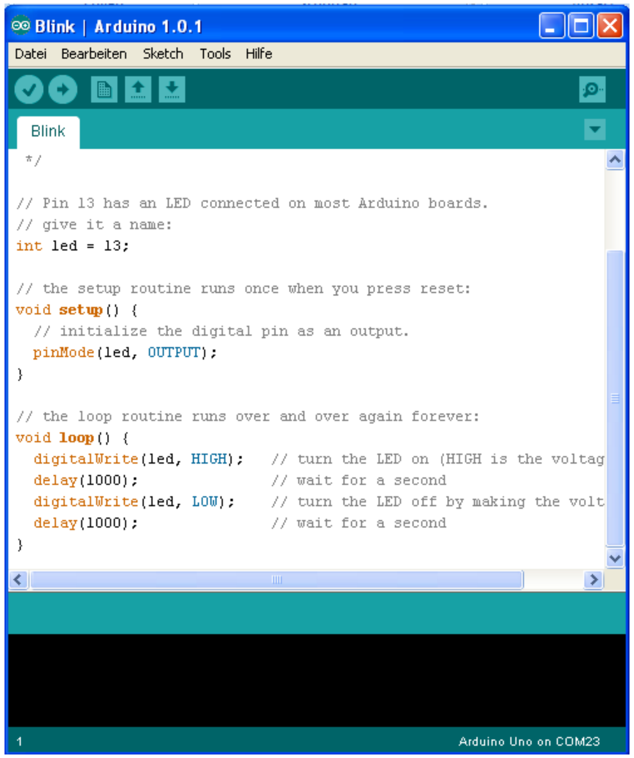
Arduino Uno

What means Arduino?

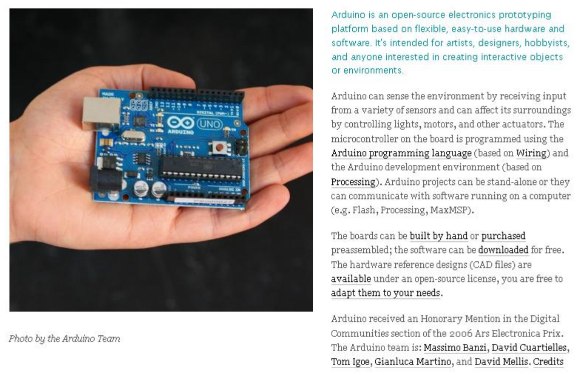
Hardware

Programming-software



```
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltag
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the volt
  delay(1000);               // wait for a second
}
```

Community



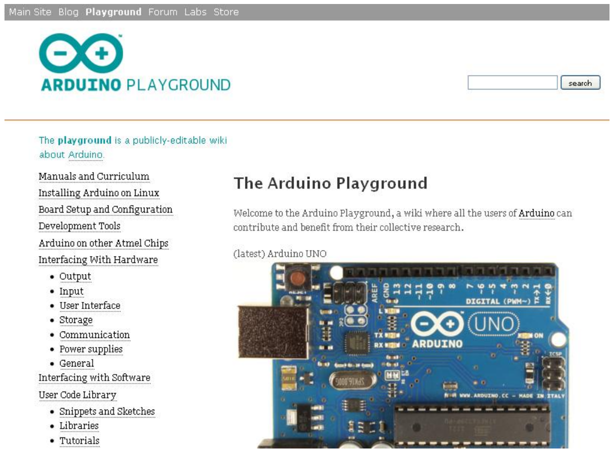
Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It's intended for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments.

Arduino can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors, and other actuators. The microcontroller on the board is programmed using the Arduino programming language (based on Wiring) and the Arduino development environment (based on Processing). Arduino projects can be stand-alone or they can communicate with software running on a computer (e.g. Flash, Processing, MaxMSP).

The boards can be built by hand or purchased preassembled; the software can be downloaded for free. The hardware reference designs (CAD files) are available under an open-source license, you are free to adapt them to your needs.

Arduino received an Honorary Mention in the Digital Communities section of the 2006 Ars Electronica Prix. The Arduino team is: Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, and David Mellis. Credits

Photo by the Arduino Team



The playground is a publicly-editable wiki about Arduino.

- Manuals and Curriculum
- Installing Arduino on Linux
- Board Setup and Configuration
- Development Tools
- Arduino on other Atmel Chips
- Interfacing With Hardware
  - Output
  - Input
  - User Interface
  - Storage
  - Communication
  - Power supplies
  - General
- Interfacing with Software
- User Code Library
  - Snippets and Sketches
  - Libraries
  - Tutorials

The Arduino Playground

Welcome to the Arduino Playground, a wiki where all the users of Arduino can contribute and benefit from their collective research.

