, 2);
//Initializing the LCD interface and specifying the number of columns and
rows
analogReference(EXTERNAL);
//Using external reference voltage
}

 //Initializing the LCD interface and specifying the number of      columns
and rows
analogReference(EXTERNAL);
          //Using external reference voltage
          }

void loop(){
for (int i=0;i<500;i++) {
val_dig_U = val_dig_U + analogRead(voltageInput); delay(1);
```

```
}
val_dig_U = val_dig_U / 500;
        //Calculating the average value of val_dig_U read from analog port1
Udiv = (val_dig_U * Uref) / 1023.0;
//Calculating the voltage obtained from the voltage divider, based on the value of val_dig_U
U = Udiv * (35 / Vcc);
//Calculating the voltage applied to the voltage divider (voltage applied to the bulb)
lcd.clear();
//Clearing the LCD screen and positioning the cursor in the top left corner
lcd.print("U=");

  //Display the text between the quotation marks on the LCD screen
lcd.print(U,1);
//Display the value of variable U on the LCD screen, with one decimal place
lcd.print("V");
            //Display the text between the quotation marks on the LCD screen
}
```

5.3. Operation Mode and Calibration

The code sequence is written. The electrical connections are made according to the diagram in Figure 11, and the code sequence is uploaded.

Because real values differ from theoretical ones, it is necessary to calibrate the measurement electronic circuit, both through modifications in the software part and in the hardware part:

- The Vcc voltage has a theoretical value of 5 V. The actual voltage will be measured using a voltmeter, and the measured value will be written in the program when declaring the Vcc variable.
- Using a voltmeter, measure the voltage on the AREF pin. Rotate the semi-adjustable resistor RS2 until it reaches a value of 1.71 V.

If an external reference is used on the AREF pin, it is mandatory to set the analog reference to EXTERNAL (using the command analogReference(EXTERNAL)) in the code sequence before calling analogRead(). Otherwise, the active reference voltage (internally generated) and the AREF pin will be short-circuited, potentially damaging the microcontroller on the Arduino board.

6. Additional Exercises and Conclusions

1. It is observed that the value of the measured current intensity varies slightly when it is 0 (val_dig_I0), due to the thermal drift of the sensor. Modify the code sequence so that for the range (val_dig_I0 - 2, val_dig_I0 + 2), the value 0 is displayed for the current intensity.
2. What is the effect of the modifications to the code sequence from step 2?
3. The code sequence should be modified so that the displayed voltage measurement is shown with 4 decimal places.

Calibrating devices or measuring instruments connected to a computer or development board can be done either through hardware or software. Calibration is performed using reference devices and instruments (or those considered as such by the user). For hardware calibration, it is necessary to modify the parameters defining some electronic components that are part of

