Laboratory 5 – Communication Interfaces

Embedded electronics refers to the interconnection of circuits (micro-processors or other integrated circuits) with the goal of creating a unified system. In order to transfer information, these circuits need to have a common communication mechanism. These mechanics can be categorized in two main classes: **serial and parallel** communication schemes.

Parallel interfaces transfer more bits at a time. They usually require busses to transmit over 8, 16 or more lines. The transmitted and received data represent massive flows of 1s and 0s. In figure 1, one can observe a simple 8-bit data bus, controlled by a clock signal, that can transmit one byte at each clock cycle (9 wires are used).

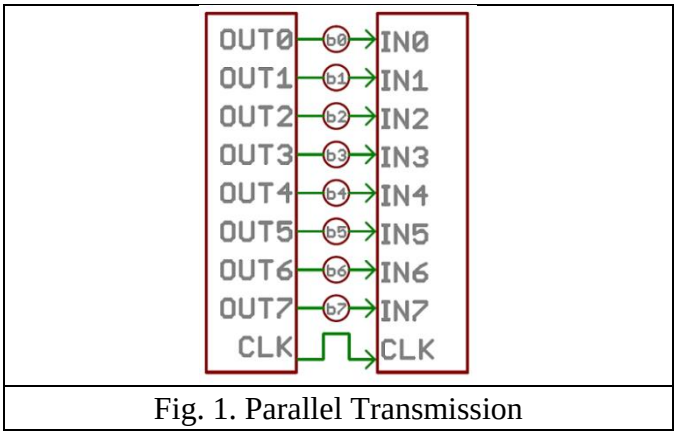

Fig. 1. Parallel Transmission

Serial interfaces send the information bit by bit. These interfaces can operate on a single wire and usually do not need more than 4 wires (minimum 1, maximum 4 wires).

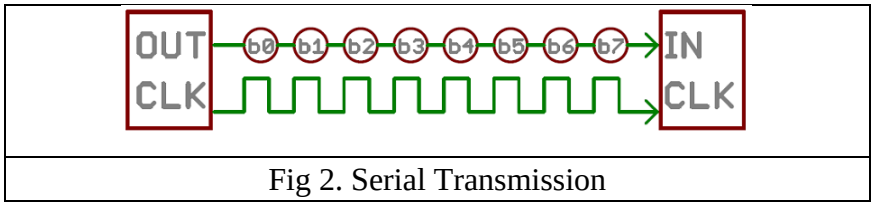

Fig 2. Serial Transmission

In figure 2, one can observe an example of a serial interface that transmits one bit at each clock cycle (2 wires are used). Although the parallel interfaces have their strengths, they require a large number of pins on the used development platform and taking into account that the number of pins on the Arduino UNO/Mega is limited, we will focus on serial communication interfaces.

Another classification used for communication interfaces refers to the communication mechanism: **synchronous or asynchronous**. A synchronous communication interface uses a clock signal at both ends of the communication link (transmitter and receiver). This kind of communication is usually faster, but it requires an extra wire between the communication devices. Examples of such communications are SPI and I2C. Asynchronous communication refers to the fact that data is being transferred without the support of a clock signal. In this manner there is no need for a clock signal, but special attention should be given to the synchronization of the transferred data.

Rules of asynchronous serial transfer – UART

Asynchronous serial transfer employs a mechanism to ensure a robust error free transmission. This mechanism contains the following:

- Baud Rate or transfer rate
- The Data Packet (Data Frame)
- Data Bits – character (data chunk)
- Synchronization Bits
- Parity Bits
- Data line (**in idle state has the logical level "1"**)

The critical part is to ensure that the devices that communicate using the serial bus obey the same communication protocol.

Baud Rate

In serial communication Baud Rate defines how fast is the data transmitted over the serial line. This is expressed in number of bits per second. Baud rate examples: 1200, 2400, 4800, 19200, 38400, 57600, 115200.

Data Packet (Data Frame)

Each block of data (usually a Byte) that is to be transmitted is embedded into a data packet (frame). The data packet is formed by adding synchronization and parity bits to the data that is to be transmitted. Figure 3 illustrates the form of a data packet.

