Report 6

Subject: Embedded Systems
Topic: Communication

Student: Marusic Diana, FAF-171
Teacher: Bragarenco Andrei

Content

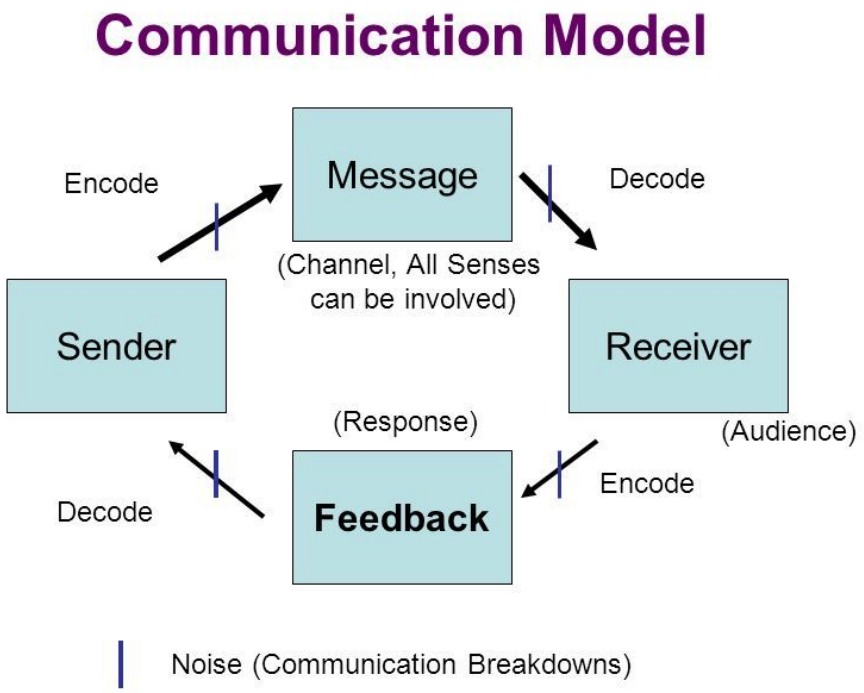
Domain

Communication

Data communications refers to the transmission of digital data between two or more computers/devices and a computer network or telecommunications network that allows computers/devices to exchange data. The physical connection between networked computing devices is established using either cable media or wireless media. The best-known computer network is the Internet.

Components of communication:

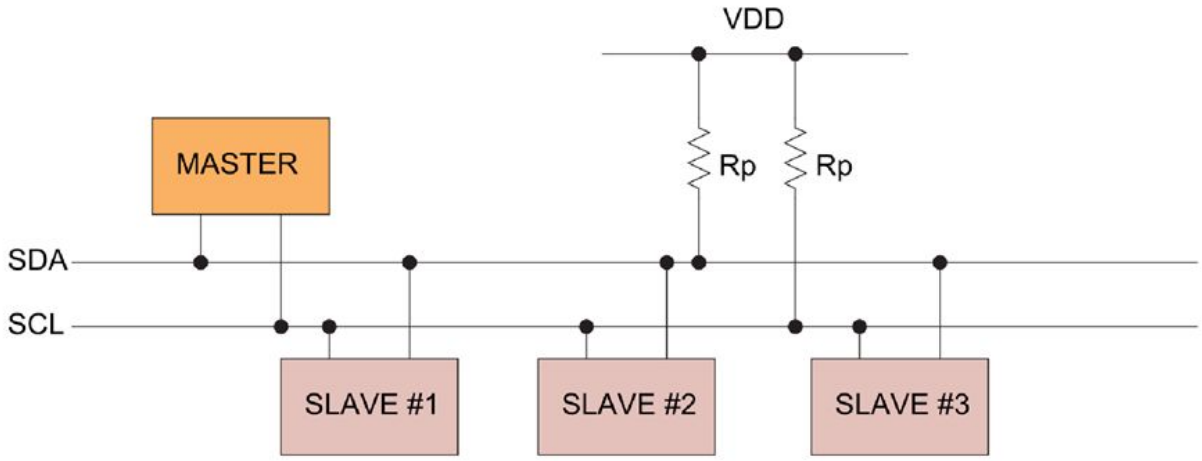
- A **sending device** that initiates an instruction to transmit data, instructions, or information.
- A communications device that connects the sending device to a communications channel.
- A **communications channel**, or transmission media on which the data, instructions, or information travel.
- A communications device that connects the communications channel to a receiving device.
- A **receiving device** that accepts the transmission of data, instructions, or information.



