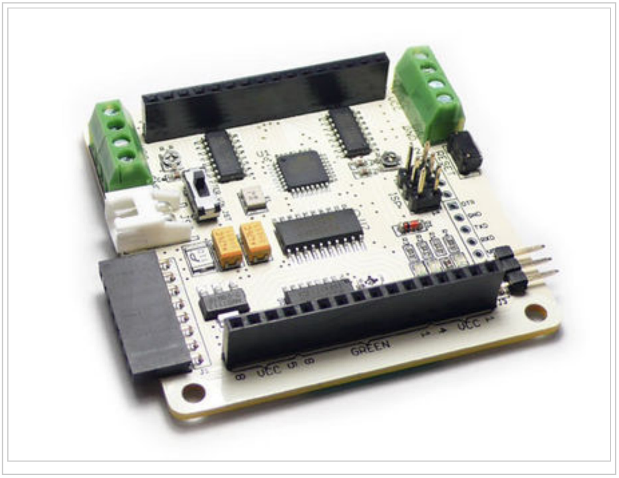
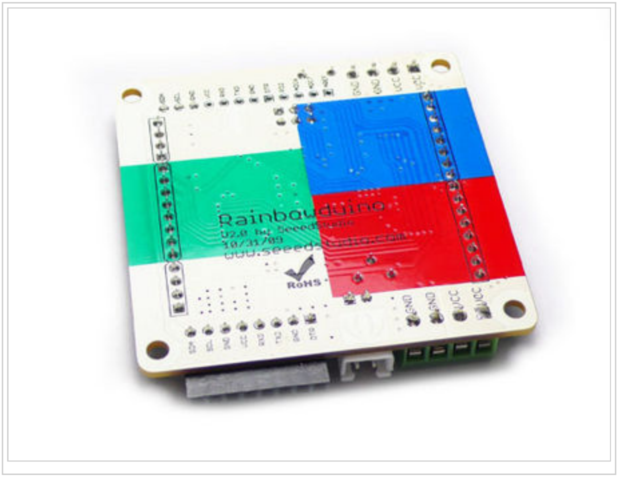
Rainbowduino LED driver platform - ATmega328

From Wiki 来自痴汉的爱

[中文 (http://www.seeedstudio.com/wiki/index.php?title=Rainbowduino_LED_driver_platform_LED驱动控制板_-_ATmega328&uselang=zh)]





Contents

Introduction

The Rainbowduino board is an Arduino compatible controller board with professional LED driving capacity. It will drive an 8x8 RGB Led Matrix (Common Anode).

- No external circuit required, plug and shine!
- 24 constant current channels of 120mA each
- 8 super source driver channel of 500mA each
- Wide output voltage adaption from 6.5V-12VDC
- Dedicated GPIO and ADC
- Hardware UART and I2C communication
- Easy cascading
- Small form and light weight

Standalone Mode (plug and shine)

Needed Hardware:

- 1 x Rainbowduino
- 1 x RGB LED Matrix

The simplest work mode, no external Systems needed (only a ttl serial adapter to upload the firmware). The LED matrix content is generated by the Rainbowduino itself.
Use Case:

- Simple real-time animations calculated by the Rainbowduino
- Display pre-stored animations, limited due the 32kb ROM of the Rainbowduino

UART Mode

Needed Hardware:

- 1x Rainbowduino
- 1x RGB LED Matrix
- 1x TTL Level converter
- 1x UART sender unit (Arduino, PC...)

Send data (LED matrix content) from your computer to one Rainbowduino. As the Rainbowduino does not have a USB connector but a TTL serial connection you need a TTL serial level converter (BusPirate, UartSBee, Arduino...).

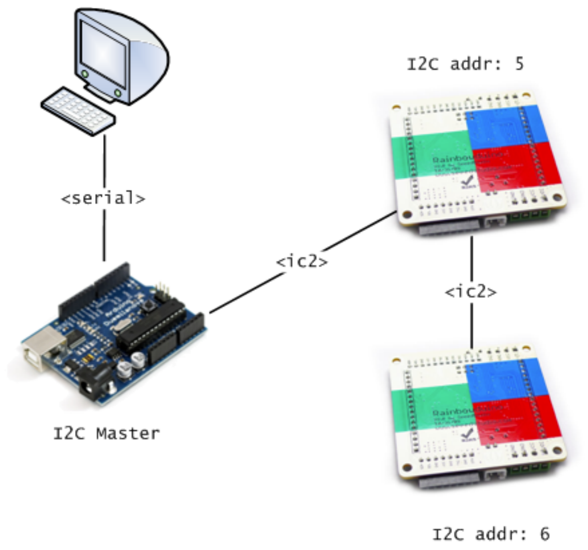
Use Case:
PC or Arduino generated frames displayed on ONE Led Matrix

I2C Mode

Needed Hardware:

- 1..n x Rainbowduino
- 1..n x RGB LED Matrix
- 1 x I2C master device (for example an Arduino)
- Some cables

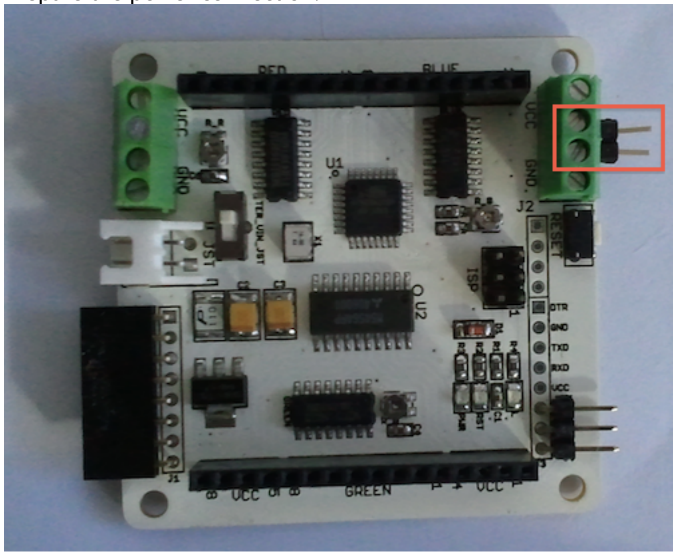
Send data (LED matrix content) from your computer to multiple Rainbowduino’s. A side-note if you use an Arduino with an FTDI USB to Serial adapter (Duemillanove, Diecimila...) - there is a Latency of ~16ms to send data from your computer to the Arduino. The new Arduino UNO have a much lower latency ~4ms.
Use Case:
PC or Arduino generated frames displayed on **multiple** Led Matrices