(latest) Arduino UNO

www.arduino.cc

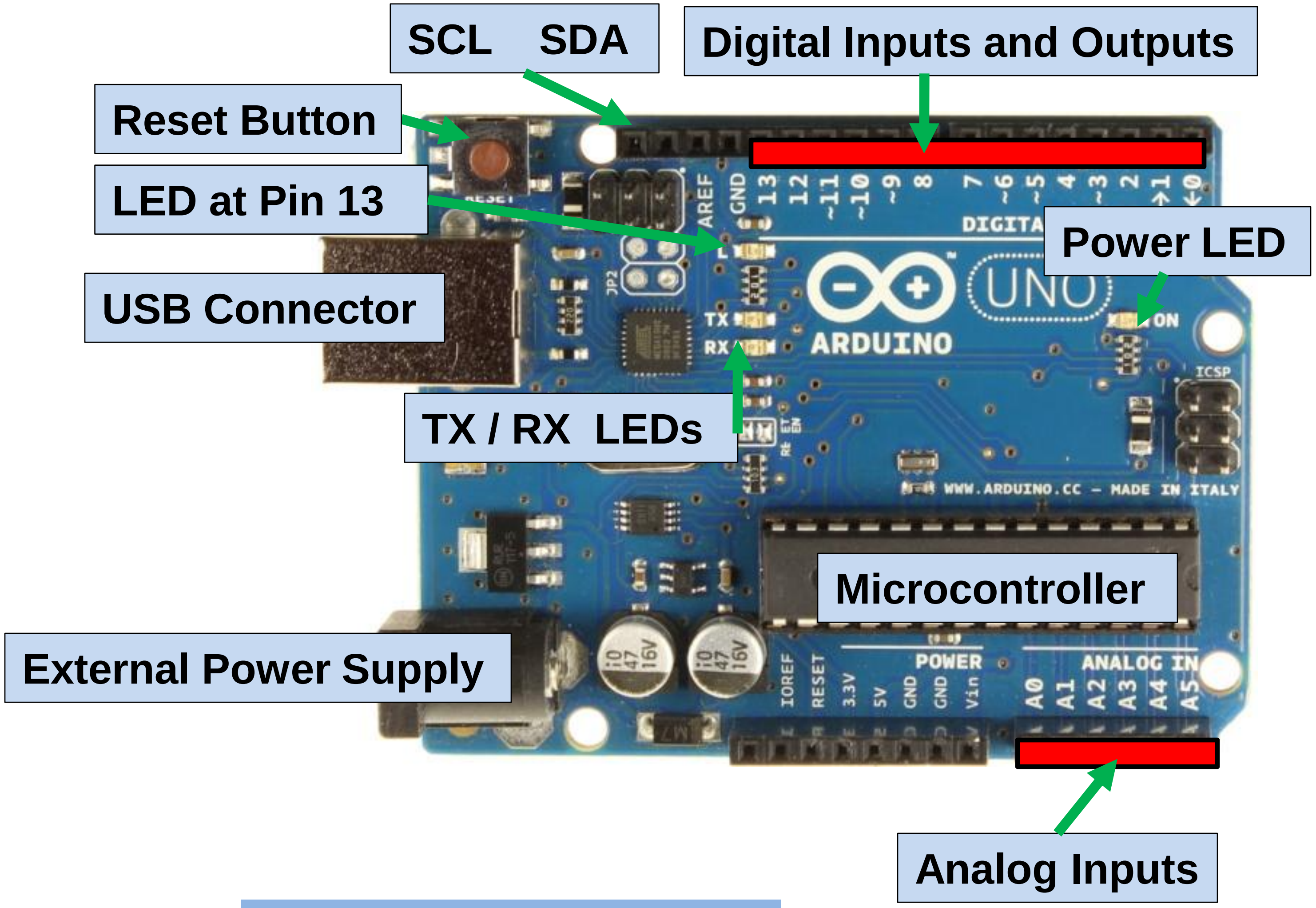
Hardware

- Cheap, fast and open
- AVR Atmega 328 Microcontroller
- C-Programming
- Programming via USB
- Power supply via USB or external