the measuring device or instrument. Software calibration involves adjusting the conversion values in programs that process the acquired data or those provided by analog inputs (e.g., analog-to-digital converter). The voltage divider has the advantage of using a simple electrical circuit and a small number of electronic components. However, it also has disadvantages such as the presence of a constant operating current, which, to have minimal influence on the load, must be much larger than the load current connected to the voltage divider, and the consumption of electrical energy during circuit operation.

BIBLIOGRAPHY

1. Iordache, V., Cormoș, A. 2019. *Senzori, traductoare și achiziții de date cu Arduino Uno.* București: Editura Politehnica Press.
2. ***. 2013. Allegro MicroSystems, LLC. *ACS711 Datasheet.* https://www.allegromicro.com/-/media/files/datasheets/acs711-datasheet.pdf
3. ***. 2015. Current Sensor Carrier. *Pololu Robotics & Electronics*: https://www.pololu.com/product/2452

Paper 8

MEASUREMENT USING A REAL-TIME CLOCK

1. Work Description

1.1. Objectives of the Work

- To create and test medium complexity circuits using external modules.
- Use of electronic shield modules.
- To develop a practical application to display a real-time clock on an LCD screen.

1.2. Theoretical Description

Introduction

The aim of this application is to understand how to make an electronic circuit that displays numerically a real time clock on an LCD screen.

For this purpose, a real-time clock module will be used, made with DS1307 integrated circuit, capable of providing day of the month information, month, year, day of the week, hour, minutes and seconds, through a serial communication of type I²C communication using the SCL and SDA data pins available on the Arduino Uno board [2].

2. Hardware component

Component/module	Features	Number of pieces	Picture
Arduino Uno		1	
Breadboard	82×52×10 mm	1	

Component/module	Features	Number of pieces	Picture
LCD Shield	Display on 2 rows of 16 characters	1	
Connecting wire	Father-Father	1	
Real-time clock	DS1307	1	

In this work, to achieve electronic assembly using external components, a breadboard test board shall be used.

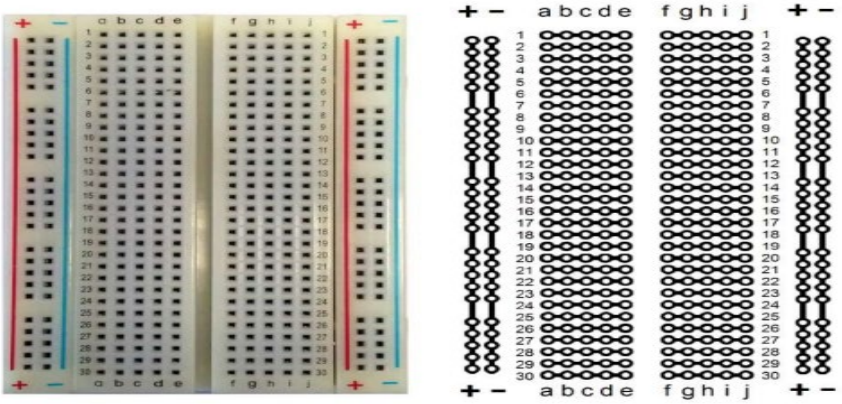


Figure 8.1. ***Breadboard and internal connections*** **(from [2])**

LCD Shield allows characters to be displayed on a liquid crystal display with LED illumination. It mounts over the Arduino board and has connectors so that the board pins will still be accessible.

The LCD screen consists of 2 lines of 16 characters each character is composed of 5×8 pixels. The shield uses the digital pins of the Arduino board from 2 to 7. The real-time clock provides information such as the day of the month, the month, year, day of the week, hour, minutes and seconds. It is made with a circuit integrated circuit DS1307 [3] and uses an oscillating circuit based on crystal oscillator with an output signal of data signal via an I2C serial connection. The time format can be set between 12 hours per bit for AM/PM or 24 hours, and the number of days of months and leap year correction is done automatically. For the day of the week the clock provides digits from 0 to 6 which corresponding to Sunday to Saturday.

