TECHNICAL UNIVERSITY OF MOLDOVA
FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS
DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATICS

Report of laboratory work №3

Theme: Sensors

Fulfilled: st. gr. FAF-202 **Popa Eugeniu**

Controlled: univ. lecturer **Moraru Dumitru**

Chișinău 2023

THE TASKS OF THE LABORATORY WORK

Main task: To design an MCU-based application that would receive a signal from at least 3 signal sources (analog and digital sensors), condition the signals, and display the physical parameter (temperature, pressure, etc.) on a serial terminal.

THE PROGRESS OF THE WORK

The laboratory work involves measuring various physical parameters using different sensors. The sensors used in this code are: LM35 temperature sensor (temperatureSensor), LDR light sensor (lightSensor), MPX4115A pressure sensor (pressureSensor), HCSR04 ultrasonic sensor (ultrasonicSensor).

The setup() function (Fig 1.1) initializes each sensor object and sets the corresponding pins for each sensor. Additionally, it initializes the serial communication at a baud rate of 9600 and prints a message to the serial monitor.

```
void setup()
{
  temperatureSensor.setPin(lm35Pin);
  lightSensor.setPin(ldrPin);
  pressureSensor.setPin(mpx4115a);
  ultrasonicSensor.setPin(trigPin, echoPin);
  ultrasonicSensor.setup();

  Serial.begin(9600);
  Serial.println("This is laboratory work nr.3");
}

void loop()
{
  int tSensorValue = temperatureSensor.getSensorValue();
  float temperatureC = temperatureSensor.getTemperature(tSensorValue);
  Serial.print("Temperature: ");
  Serial.print(temperatureC);
  Serial.println(" degrees Celsius");

  int lSensorValue = lightSensor.getSensorValue();
  float lux = lightSensor.getLight(lSensorValue);
  Serial.print("Light intensity: ");
  Serial.print(lux);
  Serial.println(" Lux");

  int pSensorValue = pressureSensor.getSensorValue();
  float pressure = pressureSensor.getPressure(pSensorValue);
  Serial.print("Pressure: ");
  Serial.print(pressure);
  Serial.println(" kPa");

  float distance = ultrasonicSensor.getData();
  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");

  delay(3000);
}
```

Fig 1.1 setup() function

The loop() function (Fig 1.2) reads data from each sensor and prints the values to the serial monitor. For each sensor, it first retrieves the raw sensor value using the getSensorValue() function of the corresponding sensor object. It then converts the raw value to a physical value using the corresponding conversion function (getTemperature(), getLight(), getPressure(), and getData() for the LM35, LDR, MPX4115A, and HCSR04 sensors respectively).

```
void loop()
{
  int tSensorValue = temperatureSensor.getSensorValue();
  float temperatureC = temperatureSensor.getTemperature(tSensorValue);
  Serial.print("Temperature: ");
  Serial.print(temperatureC);
  Serial.println(" degrees Celsius");

  int lSensorValue = lightSensor.getSensorValue();
  float lux = lightSensor.getLight(lSensorValue);
  Serial.print("Light intensity: ");
  Serial.print(lux);
  Serial.println(" Lux");

  int pSensorValue = pressureSensor.getSensorValue();
  float pressure = pressureSensor.getPressure(pSensorValue);
  Serial.print("Pressure: ");
  Serial.print(pressure);
  Serial.println(" kPa");

  float distance = ultrasonicSensor.getData();
  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");

  delay(3000);
}
```

Fig 1.2 loop() function

After each value is calculated, the loop() function prints the value and a descriptive message to the serial monitor using the Serial.print() and Serial.println() functions.

Finally, the loop() function includes a delay of 3000 milliseconds (3 seconds) to wait before reading data again. This is done to prevent reading data too frequently and overloading the serial monitor with too much information.