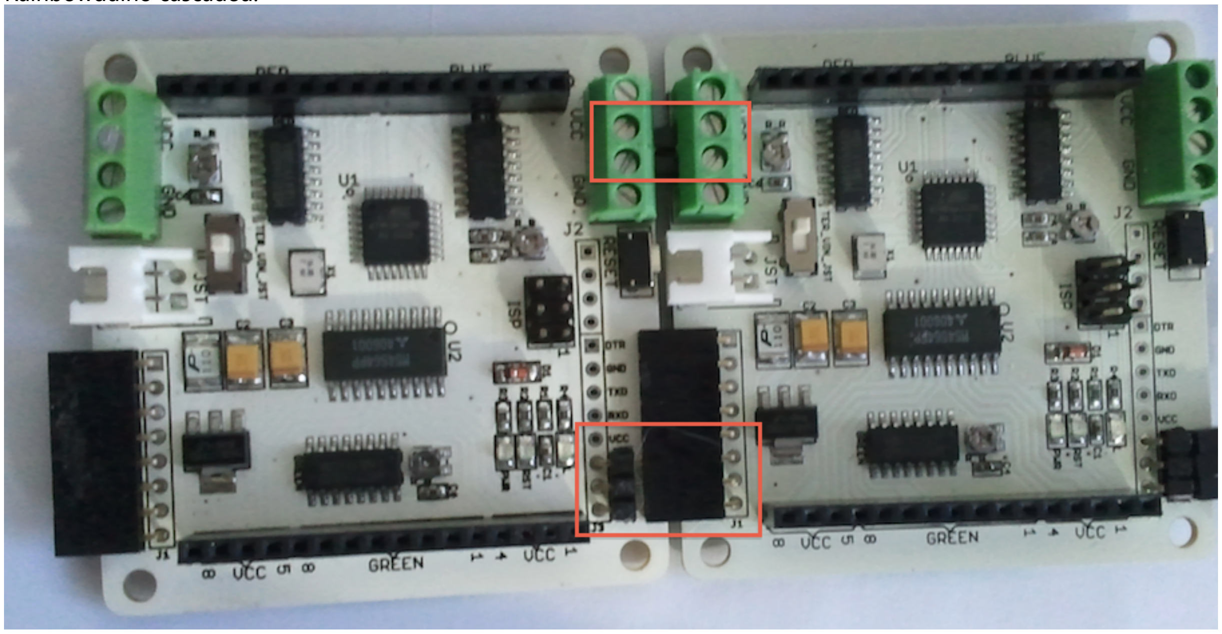
I2C Cascading

Rainbowduino is designed for easy casacading. After physically connected, power is passed on, and you may control the chain by I2C. Please note that each Rainbowduino must be assigned for a uniqe address for I2C communication.

Prepare the power connection:



Rainbowduino cascaded:

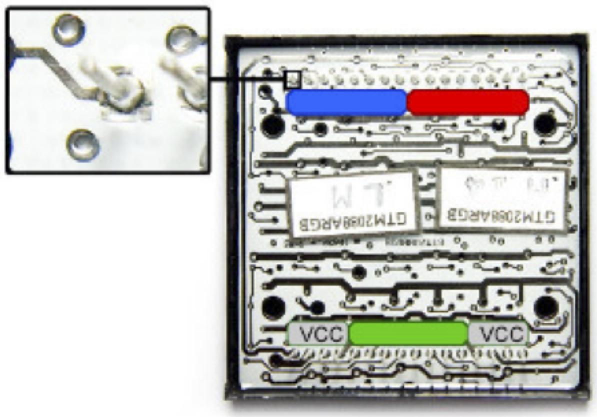


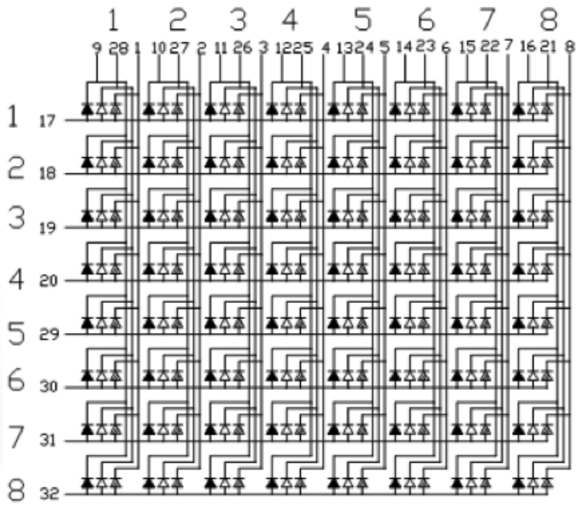
Specification

Microprocessor : Atmega 328
PCB size : 60mm * 60mm * 1.6mm
Indicators : Reset, Power, Pin 13 LED
Power supply : 6.5-12 VDC (9 VDC recommended)
Power connector : 2 pin JST Terminal Blocks, 3mm DC Jacks
Cascading Power Connector : Terminal Blocks
Program interface : UART / ISP
LED dot-matrix sockets : 32
Expansion socket : 2.54mm bended pinheader pair
Communication Protocols : I2C / UART
RHOS : Yes
Input Voltage : 6.5~12V
Global Current Consumption : 600~2000mA
Constant Current Channels (Cathode) : 24
Constant Current per channel (Cathode) : 20~120mA
Source Driver Current per channel (Common-Anode) : 500mA
Source Driver Voltage per channel (Common-Anode) : 9~12V
Source Drive Channels : 8
Drive LED count : 192
Circuit Response Time : 10ns
RGB Led Matrix color resolution per dot : 4096 : Uart Baud Rate : 115200baud