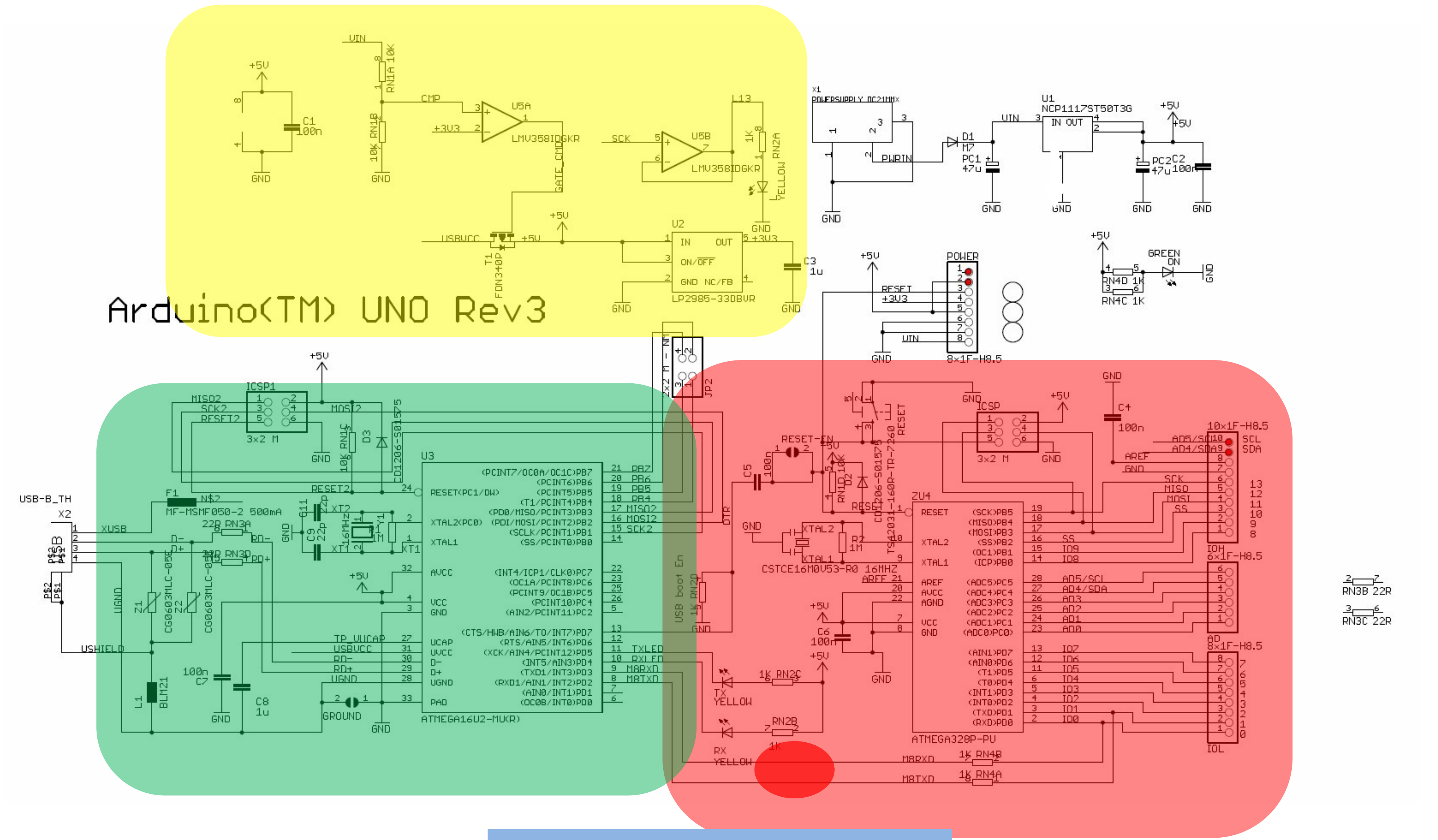
Arduino Uno Characteristics

- 32 kByte Flash Memory
- 1 kByte EEPROM
- 2 kByte SRAM
- 16 MHz Clock
- Inputs and Outputs
  - 14 digital Inputs/Outputs
  - 6 analog Inputs
  - 6 PWM-Outputs
  - I^2C-Bus, serial Bus (TX/RX)