Fig. 3. Data Packet – Data Frame

Data chunk

The most important part of each packet is represented by the data that a packet contains. This part is also named the data chunk, since the data size is not always the same. It is usually called a character. The data size can be between 5 and 9 bits – standard data size is 8 bits. After the data size is chosen, the communication devices must also agree on the **endianness** (which bit is transmitted first, the most significant bit MSB or the least significant bit LSB).

Synchronization Bits

The synchronization bits are special bits transmitted with every character. These are the **start** and **stop** bits, marking the beginning and ending of a data packet. The start bit is always 0. There can exist more than one stop bit. The stop bit is always 1.

Parity Bit

The parity bit assures a very primitive error control mechanism. The parity bit can represent odd parity, even parity, or it can be skipped from the data packet. For producing the parity bit (figure 4) all the bits from the data chunk are aggregated using the "exclusive or" operator. Parity checking is optional, not used so much in practice. It is useful to use the parity bit when transmitting through noisy medium.

$$P_{even} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0$$
$$P_{odd} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1$$

Fig. 4. Parity bit Computation

An asynchronous serial bus contains only 2 wires – one for transmitting the data and one for receiving. So, the components that communicate in serial mode must have 2 pins: receive pin (**RX**) and transmit pin (**TX**), as in figure 5.

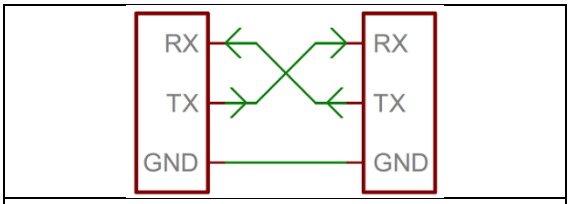

Fig. 5. Serial Connection Scheme

All the Arduino development boards contain at least a **Serial** port (known as UART or USART). Serial communication can be realized through pins 0 (RX) and 1 (TX), but also through the USB interface (the USB interface uses pins 0 and 1 to communicate with the microcontroller). This is why digital pins 0 and 1 must not be used for applications, because losing control on this pins means losing control for board programming.

The Arduino Mega development board contains 3 additional serial ports Serial1 – on pins 19 (RX) and 18 (TX), Serial2 – on pins 17 (RX) and 16 (TX) and Serial 3 – on pins 15 (RX) and 14 (TX).

In our first example in this laboratory we will implement the serial communication between the Arduino board and the PC; displaying the received message on the PC. For this example, we will use the LCD shield mounted on the development boards. The information is sent from the PC and displayed on the LCD. There exist a variety of functions for serial data manipulation. In the previous laboratories we have used a serial communication between the Arduino board and the PC for showing the state of the pressed buttons. The most used serial functions are shown in figure 6. (https://www.arduino.cc/en/Reference/Serial).

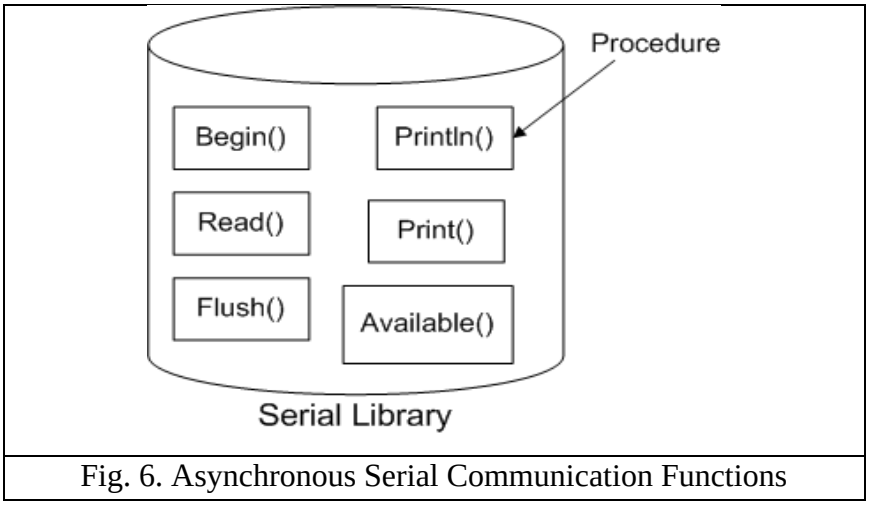

Fig. 6. Asynchronous Serial Communication Functions

Functions **print()** and **println()** from the Serial class send data through the serial port. The difference is that println() sends an additional new line character (‘\n’) and a “carriage return” character (‘\r’) at the end of the transmitted message. For the transmitted numbers you can also specify the format of the data (HEX, DEC, etc.).