The real-time clock module also contains two 4.7 kΩ resistors connected between each of the SCL, SDA and VCC data pins, acting as pull-up resistors to 1. When an I²C bus is shared by several modules, each having a set of pull-up resistors at 1, only one set will be kept, by removing the float on the jumper circled in green.

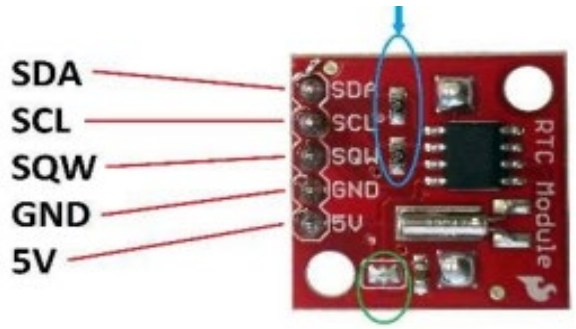


Figure 8.2. ***Using pull-up resistors*** **(from [1, 2])**

The SQW pin generates, when activated, a rectangular signal with a frequency that can be chosen from four values.

Since the I²C bus is used, the *Wire's* library, already available in the pre-installed library package in the Arduino IDE, will have to be included in the code sequence.

The real-time clock module will be powered by VCC = 5 V.

Writing real-time clock data

Since there is no way to automatically synchronize the clock with an external clock, the correct setting of the clock data is done manually by the user.

In the case of the real-time clock, a library will no longer be used. library. The data will be written directly to the internal clock registers using functions specific to the use of the I2C bus, the steps are as follows:

- Activate the I2C bus for writing by specifying the address of the clock.
- Set the address of the first register where the write will start data.
- Writing data in registers. They will first be converted from decimal to BCD format.
- Closing the I²C bus for writing.

Remarks:

- If bit 8 of the 00h register is set to 1 the oscillator is disabled (the clock stops running) in order to minimize the current consumed (clock data must be updated when used again).
- If bit 7 of register 02h is set to 0, the hours are displayed in 24-hour format.
- If bit 7 of register 02h is set to 1, the hours are displayed in 12-hour format, bit 6 providing in this case the information AM (value 0) or PM (value 1).

The content of the internal data storage registers is presented in the following table:

Address	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Function	Domain
00h	CH	Ten Seconds			Second Units				Second	00-59
01h	0	Ten Minute			Units Minute				Minute	00-59
02h	0	12	PM/AM	Tens Hour	Units Hour				Time	1-12 +AM/PM
		24	Tens Hour							00-23
03h	0	0	0	0	0	Units Day week			Day week	01-07
04h	0	0	Tens Day Month		Units Day Month				Day Month	01-31
05h	0	0	0	Tens Moon	Units Moon				Moon	01-12
06h	Ten Year				Units Year				An	00-99
07h	OUT	0	0	SQWE	0	0	RS1	RS0	Control	-
08h-3Fh									RAM 56x8	00h-FFh

CONVERSION OF DATA FROM DECIMAL TO BCD

BCD coding assigns each digit in the decimal system (0, 1,9) to a four-bit binary code (0000, 0001,1001). If a number in decimal system has n digits, BCD coding will give it a consisting of $n * 4$ bits (e.g. the number 19 in decimal is written as 0001 1001 in BCD).

The watch's internal registers store 8-bit BCD-coded data, i.e. by decoding it we get a decimal number consisting of two digits (the of tens and units).

