Hochschule Konstanz
Department of Computer Science

Josef Müller
Ubiquitous Computing Lab

Degree: Applied Informatics (AIN)
Lecture: Ubiquitous Computing

Report for the Laboratory 2:
Communication between IoT Devices

1 General Task

In this task, the different I/O devices should be experimented with. Arduino offers various interfaces, such as I2C, SPI or Serial. The interface must of course be adapted to the use case. Based on previous experience, the serial interface seems to be the simplest, most uncomplicated and most transparent, since, for example, with the help of the "SoftwareSerial" library, most ports can be converted to serial interfaces without much effort.

As already mentioned, the interfaces must be adapted to the use case. The Arduino offers the possibility to connect numerous input and output devices. Be it simple LEDs or more complex screens, LED strips, thermometers, LED matrices, motors or buttons.

In this general task, it is to be shown how such a device can be controlled. The LED matrix was chosen. The task is to display the letter "U".

1.1 Implementation

LED matrices can be controlled in different ways - some simpler, others more complicated. There are controllers that simplify the control. One example is the MAX7219, which can control an 8x8 matrix via SPI. If the right libraries, such as those from "MD", are used, several matrices can even be connected in series.

In this case, however, it remains a simple 8x8 matrix.

Before the setup the display is initialized globally. With the help of "MD_Parola" the screen can be controlled. The library is specially designed to display words and numbers. In the set-up process the screen is configured. In the loop method the letter is written.

The example shown here is very simple. A use case for this would be, for example, a smartwatch that shows information only via individual letters: "W" as in WhatsApp or "M" as in Mail. Or it could display the time in a very minimalistic way like binary or via the number of bright pixels.

1.1.1 Code

```
// Including the required Arduino libraries
#include <MD_Parola.h>
#include <MD_MAX72xx.h>
#include <SPI.h>

// Uncomment according to your hardware type
//#define HARDWARE_TYPE MD_MAX72XX::FC16_HW
#define HARDWARE_TYPE MD_MAX72XX::GENERIC_HW

// Defining size, and output pins
#define MAX_DEVICES 1
#define CS_PIN 10

// Create a new instance of the MD_Parola class with hardware SPI connection
MD_Parola myDisplay = MD_Parola(HARDWARE_TYPE, CS_PIN, MAX_DEVICES);

void setup() {
  // Intialize the object
  myDisplay.begin();
  myDisplay.setIntensity(15);
  myDisplay.displayClear();
}

void loop() {
  myDisplay.setTextAlignment(PA_CENTER);
  myDisplay.print("U");
  delay(500);
}
```

1.2 Result

As can be seen in the figure, the letter "U" is displayed. The control of the matrix is done by the controller "MAX7219", which is connected to the Arduino via SPI.

