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

Laboratory Work No. 7

Communication

Student: Pogorevici Daniel FAF-202
Teacher: univ.lecturer Moraru Dumitru

Chisinau 2023

The task of the laboratory work:

1. Implement a digital sensor using the I2C digital sensor:
    - The functionality should include data collection using the sensor interface.
    - The MCU that is connected to the sensor should respond to data requests.

2. Request execution using the I2C interface for the given sensor.

Progress of the work:

1. Description of C++ functions:

The most important function from my code is the send Packet function. Here is a code snippet for it and a brief description of it afterwards:

```
void sendPacket(byte* bytes, byte bytes_length)
{
    byte packetSize = 13 + bytes_length;
    byte packet[packetSize];
    // Start Byte
    packet[0] = 0xAA;
    // End Byte
    packet[1] = 0xBB;
    // Packet Counter
    static byte packetCounter = 0;
    packet[2] = packetCounter++;
    // Sender ID
    packet[3] = 0x01; // Example ID for MCU1
    // Receiver ID
    packet[4] = 0x02; // Example ID for MCU2
    // Packet Type
    packet[5] = 0x01; // Example packet type
    // Packet Length
    packet[6] = bytes_length;
    // Payload (float value)
    // byte *floatArray = (byte *)&value;
    for (byte i = 0; i < bytes_length; i++)
    {
        packet[7 + i] = bytes[i];
    }
    // Checksum calculation
    int checksum = 0;
    for (int i = 0; i < packetSize - 1; i++)
    {
        checksum += packet[i];
    }
    packet[packetSize - 1] = checksum & 0xFF;
    // Send the packet over I2C to MCU2
    Wire.beginTransmission(i2cAddress);
    Wire.write(packet, packetSize);
    Wire.endTransmission();
}
```

The function sendPacket takes in two arguments: bytes, which is a pointer to an array of bytes representing the payload to be sent, and bytes_length, which is the length of the payload in bytes.

The first line of the function calculates the total size of the packet to be sent, which includes the protocol fields and the payload. The size of the protocol fields is 13 bytes, so the total packet size is 13 + bytes_length. Next, an array packet of size packetSize is created to hold the packet data. The first two elements of the packet are set to the start and end bytes, respectively. These are fixed values of 0xAA and 0xBB.

The third element of the packet is the packet counter. This is a static variable that is incremented every time a packet is sent. The fourth and fifth elements of the packet are the sender and receiver IDs. In this example, they are set to 0x01 for MCU1 and 0x02 for MCU2. The sixth element of the packet is the packet type. In this example, it is set to 0x01. The seventh element of the packet is the length of the payload in bytes. The next bytes_length elements of the packet are set to the payload data.

After that, a checksum is calculated by summing all the bytes in the packet except for the last byte. The least significant byte of this sum is then stored in the last element of the packet. Finally, the packet is sent over I2C to MCU2 using the Wire library. The I2C address of MCU2 must be specified in a variable called i2cAddress.

```
void receivePacket(int byteCount)
{
    Serial.println("------------------------------------------------");
    byte packet[packetSize];
    for (int i = 0; i < packetSize; i++)
    {
        packet[i] = Wire.read();
    }
    // Start Byte verification
    if (packet[0] != 0xAA)
    {
        return; // Invalid packet, discard
    }
    // End Byte verification
    if (packet[1] != 0xBB)
    {
        return; // Invalid packet, discard
    }
    // Checksum calculation
    int receivedChecksum = 0;
    for (int i = 0; i < packetSize - 1; i++)
    {
        receivedChecksum += packet[i];
    }
    receivedChecksum &= 0xFF;
    // Checksum verification
    if (receivedChecksum != packet[packetSize - 1])
    {
        return; // Invalid packet, discard
    }
    // Extracting fields from the packet
    byte packetCounter = packet[2];
    byte senderID = packet[3];
    byte receiverID = packet[4];
    byte packetType = packet[5];
    byte payloadLength = packet[6];
    // Payload (float value)
    float floatArray[payloadLength / sizeof(float)];
    for (int i = 0; i < payloadLength / sizeof(float); i++)
    {
        byte floatb[sizeof(float)]; // the float containing the distance
        floatb[0] = packet[7 + i * sizeof(float)];
        floatb[1] = packet[7 + i * sizeof(float) + 1];
        floatb[2] = packet[7 + i * sizeof(float) + 2];
        floatb[3] = packet[7 + i * sizeof(float) + 3];

        float realFloat;
        memcpy(&realFloat, floatb, sizeof(float));
        floatArray[i] = realFloat;
    }
    // Convert byte array to float
    float receivedFloat;
    memcpy(&receivedFloat, floatArray, sizeof(float));
    // Use the received float value
    // Example: print it to the serial monitor
    // Serial.print("Received Float: ");
    for (int i = 0; i < payloadLength / sizeof(float); i++)
    {
        Serial.println(floatArray[i]);
    }
}
```