The Arduino board takes the data to be written to the clock registers in decimal (as written by the user) and transmits it in binary code, without knowing that the clock will interpret it as BCD.

Thus, after example above, the number 19 (0001 1101 in BCD) will be transmitted in binary as 0001 0011 and will mean for the clock 13. Therefore, before being transmitted, the data must be converted from decimal to BCD.

The transformation from decimal to BCD is done in the code sequence and involves the following operations:

- Divide the initial number by 10. Dividing two numbers gives the result without subtraction, i.e. the number of tens (19 / 10 = 1, i.e. 0000 0001). Then translate the bits of the result by 4 positions to the left (code 0000 0001 will become 0000 0001 0000) by which basically preserves the four bits representing the digit the digit of the decimal point, and are brought to the decimal point in BCD format).
- Perform the modulo operation (return the remainder of the division of two integers) between the original number and 10 (19 % 10 = 9), the result is the number of units in the original number (0000 1001).
- Operating a logical OR between the two results will result in the code 0001 1001, i.e. 19 in BCD.

Reading clock data in real time

Data will be read directly from the internal registers of the watch using functions specific to the use of the I²C bus, the steps are as follows:

- Activate the I2C bus for writing, mentioning the address of the clock.

- Set the address of the first register from which reading will start the data.

- Close the I2C bus for writing.

- Activate the I2C bus for reading.

- Read data from registers and assign them to variables. Thanks to the fact that the data is stored in registers in BCD (decimal coded binary), they will first be converted to decimal format.

When the 12-hour format is chosen, from the byte byte corresponding to the hour, bits 1...5 are extracted containing the time information and bit 6 containing the AM or PM information.

Converting data from BCD to decimal

The Arduino board receives each byte of data by interpreting it as a classical number using all 8 bits, not knowing that they are actually BCD encoded. So the combination 0001 1001 will mean 25 in decimal (and not 19 as it should be).

The conversion from BCD to decimal is done in the code sequence and involves the following operations:

- Translating the bits of the 4-position data byte to the right (the code 0001 1001 will become 0000 0001; it will basically keep the four bits representing the tens digit) and multiplying by 10 (the final result will be $1 \cdot 10 = 10$).
- Operation of a logical AND between the data byte and a byte with value 0000 1111 (0001 1001 & 0000 1111 = 0000 1001 = 9; the four bits representing the digit of the units are preserved).
- Add the results of the above operations (10 + 9 = 19).

3. Software component

LiquidCrystal.h is the library containing the commands for the LCD SHIELD.

Wire.h is the library containing the commands for the I2C bus.

LiquidCrystal creates a variable specifying the digital pins used to control the LCD shield.