LED devices Compatibility

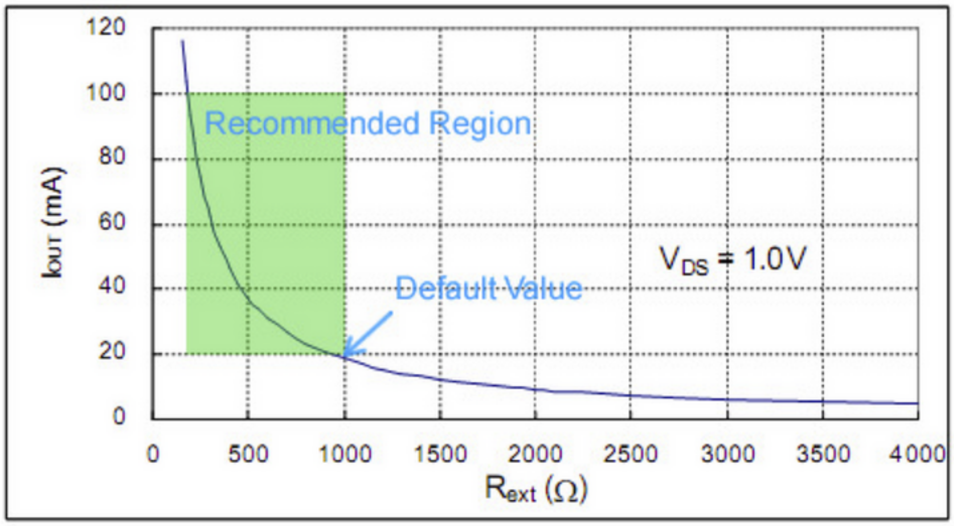
Before direct plug into the female pin-headers, please verify if the RGB dot matrix are proven compatible. The concern is mainly on the pin out, where same color LEDs are in cluster, here we attach the scheme and photo demonstration. The color sequence might change, since the controlling logic are open source and easily reprogrammable.





The power of Rainbowduino is well beyond driving a RGB dot-matrix. With 192 output count, and up to 120mA constant current capacity, you may easily populate massive LED setups. The output current of each channel (IOUT) is set by an external resistor, Rext. The relationship between Iout and Rext is shown in the following figure. Please refer to MBI5168 data-sheet for more details. Adjusting the 1k Potentiometer clockwise to reduce the output current (default minimal 20mA for RGB dotmatrix), rotating counter-clockwise to increase the output current. The potentiometers are single circle, please NOTE that strong force will break it into unlimited rotatable, then you would need a multimeter to adjust. :)

This means you can build your own LED matrix without any additional resistors.



Resistance of the external resistor, R_{ext}, in Ω

Demonstration

Requirements

- Rainbowduino board
- A Common Anode RGB Matrix
- An Arduino board (Optional)

Prepare Rainbowduino Hardware

Connect the RGB LED Matrix to the Rainbowduino and connect "Pin 1" to the red connection block. Pin1 is marked by a square solder point, while the other pins use a round solder point.

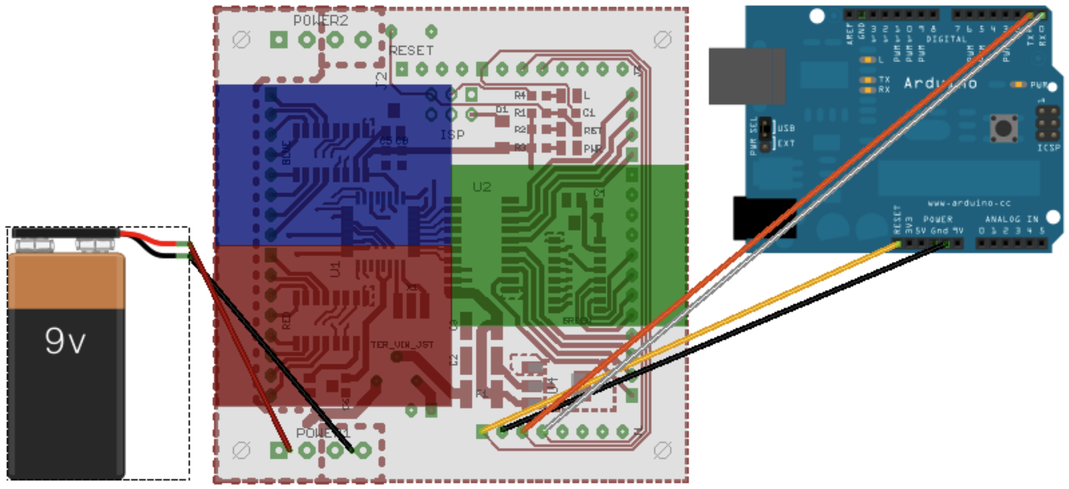
Upload Firmware

1.Upload a piece of code to Arduino first: In order to use the Arduino to upload the firmware to the Rainbowduino, make sure that the Arduino is clean - we need to upload an empty firmware sketch as below shows to it.

```
void setup() {}

void loop() {}
```

2. Upload Firmware to Rainbowduino

Open the Rainbowduino firmware, **select the correct board** (Tools-->board--> Arduino Duemilanove or Nano w/ ATmega328) and upload the Firmware. At least that’ s the theory ;)
For your viewing pleasure, here is the connection scheme:

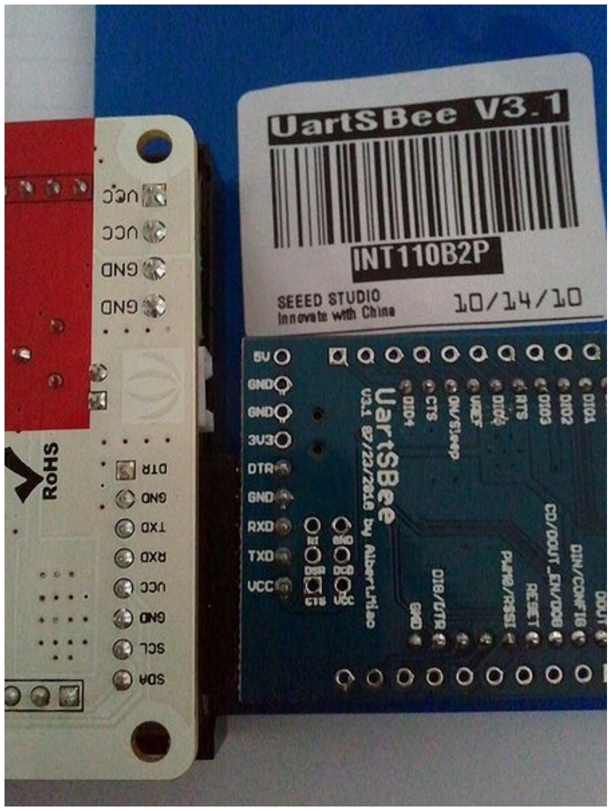


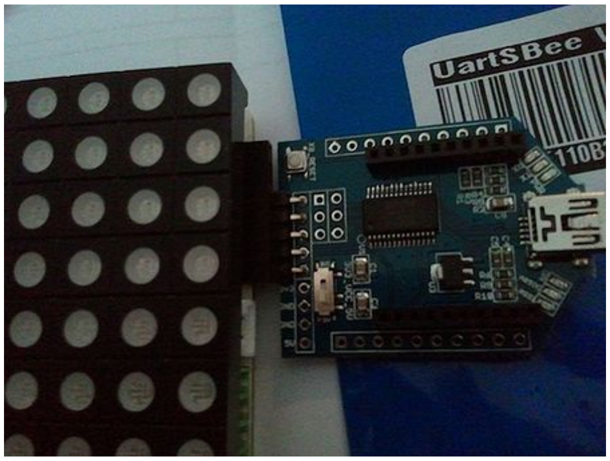
We use an external power source, however you could also use the 5V from the Arduino.
NOTE: If you own a Rainbowduino v1 board, you need to select the "Arduino Diecimila, Duemilanove, or Nano w/ ATmega168"!

Arduino	Rainbowduino
RESET	DTR
GND	GND
RX	RX
TX	TX

3.Use UartSB to Upload firmware

Those screenshot's shows how to connect the UartSBee to the Rainbowduino:





If you connect the UartSBee to the USB bus, it should register a new serial port. Now simply upload your firmware using the new serial port.

4.Use a Buspirate to Upload firmware / bootloader

I'm explaining three methods of programming (all using the buspirate):

- programming through the ISP. - programming using avrdude and manual reset (no patching necessary) - programming through avrdude with a tiny patch.

DISCONNECT THE RAINBOWDUINO FROM THE DISPLAY AND POWER.

STEP 1: To use the Buspirate you need a new version of avrdude [1] (http://download.savannah.gnu.org/releases/avrdude/) . I'm using version 5.10 and that recognizes the '-c buspirate' programmer option. You can test this with

```
./avrdude -c buspirate -C ./avrdude.conf
```

If this complains about the programmer, then you need a newer version of the buspirate.

STEP 2: connect the buspirate to the rainbowduino ISP connector like this:

Buspirate	ISP	ISP pin
GND	GND	6
+5V	Vcc	2
CS	RESET	5
MOSI	MOSI	4
MISO	MISO	1
SCL/CLK	SCK	3

STEP 3: find the correct bootloader (I'm using the tiny optiboot firmware). Copy this file to your freshly compiled avrdude directory.

STEP 4: program the atmega 328p with

```
./avrdude -v -c buspirate -p m328p -C ./avrdude.conf -P /dev/ttyUSB0 -U flash:w:optiboot_atmega328.hex
```

This takes a very long time...

I started with uploading firmwares without the bootloader and that works fine. Trick is to get the HEX files from the arduino IDE. In version 0.22-Linux they are stored in /tmp/buildXXXXXXXXXXXX and NOT in the sketches directory. Just issue the 'Upload' command without any programmer connected (press <shift> during while pressing the 'upload' button to get much debug info from the arduino ide).

After you have the bootloader on the rainbowduino you can use the transparent serial interface of the buspirate. Set the baudrate to 115200 and enter the '(3)' macro to activate transparent mode. The buspirate now acts like a USB-serial converter (any other FTDI like usb-serial converter could be used). Issue with the buspirate is that there is no DTR to reset the arduino with. You now have to time the reset and upload manually. Pressing reset after starting the upload seems to work best.

```
HiZ>m
1. HiZ
2. 1-WIRE
3. UART
4. I2C
5. SPI
6. JTAG
7. RAW2WIRE
8. RAW3WIRE
9. PC KEYBOARD
10. LCD
(1) >3
Mode selected
Set serial port speed: (bps)
1. 300
2. 1200
3. 2400
4. 4800
5. 9600
6. 19200
7. 38400
8. 57600
9. 115200
10. 31250 (MIDI)
(1) >9
Data bits and parity:
1. 8, NONE *default
2. 8, EVEN
3. 8, ODD
4. 9, NONE
(1) >
Stop bits:
1. 1 *default
2. 2
(1) >
Receive polarity:
1. Idle 1 *default
2. Idle 0
(1) >
Select output type:
1. Open drain (H=Hi-Z, L=GND)
2. Normal (H=3.3V, L=GND)
(1) >2
READY
UART>(3)
UART bridge. Space continues, anything else exits.
Reset to exit.
```