HCSR04.cpp (Fig 1.3) is the implementation of a class for the HCSR04 ultrasonic sensor.
The class has the following member functions:

HCSR04(): Constructor of the class. It does not receive any parameter and does not do anything in this implementation.

void setPin(int pin1, int pin2): This function receives two integer parameters that represent the pin numbers for the trigger and echo pins of the sensor. It sets the member variables trigPin and echoPin with these values.

void setup(): This function configures the pins defined by trigPin and echoPin as OUTPUT and INPUT respectively.

float getData(): This function triggers the sensor to send an ultrasonic pulse and measures the time it takes for the pulse to bounce back to the sensor. It then calculates the distance based on the time measurement and returns the result as a float value.

```cpp
#include "HCSR04.h"

HCSR04::HCSR04()
{
}

void HCSR04::setPin(int pin1, int pin2)
{
    trigPin = pin1;
    echoPin = pin2;
}

void HCSR04::setup()
{
    pinMode(trigPin, OUTPUT);
    pinMode(echoPin, INPUT);
}

float HCSR04::getData()
{
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);

    long duration = pulseIn(echoPin, HIGH);
    float distance = duration * 0.034 / 2.0;

    return distance;
}
```

Fig 1.3 HCSR04.cpp – Implementation of the ultrasonic sensor

HCSR04.h (Fig 1.4) is the header file for the same class. It includes the definition of the class, its member variables and functions, as well as an include for the Arduino.h library, which is required to use the pinMode() and pulseIn() functions in the implementation of the class.

The #ifndef, #define, and #endif are preprocessor directives that are used to prevent the header file from being included more than once in the same code. They ensure that the contents of the header file are only included once, even if the file is included multiple times in the same program. This is done to avoid compiler errors and to save memory.

```
#ifndef HCSR04_H
#define HCSR04_H

#include <Arduino.h>

class HCSR04
{
private:
    int trigPin;
    int echoPin;

public:
    HCSR04();
    void setPin(int pin1, int pin2);
    void setup();
    float getData();
};

#endif
```

Fig 1.4 HCSR04.h – Header file of the ultrasonic sensor

LDR.cpp (Fig 1.5) is the implementation of a class for an LDR (Light Dependent Resistor) sensor. The class has the following member functions:

LDR(): Constructor of the class. It sets the default values for some of the member variables used in the calculation of the light intensity based on the LDR resistance.

void setPin(int pin): This function receives an integer parameter that represents the pin number for the LDR sensor. It sets the member variable sensorPin with this value.

int getSensorValue(): This function reads the sensor value from the pin defined by sensorPin and returns the raw ADC (Analog-to-Digital Converter) value.

float getLight(int sensorValue): This function receives the raw ADC value as a parameter and calculates the resistance of the LDR based on the voltage read from the pin. It then calculates the light intensity in lux using a formula that takes into account the resistance of the LDR, a scalar value and an exponent.

```cpp
#include "LDR.h"

LDR::LDR()
{
    maxAdcReading = 1023;
    adcRefVoltage = 5.0;
    refResistance = 5;
    luxCalcScalar = 12518931;
    luxCalcExponent = -1.405;
}

void LDR::setPin(int pin)
{
    sensorPin = pin;
}

int LDR::getSensorValue()
{
    return analogRead(sensorPin);
}

float LDR::getLight(int sensorValue)
{
    float resistorVoltage = (float)sensorValue / maxAdcReading * adcRefVoltage;
    float ldrVoltage = adcRefVoltage - resistorVoltage;
    float ldrResistance = ldrVoltage / resistorVoltage * refResistance;
    float lux = luxCalcScalar * pow(ldrResistance, luxCalcExponent);
    return lux;
}
```

Fig 1.5 LDR.cpp – Implementation of the light sensor

LDR.h (Fig 1.6) is the header file for the same class. It includes the definition of the class, its member variables and functions, as well as an include for the Arduino.h library, which is required to use the analogRead() and pow() functions in the implementation of the class.