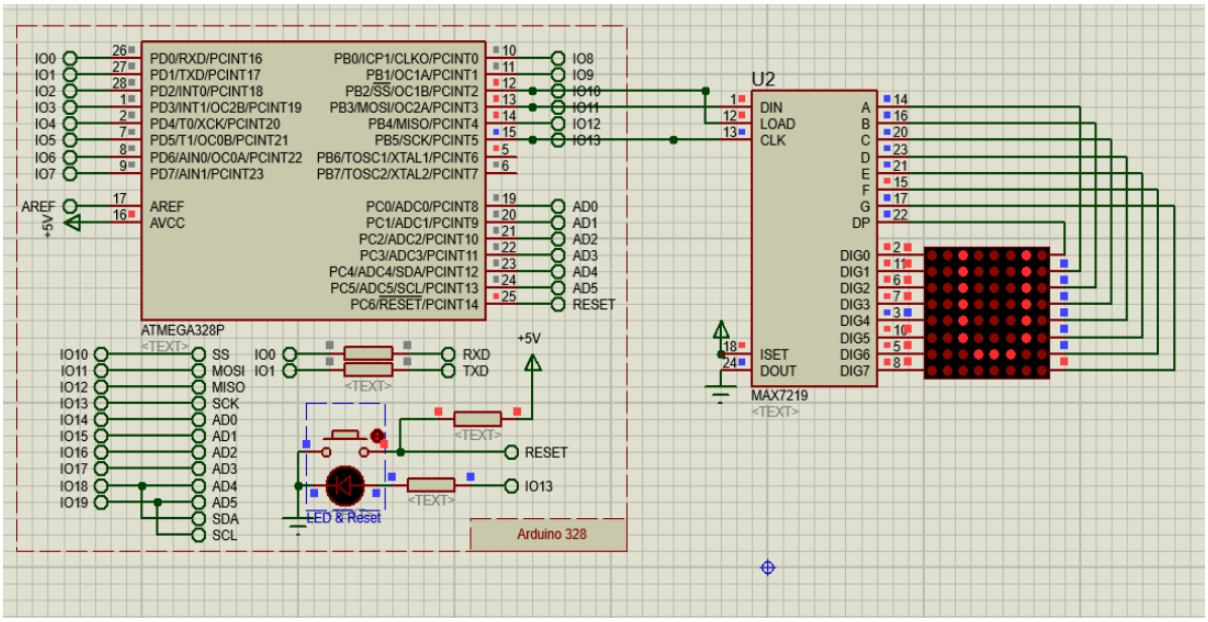

Figure 1: LED Matrix that displays the letter "U"

2 Individual Task

2.1 Explanation

In this task, two Arduinos should communicate with each other. There are different interfaces that can be used for communication. In this case, the serial interface should be used to send commands from one Arduino to another Arduino.

When information is transmitted from one instance to another, for example from the router to the device, it must be checked for correctness after arrival, otherwise it can lead to undesired behaviour.

A simple example: A boss tells an employee what to do. Since misunderstandings or missing information can occur, the employee, after receiving his tasks, asks his boss if he understood everything correctly. When data is transmitted, it is therefore ensured that it is complete. This is called a checksum. This is calculated from the sent data and is sent along. The recipient decrypts the information and uses the checksum to check whether the data is complete.

In this task, the checksum CRC8 (Cyclic Redundancy Checksum) should be used. The procedure is based on polynomial division. This means that the binary numbers can be represented as polynomials. The calculation is done by stepwise XOR.

First, the data are extended by the length of a so-called generator polynomial. Then the polynomial division takes place, in which the data is divided by the polynomial as a bit sequence. The remainder forms the CRC value, which is appended to the data block.

In order for the receiver to check the completeness of the data, it performs the same steps. Since the remainder is appended, the result of the procedure should be zero. This ensures that the data is complete. If the result is different, an error occurred during transmission.

2.2 Implementation

2.2.1 Client

The two Arduinos have a master-slave or client-server relationship. The client or master gives the slave or server tasks to process. For the sake of simplicity, we will only refer to the client here. The client receives commands via a virtual terminal. The data transfer to the virtual terminal takes place via a serial interface. Since the Arduino used here only has one serial interface, the use of the "SoftwareSerial" library was a remedy. With the help of this library, almost all PINs of the Arduino can be used for a serial interface. As soon as the serial instance is created, it can be used like the "Serial" instance: "readBytes", "println", etc. can be used without further ado.

The standard serial interface was used for communication to the slave. To transmit the data to the slave, it is first entered via the virtual terminal, which communicates via the serial interface instance "Input".

Once the data is read in, it is trimmed and parsed using a simple string comparison. Depending on the fulfilled condition of the command word, one from 11 byte-arrays is selected and copied in using "memcpy(...)". The fact that a byte array is used as encoding is necessary because the calculation of the CRC value expects a byte array and also returns a byte. The calculated value is written to the penultimate position of the array. The last position for the terminator character.

Afterwards, the finished array is sent to the slave via the serial interface after a cast to the string.