This is a function that receives a packet of data over I2C from another microcontroller (MCU1). The function takes in one argument: byteCount, which is the number of bytes received over I2C. The first line of the function prints a separator line to the serial monitor for readability. Next, an array packet of size packetSize is created to hold the received packet data. The packet data is read from the I2C buffer using the Wire.read() function and stored in the packet array.

The start and end bytes of the packet are then verified to ensure that the packet is valid. If either of these bytes is not equal to the expected value (0xAA for the start byte and 0xBB for the end byte), the function returns without processing the packet further. Next, a checksum is calculated by summing all the bytes in the packet except for the last byte. The least significant byte of this sum is then compared to the checksum value stored in the last element of the packet. If these values do not match, the function returns without processing the packet further.

After verifying that the packet is valid, several fields are extracted from the packet, including the packet counter, sender and receiver IDs, packet type and payload length. The payload data is then extracted from the packet and converted into an array of floats.

This is done by first creating an array floatArray of size payloadLength / sizeof(float). Then, for each float in the array, four bytes are read from the packet and stored in a temporary array floatb. The memcpy function is then used to copy these bytes into a float variable realFloat, which is then stored in floatArray.

Finally, the received float values are printed to the serial monitor.

2. Block diagram of program

The first diagram describes the first program. The second diagram is the representation of the second program.