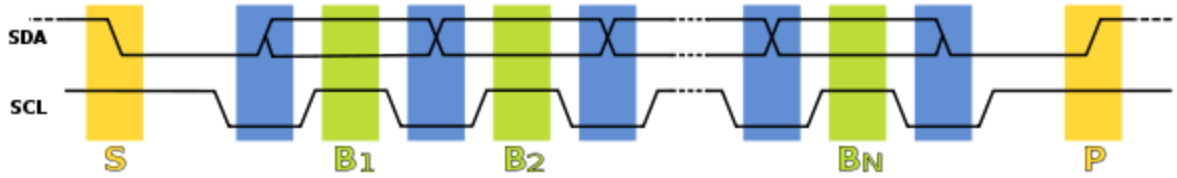
Serial port - communication

In telecommunication and data transmission, serial communication is the process of sending data one bit at a time, sequentially, over a communication channel or computer bus. This is in contrast to parallel communication, where several bits are sent as a whole, on a link with several parallel channels.
In Arduino, the Serial interface is used for communication between the Arduino board and a computer or other devices. All Arduino boards have at least one serial port (also known as a UART or USART), and some have several.

I2C communication



I2C is a serial protocol for two-wire interface to connect low-speed devices like microcontrollers, EEPROMs, A/D and D/A converters, I/O interfaces and other similar peripherals in embedded systems. Each I2C slave device needs an address – they must be obtained from NXP (formerly Philips semiconductors).



I2C communication protocol follows a master/slave hierarchy, wherein the master is defined as the device that clocks the bus, addresses slaves and writes or reads data to and from registers in the slaves. The slaves are devices that respond only when interrogated by the master, through their unique address. Hence it is imperative to avoid duplication of addresses among slaves. Slaves never initiate a data transfer.

The I2C bus uses only two bidirectional lines, Serial Data Line (SDA) and a Serial Clock Line (SCL). I2C compatible devices connect to the bus with open collector or open drain pins which pull the line LOW. When there is no transmission of data the I2C the bus lines idle in a HIGH state; the lines are passively pulled high.

Transmission occurs by toggling the lines by pulling LOW and releasing HIGH. Bits are clocked on falling clock edges. The standard data transfer rate is 100kbits/s while the Fast Mode transfer rate is 400kbits/s

Basic I2C communication is using transfers of 8 bits or bytes. Each I2C slave device has a 7-bit address that needs to be unique on the bus.

In normal state both lines (SCL and SDA) are high. The communication is initiated by the master device. It generates the Start condition (S) followed by the address of the slave device (B1). If the bit 0 of the address byte was set to 0 the master device will write to the slave device (B2). Otherwise, the next byte will be read from the slave device. Once all bytes are read or written (Bn) the master device generates Stop condition (P). This signals to other devices on the bus that the communication has ended and another device may use the bus.

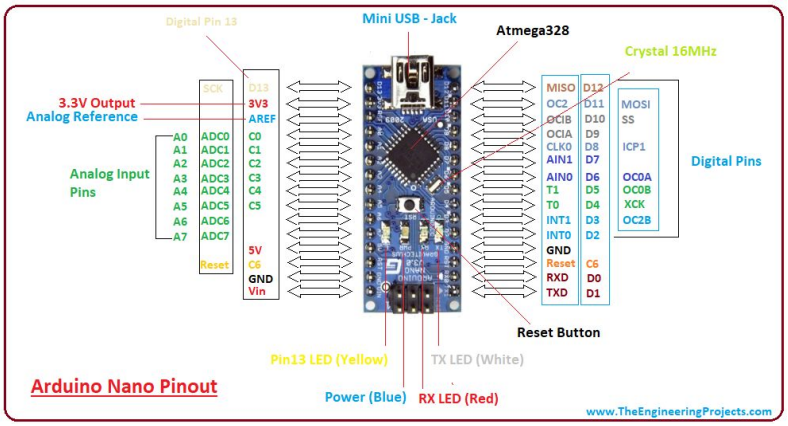
Most I2C devices support repeated start condition. This means that before the communication ends with a stop condition, the master device can repeat the start condition with address byte and change the mode from writing to reading.

