REXduino – Arduino microcontroller in the REX Control System

User guide

Jaroslav Sobota

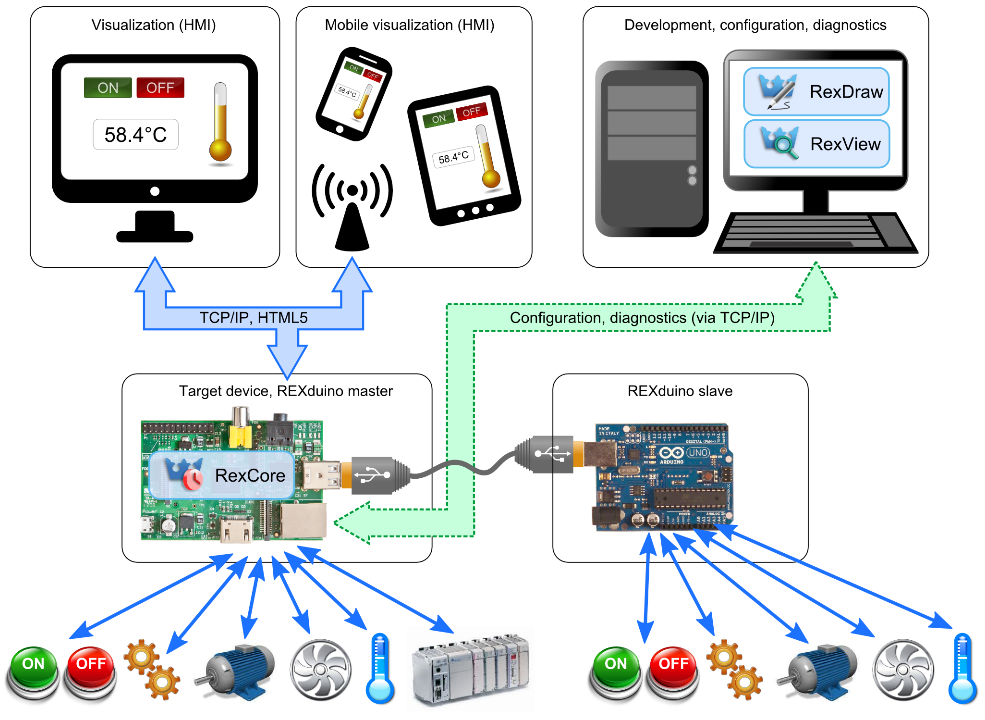
Version 1.2.0

Contents

Chapter 1

Introduction

The symbiosis of the REX Control System[1] and an Arduino board is based on a simple communication protocol, whose slave part is implemented in the Arduino while the master part is running in the target device of the REX Control System (e.g. Raspberry Pi or laptop or desktop PC running the RexCore runtime module). In fact, the Arduino serves only as an input/output unit, the control algorithm is running in the runtime module of the REX Control System. The Arduino is connected using the USB cable.


Visualization (HMI)
ON
OFF
58.4°C
Mobile visualization (HMI)
Development, configuration, diagnostics
RexDraw
RexView
TCP/IP, HTML5
Configuration, diagnostics (via TCP/IP)
Target device, REXduino master
RexCore
REXduino slave
ON
OFF
ON
OFF

[1] www.rexcontrols.com/rex

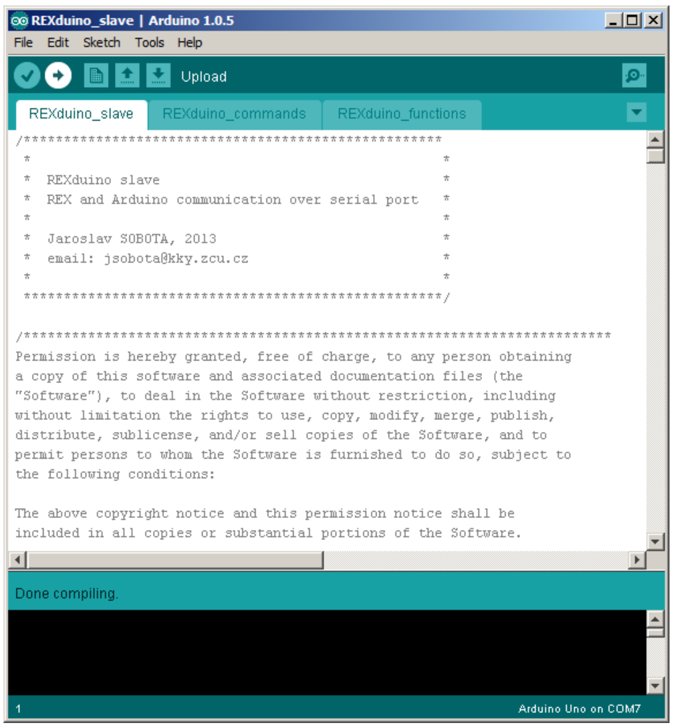
Chapter 2

Installation

2.1 Preparing the slave device

The Arduino must be programmed first to act as REXduino slave. For that, you have to