const has the meaning of a constant modifying the behavior of a variable. The variable will become read-only i.e. its value will not be able to be changed.

int variable = value sets a value for a variable of type 16-bit integer variable with sign (-32.768 to 32.767).

byte variable sets an unsigned byte variable.

void setup() is a function (which returns no data and has no parameters) that runs once at the start of the program. This sets the general instructions for setting up the program (setting pins, enabling serial ports, etc.).

void loop() is the main function of the program (which does not return data and has no parameters) and is executed continuously as long as the board is running and not reset.

if(condition) {instructions} else {instructions/instructions} tests whether a condition is met.

Wire.begin() is a function that initializes the I2C bus.

Wire.write(byte(register_pointer)) is a function that sets the register from which to start the data read operation.

Wire.requestFrom(address, n) is a function that opens the I 2C in data read mode (a number of n registers) from the specified address.

Wire.read() is a function that reads data register by register and provides the result.

lcd.begin(columns, rows) initializes the interface to the LCD screen and specifies the number of rows and columns.

lcd.setCursor(column, row) sets the position of the LCD cursor. For the LCD used in this application the number of columns is 0 to 15, and the rows from 0 to 1.

lcd.clear().

lcd.print() displays data (values of some variables)/text in brackets. To display a text it is necessary that it must be enclosed in quotation marks (“text”).

To display the value of a char, byte, int, long, or string variable, write the variable name and, optionally, its numbering base (variable, BIN or DEC or OCT or HEX). To display the value of a float or double variable type write the variable name and after the decimal point, Serial.begin(speed) sets the data transfer rate for the serial port in bits/second (BAUD).

Serial.println("text") prints the text as ASCII characters using the serial port, adding a newline after it. The return value is used to terminate the execution of a function and to return a value. delay(ms) pauses the program for a duration of time specified in milliseconds.

variable >> n is an operator that moves the bits of the variable by n positions to the right.

variable << n is an operator that moves the bits of the variable by a number of n positions to the left.

& is the logical AND operator.

| is the logical OR operator.

% is the modulo operator that calculates and gives the remainder of the division between two integers.

/ is the division operator which, when dividing two numbers gives the result without remainder.

== means equal to.

4. Application

4.1. Electronic Assembly

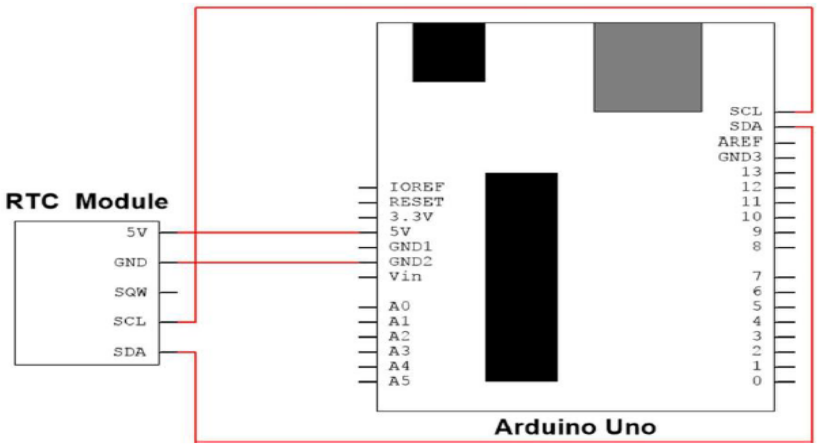


Figure 8.3. ***Principle diagram*** **(from [1, 2])**

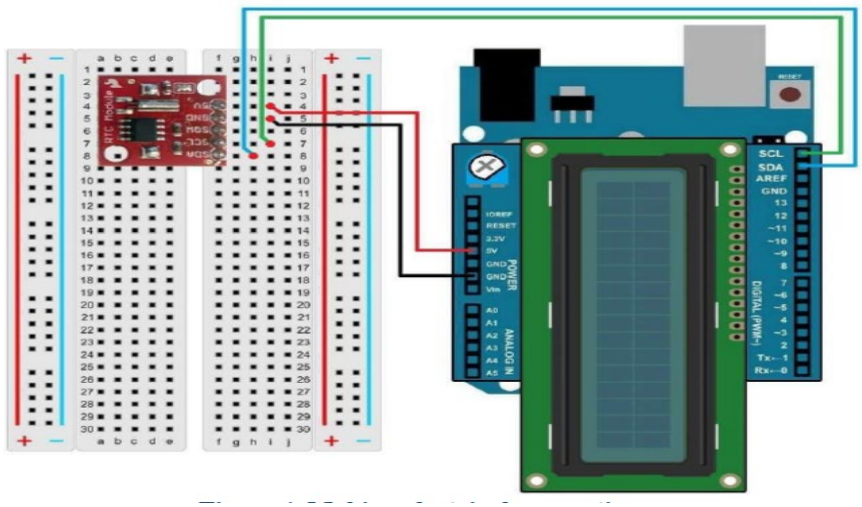
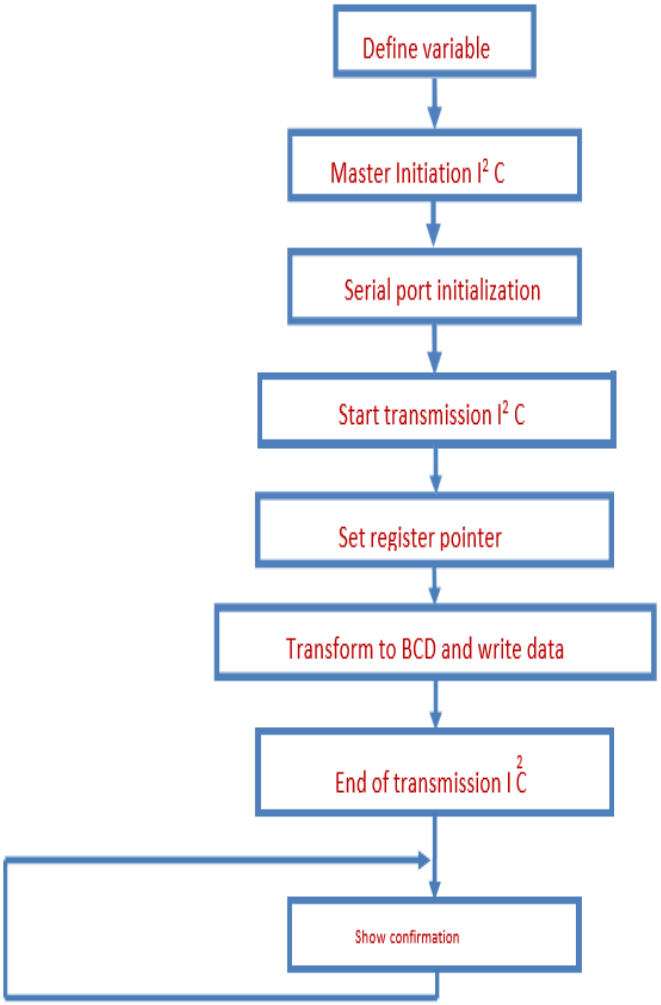


Figure 8.4. ***Electrical connections*** **(from [1, 2])**

4.2. Logic diagram and code sequence

Writing (setting) real-time clock data



Code sequence