After that you can program the arduino with the bootloader:

```
./avrdude -v -c stk500v1 -p m328p -b 115200 -F -C ./avrdude.conf -P /dev/ttyUSB0 -U flash:w:Rainbow_Plasma.cpp.hex
```

One step further is to patch avrdude in the file 'arduino.c'. The buspirate sends the 'rts' signal with the wrong polarity to the arduino but by swapping 1 for 0 and 0 for 1 that is fixed.From then on you have to choose 'arduino' as programmer instead of 'stk500v1'.

```
static int arduino_open(PROGRAMMER * pgm, char * port)
{
  fprintf(stderr, "arduino_open...\n");
  strcpy(pgm->port, port);
  serial_open(port, pgm->baudrate? pgm->baudrate: 115200, &pgm->fd);

  /* Clear DTR and RTS to unload the RESET capacitor
   * (for example in Arduino) */
  serial_set_dtr_rts(&pgm->fd, 1);
  usleep(50*1000);
  /* Set DTR and RTS back to high */
  serial_set_dtr_rts(&pgm->fd, 0);
  usleep(50*1000);

  /*
   * drain any extraneous input
   */
  stk500_drain(pgm, 0);

  if (stk500_getsync(pgm) < 0)
    return -1;

  return 0;
}
```

Note: change the programmer type used by the arduino ide in the 'boards.txt' file.

Source: buspirate-avr-programming [2] (http://hintshop.ludvig.co.nz/show/buspirate-avr-programming/) , Bus_Pirate_AVR_Programming [3] (http://dangerousprototypes.com/docs/Bus_Pirate_AVR_Programming) , Optiboot [4] (http://code.google.com/p/optiboot/)

Rainbowdunio Firmware

Neorainbowduino Firmware

This firmware bundle comes with two firmwares (one for a Arduino, one for the Rainbowduino) and a Processing library. You can send data from any Application via the serial line to the Arduino - the Arduino then sends the images to the corresponding Rainbowduino device. I created an easy-to-use Processing library to get started.

Source: http://code.google.com/p/neorainbowduino/

Features:

- I2C enabled firmware (supports multiple Rainbowduino’ s)
- Processing library, so you can easily control your Rainbowduino from Processing!
- Send full frames from Processing to Rainbowduino
- Send frames from Processing to your RGB matrix, each frame has a size of 8x8 pixel, 12bit color resolution (4096 colors). The color conversion is handled by the library
- Optimized processing lib - send only frames to Rainbowduino if needed (save ~50% of traffic - of course it depends on your frames)
- Fixed buffer swapping (no more flickering)
- Added i2c bus scanner, find your Rainbowduinos if you forget their addresses

Supported Work Modes: I2C

Requirements

This firmware allows you to use Processing to control the rainbowduino, so its obvious you need:

- Processing Software, get it from http://processing.org/

If you don't like Processing (JAVA) you are not limited to it. Check http://wish.seeedstudio.com/?p=320 for an example using autoitscript (http://www.autoitscript.com/autoit3/index.shtml) sending data to the Arduino.

Patches for Arduino IDE

Because the neorainbowduino firmware sends full frames via I2c (92 bytes) we need to patch the I2c buffer size for the arduino (to optimize transfer speed). I hope the Arduino supports variable buffer size in near future. Make sure your **Arduino IDE is closed** if you patch the files!

File to patch: Java/libraries/Wire/utility/twi.h
Reason: Increase the I2C speed from 100kHz to 400kHz, increase the I2C buffer size from 32 bytes to 98 bytes

Original File	Patched File
`#ifndef TWI_FREQ`	`#ifndef TWI_FREQ`
`#define TWI_FREQ 100000L`	`#define TWI_FREQ` **`400000L`**
`#endif`	`#endif`
`#ifndef TWI_BUFFER_LENGTH`	`#ifndef TWI_BUFFER_LENGTH`
`#define TWI_BUFFER_LENGTH 32`	`#define TWI_BUFFER_LENGTH` **`98`**
`#endif`	`#endif`

File to patch: Java/libraries/Wire/Wire.h
Reason: Increase the Serial buffer size from 32 bytes to 98 bytes

Original File	Patched File
`#define BUFFER_LENGTH 32`	`#define BUFFER_LENGTH` **`98`**

Upload Firmware to Rainbowduino

Upload the firmware (see Upload Firmware), the firmware file you need is **rainbowduinoFw/Rainbow_V2_71/Rainbow_V2_71.pde**.

Note: This firmware use the I2C protocol to communicate - each Rainbowduino needs a unique I2C address. The address can be configured by editing the Rainbowduino.h file (`#define I2C_DEVICE_ADDRESS 0x06`). So dont forget to change the address if you upload this firmware to more than one rainbowduino's!