1. Open the `REXduino_slave.ino` sketch in Arduino IDE.
2. Compile it.
3. Program Arduino with it.



```
/*****************************************************
 *                                                   *
 *  REXduino slave                                   *
 *  REX and Arduino communication over serial port   *
 *                                                   *
 *  Jaroslav SOBOTA, 2013                            *
 *  email: jsobota@kky.zcu.cz                        *
 *                                                   *
 *****************************************************/

/*************************************************************************
Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.
```

```
Done compiling.
```

```
1                                                  Arduino Uno on COM7
```

2.2 Preparing the master device

The master part of REXduino requires only the RexCore runtime module of the REX Control System to run. See the *Getting started with REX* user guides at `www.rexcontrols.com` for instructions.

Apart from the runtime module only a system driver for the Arduino USB port is required for the Arduino to appear as a serial device

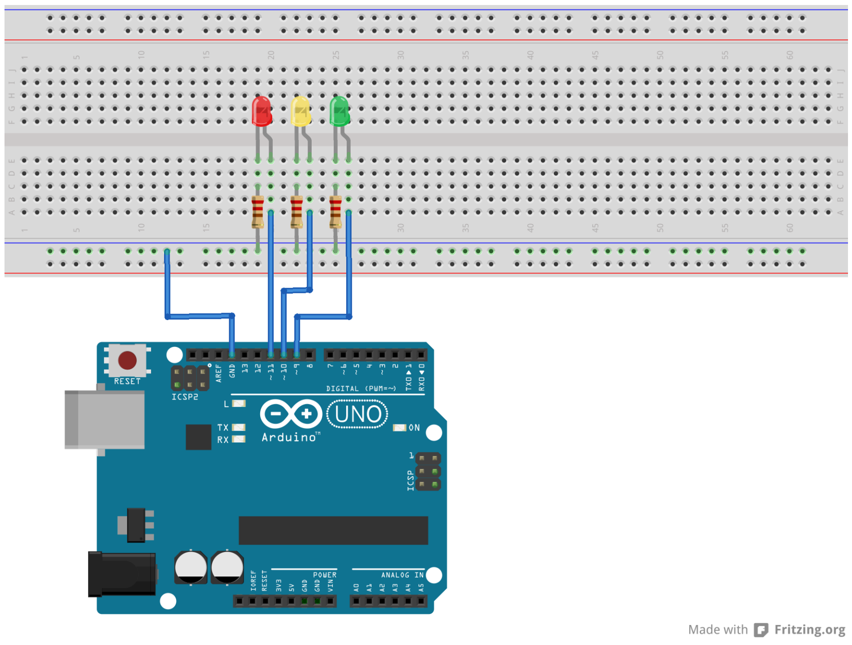
- **Windows** – the driver is part of the Arduino IDE installation
- **Raspberry Pi** – the driver is included in the Raspbian distribution by default
- **OpenWrt** – the `kmod-usb-acm` package must be installed

You only need to know the serial port identifier. In Windows it will be e.g. `COM5`, in Linux it will be most probably `/dev/ttyACM0` or `/dev/ttyUSB0` but the numbers in the identifiers may differ.