The #ifndef, #define, and #endif are preprocessor directives that are used to prevent the header file from being included more than once in the same code. They ensure that the contents of the header file are only included once, even if the file is included multiple times in the same program. This is done to avoid compiler errors and to save memory.

```
#ifndef LDR_H
#define LDR_H

#include <Arduino.h>

class LDR
{
private:
    int sensorPin;
    int maxAdcReading;
    float adcRefVoltage;
    float refResistance;
    float luxCalcScalar;
    float luxCalcExponent;

public:
    LDR();
    void setPin(int pin);
    int getSensorValue();
    float getLight(int sensorValue);
};

#endif
```

Fig 1.6 LDR.h – Header file of the light sensor

LM35.cpp (Fig 1.7) is the implementation file for the LM35 class.

The LM35 class contains a constructor, setPin function, getSensorValue function, and getTemperature function. In the constructor, the resolution of the LM35 sensor is set to 10, which is used to convert the sensor value to temperature in the getTemperature function. The setPin function takes an integer parameter and sets it as the sensorPin. The getSensorValue function reads the analog value from the LM35 sensor connected to the sensorPin and returns it. The getTemperature function takes an integer parameter, which is the sensor value obtained from the getSensorValue function, and returns the temperature in Celsius, calculated as the sensor value divided by the resolution of the sensor.

```
#include "LM35.h"

LM35::LM35()
{
    resolution = 10;
}

void LM35::setPin(int pin)
{
    sensorPin = pin;
}

int LM35::getSensorValue()
{
    return analogRead(sensorPin);
}

float LM35::getTemperature(int sensorValue)
{
    float temperatureC = sensorValue / resolution;
    return temperatureC;
}
```

Fig 1.7 LM35.cpp – Implementation of the temperature sensor

LM35.h (Fig 1.8) is the header file for the LM35 class. It starts with the standard #ifndef and #define directives used to prevent multiple inclusions of the same header file.

Next, the header includes the Arduino.h library, which is necessary to use the analogRead() function and other Arduino-specific functions and constants.

The LM35 class is then defined with a private section containing two variables: sensorPin, which stores the pin number of the LM35 sensor, and resolution, which sets the number of bits of resolution for the sensor.

The public section of the class defines three methods:

LM35(): the constructor for the class, which sets the resolution to a default value of 10 bits.

setPin(int pin): a method that takes an integer argument pin and sets the sensorPin to that value.

getSensorValue(): a method that returns the analog value read from the LM35 sensor pin.

getTemperature(int sensorValue): a method that takes an integer argument sensorValue and returns the temperature value in Celsius degrees, calculated based on the resolution and the input sensor value.

Finally, the header file ends with the standard #endif directive to close the #ifndef block.

```
#ifndef LM35_H
#define LM35_H

#include <Arduino.h>

class LM35
{
private:
    int sensorPin;
    float resolution;

public:
    LM35();
    void setPin(int pin);
    int getSensorValue();
    float getTemperature(int sensorValue);
};

#endif
```

Fig 1.8 LM35.h – Header file of the temperature sensor

MPX4115A.cpp (Fig 1.9) defines a class called MPX4115A. The constructor function initializes the class, and the setPin() function sets the analog pin of the sensor. The getSensorValue() function reads the analog value from the sensor pin, and the getPressure() function calculates the pressure value based on the sensor value.

The pressure is calculated using the formula pressure = (sensorValue/1023 + 0.095) / 0.009, where sensorValue is the analog value read from the sensor. The formula is specific to the MPX4115A pressure sensor, which has a transfer function that relates the output voltage to the applied pressure.

The header file (MPX4115A.h) defines the class and its public and private member functions and variables. It also includes the Arduino.h library, which is required to use the analogRead() function.

```
#include "MPX4115A.h"

MPX4115A::MPX4115A()
{
}

void MPX4115A::setPin(int pin)
{
    sensorPin = pin;
}

int MPX4115A::getSensorValue()
{
    return analogRead(sensorPin);
}

float MPX4115A::getPressure(int sensorValue)
{
    float pressure = ((float)sensorValue / (float)1023 + 0.095) / 0.009;
    return pressure;
}
```

Fig 1.9 MPX4115A.cpp – Implementation of the pressure sensor

MPX4115A.h (Fig 1.10) is the header file for the MPX4115A class, which is used to interface with an MPX4115A pressure sensor. It includes the Arduino.h library which provides access to the analogRead function used to read the analog sensor data.