Function **begin()** is used to initialize the baud rate of the serial communication. For serial communication with the PC the following baud rates are usually used: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200. Optionally, you can add an additional parameter for configuring the number of bits sent, parity and number of stop bits. The following values are implicit: **8 data bits, no parity and 1 stop bit**.

Function **read()** is used to read the date received from the serial interface. The sintax is the following: **IncomingByte = Serial.read();**

Function **write()** sends a series of bytes over the serial line. For sending numbers it is recommended to use the function **print()**.

The **flush()** instruction waits for the serial transmission to complete.

Function **available()** returns the number of bytes that can be read from the serial port. These data have already been received and are available in the serial receive buffer.

A useful function is **serialEvent().** The function is defined by the user and will be automatically called when new data is ready to be read from the serial receive buffer.

The following example illustrates receiving data from the serial monitor and displaying it on the LCD.

```
//include LCD library
#include <LiquidCrystal.h>
String inputString = "";         // create an empty string to hold the incoming serial data
// Boolean flag to test whether a complete string has been received (enter pressed in serial monitor)
boolean stringComplete = false;
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);

void setup() {
  // initialize serial interface
  Serial.begin(9600); // implicit serial frame format
  //initialize lcd
   lcd.begin(16, 2);
  // reserve 200 bytes for the string
  inputString.reserve(200);
}

void loop() {
  // display the string when new line is received
  if (stringComplete) {
     lcd.setCursor(0, 0);
    lcd.print(inputString);
    Serial.println(inputString);
    // empty the string
    inputString = "";
  // reset the flag
    stringComplete = false;
  }
}
```

```
/*
 SerialEvent is called when new data is received on the RX port
This function is automatically called when the loop is called. If we add delays in loop we will also
delay the showing of the result. */
void serialEvent() {
  while (Serial.available()) {
    // read the new received byte
    char inChar = (char)Serial.read();
    // check if new line character has been received; if not, add it to the string
    // we do not add new line in input string since it will show an extra character in the LCD
    if (inChar != '\n')
    inputString += inChar;
    // dif the receive character is new line, set the flag so that the loop will know to display the
received data
    if (inChar == '\n') {
      stringComplete = true;
    }
  }
}
```

For transmitting data to the Arduino board, use the Serial Monitor, opened from the Tools menu.

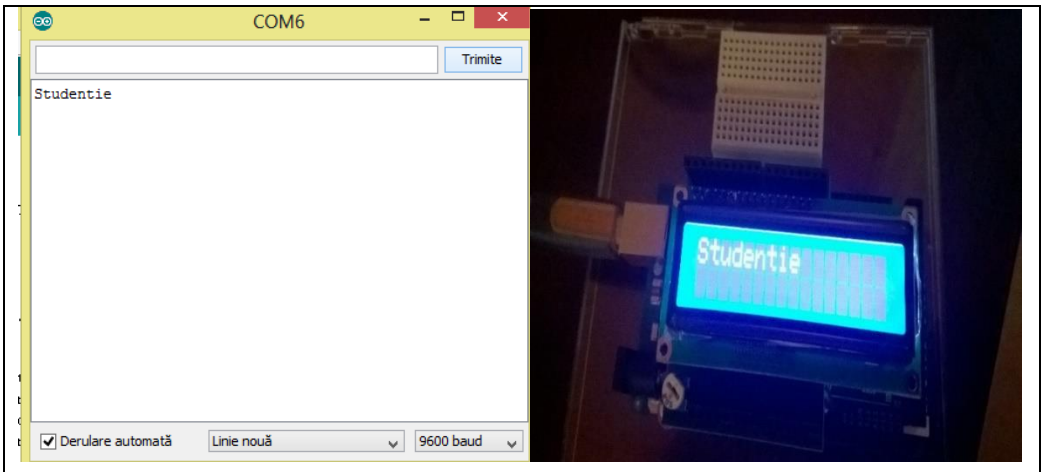

Fig. 7. Serial Data transmission in Serial Monitor

Inter-Integrated Circuit (I2C) protocol

The **Inter Integrated Circuit (I2C)** protocol was created for connecting more "slave" integrated circuits with one or more "master" circuit. This communication protocol is intended for short distance communications and similarly to the UART protocol or RS232 requires two wires for sending / receiving information.