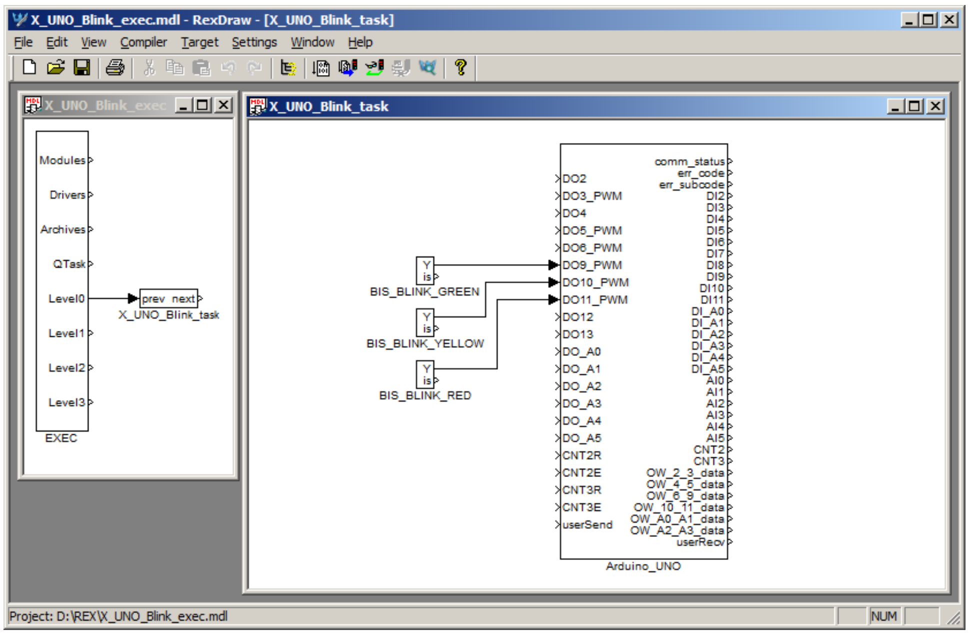
2.3 Using REXduino

The so-called masked subsystems are prepared for you to include Arduino in your projects almost instantly.

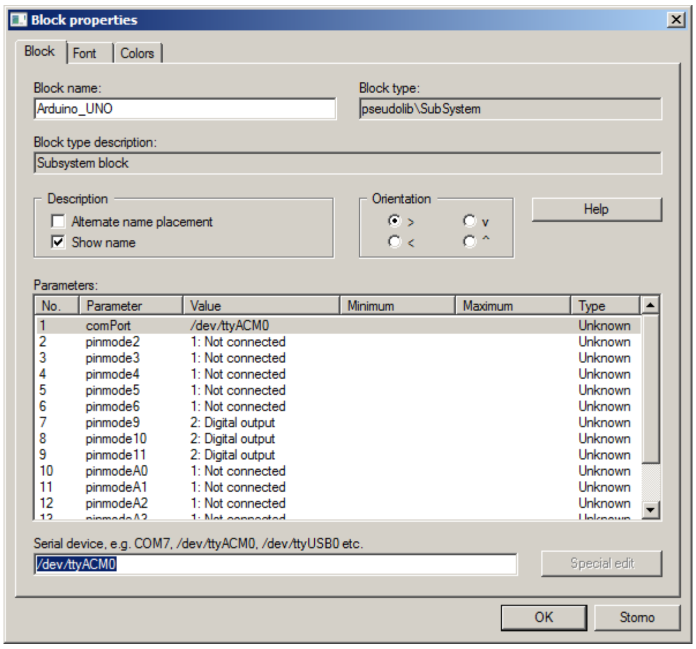
A simple setup to start with might include 3 LED lights and 3 resistors:



And the corresponding project in the REX Control System follows:



The working modes of the individual input/output pins must be defined upon double-clicking the Arduino mask in the RexDraw development environment. Also the serial device to use must be specified.



Block name: Arduino_UNO

Block type: pseudolib\SubSystem

Block type description: Subsystem block

Description: [ ] Alternate name placement [✓ Show name]

Orientation: [✓ >] v < ^

Parameters:

No.	Parameter	Value	Minimum	Maximum	Type
1	comPort	/dev/ttyACM0			Unknown
2	pinmode2	1: Not connected			Unknown
3	pinmode3	1: Not connected			Unknown
4	pinmode4	1: Not connected			Unknown
5	pinmode5	1: Not connected			Unknown
6	pinmode6	1: Not connected			Unknown
7	pinmode9	2: Digital output			Unknown
8	pinmode10	2: Digital output			Unknown
9	pinmode11	2: Digital output			Unknown
10	pinmodeA0	1: Not connected			Unknown
11	pinmodeA1	1: Not connected			Unknown
12	pinmodeA2	1: Not connected			Unknown
13	pinmodeA3	1: Not connected			Unknown

Serial device, e.g. COM7, /dev/ttyACM0, /dev/ttyUSB0 etc.
/dev/ttyACM0

As soon as you compile the project and download it to the target device, the communication with Arduino will be initialized and the LEDs will start to blink.

In the background, the REXduino master sends commands to the Arduino to initialize communication, switch pins to the corresponding modes and read/write data.