2.2.1.1 Code

```
// code for client (sender)
#include <stdio.h>
#include <string.h>
#include <SoftwareSerial.h>

SoftwareSerial Input(3, 2); // RX -> TX, TX -> RX

byte CRC8(const uint8_t *data, uint8_t len) {
  const uint8_t polynome = 0xD5;
  uint8_t crc = 0x00;
  while (len--) {
    crc ^= *data++;
    for (uint8_t i = 8; i; i--) {
      if (crc & 0x80) {
        crc = (crc << 1) ^ polynome;
      } else {
        crc <<= 1;
      }
    }
  }
  return crc;
}

const int BUFFER_SIZE = 8;
byte cmdAsByte[BUFFER_SIZE + 2]; // +1 for CRC // +1 for \0

byte cmds[11][9] = {
  {0x30, 0x30, 0x30, 0x31, 0x30, 0x30, 0x30, 0x30, 0x00}, // off
  {0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x00}, // 0
  {0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x31, 0x00}, // 1
  {0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x31, 0x30, 0x00}, // ...
  {0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x31, 0x31, 0x00},
  {0x30, 0x30, 0x30, 0x30, 0x30, 0x31, 0x30, 0x30, 0x00},
  {0x30, 0x30, 0x30, 0x30, 0x30, 0x31, 0x30, 0x31, 0x00},
  {0x30, 0x30, 0x30, 0x30, 0x30, 0x31, 0x31, 0x30, 0x00},
  {0x30, 0x30, 0x30, 0x30, 0x30, 0x31, 0x31, 0x31, 0x00},
  {0x30, 0x30, 0x30, 0x30, 0x31, 0x30, 0x30, 0x30, 0x00},
  {0x30, 0x30, 0x30, 0x30, 0x31, 0x30, 0x30, 0x31, 0x00},
};

void setup() {
  Serial.begin(9600); // send
  Input.begin(9600); // receive
}

void loop() {
  // check if sending is possible
    delay(1);
    // read from virt terminal
    if (Input.available() > 0) {
      // write bytes into buffer
      Input.println("Zu Befehl, Sir!");
```

```
      String cmdAsString = Input.readStringUntil('\n');
      cmdAsString.trim(); // is important
      Input.println(cmdAsString);

      // parse cmd
      if (cmdAsString == "off") {
        memcpy(cmdAsByte, cmds[0], 9);
      } else {
        uint8_t number = cmdAsString.substring(11, 12).toInt();
        memcpy(cmdAsByte, cmds[number + 1 /* because -1 is at index 0 */], 9);
      }

      uint8_t crc = CRC8(cmdAsByte, 8);
      Input.println(String(crc));
      cmdAsByte[8] = (char) crc;
      // Input.println(String(crc));
      Input.println((char *) cmdAsByte);
    }

    Serial.println((char *) cmdAsByte); // byte array to char array
}
```

2.2.2 Server

After the data has been sent by the client, it takes some time for it to arrive at the server via the serial interface. The server reads from the interface every 500 ms and thus checks whether data are present. If this is the case, the CRC algorithm is applied to them.

In order to have the data better prepared, bytes are read instead of a string. In addition, the sent data always have the same length, and thus exactly 9 bytes can be read. If the calculated CRC value is 0, the data is complete. The reason for this lies in the meaning of the CRC value. It forms the remainder of the polynomial division. So if the remainder is added to the data, the division can be done without a remainder. This cannot be compared to a normal division with numbers, because the polynomial division returns the remainder as a polynomial.

Afterwards, the received message is terminated at the 8th position for the following string comparison using "strcmp(...)" - everything above this, such as the CRC value, is no longer important. To parse the message, iterate through the array of byte arrays and compare each byte array with the message. This works because the message is also a byte array and thus the string comparison after a cast to (char *) works without further ado.

As soon as message and byte array are identical, the index is used to execute the corresponding commands. If the index is zero, the LED matrix is switched off. This simply resets the matrix. All other values are displayed on the matrix.