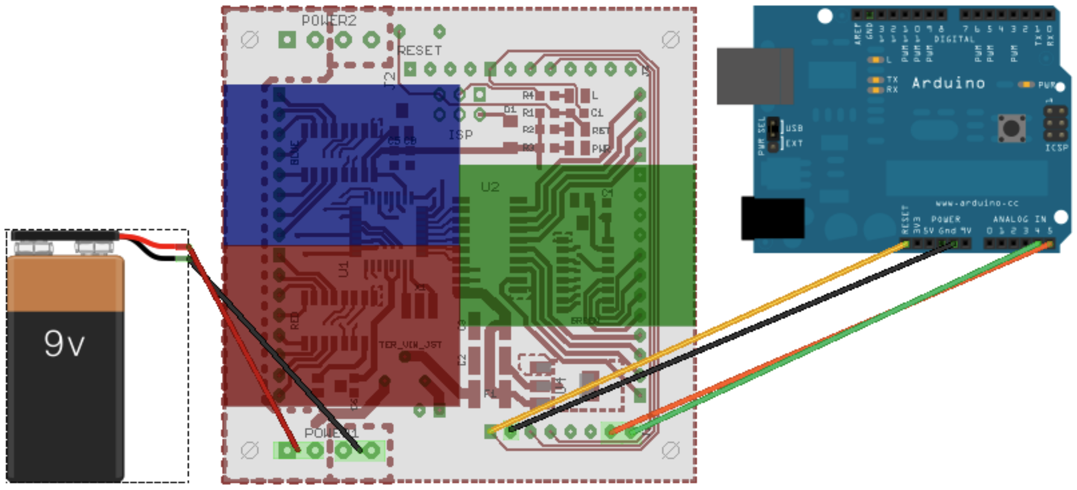
Upload Firmware to Arduino

Disconnect the RX/TX lines between Rainbowduino and Arduino. Upload the Arduino firmware **arduinoFw/neoLed/neoLed.pde** to the Arduino.

Interact with Rainbowduino

This chapter will show you a **simple way to communicate** with your Rainbowduino. You need an Arduino (working as a serial to I2C gateway) and a Rainbowduino with an I2C address of 0x06.

The connection between the Rainbowduino and Arduino should look like this:



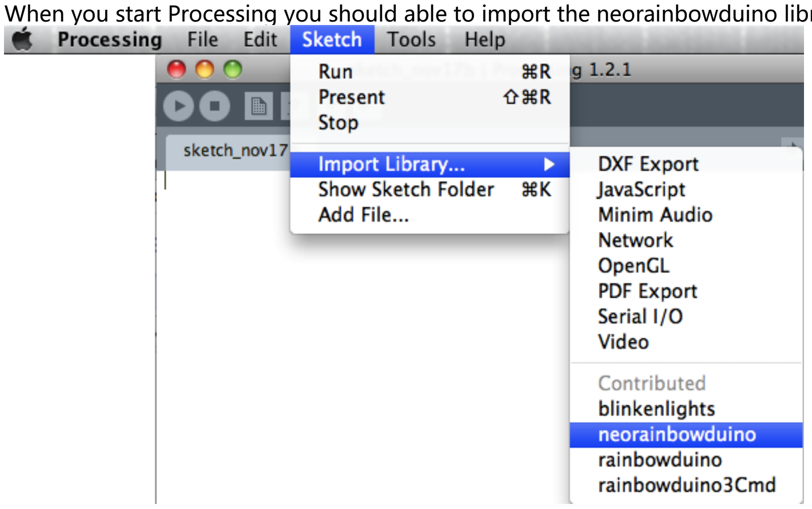
We use an external power source, however you could also use the 5V from the Arduino.

Arduino	Rainbowduino
RESET	DTR
GND	GND
Analog IN 4	SDA
Analog IN 5	SDL

Install Processing Libraries

After you installed the Processing Software, you'll need to install the neorainbowduino libraray. You can find the processing library in the **processingLib\distribution\neorainbowduino-x.y\download** directory. Unpack the zip-file to your Processing home folder (there is a README.TXT file inside for detailed instructions, how to install).

When you start Processing you should able to import the neorainbowduino library):



Simple Example

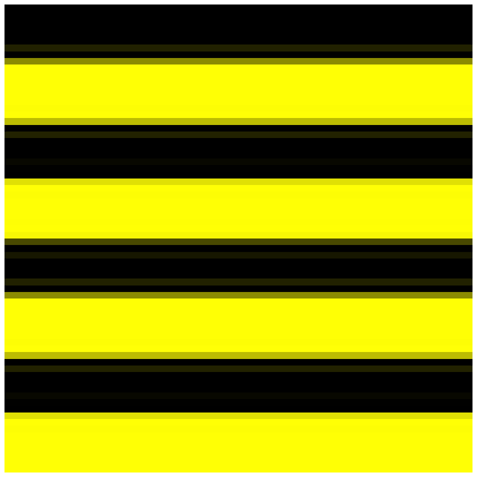
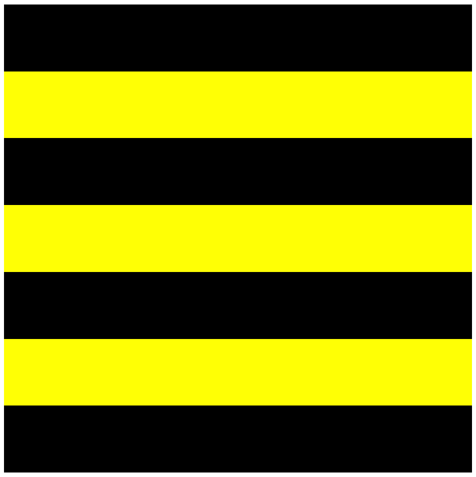
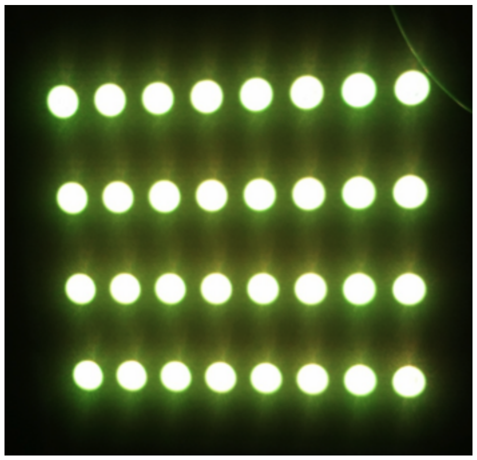
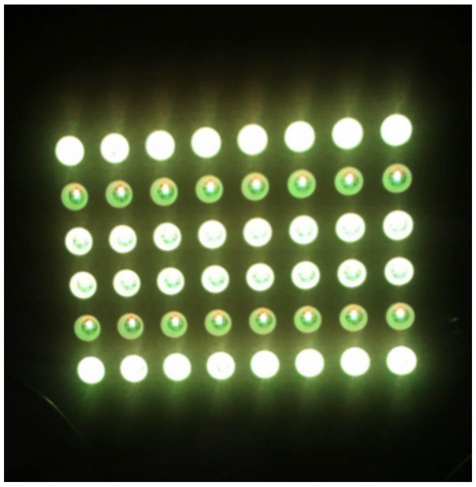
Here is a very simple Processing sketch to send som random rectangles to the rainbowduino.

```
import processing.serial.*;
import com.neophob.lib.rainbowduino.test.*;
import com.neophob.lib.rainbowduino.*;

static final int SIZE = 400;
Rainbowduino r;

void setup() {
  frameRate(15);
  background(0);
  size(SIZE, SIZE);

  //initialize rainbowduino
  List<Integer> list = new ArrayList<Integer>();
  list.add(6);         //use rainbowduino with slave id 6
  try {
    r = new Rainbowduino(this, list);
    System.out.println("ping: "+r.ping());
  } catch (Exception e) {
    println("FAILED to open serial port!!");
    e.printStackTrace();
  }

  smooth();
  noStroke();
}

void draw() {
  //draw some simple stuff on screen
  color c1 = color(128+(int)random(64), 128, (int)random(255));
  fill(c1);

  int size = 80+(int)random(80);
  int x = (int)random(SIZE);
  int y = (int)random(SIZE);
  rect(x, y, size, size);<br>
  //send PApplet to the Rainbowduino lib - and send it to slave id 6
  r.sendRgbFrame((byte)6, this);
}
```