```
#include < Wire.h>
//include I2C bus command library in program
const intADDRESS_CEAS = 0x68;
```

```
//defining the variable corresponding to the clockaddress
 int second = 10;
//defining variables corresponding to clock data
Int minute = 29;
int now = 4;

void setup(){
Wire. begin();
//initialisation of the I2C bus
Serial. begin(9600);
//enables serial port output at 9600 baud rate
Wire. beginTransmission(address_CEAS);
//open the I2C bus in data transmission mode to the specified address
Wire. write(byte(0x00));
//set the register where the data write operation will start.
Wire.write(ZecinBCD(second));
//write seconds value after conversion from decimal to BCD
Wire. write(ZecinBCD(minute));
//write minute value, after conversion from decimal to BCD Wire.
write(ZecinBCD(hour));
//write hour value after conversion from decimal to BCD - for 24-hour format
//Wire.write(ZecinBCD(hour) | 0b1100000);
//write hour value, after conversion from decimal to BCD - for 12-hour
format, set bit 6 to 1 for PM (or 0 for AM) and set bit 7 to 1
Wire. write(ZecinBCD(weekday));
//write weekday value after conversion from decimal to BCD
Wire. write(ZecinBCD(dayMonth));
```

```
//write day month value after conversion from decimal to BCD Wire.
write(ZecinBCD(month));
//write month value, after conversion from decimal to BCD Wire.
write(ZecinBCD(year));
//write year value after conversion from decimal to BCD Wire.
endTransmission();
//end data transmission
}
void loop(){
//the content of the loop loop is to inform about the completion of the clock
update procedure
Serial. println("The clock has been updated");
//scratch the text between quotes on the serial monitor delay(10000);
//delay 10 seconds
}
byte ZecinBCD(byte value) {
return ( ((value/10)<<4) | (value%10) );
//data conversion function from decimal to BCD - formula (1)
}
```

Real-time clock data reading

```
#include < Wire.h>
//including the I2C bus command library in the program const int
ADDRESS_CEAS = 0x68;
//defining the variable corresponding to the clock address
int second, minute, hour, dayWeek, dayMonth, month,year;
//defining variables for the data provided by the clock
```

```
//byte AMPM;
#include < LiquidCrystal.h>
//include command library in theprogram

LCrl lcd( )
//initialize the library

//initialize I2C buslcd
lcd.begin(16, 2);
/initialize the LCD interface and specify the number of rows andcolumns of the LCD screen
}
  void loop(){ Wire. beginTransmission(CEAS_ADDRESS);
//open the I2C bus in data transmission mode to the specified addressWire. write(byte(0x00));
//set the register where the data read operation will start. Wire.endTransmission();
//end data transmission
Wire. requestFrom(ADDRESS_CEAS, 7);
//open the I2C bus in data read mode (from 7 registers) from thespecified address
second = BCDinZec(Wire. read());
//read value seconds, conversion from BCD to decimal and variableallocation
minute = BCDinZec(Wire. read());
//reading minute value and conversion from BCD to decimal and variable allocation
hour = BCDinZec(Wire. read());
```