2.3.1 Communication status

There are 4 states of the communication with the Arduino:

- 0: Arduino not responding, waiting for timeout to try again
- 1: Serial port open, trying to initialize communication
- 2: Communication initialized, setting pin modes
- 3: Arduino OK, pin modes set, data valid

2.4 Example projects

Examples can be found in the REXduino_master folder, all of them start with `X_`.

If you wish to start from scratch, copy and paste the block of your choice from `REXduino_masks.mdl`.

2.5 IMPORTANT NOTICE

The REXduino master part is implemented using the programmable function block REXLANG. The `REXduino_master.stl` file contains the code and therefore it is an essential part of each project using the Arduino in the REX Control System. Therefore,

!!! NEVER FORGET TO INCLUDE IT IN THE PROJECT FOLDER !!!

Your project will always contain at least 3 files:

- the executive configuration file
- the so-called task with the control algorithm
- the `REXduino_master.stl` file

Chapter 3

Pin modes

3.1 Digital out - O

Digital output, default LOW. Control LEDs, valves, relays, etc.

Bring the control signal to the `DOn` input of the block, where `n` is the pin number. Or `DO_An` if you are using analog pins as digital outputs.

3.2 Digital out - Q

Digital output, default HIGH. Useful for controlling devices with inverted logic (relay boards, typically).

Bring the control signal to the `DOn` input of the block, where `n` is the pin number. Or `DO_An` if you are using analog pins as digital outputs.

3.3 Digital input - I

Digital input. Intended for reading a switch, the pin should always be connected to HIGH (5V) or LOW (0V).

The state of the pin can be read from the `DIn` output of the block, where `n` is the pin number. Or `DI_An` if you are using analog pins as digital inputs.

3.4 Digital input with pullup - J

Digital input. Intended for reading a push button. The pin is connected to LOW (0V) when the button is pressed, otherwise it is drawn to HIGH (5V) by the internal 20k pull-up resistor to avoid a "floating pin". Note the inverted logic.

The state of the pin can be read from the `DIn` output of the block, where `n` is the pin number. Or `DI_An` if you are using analog pins as digital inputs.

3.5 Analog out (PWM) - P

Pulse-width modulated output. Can be used for dimming LEDs, smooth control of motors etc.

Bring the control signal to the `DOn_PWM` input of the block, where `n` is the pin number. The range is 0 to 255.

3.6 Analog in - A

Analog input. Can be used to read potentiometers, NTC thermistors etc.

The measured value is available at the `AIn` output of the block, where `n` is the pin number. The range is 0 to 1023.

3.7 Counter - C

Counter for counting pulses from e.g. water consumption meters. Bring the pulses to pin 2 and the count direction to pin 4 (HIGH=count up, LOW=count down). The rising edges on pin 2 are counted. The mode of pin 4 must be set to `Counter DIR`.

The same applies for the second counter which is available on pins 3 and 5.

The values of the counters are available at `CNT2` and `CNT3` outputs.

You have to enable the counter by setting the `CNT2E` (`CNT3E`) input to `1`.

You can reset the counter by setting the `CNT2R` (`CNT3R`) input to `1`.

3.8 Encoder - E

Reading encoder signals. Bring the encoder signal A to pin 2 and the encoder signal B to pin 4. Both the rising and falling edges of signal A are counted, the direction is given by the encoder B signal. The mode of pin 4 must be set to `Encoder B`.

You can also connect another encoder to pins 3 and 5.

The values of the encoders are available at `CNT2` and `CNT3` outputs.

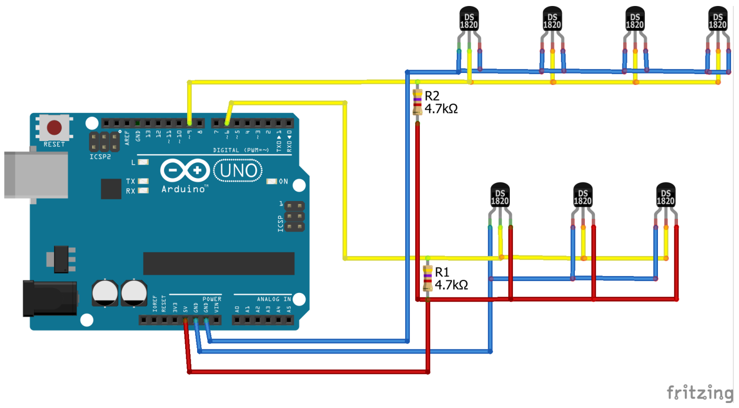
You have to enable encoder evaluation by setting the `CNT2E` (`CNT3E`) input to `1`.