The matrix was taken over from the last delivery and is controlled by the "MD_Parola" library. A string value can be passed to the "MD_Parola" instance, which is displayed on the matrix without any problems. The matrix is controlled via SPI, a serial interface.

2.2.2.1 Code

```
// code for server (receiver)
#include <SoftwareSerial.h>
#include <MD_Parola.h>
#include <MD_MAX72xx.h>
#include <SPI.h>

// Uncomment according to your hardware type
//#define HARDWARE_TYPE MD_MAX72XX::FC16_HW
#define HARDWARE_TYPE MD_MAX72XX::GENERIC_HW

// Defining size, and output pins
#define MAX_DEVICES 1
#define CS_PIN 10

// Create a new instance of the MD_Parola class with hardware SPI connection
MD_Parola myDisplay = MD_Parola(HARDWARE_TYPE, CS_PIN, MAX_DEVICES);


SoftwareSerial Output(9, 8); // RX -> TX, TX -> RX

byte CRC8(const uint8_t *data, uint8_t len) {
  const uint8_t polynome = 0xD5;
  uint8_t crc = 0x00;
  while (len--) {
    crc ^= *data++;
    for (uint8_t i = 8; i; i--) {
      if (crc & 0x80) {
        crc = (crc << 1) ^ polynome;
      } else {
        crc <<= 1;
      }
    }
  }
  return crc;
}

const uint8_t CMD_LENGTH = 10; // 8 + 2
byte Message[CMD_LENGTH];

// same as client
byte cmds[11][9] = {
  {0x30, 0x30, 0x30, 0x31, 0x30, 0x30, 0x30, 0x30, 0x00}, // off
  {0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x00}, // 0
  {0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x31, 0x00}, // 1
  {0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x31, 0x30, 0x00}, // ...
  {0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x31, 0x31, 0x00},
  {0x30, 0x30, 0x30, 0x30, 0x30, 0x31, 0x30, 0x30, 0x00},
  {0x30, 0x30, 0x30, 0x30, 0x30, 0x31, 0x30, 0x31, 0x00},
  {0x30, 0x30, 0x30, 0x30, 0x30, 0x31, 0x31, 0x30, 0x00},
  {0x30, 0x30, 0x30, 0x30, 0x30, 0x31, 0x31, 0x31, 0x00},
  {0x30, 0x30, 0x30, 0x30, 0x31, 0x30, 0x30, 0x30, 0x00},
  {0x30, 0x30, 0x30, 0x30, 0x31, 0x30, 0x30, 0x31, 0x00},
};
```

```
void setup() {
  myDisplay.begin();
  myDisplay.setIntensity(15);
  myDisplay.displayClear();
  Serial.begin(9600);
  Output.begin(9600);
}

void loop() {
  if (Serial.available() > 0) {
    Serial.readBytes(Message, 9);
    uint8_t crc_out = CRC8(Message, 9);
    // if crc_out is 0 then cmd was send successfully
    int i;
    if (crc_out == 0) {
      Output.println("Succesfully received cmd");
      Message[8] = 0x00;
      Output.println((char *) Message);
      Output.println("Start comparison");
      for (i = 0; i < 11; i++) {
        uint8_t success = 0;
        // Output.println((char *) cmds[i]);
        if (strcmp((char *) cmds[i], (char *)Message) == 0) {
          Output.println("Success!!");
          break;
        }
      }
      Output.println(String(i));
      if (i == 0) {
        // off
        myDisplay.setIntensity(15);
        myDisplay.displayClear();
      } else {
        myDisplay.print(String(i - 1));
      }
    }
  }

  delay(500);
}
```

2.2.3 Result

The figure shows the wiring of the two Arduinos, the two virtual terminals and the input and output. The upper virtual terminal receives command words and sends them to the Arduino. The latter processes the input and sends the coded command to the slave Arduino via the serial interface. The slave Arduino reads from the interface, checks the received 9-byte long message using CRC and processes it. The processing is done by means of a string comparison. The index of the string determines which command is executed. The LED matrix is then controlled accordingly.