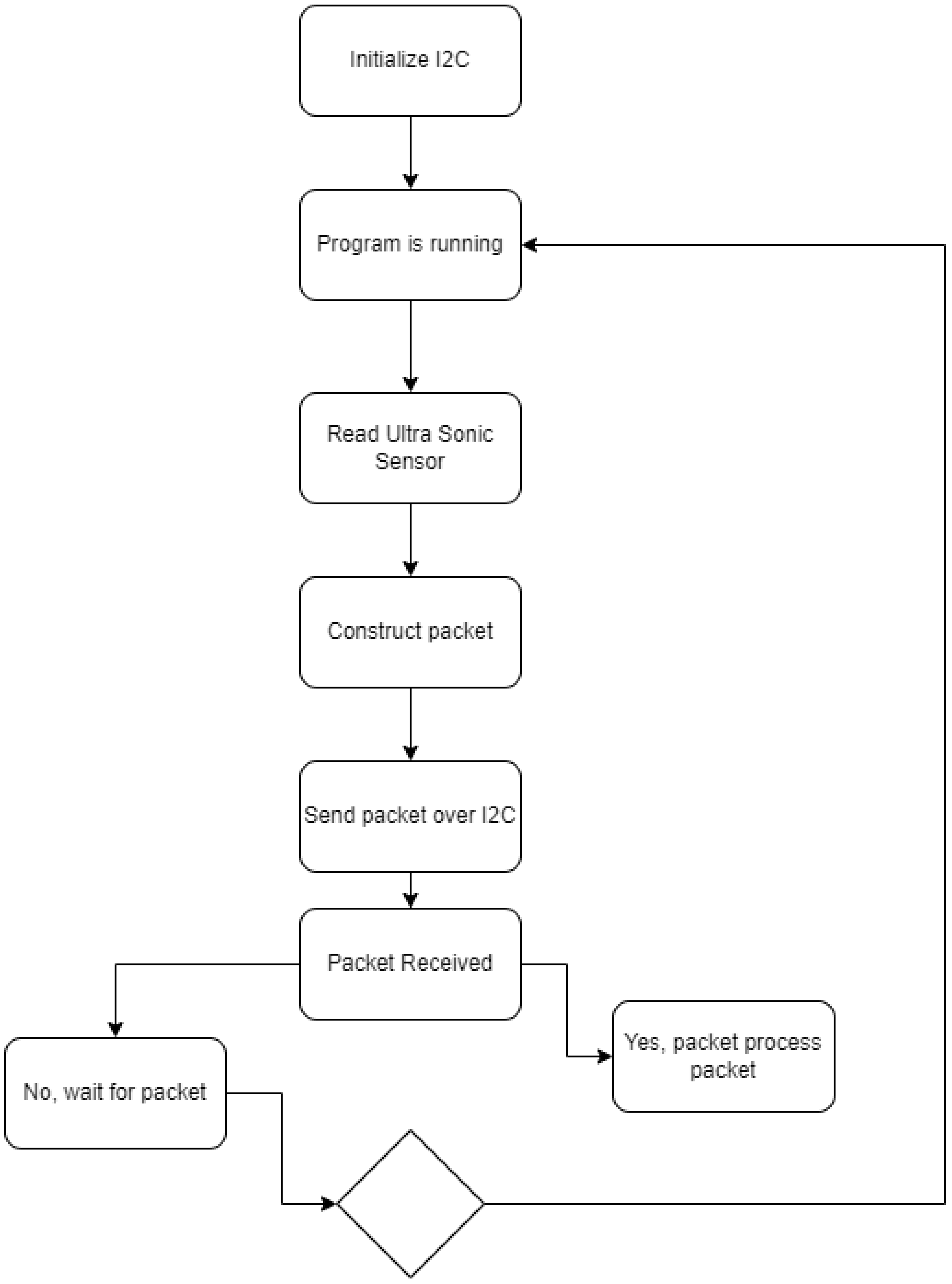


Figure 1. Schematic capture

3. Electric scheme:

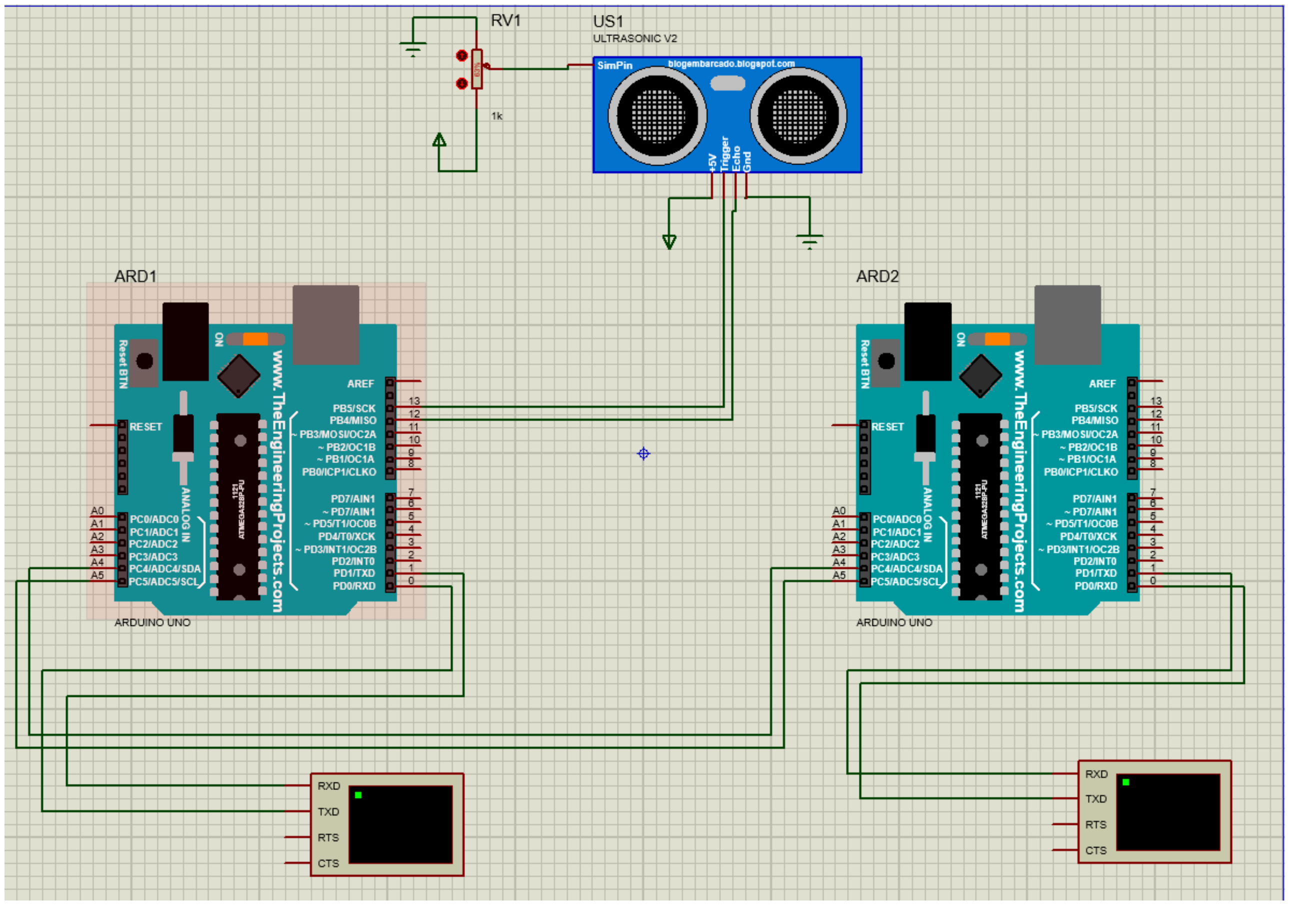
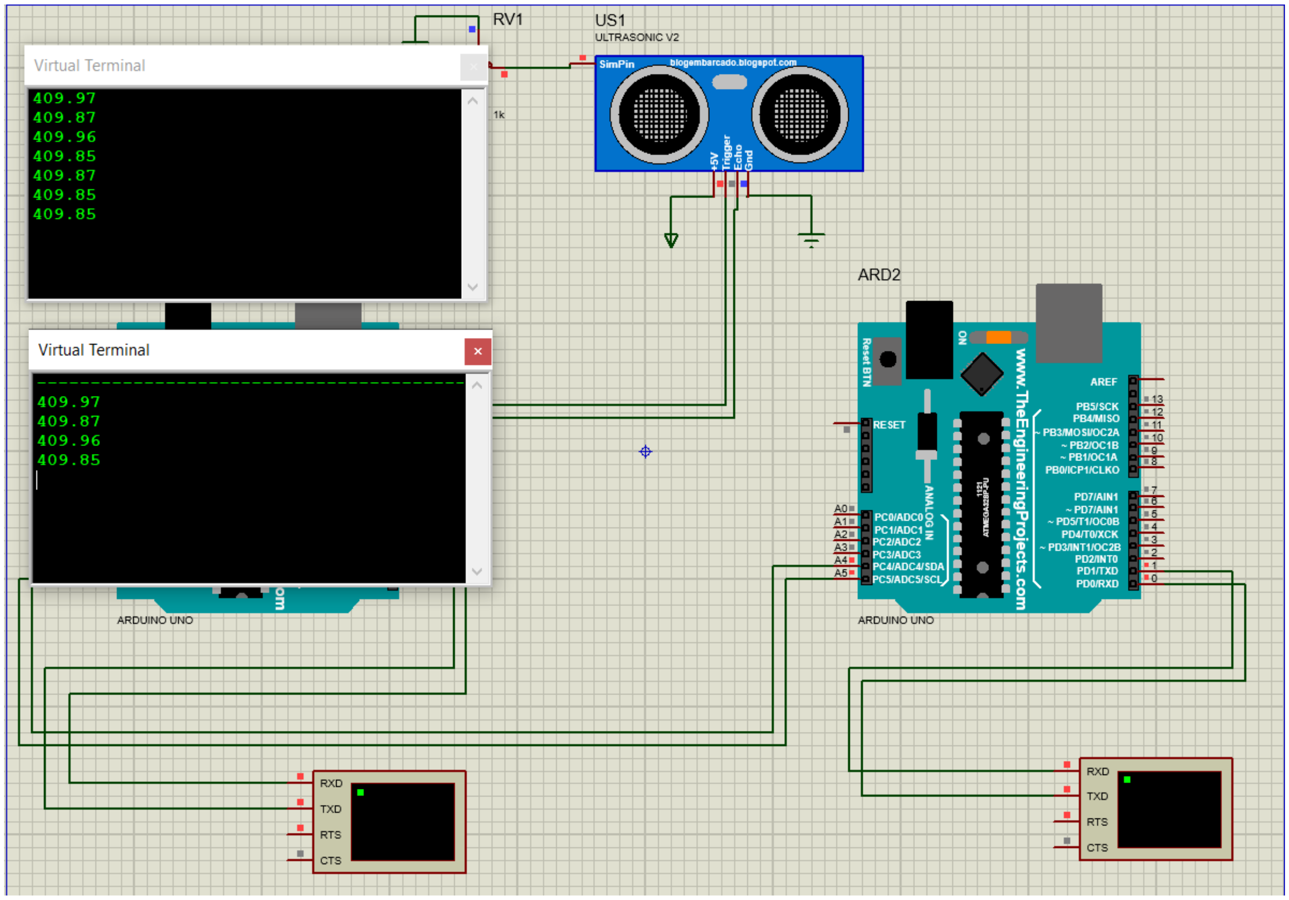