You can reset the encoder value by setting the `CNT2R` (`CNT3R`) input to `1`.

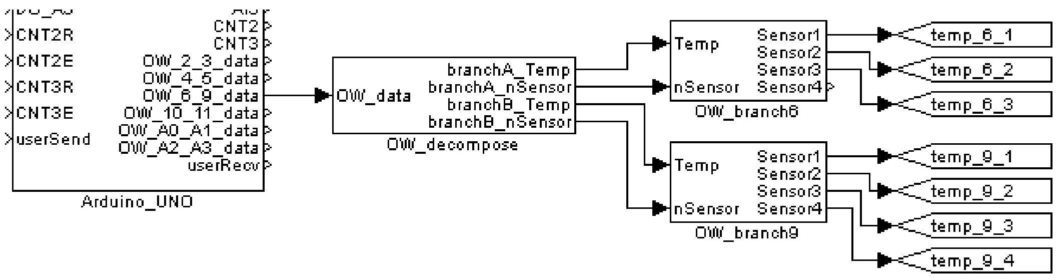
3.9 OneWire - W

Reading 1-Wire temperature sensors. In order to use this mode, you have to enable 1-Wire in the REXduino slave sketch. For that, you must include the OneWire library in your Arduino IDE and uncomment the `//#define USE1WIRE` line. Compile the sketch with this modification and program it into the REXduino slave device (Arduino).

You can use both powered and parasitic sensors as shown in the wiring example below.



The temperature data is available at the `OW_m_n` output of the block, where `m` and `n` are pin numbers. The data must be further processed by the `OW_decompose` block and the `OW_sensors_1_4` as shown below. The function blocks can be found in the `REXduino_masks.mdl` file.



The sensors are ordered by their ROM IDs.
Maximum 15 sensors in one branch!

Chapter 4

Communication protocol

The following table shows the individual commands of the serial communication. Lowercase versions of the commands are also supported for human-readable communication, which is useful for debugging using the serial line monitor in Arduino IDE. The corresponding port must be selected, the settings are baudrate 57600 and "no line ending".

4.1 Commands

communication initialization			
C	0	;	
switch pin mode			
M	nPin	pinMode	;
analog read (analog input)			
A	nPin	;	
PWM write (analog output)			
P	nPin	val	;
read digital input			
I	nPin	;	
set digital output			
O	nPin	val	;
read 1-wire temperature			
T	nPin	;	
read counter or encoder value			
N	nCnt	N	;
reset counter			
N	nCnt	R	;
enable counter			
N	nCnt	E	;
disable counter			
N	nCnt	D	;

4.2 Responses

initialization acknowledgement									
C	0	;							
pin mode acknowledgement									
M	nPin	pinMode	;						
analog input value									
A	nPin	byteH	byteL	;					
PWM value acknowledgement									
P	nPin	val	;						
digital input value									
I	nPin	val	;						
digital output acknowledgement									
O	nPin	val	;						
1-wire temperature measurement status									
T	nPin	nStat	;						
1-wire temperature data									
T	nPin	nSens	byteH	byteL	;				
counter or encoder data									
N	nCnt	N	wordH, byteH	wordH, byteL	wordL, byteH	wordL, byteL	;		
counter enabled									
N	nCnt	E	;						
counter disabled									
N	nCnt	D	;						
counter reset									
N	nCnt	R	;						
error code									
E	err	err_sub	;						

Chapter 5

Troubleshooting

- Check if the REXduino slave responds. You can act as REXduino master. In the Arduino IDE, open the serial monitor, set baudrate 57600 and "no line ending" and enter the message
  `c0;`
  The reply must be `C0;`
- Start with only one or two signals. If it works, add additional signals one-by-one until you reach the problem.
- If the communication status is 2 and never reaches 3, you are probably trying to read 1-Wire temperature but your REXduino slave was compiled without 1-Wire support. See the 1-Wire pin mode for instructions.
- You can print all the outgoing and incoming serial data to the REX Control System log. This will allow you to find the problem. Go inside the Arduino mask and find the blocks `CNB_LOG_INCOMING` and `CNB_LOG_OUTGOING`. Set them both to `on`. Afterwards, in RexView, go to menu `Target–Diagnostic messages` and tick `Information` in the Function block messages box. Then you can switch to system log tab and analyze the serial communication. It is recommended to use a sampling period of 1 second for debugging purposes.