In this example the following command was passed via virtual terminal: “set number 3”. As can be seen number 3 is displayed on the LED matrix.

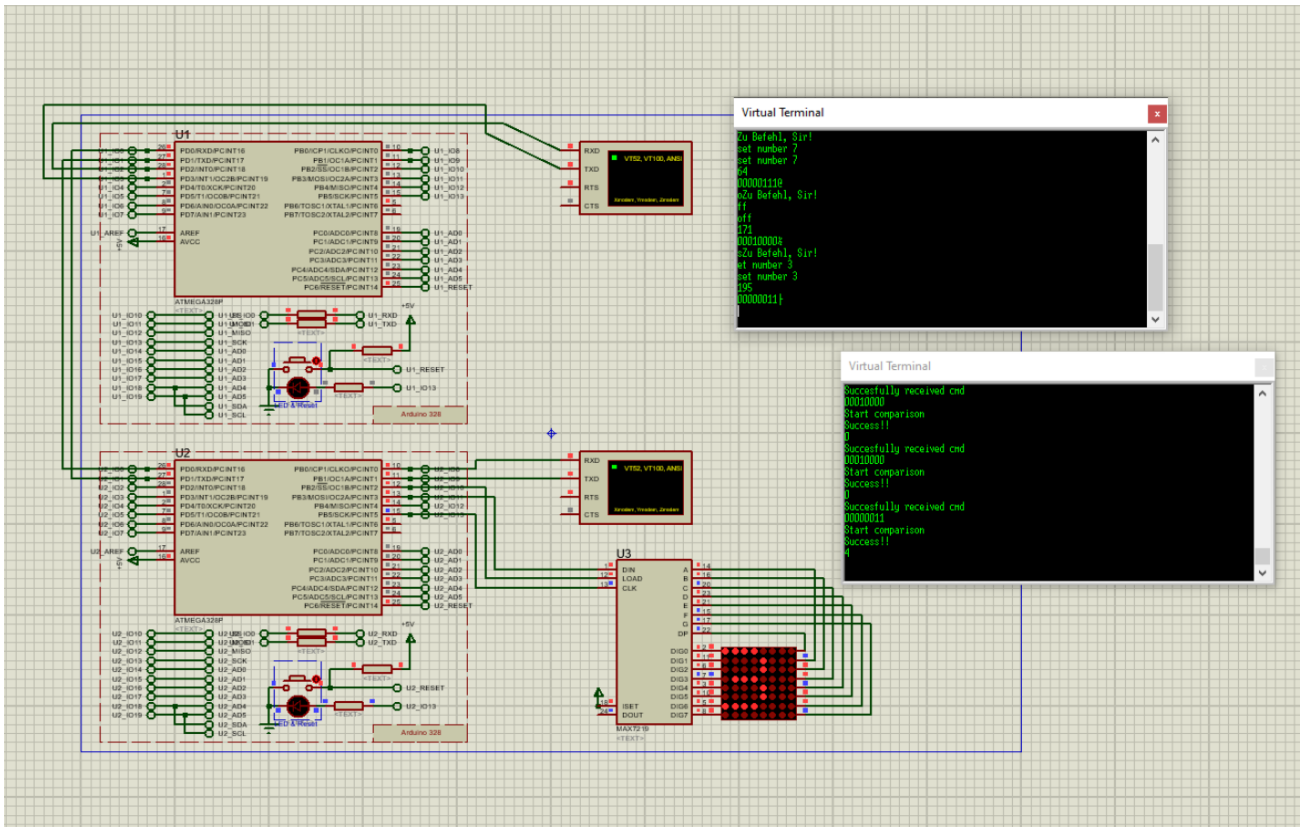

Figure 2: Two Arduinos communicating via serial. The digit 3 is displayed because a corresponding command was passed via terminal