Figure 2. Schematic capture of lab tasks.

As you can see, there are 2 MCUs, 2 terminals and an ultrasonic sensor.



```
409.97
409.87
409.96
409.85
409.87
409.85
409.85
```

```
--------------------------------------------
409.97
409.87
409.96
409.85
```

Figure 3. Screenshot of initial state in that MCU application simulation

As you can see, there are 2 MCUs, 2 terminals and an ultrasonic sensor. The first terminal is for the first MCU and prints the data every 500ms and the second terminal prints the results in batches every 1.5seconds.

CONCLUSIONS

The first function, sendPacket, is responsible for sending a packet of data over the Inter-Integrated Circuit (I2C) communication protocol from one microcontroller unit (MCU1) to another (MCU2). The function takes as input an array of bytes representing the payload to be sent and the length of the payload in bytes. The packet includes protocol fields such as start and end bytes, packet counter, sender and receiver IDs, packet type, packet length, payload and checksum.

The payload is the input array of bytes. The checksum is calculated by summing all the bytes in the packet except for the last byte and taking the least significant byte of the sum. The packet is then transmitted over I2C to MCU2.

The second function, receivePacket, is responsible for receiving a packet of data over I2C from MCU1. The function takes input the number of bytes received over I2C. Upon receiving a packet, the function verifies its validity by checking the start and end bytes and the checksum. If the packet is valid, the payload data is extracted from the packet and converted into an array of floats. These float values are then printed to the serial monitor for further processing.

BIBLIOGRAPHY

1. ARDUINO: *Arduino MEGA*. The oficial site of Arduino's modules, ©2020 [citat30.03.2020]. Disponibil: https://www.arduino.cc/.
2. THE ENGINEERING PROJECTS: Circuit Designing. Site of free project in arduino rasberry pi and other good information, ©2020 [citat 02.03.2020]. Disponibil: https://www.theengineeringprojects.com.
3. INSTRUCTABLES: *Instructables*. Circuit Designing. Site of online courses, ©2023 [citat 01.02.2017]. Disponibil: https://www.instructables.com.
4. ELECTRONIC LOVERS: *How to use virtual terminal in proteus*. Site for providing information ,publishing tutorials on electronic circuit and technology news, ©2023 [citat 13.04.2015]. Disponibil: https://electronicslovers.com.
5. ARDUINO: *Constrain*. The oficial site of Arduino's modules, ©2020 [citat 30.03.2020]. Disponibil: https://cdn.arduino.cc/reference/en/language/functions/math/constrain/.
6. ROBO INDIA: *Arduino Frequency Meter*. Site of free project in arduino rasberry pi and other goodinformation in embeded and robotic fields, ©2023 [citat 04.06.2019].
7. Complete Guide for HC-SR04 Ultrasonic Sensor with Arduino: https://www.circuitgeeks.com/hc-sr04-ultrasonic-sensor-with-arduino/