Arduino Uno R3 Board

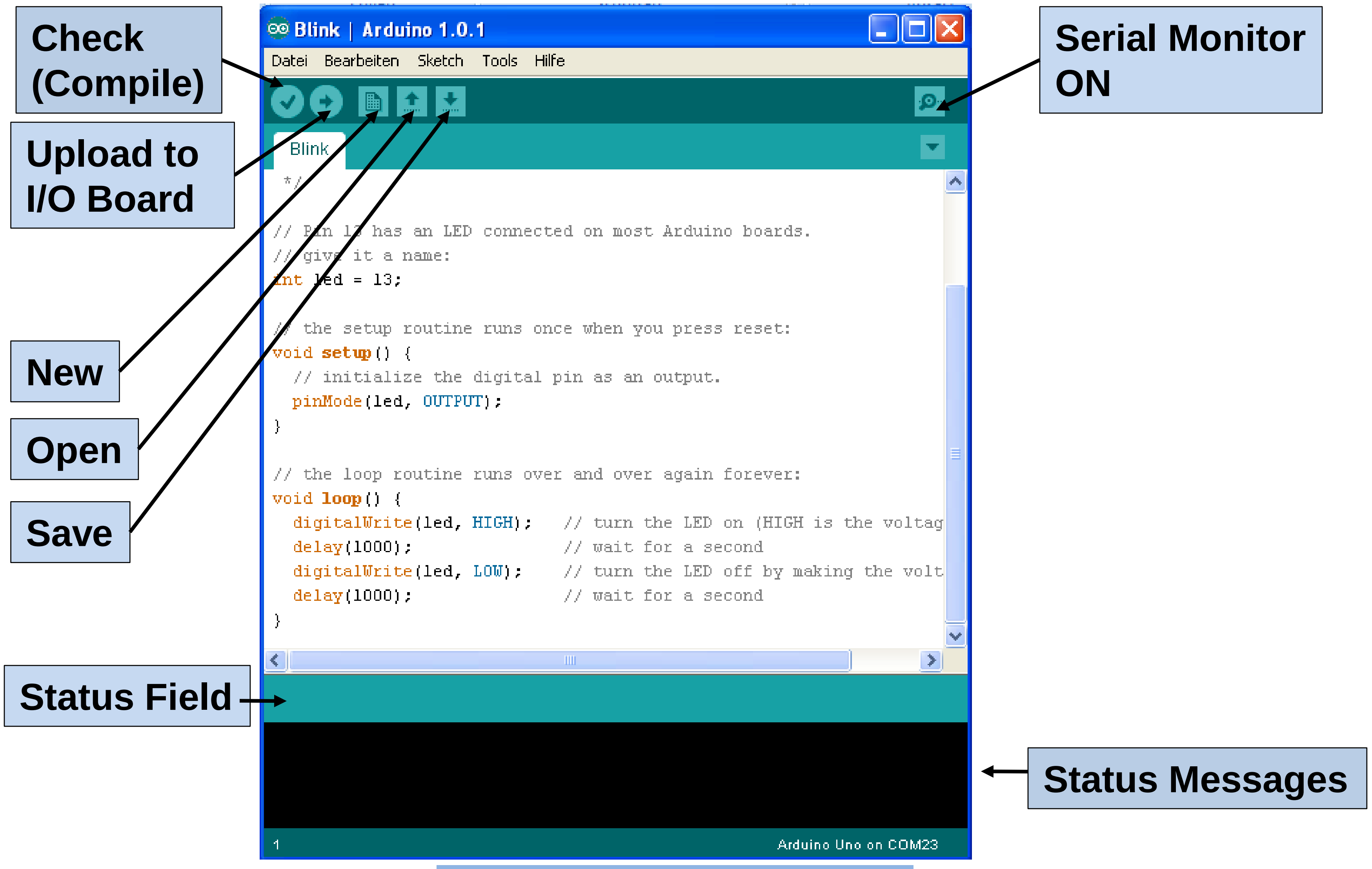


Arduino Uno R3 Schematic



Arduino Uno

Arduino-Software



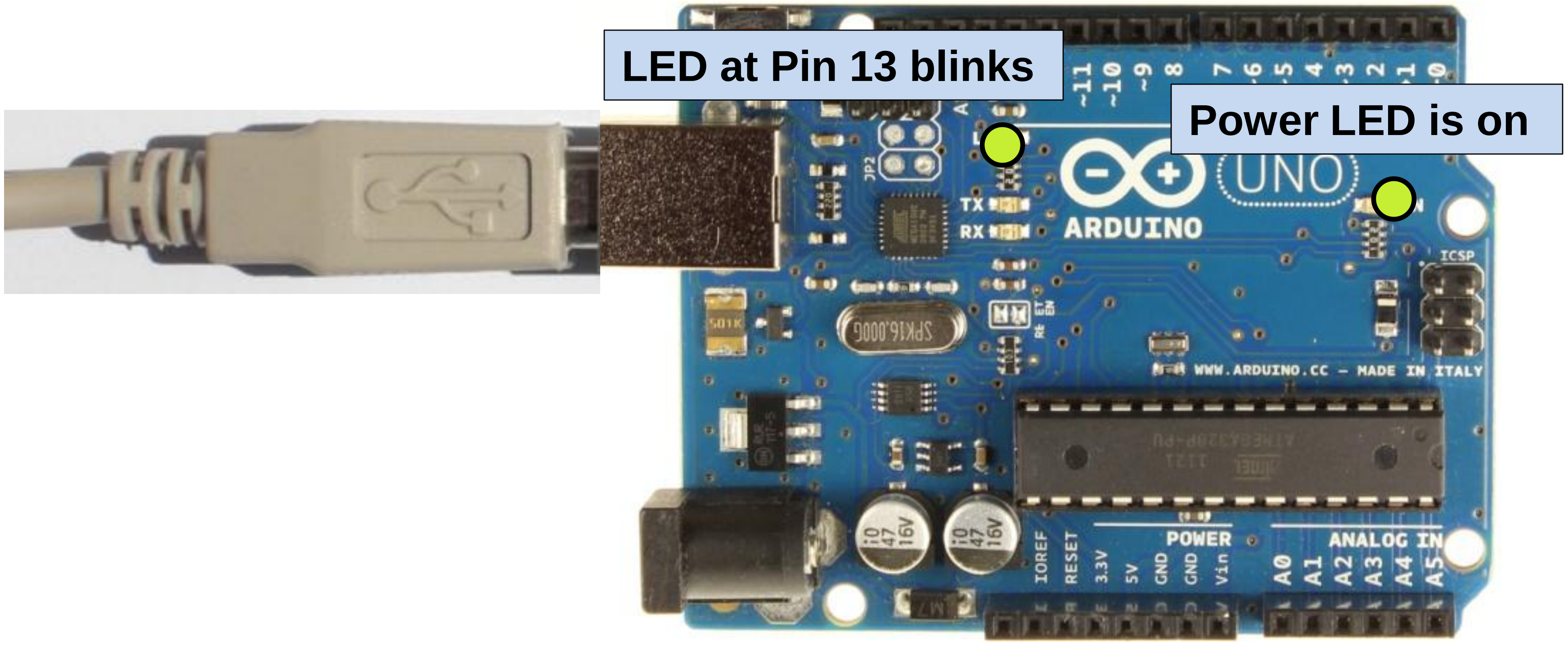
```
*/

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltag
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the volt
  delay(1000);               // wait for a second
}
```

Arduino Uno

Installation

- Unzip of the Arduino-software
- Connection of the Arduino-board
- Installation of the drivers (administrator rights needed)
- Run the Arduino-software
- Go on……..