TODO add some screenshots

How Image resizing works

The image will be resized using an Area Averaging Filter. So its important to know, that the image should be correctly aligned. Aligned means, that the result looks good if the image can be divided by 8. Here is a good and bad example:

Good example (aligned)	Bad example (not aligned)
	
	

mtXcontrol Firmware

Source: http://www.rngtng.com/mtxcontrol/
Features:

- mtXcontrol is an editor written in Processing to easily create image sequences for several output devices containing multicolor LED matrix.

Supported Work Modes: ???

Firmware 3

Source: http://code.google.com/p/rainbowduino-firmware/

Features:

- double-buffering synced with refresh rate
- 4 auxiliary buffers
- hi-level instruction set
- multiple controlled hardware
- I2C communication protocol
- permanent data storage in Eeprom

Supported Work Modes: I2C

RainbowDashboard

Source: http://code.google.com/p/rainbowdash/

Features:

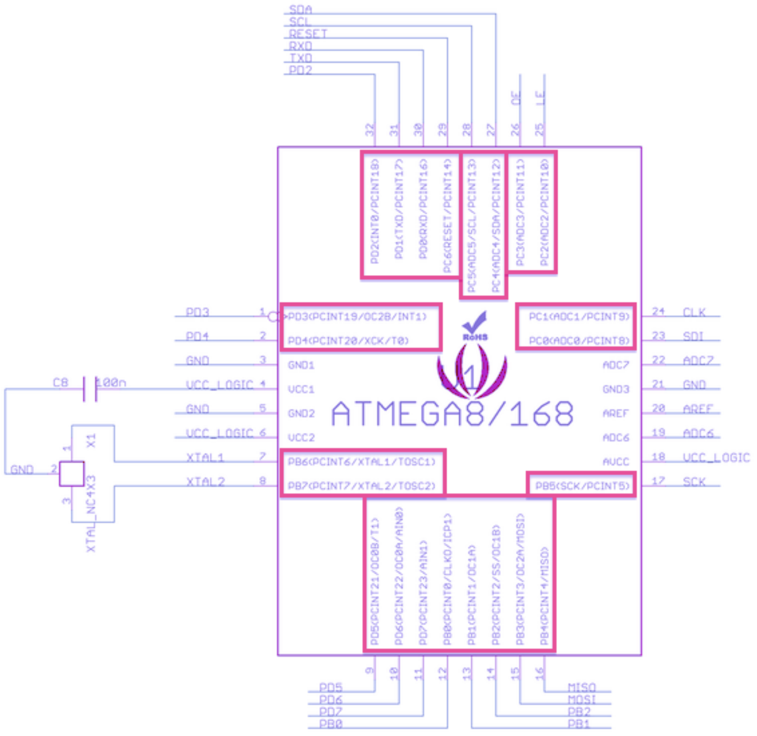
- Clean, maintainable code base.
- Compatible with standard firmware.
- Supports UART mode (no Arduino host needed - talk to Rainbowduino directly).
- Double-buffered graphics operations.
- Software real-time clock.
- Animation driven by the Rainbowduino itself.
- Full Windows ANSI (CP1252) character set.
- High-level command set.

Supported Work Modes: UART

Can easily be changed to use I2C; only one file (RainbowDash.pde) needs to be changed.