The MPX4115A class has a private member variable called sensorPin, which is used to store the Arduino pin number that the sensor is connected to.

The public methods of the MPX4115A class are:

MPX4115A(): A constructor that initializes the MPX4115A object.

void setPin(int pin): A method that sets the sensorPin member variable to the specified pin number.

int getSensorValue(): A method that reads the analog value from the sensorPin and returns it as an integer value between 0 and 1023.

float getPressure(int sensorValue): A method that takes an integer sensorValue as input, converts it to a pressure value in kilopascals using a formula specific to the MPX4115A sensor, and returns the calculated pressure value as a float.

```
#ifndef MPX4115A_H
#define MPX4115A_H

#include <Arduino.h>

class MPX4115A
{
private:
    int sensorPin;
public:
    MPX4115A();
    void setPin(int pin);
    int getSensorValue();
    float getPressure(int sensorValue);
};

#endif
```

Fig 1.10 MPX4115A.h – Header file of the pressure sensor

Electrical scheme made in Proteus

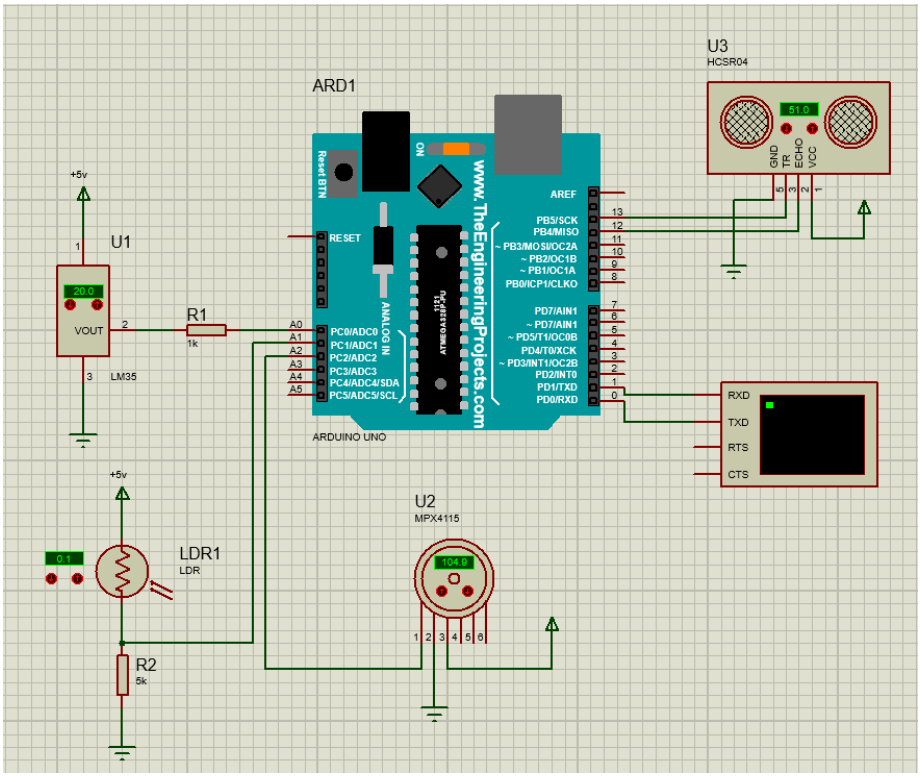

Fig 2 Electrical scheme

CONCLUSIONS

In the following laboratory work, I developed a program that can gather temperature, humidity, light, distance, and pressure data, making it suitable for a wide range of applications like environmental monitoring and robotics. The program utilizes various Arduino functions and libraries to interact with sensors, making it an excellent resource for both novice and experienced Arduino users. I also employed multiple libraries to communicate with the sensors and assign pins for each one. The program continuously reads data from each sensor and displays the results in the Serial Monitor, with a 3-second delay between each iteration.

Overall, I gained valuable experience working with various types of sensors and enhanced my knowledge in this field.