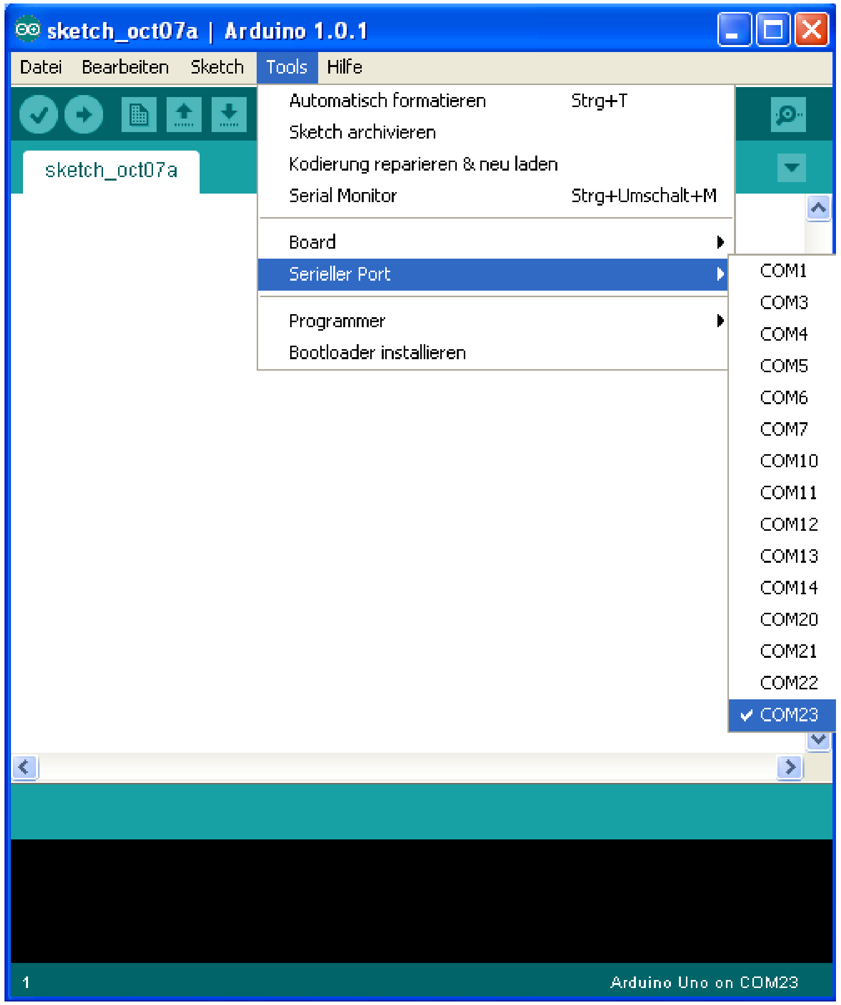
Connection of the Arduino-Board



Installing drivers

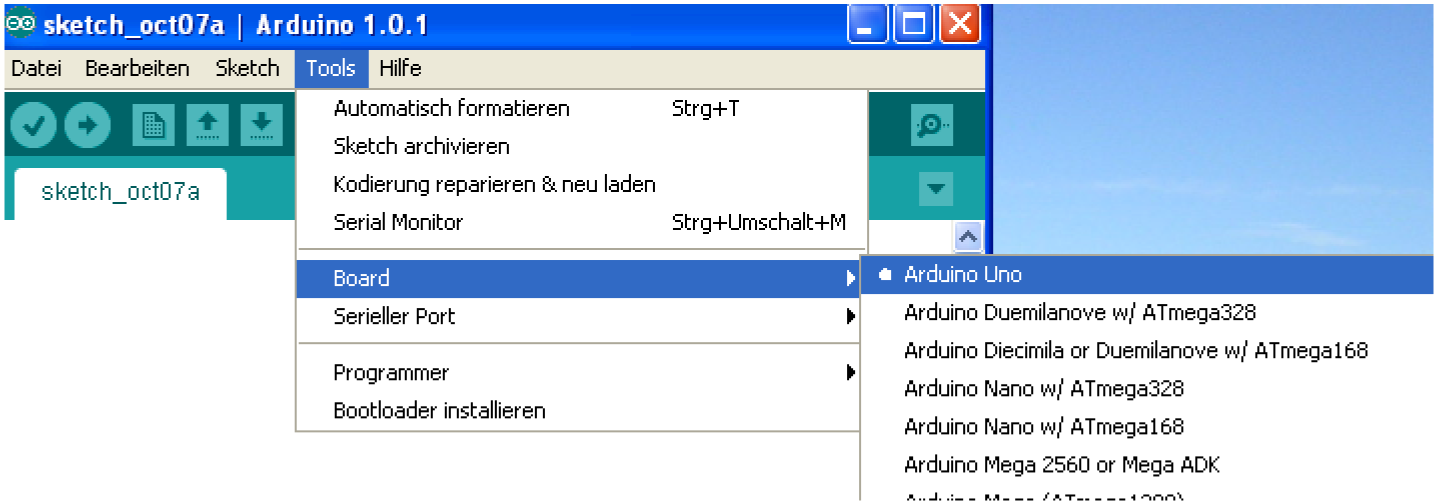
- **Installing drivers for the Arduino Uno with Windows7, Vista, or XP:**
- Plug in your board and wait for Windows to begin it's driver installation process. After a few moments, the process will fail.
- Click on the Start Menu, and open up the Control Panel.
- While in the Control Panel, navigate to System and Security. Next, click on System. Once the System window is up, open the Device Manager.
- Look under Ports (COM & LPT). Â You should see an open port named "Arduino UNO (COMxx)"
- Right click on the "Arduino UNO (COmxx)" port and choose the "Update Driver Software" option.
- Next, choose the "Browse my computer for Driver software" option.
- Finally, navigate to and select the Uno's driver file, named **"ArduinoUNO.inf"**, located in the "Drivers" folder of the Arduino Software download (not the "FTDI USB Drivers" sub-directory).
- Windows will finish up the driver installation from there.

Selecting the COM-Port



Arduino Uno

Selecting the Board


sketch_oct07a | Arduino 1.0.1

Datei Bearbeiten Sketch Tools Hilfe

sketch_oct07a

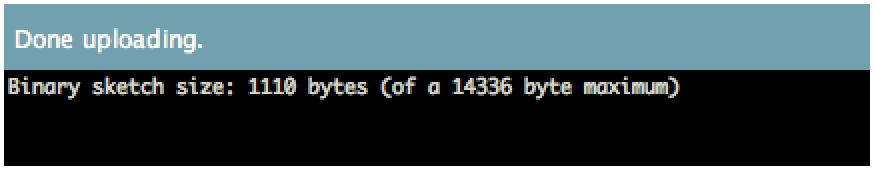
Automatisch formatieren Strg+T
Sketch archivieren
Kodierung reparieren & neu laden
Serial Monitor Strg+Umschalt+M
Board
Serieller Port
Programmer
Bootloader installieren

● Arduino Uno
Arduino Duemilanove w/ ATmega328
Arduino Diecimila or Duemilanove w/ ATmega168
Arduino Nano w/ ATmega328
Arduino Nano w/ ATmega168
Arduino Mega 2560 or Mega ADK

Arduino Uno

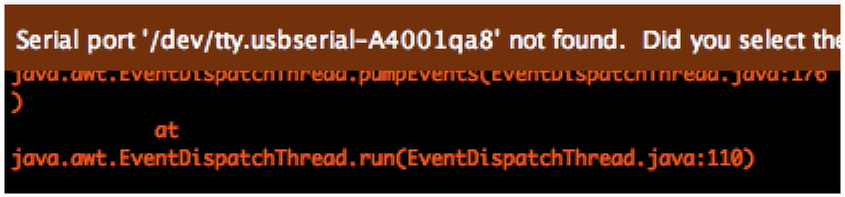
Status-Messages

Upload done



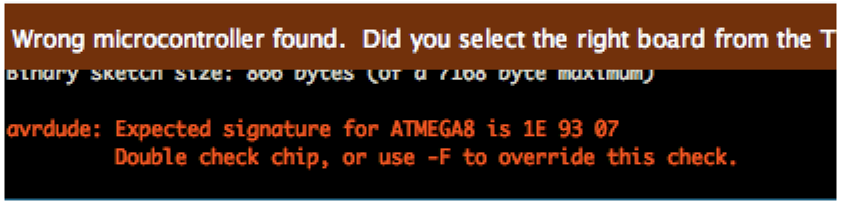
```
Done uploading.
Binary sketch size: 1110 bytes (of a 14336 byte maximum)
```