Since serial ports are asynchronous the components that use them must agree in time upon the data transfer rate. The components must have clock signals with similar frequency, in order for the communication to work. At every transfer of a UART communication we transmit at least 10 bits in one packet, when using 8 data bits, which slows down the actual data transfer rate. The main advantage of UART serial communication is the fact that the communication can be done only

between 2 components (although you can connect more devices on the serial bus "bus contention" – with the risk of destroying the components).

In the case of I2C communication there can be more than 2 devices communicating on the bus. The data transfer rates for UART are usually fixed at a set of known baud rates, while in I2C the transfer rate is much higher.

The I2C bus is composed of 2 signals: SCL and SDA. SCL represents the clock signal; SDA the data signal. The clock signal is always generated by the master. (Some slave components can force the clock signal to low in order to signal the master to introduce a delay in the data transmission – this is called clock stretching).

Unlike other serial communication protocols, I2C is an "open drain" bus, that is a signal line can be pulled to 0 logic level ("low") but it cannot be pulled to 1 logical level ("high"). In this manner "bus contentions" are eliminated – a device tries to pull a signal to "high" while another devices tries to pull the signal to "low". Each signal has a pull-up resistor in order for the device to be able to set the signal to "high" when no other device is trying to pull the signal to "low".

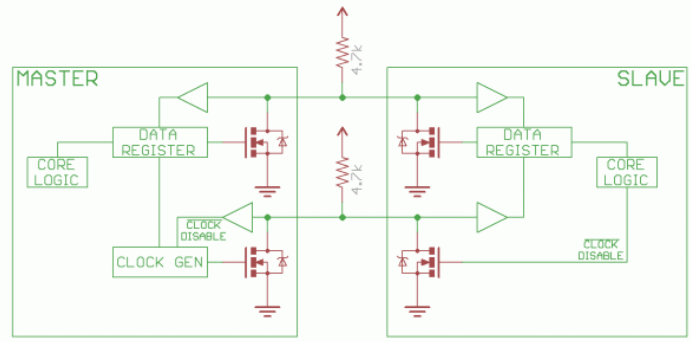

Fig. 8. I2C connection scheme. There is a resistor on each signal.

The selection for the resistor varies with the devices that use the bus, but as a rule start with 4.7k resistors and decrease them if necessary.

Protocol Description

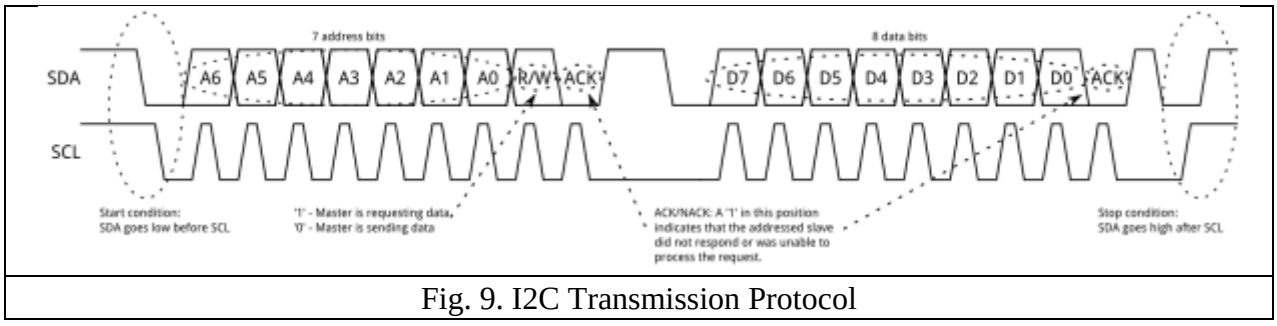

Fig. 9. I2C Transmission Protocol

I2C message are split in 2 types of frames: address frames (where the master selects the slave to which the data will be sent) and data frames that contain 8 bits of data to be passes from master to slave or from slave to master. The data are put on the SDA line after SCL is "low" and are sampled when SCL reaches the "high" logical level.