Component description

HC-SR04 ultrasonic sensor

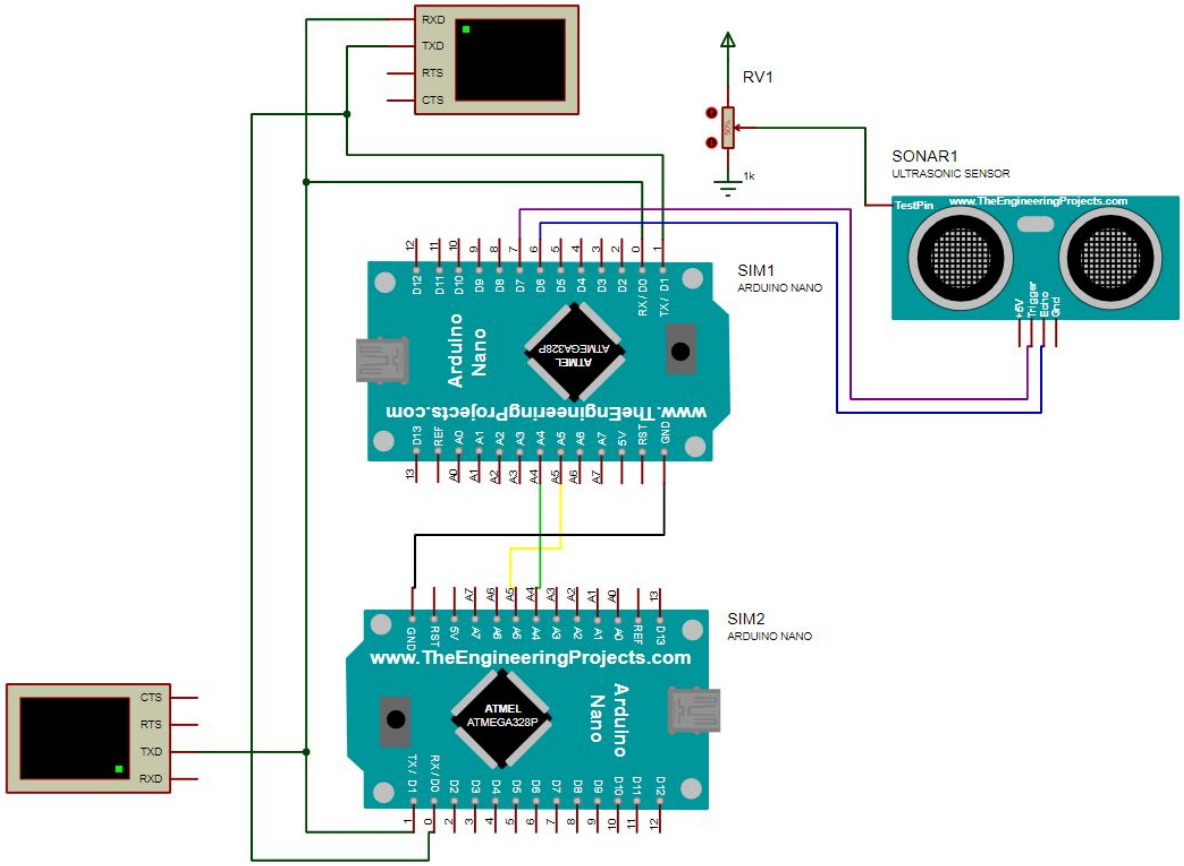
The HC-SR04 ultrasonic sensor uses sonar to determine distance to an object like bats do. It offers excellent non-contact range detection with high accuracy and stable readings in an easy-to-use package. It comes complete with ultrasonic transmitter and receiver modules.