How the Firmware works

Microprocessor - Atmega 168/328

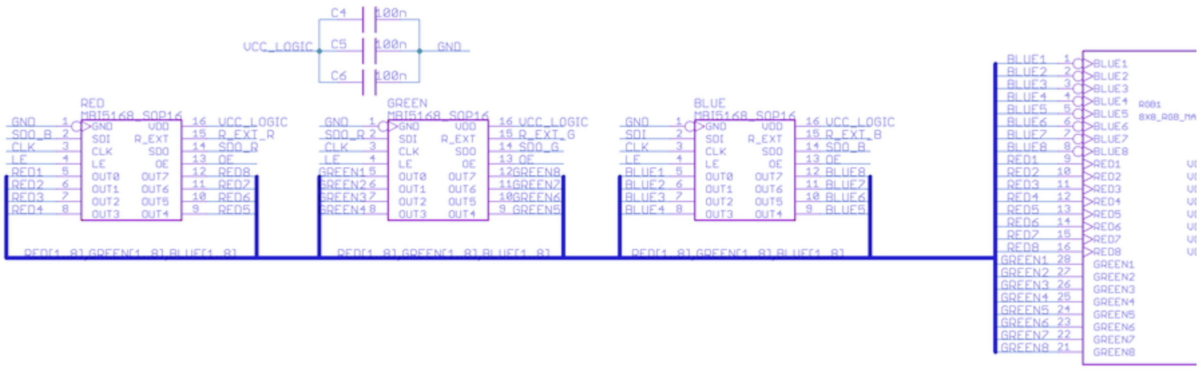


PORTD	PORTB	PORTC
pin02 / PD0 / RXD	pin14 / PB0 / INT0	pin23 / PC0 / SDI
pin03 / PD1 / TXD	pin15 / PB1 / INT1	pin24 / PC1 / CLK
pin04 / PD2 / INT0	pin16 / PB2 / INT2	pin25 / PC2 / LE
pin05 / PD3 / INT19	pin17 / PB3 / INT3	pin26 / PC3 / OE
pin06 / PD4 / INT20	pin18 / PB4 / INT4	pin27 / PC4 / SDA
pin11 / PD5 / INT21	pin19 / PB5 / INT5/SCK	pin28 / PC5 / SDL
pin12 / PD6 / INT22		
pin13 / PD7 / INT23		

PORTB maps to Arduino digital pins 8 to 13 The two high bits (6 & 7) map to the crystal pins and are not usable.
PORTC maps to Arduino analog pins 0 to 5. Pins 6 & 7 are only accessible on the Arduino Mini.
PORTD maps to Arduino digital pins 0 to 7.

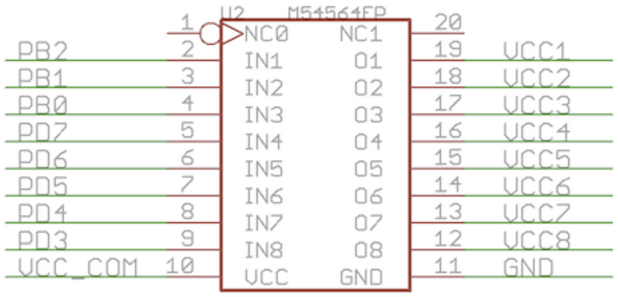
Constant Current LED driver

This driver uses the MBI5168. The MBI5168 is a 8bit shift register (http://en.wikipedia.org/wiki/Shift_register) . It converts the serial data to parallel data. All 3 MBI5168 share the LE,CLK and OE input.



Name	Desc
OE	Output Enabled, when (active) low, the output drivers are enabled; when high, all output drivers are turned OFF (blanked).
LE	Data strobe input terminal, Serial data is transfered to the respective latch when LE is high. The data is latched when LE goes low.
SDI	Serial data input to the shift register.
SDO	Serial data output to the following SDI of next driver IC.
R-EXT	Input terminal used to connect an external resistor dor setting up output current for all output channels.
CLK	Clock input terminal for data shift on rising edge

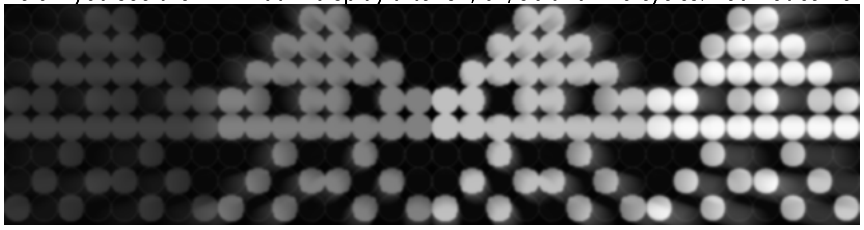
Super Source Driver



Shift out data

To display a full frame on the LED Matrix, the Rainbowduino interrupt method needs to be called 128 times. There are 8 lines and 16 brightness levels. Each time the displayNextLine() method gets called, one line gets updated by the current brightness level. After all 8 lines are updated the brightness level gets updated. That' s why this function needs 128 cycles until a full frame is populated on the LED Matrix.

Below you see the LED Matrix display after 32, 64, 96 and 128 cycles. You notice how the brightness is increased.



Support more than 4096 colors (12bit)

The stock firmware (and most 3rd party firmwares) support 12bit color resolution. It is possible to increase this:

Color Resolution	Payload	Brightness Level
12 bit (4bit per color), 4096 Colors	96 bytes (12bit*64=768bit)	16
15 bit (5bit per color), 32768 Colors	120 bytes (15bit*64=960bit)	32

The benefit of using 4bits per color is the data storage, one byte takes 2 color values - thus it's easy to get the color from a byte buffer. Using 5bits per color needs more cpu power or more buffer space (use 2 bytes for 3 color values - wasting 1bit per color).

To achieve 15 bit color resolution, the firmware needs two changes:

- loop over 32 instead 16 brightness levels
- change the shift out function

Resources

- A Huge DIY LED Matrix (http://www.neophob.com/2010/11/huge-rgb-led-matrix/)
- Generic Rainbowduino information (http://www.neophob.com/2010/07/rainbowduino-fun-aka-neorainbowduino/)
- File:RAINBOW-MBI5168 Datasheet VA.02-English.pdf
- [prod_documents (http://www.atmel.com/dyn/resources/prod_documents/doc2545.pdf)]
- File:RAINBOW-MBI5168 Datasheet VA.02-English.pdf