Assignment 05

1. Serial communication support of the Arduino platform

There are different ways to transmit information between different devices, such as SBUS, PPM, USB, MIDI, I2C, SPI, and UART among others. For the course of this study we are going to focus only on the last three ones.
UART, SPI, and I2C are three different types of serial communication used by Arduino devices. The main difference between them is that one of them, (UART) works in an asynchronous way, while SPI and I2C work in a synchronous way.

1.1 UART

Stands for Universal Asynchronous Reception and Transmission. Is the rst protocol that we are going to analyse. It uses only two wires to send data (digital pin 1 - TX), and to receive data(digital pin 0 - RX), and a common ground between the Arduinos. Also, Arduino has a UART communication via USB, the one that can be connected to a Windows, Mac or Linux OS to receive the data of the programs and to send back data to the connected device.
UART does not require a clock to transmit data. In this case, the user gives UART all the necessary. Instead to have a common clock signal, both Arduinos use internal clock signals. For this reason, if transmitter and receiver have not been configured for the same data-transmission frequency, the two devices will not be able to communicate.

1.1.1 Characteristics

To transmit data, UART requires three main characteristics: a start bit, a stop bit and baud rate.
To be able to transfer a message UART needs to synchronize data between the two devices. This is achieved by grouping the message that is going to be send in packages of known dimensions. The transmitter indicates to the receptor the moment that a package will start transmitting, and from that moment the clocks of both Arduinos work in parallel. After sending the package the devices go back to the resting state and a new package can be sent.

1.1.2 Start bit

The resting state in UART is a continuous high voltage or a one (1), due to is the best way to check that the connection is working. If the initial state is a low voltage or a zero (0), there would be no difference between resting state and no signal for an error in the transmission.
The transmitter sends the starting bit, indicating the receptor that the message is starting. This bit is always a zero (0).

1.1.3 Data Transmission

After receiving the start bit the receiver knows that a message is starting and it has to start reading the data from the next period. The measurement is taken in the middle part of each bit to avoid mistakes related to the voltage changing at the beginning or the end of the bit. To achieve this data transmission has to be known for both transmitter and receiver.

1.1.4 Parity bit

Parity bit is an error detection added at the end of the data transmitted. The parity bit will be equal to the sum of all the bits. This means that if the amount of hight bits transmitted is even, the parity bit will be a zero (0), and if the amount of high bits transmitted is odd, the parity bit will be one (1) .
We can make example of parity for a 8-bit message: 01100110, in this case the amount of high bits is four (4). In this case, the parity bit would be a zero (0). In this example the full data sent by the transmitter would be: 011001100, including the parity bit.
If for any reason the message received is 011000100, the receiver will add the rst 8 bits and will compare the result with the parity bit. In this example the sum of the 8 bits would be one (1), and after comparing with the received parity bit, the receiver would know that there is an error in the data received.
However this is not the most efficient way to corroborate the data transmission, since in if two bits are gotten wrong, the sum to the parity bit will be correct. We can consider our example with two errors in it, e.g. 011000110. In this case the receiver will add 0 and after comparing with the parity bit will detect that the data received is correct even though there are two incorrect bits.

1.1.5 Stop bit

Once the data package has been sent, the transmitter sends the nal or stop bit. The stop bit is always a one (1), and this will let the receiver know that the message has nished.

1.1.6 Baud rate

To achieve a proper data transmission, the speed of the bit transmission has to be known by the transmitter and the receiver. This speed is know as Baud rate and it is measured in bauds per second. In the basic UART system, this is equivalent to bits per second. The bigger the baud rate, the less time that a bit needs to be transmitted. With this idea, the best option would be to use the highest possible baud rate, but the higher the speed Microcontrollers
transmission, the more possibilities there are to have transmission errors. The baud rate has to be slower than the processor speed .

1.2 SPI

The SPI is a serial communication system based on four wires. Two wires; the MOSI, that connects the output of the master with the input of the slave; and the MISO, that connects the input of the master with the output of the slave. Besides this, SPI allows a single master to connect with several different slaves through an address. For this purpose we use a third line SS Slave Select and nally, to synchronize the master device with the slaves, we use a SCK, serial clock signal .
In the Arduino world we can communicate with SPI in the following way

- PIN D10 - SS (Slave Select)
- PIN D11 - MOSI (Master out, slave in)
- PIN D12 - MISO (Master in, slave out)
- PIN D13 - SCK (Serial clock)

1.3 I2C

The I2C is a serial communication protocol based in two wires, a data line SDA, and a clock line SCL. This communication protocol allow us to connect di
erent Slaves with a single
Master with only one data line. for this purpose, it is needed to send the information of the Slave that is required to access into the same message, the server will check if it is the one

who has been selected and will proceed to receive and send information, if the address is not the one that corresponds to it, it will ignore the call. To connect this protocol in the Arduino world the following PINs are used:

- PIN A4 or SDA
- PIN A5 or SCL

2. **Implementation:**
    - URAT Communication: https://www.tinkercad.com/things/1LLdmjoTcWp
    - I2C communication: https://www.tinkercad.com/things/bzwJ0xpcJom
    - SPI communication: https://www.tinkercad.com/things/hXP52SrQTHA

3. **Pros & cons:**
    - **SPI:**
        - **+ Ideal for high-speed data communication (up to 10Mbps)**
        - **+ Configurable master-slave status**
        - **+ Can have unlimited slave number**
        - **- It needs more wiring than I2C**
    - **I2C:**
        - **+ Ideal for onboard communication**
        - **+ Fewer wires required (2 wires) than SPI**
        - **+ Easy to set up**
        - **- Its speed (400Kbps) is slower than SPI**
        - **- Limited connectible devices as it uses a 10 or 7-bits addressing system.**
    - **UART:**
        - **+ Only requires two wires which save cost for wires**
        - **+ No clock signal required**
        - **+ Less complex than USART**
        - **+ Variable data transfer rate**
        - **- Only asynchronous mode available**
        - **- Slower than USART**
        - **- The receiver must recognize the transmitter's baud rate**
        - **- How USART works**