```
//read hour value and conversion from BCD to decimal and variableallocation
//time = Wire.read();
//read value hour
//AMPM = time & 0b100000;
//retain bit 6 of hour value (if 0 then it is AM)
//time = BCDinZec(time & 0b11111);
//save bits 1...5 of hour value (actual time data) and update hourvariable
dayWeek = BCDinZec(Wire. read());
//read weekday value and conversion from BCD to decimal and variable allocation
dayMonth = BCDinZec(Wire. read());
//read day month value and conversion from BCD to decimal and variable allocation
Month = BCDinZec(Wire. read());
//month value reading and conversion from BCD to decimal and variable allocation
year = BCDinZec(Wire. read());
//year value reading and conversion from BCD to decimal and variable allocation
lcd. clear();
//deleting LCD screen content.
lcd.prtdayMoon);
///displays on the LCD screen the value of the variable dayMonth
lcd. prnt("-");
///displays on the LCD screen the text in quotes lcd. prnt(month);
///displays on the LCD screen the value of the variable month.
lcd.print("-");
```

```
///displays on the LCD screen the text in quotes lcd. print(year + 2000);
///displays on the LCD screen the value of the variable an
lcd. setCursor(0, 1);
//move cursor to column 1, row 2
lcd. print(time);
///displays the value of the time variable on the LCD screen.
lcd.print(":");
///displays on the LCD screen the text in quotes
lcd.print(minute);
///displays on the LCD screen the value of thevariable minute
lcd. print(":");
///displays on the LCD screen the text in quotes
lcd.print(second);
  ///displays on the LCD screen the value of the second variabl
//if (AMPM == 0) {
  //if AMPM = 0 AM is displayed, otherwise PM is displayed
   //lcd.print("AM")
  // }  // else {
    /lcd.print("PM")
  // }
    delay(1000);
  //delay 1 second    }
          byte BCDinZec(byte value) {
     return ( ((value>>4)*10) + (value&0b1111) );
     //data conversion function from BCD to decimal - formula (2)
       }
```

ADDITIONAL EXERCISES AND CONCLUSIONS

1. Change the code sequence so that months, hours, minutes or seconds from 0 to 9 are displayed as 00, 01, ... 09.
2. Modify the code sequence so that the day of the week is alsodisplayed on the screen in abbreviated form (Mon, Tue, Wed, etc.).
3. Write a program that provides a timer function - displaying a predefined message when the user reaches a preset time.
4. Write a program to perform certain read or write operations to the ports of the development board at certain points in time (to simulatethe timing clock used in the operation of automatic traffic and traffic light installations).

The clock or clock signal is very important in the synchronization of electronic circuits or in the sequential approach to the steps of an industrial process (in the case of transport, traffic management can be considered as such a process). Transmissions between the different components of a system can be synchronous or asynchronous. Synchronous transmissions are made by the use of a common clock by all equipment transmitting data within a system. Asynchronous transmissions are made by including timing information in the transmitted messages or data (in particular for defining the bit range).

Synchronization of the different components of a traffic management system can be done by using clock signals from *Global Navigation Satellite Systems* (GNSS), such as GPS, or by use of a dedicated clock channel. Synchronization is very important together with maintaining a common time reference for all synchronized equipment (time t = 0). The clock signal is also important in the development of equipment based on digital integrated circuits, as it has the role of both synchronizing all existing circuits and ensuring a sequential approach to the program or logic implemented in the equipment.

BIBLIOGRAPHY

1. Iordache, V., Cormoș, A. 2019. *Senzori, traductoare și achiziții de date cu Arduino Uno.* București: Editura Politehnica Press.
2. ***. 2008. Arduino Reference. *Bitshift Operators.* https://www.arduino.cc/en/ Reference/Bitshift.
3. ***. 2008. Maxim Integrated. *DS1307, I2C Real-Time Clock – Datasheet.* https://www.microcrystal.com/en/products/real-time-clock-rtc-modules/