This code implements a **simplified protocol** to allow managing Arduino Uno R3 I/O from whatever platform capable to manage UART data flow, including Raspberry PI or your laptop.

Arduino I/O is controlld by means of a set of commands sent to Arduino via its UART interface.

I2C (TWI) and SPI are not (yet) supported.

# SET UP

To prepare Arduino please download the sketch serPiArduino2.ino on Arduino.

Connect Arduino’s UART pins and power it on (or simply connect USB port).

On the platform you want to use to manage Arduino, connect UART (by cross-wiring RX and TX and connecting GND) pins or simply connect USB cable.

## THE “PROTOCOL”

The simplified protocol provides a master / slave mechanism where Arduino is slave and your platform is master.

The protocol commands (from Master di Arduino) have the following frame structure:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0xF7  START BYTE  (1 byte) | COMMAND  (1 byte) | PIN  Number  (1 byte) | Value  (2 bytes) | CRC (mode 256)  (1 byte) | 0xF6  END BYTE  (1 byte) |

CRC is evaluated by adding bytes from COMMAND up to VALUE fields. (0xF7 start character is not taken into account).

Arduino will answer the received command with:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0xF7  START BYTE  (1 byte) | COMMAND ANSWER  (1 byte) | PIN  Number  (1 byte) | Value  (2 bytes) | Error Code  (1 byte) | CRC (mode 256)  (1 byte) | 0xF8  END BYTE  (1 byte) |

CRC is evaluated by adding bytes from COMMAND ANSWER up to “ERROR CODE” fields. (0xF7 start character is not taken into account).

These are the implemented COMMANDS:

* 0x01 READ PIN VALUE,
* 0x02 WRITE PIN VALUE,
* 0x03 SET PIN MODE,
* 0x04 GET PIN MODE

Arduino will answer (acknowledge) the received commands by issuing these answers:

* 0x31 PIN READ VALUE,
* 0x32 PIN WRITEN VALUE,
* 0x33 PIN MODE SET,
* 0x34 GET PIN MODE ANSWER

PIN number are AS FOLLOWS:

* Digital I/Os from 0 (Rx) up to 13 (debug built-in led) are referenced with their PIN number on Arduino connector (i.e from 0 to 13)
* Analog from A0 up to A5 are referenced with 16 (0x10) up to 21 (0x15).

Value is a 2 bytes field with these meaning according to used command:

|  |  |  |
| --- | --- | --- |
| COMMAND | VALUE FIELD (HIGH BYTE) | VALUE FIELD (LOW BYTE) |
| 0x01 READ PIN VALUE | Dummy byte | Dummy byte |
| 0x02 WRITE PIN VALUE | HIGH BYTE VALUE TO WRITE | LOW BYTE VALUE TO WRITE |
| 0x03 SET PIN MODE | HIGH BYTE MODE TO SET (0x00) | LOW BYTE MODE TO SET (see legend) |
| 0x04 GET PIN MODE | Dummy byte | Dummy byte |

Value in answer messages from Arduino are coded as follows:

|  |  |  |
| --- | --- | --- |
| ANSWER | VALUE FIELD (HIGH BYTE) | VALUE FIELD (LOW BYTE) |
| 0x31 PIN READ VALUE | Read value High Byte | Read value Low Byte |
| 0x32 PIN WRITEN VALUE | Written value High Byte | Written value Low Byte |
| 0x33 PIN MODE SET | HIGH BYTE MODE SET (0x00) | LOW BYTE MODE SET (see legend) |
| 0x34 GET PIN MODE ANSWER | HIGH BYTE CURRENT SET MODE (0x00) | LOW BYTE CURRENT SET MODE (see legend) |

These are the possible values for **MODE** field:

|  |  |
| --- | --- |
| MODE FIELD VALUE | MEANING |
| 0x00 | Digital Input |
| 0x01 | Digital Input Pull-Up |
| 0x02 | Digital Output |
| 0x03 | Digital Output PWM |
| 0x04 | Analogue Input |

These are the possible values for **ERROR CODE** field:

|  |  |
| --- | --- |
| ERROR CODE FIELD VALUE | MEANING |
| 0x00 =0 | OK (No Error) |
| 0xE1 =225 | Invalid CRC |
| 0xE2 =226 | Required WRITE operation on Input or Required READ on output |
| 0xE3 = 227 | No valid mode for PIN number |
| 0xE4 = 228 | Not valid PIN number |
| 0xE5 = 229e | Not valid MODE code |
| 0xE6 = 230 | Not valid COMMAND code |

**EXAMPLE**

The code comes with an example program written in Python providing you with a textual interface.

You can use this code to get familiar with the functionalities offered by the program running on Arduino.

Python code is made up of three files:

* **interfaccia\_Arduino.py**: this is the main program you have to run
* **lib\_Arduino\_Funcs.py**: this implements all the functions to generate and decode the messages to be exchanged with Arduino
* **lib\_menu.py**: this provide a function to generate and manage textual menus

Add: description description of config file

The only required adaptation is that you should enter in **interfaccia\_Arduino.py** the COM port of your platform where you will connect Arduino:

*ser = serial.Serial(****'COM31'****, 9600, timeout=3)*

In this case the Python code is running on a Windows laptop and Arduino has been connected to the USB port mapped on COM31.

**NOTE:** If you want to change data rate, you should set the same value on both Arduino and your platform. In this case you have to edit this line in SerPiArduino2.ino:

*Serial.begin(****9600****);*

and:

*ser = serial.Serial('COM31',* ***9600****, timeout=3)*

on Python code.