Start Condition

In order to initiate the address frame, the master leaves SCL to high and pulls SDA low. This prepares all the slaves for transmission. If 2 master devices want to send data over the I2C bus, the one that pulls SDA low first is granted control of the bus.

Address Frame

The address frame is always the first in message in the I2C communication. The address bits are transmitted first, MSB to LSB, followed by the R/W bit, showing if the operation is a read (1) or a write (0). The 9th bit of the address frame is the NACK / ACK bit. After the first 8 bits of the address frame are transmitted the receiving device gets control of the SDA line. If the receiving device does not pull the SDA line to 0 logic before the 9th clock cycle, it can be inferred that the receiving device either did not receive the message or did not correctly interpret the message. In this case the communication is stopped and the master decides what to do next.

Data Frames

After the address frame has been sent, the useful information can be transmitted on the I2C bus. The master will continue to generate the clock signal, at a regular interval and the data will be set on the SDA line, either by the master or by the slave (bit R/W indicates if it is a read or write operation). The number of data frames is arbitrary.

Stop condition

As soon as all the data frames have been transmitted, the master will generate a stop conditions. Stop conditions are defined as transitions from low to high (0 → 1) on the SDA line, after a 0 → 1 transition on SCL with SCL remaining high. During the write operations the value on SDA should not change when SCL is high, in order to avoid false stop conditions.

Example

For testing the I2C protocol functionality recreate the mounting from figure 10 or 11 (depending on the used boards).