Arduino Nano



The **Arduino Nano** is a small, complete, and breadboard-friendly board based on the ATmega328P (Arduino Nano 3. x). It has more or less the same functionality of the Arduino Duemilanove, but in a different package. It lacks only a DC power jack, and works with a Mini-B USB cable instead of a standard one. It has 14 digital pins (D0-D13) and 8 analog input pins (D0-D7). Its small dimensions make it a good board for compact projects.

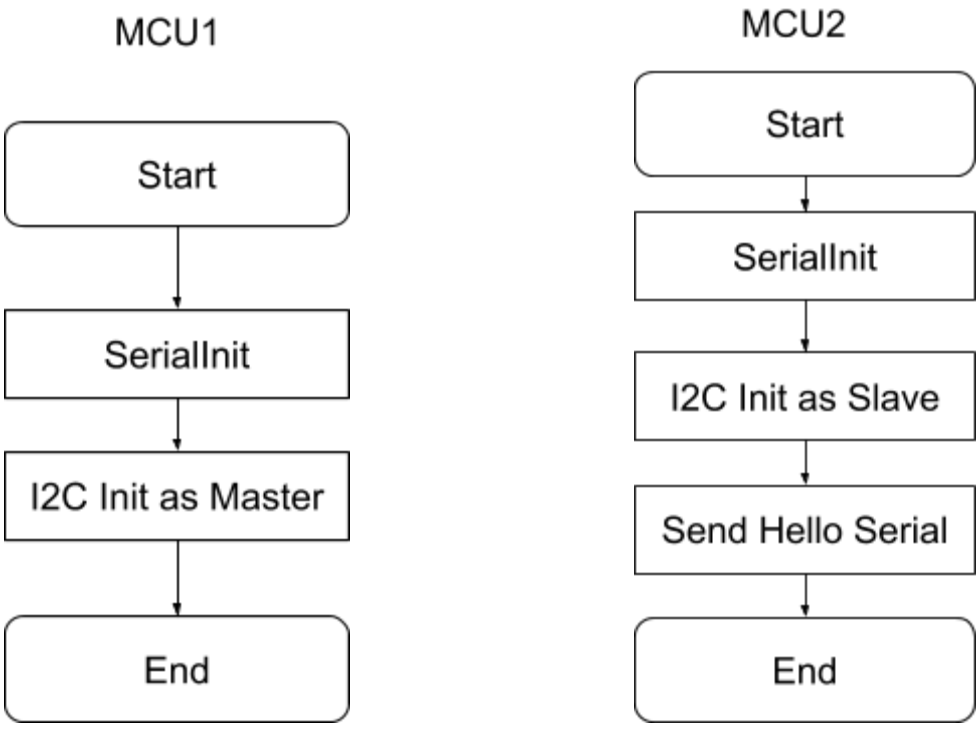
Implementation



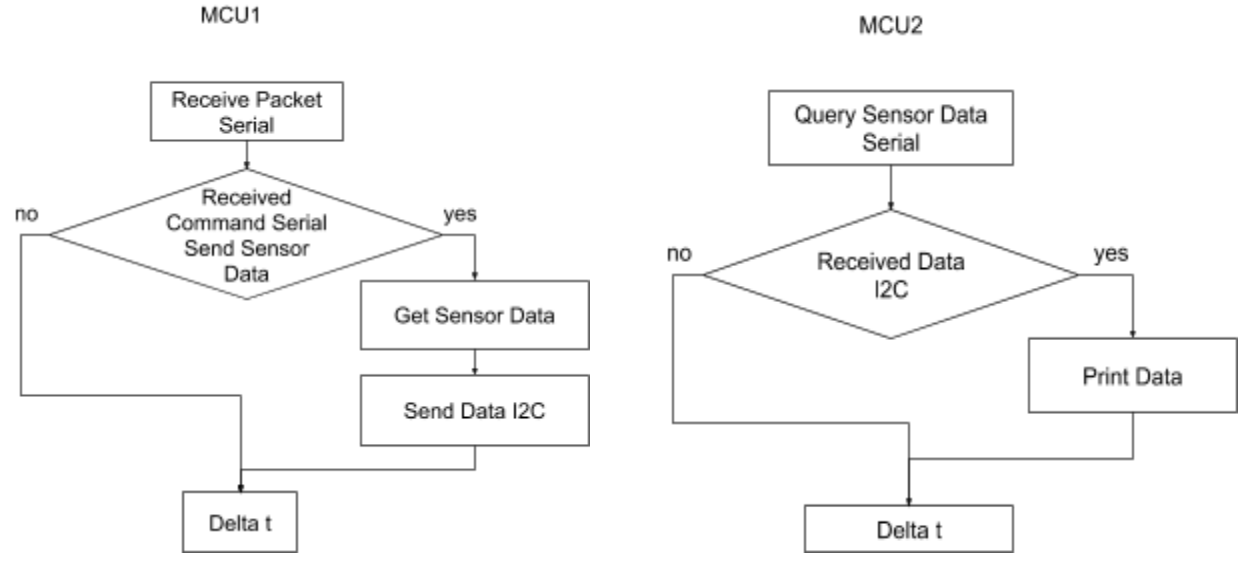
The circuit was assembled according to the scheme above, but also one Arduino nr 1 was connected to the computer and from Arduino 1 a wire was connected from 5V to Arduino 2 VIN pin, to assure enough power. Arduino 1 was connected to Arduino 2 via serial port by connecting the TX pin of one controller to the RX pin of the second controller and vice versa (RX to TX). Also, for I2C communication, connect the A4 pin of both controllers together and A5 pin of the first controller to the A5 pin of the second controller. A4 pin is the SDA(data) wire of I2C port and A5 pin is the SCL(clock) for the I2C interface.
The Ultrasonic sensor (hc-sr04) was connected to the 6 (echo) and 7 (trig) pins of the Arduino 1. In this way, by using Arduino 1, it was possible to transform the ultrasonic sensor to an I2C interface sensor.

The data was passed from one Arduino to another by following the scheme:

Setup:


MCU1
Start
SerialInit
I2C Init as Master
End
MCU2
Start
SerialInit
I2C Init as Slave
Send Hello Serial
End

Loop:

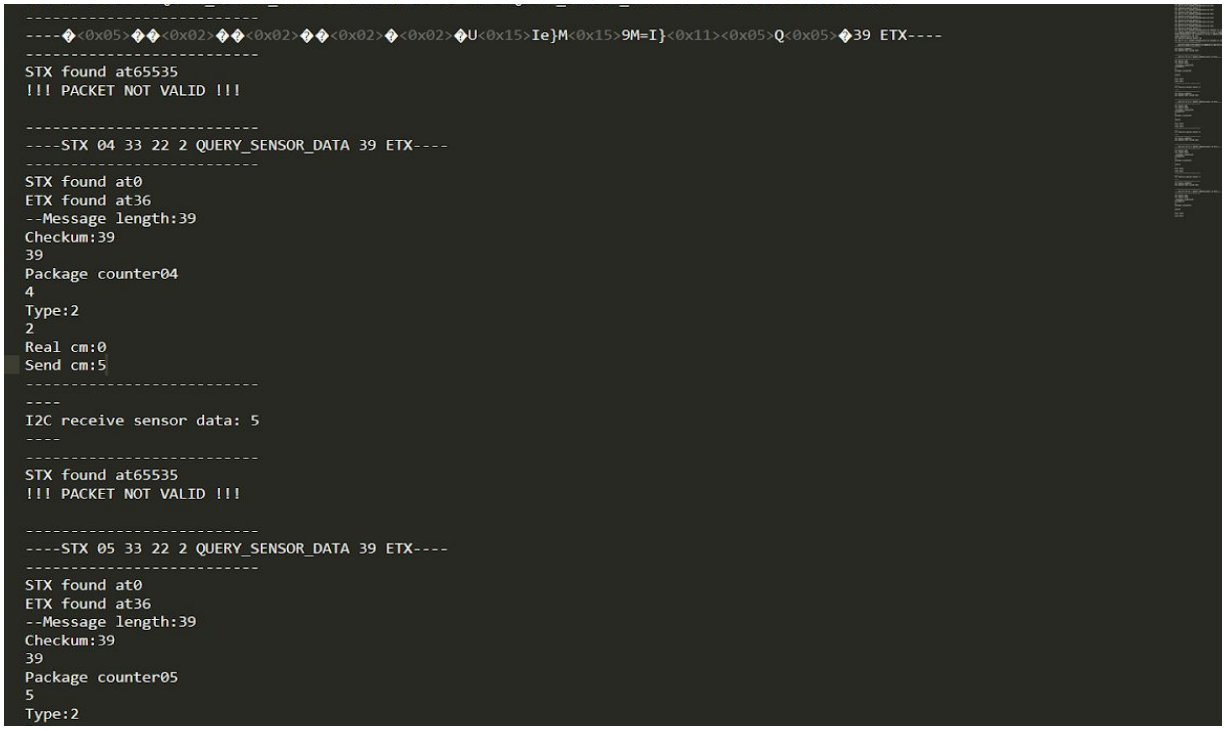

MCU1
Receive Packet Serial
Received Command Serial Send Sensor Data
no
yes
Get Sensor Data
Send Data I2C
Delta t
MCU2
Query Sensor Data Serial
Received Data I2C
no
yes
Print Data
Delta t

Proof

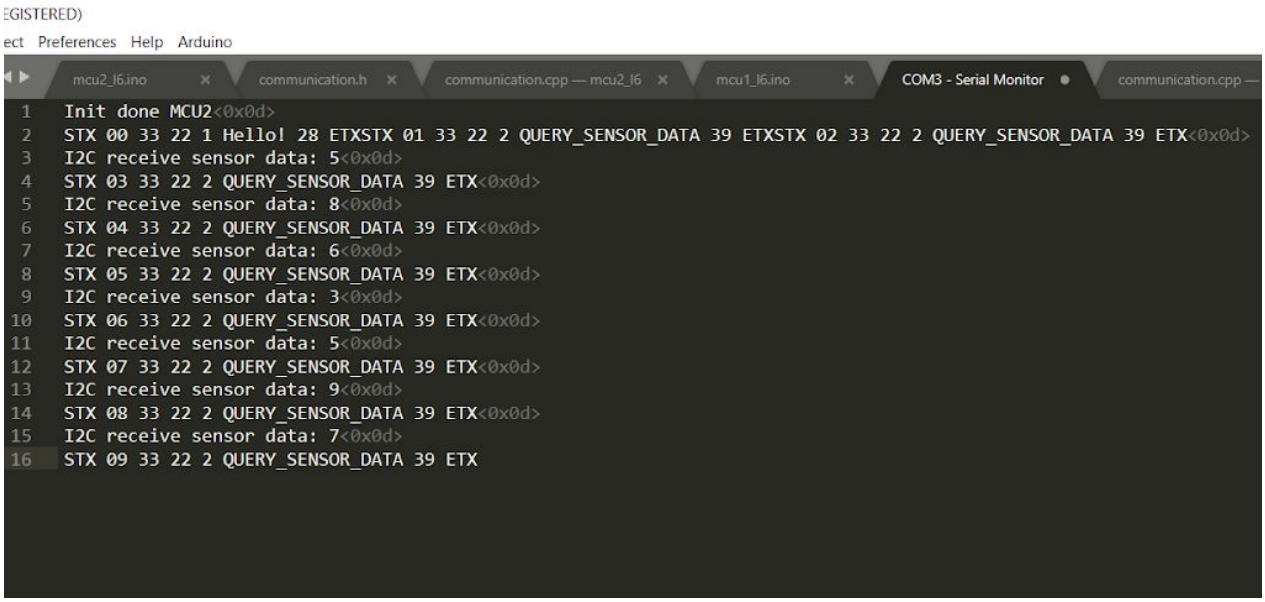
Video proof on real circuit:

https://www.youtube.com/watch?v=fQawkNxjYn8

MCU1 Output example:



MCU2 Output example:



Annex 1 - source code for Arduino 1

Main file

```
// Master I2C

#include <Arduino.h>
#include "communication.h"
#include "myi2c.h"

const int pingPin = 7; // Trigger Pin of Ultrasonic Sensor
const int echoPin = 6; // Echo Pin of Ultrasonic Sensor


long microsecondsToCentimeters(long microseconds)
{
   return microseconds / 29 / 2;
}


void setup() {
        CommunicationInit();
    InitI2C();
        Serial.println("Initialization done");
}

int GetSensorData() {
  long duration, inches, cm;
    pinMode(pingPin, OUTPUT);
    digitalWrite(pingPin, LOW);
    delayMicroseconds(2);
    digitalWrite(pingPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(pingPin, LOW);

    pinMode(echoPin, INPUT);
    duration = pulseIn(echoPin, HIGH);
    cm = microsecondsToCentimeters(duration);
```

```
    Serial.print("Real cm:");
    Serial.print(cm);
    Serial.println();

    cm += random(1, 10);
    Serial.print("Send cm:");
    Serial.print(cm);
    Serial.println();

    // Simulation without sensor:
    return cm;
}

void loop() {
          int command = ReceivePacket();
          if(command == COMMAND_SEND_SENSOR_DATA) {
                    int cm = GetSensorData();
        SendI2C(cm);
  }
          delay(1000);
}

-------------------------
communication.cpp
-------------------------#include "communication.h"
#include <Arduino.h>
#include <stdlib.h>
#include <Stream.h>
#include <string.h>


// Package structure:
/*
start package indicator
package counter (nr)
ID sender
ID receiver
type
```

```
<optional fields>
payload
checksum
end indicator
*/

// package number
uint8_t p_nr = 0;

String mystr;

void CommunicationInit() {
        Serial.begin(9600);
}

int calculateChecksum(char * message, uint8_t type, uint8_t p_nr) {
        // return 13 + length(MY_ID) + length(RECEIVER_ID) + length(message) +
length(type) + length(checksum);
        return 22 + strlen(message);
}

void SendPacket(char * message, uint8_t type) {
        Serial.print("STX");
        Serial.print(" ");
        if(p_nr < 10) {
                Serial.print("0");
        } else if(p_nr >99) {
   p_nr = 0;
  }
        Serial.print(p_nr++);
        Serial.print(" ");
  // Sender ID
        Serial.print(MCU2_ID, HEX);
  // Receiver ID
        Serial.print(" ");
        Serial.print(MCU1_ID, HEX);

        Serial.print(" ");
        Serial.print(type);
```

```
        Serial.print(" ");
        Serial.print(message);
        Serial.print(" ");

        int checksum = calculateChecksum(message, type, p_nr);
        Serial.print(checksum);

        Serial.print(" ");
        Serial.print("ETX");
}


int ReceivePacket() {
   int validPacket = 1;
   if(Serial.available() ){
   mystr = Serial.readString(); //Read the serial data and store in var
   Serial.println("-------------------------");
   Serial.println("----"+mystr+"----");
   Serial.println("-------------------------");

        size_t stx_pos = mystr.indexOf("STX");
        Serial.print("STX found at");
        Serial.print(stx_pos);

        if(stx_pos>2) {
                validPacket = 0;
     Serial.println();
                Serial.println("!!! PACKET NOT VALID !!!");
     Serial.println();
                return -1;
        }
   Serial.println();
   size_t etx_pos = mystr.indexOf("ETX");
   Serial.print("ETX found at");
   Serial.println(etx_pos);

   int messageLen = etx_pos+3;
   Serial.print("--Message length:");
   Serial.println(messageLen);
```

```
  if(messageLen-6 <0) {
        validPacket = 0;
        Serial.println("PACKET NOT VALID");
        return -1;
  }

  String checksum_s = mystr.substring(messageLen-6, messageLen-4);
  Serial.print("Checkum:");
  Serial.println(checksum_s);
  int checksum = checksum_s.toInt();
  Serial.println(checksum);

 if(checksum != messageLen) {
  validPacket = 0;
  Serial.println("PACKET NOT VALID");
  return -1;
 }

 String packageCnt = mystr.substring(4, 6);
 Serial.print("Package counter");
 Serial.println(packageCnt);
 Serial.println(packageCnt.toInt());

 String type_s = mystr.substring(13, 15);
 Serial.print("Type:");
 Serial.println(type_s);
 int type = type_s.toInt();
 Serial.println(type);
 return type;
 }
 else {
  return -2;
 }
}
```