Wrong serial port



```
Serial port '/dev/tty.usbserial-A4001qa8' not found.  Did you select the
java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:176
)
            at
java.awt.EventDispatchThread.run(EventDispatchThread.java:110)
```

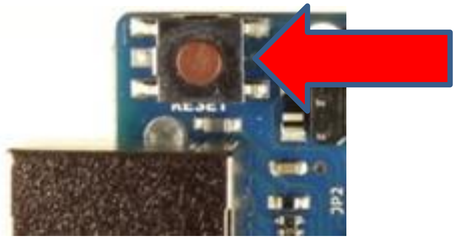
Wrong board



```
Wrong microcontroller found.  Did you select the right board from the T
Binary sketch size: 866 bytes (of a 7168 byte maximum)

avrdude: Expected signature for ATMEGA8 is 1E 93 07
         Double check chip, or use -F to override this check.
```

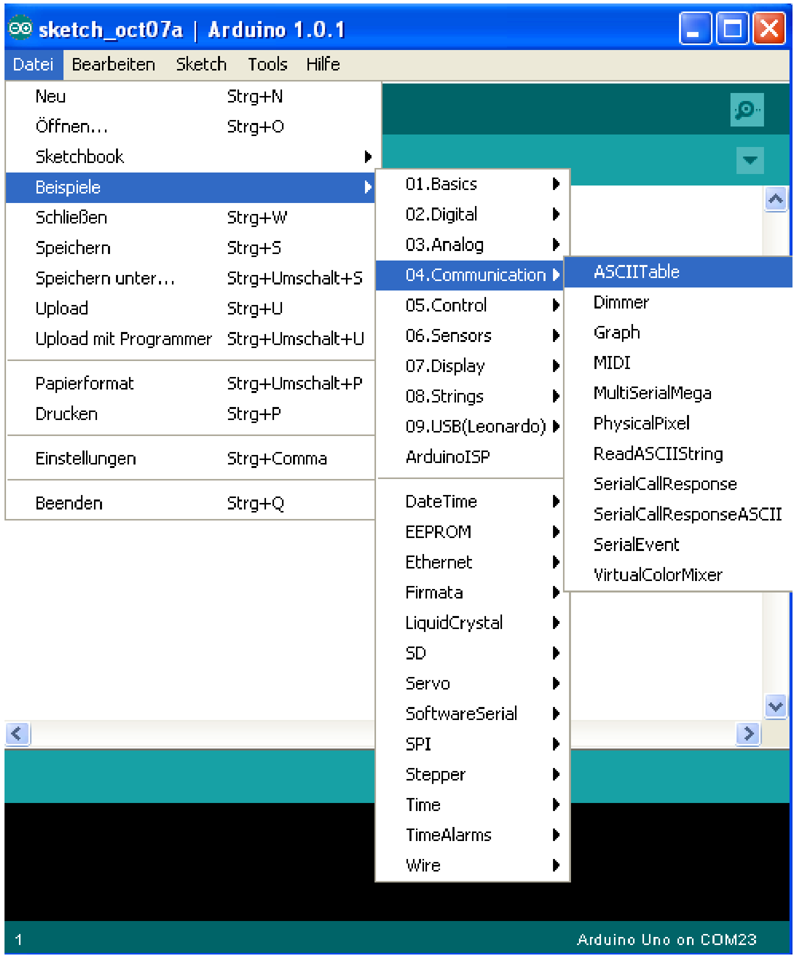
Troubleshooting

- Press the reset-button on Arduino and try again



- Check the serial port (Connection and number)
- Read the red text (Debugging output) at the bottom to determine the problem
- The status area shows what is wrong

Program Examples



Program-Structure

- Declaration of variables

```
int ledPin = 13;                              // LED connected to digital pin 13
```