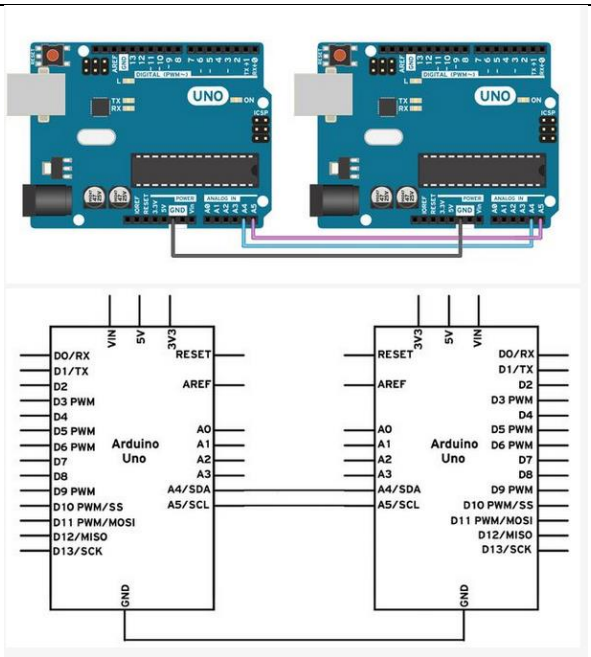

Fig. 10. Connecting 2 Arduino Uno Boards

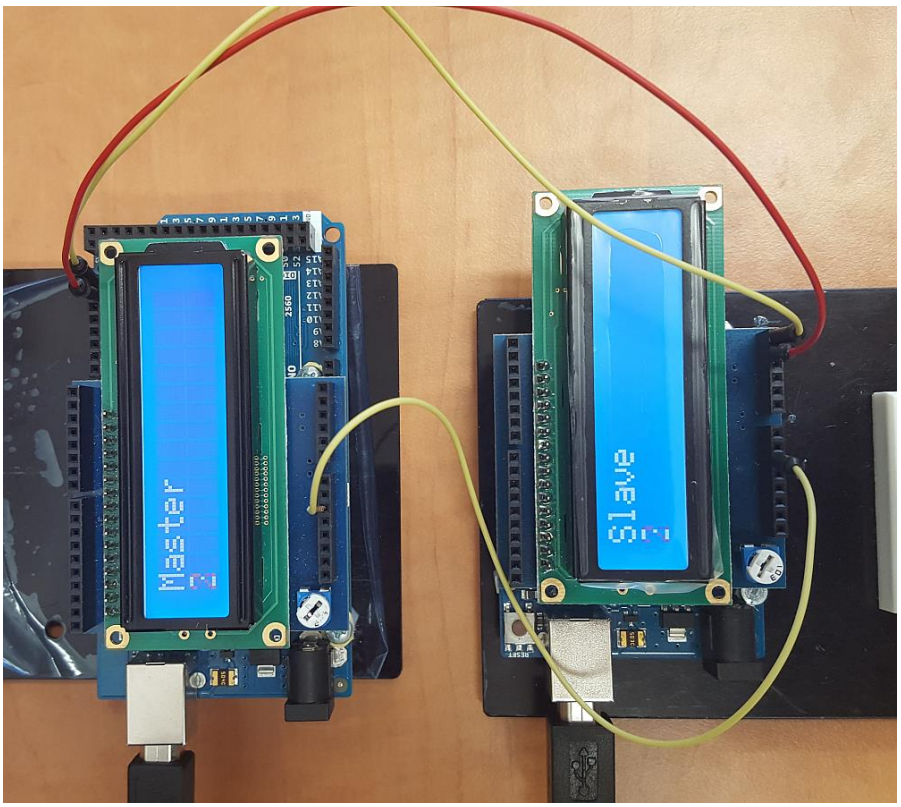

Fig. 11. Connecting an Arduino Mega Board with an Arduino Uno Board

If you are using an Arduino UNO / dumilanove / mini board connect pins A4 and A5 of one board to the same pins on the other board. Also connect the GND signal from the two boards.

Do not connect the voltage pins from the two boards together and be sure the boards are powered up at the same voltage level.

In order to write the code we will use the Wire library from Arduino environment (https://www.arduino.cc/en/Reference/Wire).

In the following table you can see the I2C pin locations on different Arduino boards.

Board	**I2C**
UNO, Ethernet	A4 (SDA), A5 (SCL)
MEGA 2560	20 (SDA), 21 (SCL)
Leonardo	2 (SDA), 3 (SCL)
Due	20 (SDA), 21 (SCL), SDA1, SCL1

Codul pentru dispozitivul slave:

```
#include <LiquidCrystal.h>

// include I2C library
#include <Wire.h>

int x = 0;
```

```
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);

void setup() {
  // Start i2C slave at address 9
  Wire.begin(9);
  // attach a function to be called when we receive something on the I2C bus
  Wire.onReceive(receiveEvent);

  lcd.begin(16,2);
  lcd.print("Slave");
}

void receiveEvent(int bytes) {
  x = Wire.read();    //read I2C received character
}

void loop() {
  lcd.setCursor(0,1); // display received character
  lcd.print(x);
}
```

Codul pentru master:

```
#include <LiquidCrystal.h>

// include I2C library
#include <Wire.h>

LiquidCrystal lcd(7, 6, 5, 4, 3, 2);
int x = 0;

void setup() {
  // Open I2C bus as master
  Wire.begin();
  lcd.begin(16,2);
  lcd.print("Master");
}

void loop() {
  Wire.beginTransmission(9); // transmit to device #9
  Wire.write(x);             // transmit x
  Wire.endTransmission();    // stop transmission

  lcd.setCursor(0,1);  // display sent character on LCD
  lcd.print(x);

  x++; // increment x
  if (x > 5) x = 0; // reset x when it reaches 6
  delay(500);
}
```

Individual Work

1. Test the examples from the laboratory. Ask your TA if you have any problems in connecting the wires.
2. Implement a communication system between 2 PCs by using the Arduino boards. The boards are connecting to the PC via USB and between them through I2C. The text written in Serial Monitor on the PC connected to the master will be written in the Serial Monitor of the slave PC.
3. Implement the same setup as in problem 2, but bi-directional. For transmitting from slave to master read: https://www.arduino.cc/en/Tutorial/MasterReader
4. Implement a network with one master and 2 slaves. The master is connected to the PC and receives messages through the serial interface of the following type:
    a. s1-hello
    b. s2-goodbye

    According to the number after the s character, the message after the '-' will be sent to the appropriate slave. The slave boards will display on LCD only the message addressed to them.