---
communication.h

```
#ifndef COMMUNICATION_H_
#define COMMUNICATION_H_

#include <stdlib.h>
#include <Arduino.h>

#define MCU1_ID 0x22
#define MCU2_ID 0x33

#define TYPE_HELLO 1
#define TYPE_QUERY_DATA_I2C 2
#define COMMAND_SEND_SENSOR_DATA 2
#define SENSOR_DATA_CM 3

void CommunicationInit();
void SendPacket(char * message, uint8_t type);
int ReceivePacket();
#endif
```

---
myi2c.cpp

```
#include "myi2c.h"


void InitI2C() {
	Wire.begin();
}

void SendI2C(int value) {
	Wire.beginTransmission(9); // transmit to device #9
	Wire.write(value);		// sends x
	Wire.endTransmission();
}
```

---
myi2c.h

```
#ifndef MYI2C_H_
#define MYI2C_H_

#include <Arduino.h>
#include <Wire.h>


void InitI2C();
void SendI2C(int value);

#endif
```

Annex 2 - source code for Arduino 2

Main file

```
// Slave I2C

#include <Arduino.h>
#include "communication.h"
#include <Wire.h>
int sensorData;

void InitI2C() {
	Wire.begin(9);
	// Attach a function to trigger when something is received.
	Wire.onReceive(ReceiveEvent);
}

void setup() {
	CommunicationInit();
	InitI2C();
```

```
	Serial.println("Init done MCU2");
	delay(1000);
	SendHello();

	int data = ReceiveData();
	if(data != -1) {
		Serial.print("Received data:");
		Serial.println(data);
	}
}


void ReceiveEvent(int bytes) {
	sensorData = Wire.read();    // read one character from the I2C
	Serial.println();
	Serial.print("I2C receive sensor data: ");
	Serial.print(sensorData);
	Serial.println();
}

void loop() {
	// cerere prin serial de date de la MCU1
	QuerySensorData();
	delay(5000);
}
```

---

communication.cpp

```
#include "communication.h"
#include <Arduino.h>
#include <stdlib.h>
#include <Wire.h>

// Package structure:
/*
start package indicator
package counter (nr)
```

```
ID sender
ID receiver
type
<optional fields>
payload
checksum
end indicator
*/

// package number
uint8_t p_nr = 0;

int data = -1;

void CommunicationInit() {
        Serial.begin(9600);
}

int calculateChecksum(char * message, uint8_t type, uint8_t p_nr) {
   return 22 + strlen(message);
}

 void SendPacket(char * message, uint8_t type) {
  // start indicator
  Serial.print("STX");
  Serial.print(" ");
  if(p_nr < 10) {
    Serial.print("0");
  } else if(p_nr >99) {
    p_nr = 0;
  }
  Serial.print(p_nr++);
  Serial.print(" ");
  // Sender ID
  Serial.print(MCU2_ID, HEX);
  // Receiver ID
  Serial.print(" ");
  Serial.print(MCU1_ID, HEX);
```

```
  Serial.print(" ");
  // Serial.print(QUERY_DATA_TYPE);
  Serial.print(type);
  Serial.print(" ");
  Serial.print(message);
  Serial.print(" ");

  int checksum = calculateChecksum(message, type, p_nr);
  Serial.print(checksum);

  Serial.print(" ");
  Serial.print("ETX");
}


// MCU2 sends query to MCU1 to send data
void QuerySensorData() {
        SendPacket("QUERY_SENSOR_DATA", TYPE_QUERY_DATA_I2C);
}

void SendHello() {
        SendPacket("Hello!", TYPE_HELLO);
}



int ReceiveData() {
        return data;
}


--------------------------
communication.h
--------------------------

#ifndef COMMUNICATION_H_
#define COMMUNICATION_H_

#define MCU1_ID 0x22
```

```
#define MCU2_ID 0x33

#define TYPE_HELLO 1
#define TYPE_QUERY_DATA_I2C 2

extern int data;

void CommunicationInit();
void QuerySensorData();
int ReceiveData();
void SendHello();


#endif
```