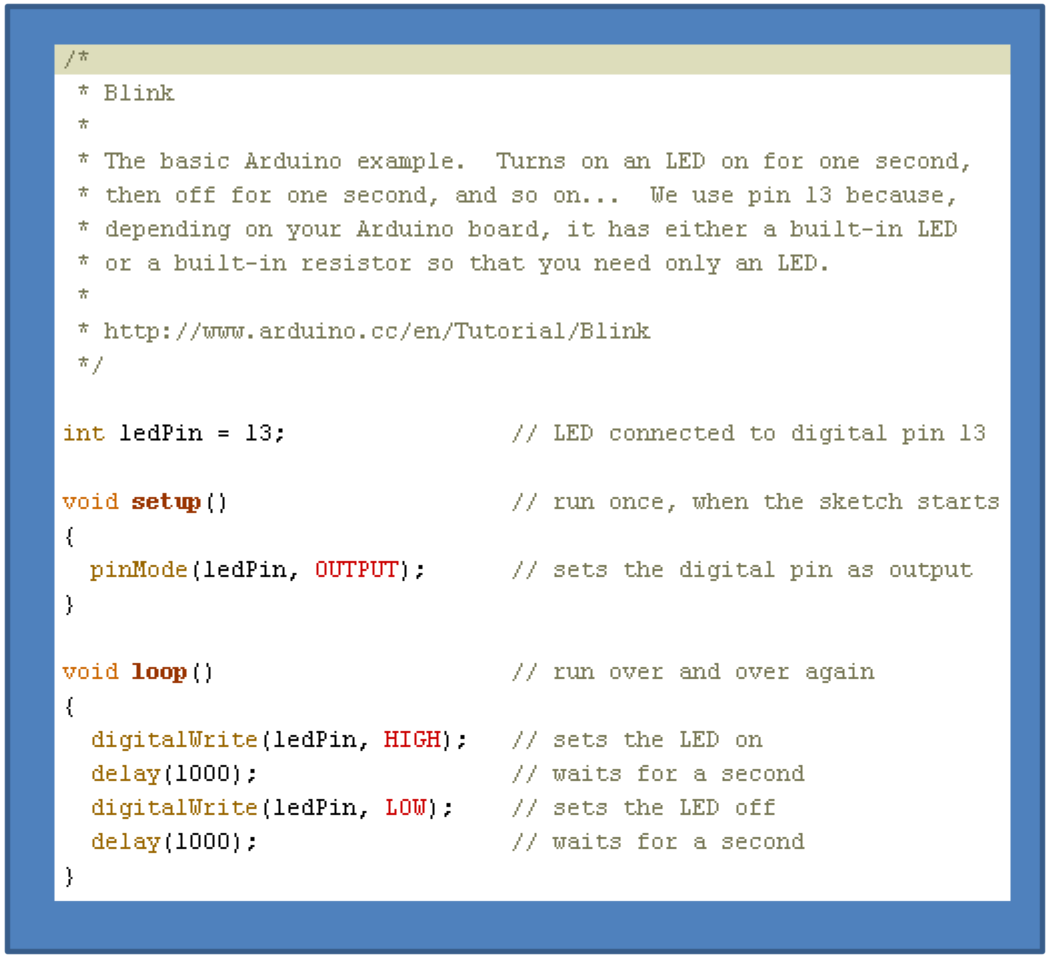
- Initialization
  - **setup( )** Set the inputs and outputs

```
void setup()                                  // run once, when the sketch starts
{
  pinMode(ledPin, OUTPUT);                    // sets the digital pin as output
}
```

- Main program
  - **loop( )** Loop without end

```
void loop()                                   // run over and over again
{
  digitalWrite(ledPin, HIGH);                 // sets the LED on
  delay(1000);                                // waits for a second
  digitalWrite(ledPin, LOW);                  // sets the LED off
  delay(1000);                                // waits for a second
}
```

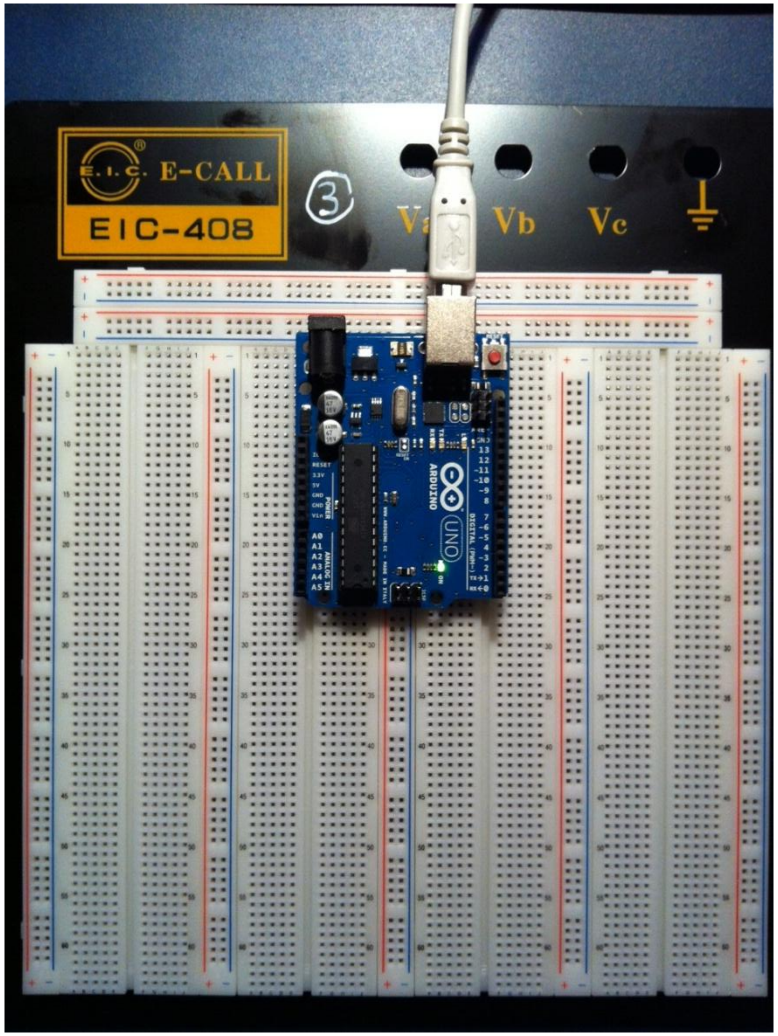
The blinking LED



```
/*
 * Blink
 *
 * The basic Arduino example.  Turns on an LED on for one second,
 * then off for one second, and so on...  We use pin 13 because,
 * depending on your Arduino board, it has either a built-in LED
 * or a built-in resistor so that you need only an LED.
 *
 * http://www.arduino.cc/en/Tutorial/Blink
 */

int ledPin = 13;                 // LED connected to digital pin 13

void setup()                     // run once, when the sketch starts
{
  pinMode(ledPin, OUTPUT);       // sets the digital pin as output
}

void loop()                      // run over and over again
{
  digitalWrite(ledPin, HIGH);    // sets the LED on
  delay(1000);                   // waits for a second
  digitalWrite(ledPin, LOW);     // sets the LED off
  delay(1000);                   // waits for a second
}
```

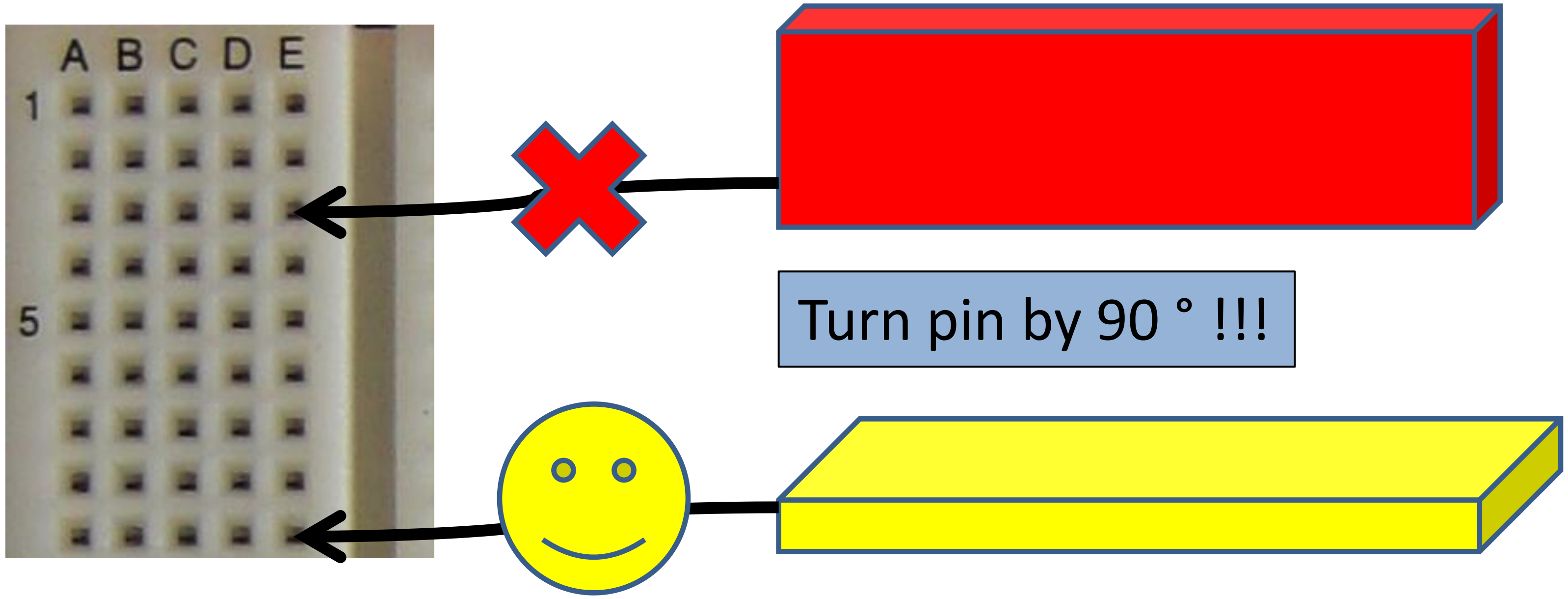
Arduino Uno

Hardware

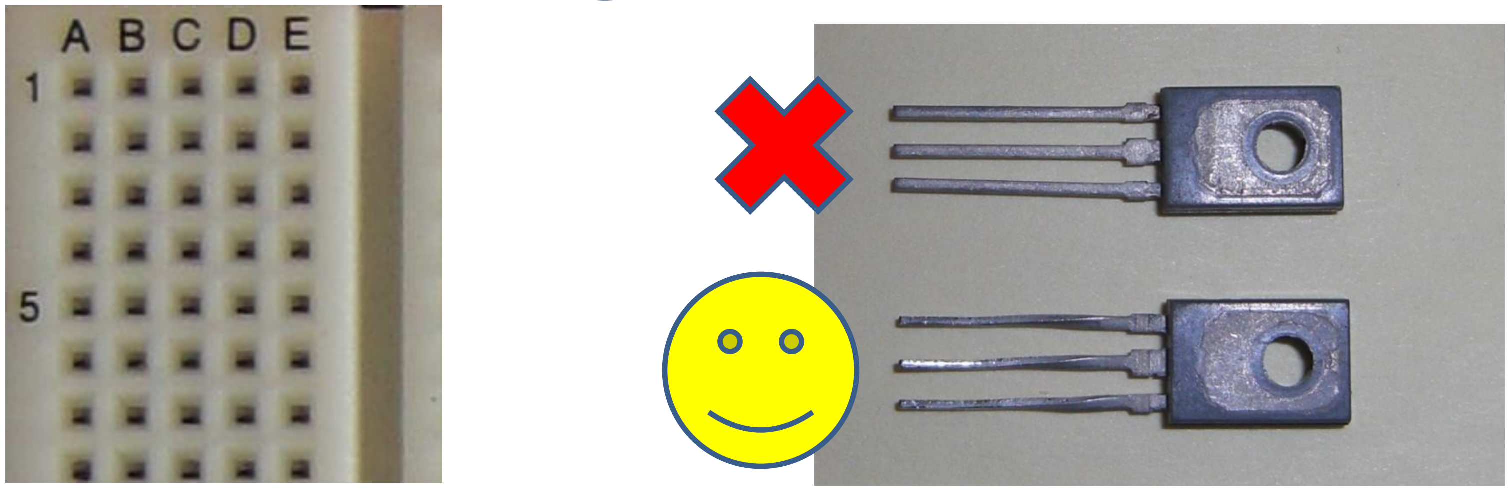


Arduino Uno

Preparing special pins



Turn pin by 90 ° !!!

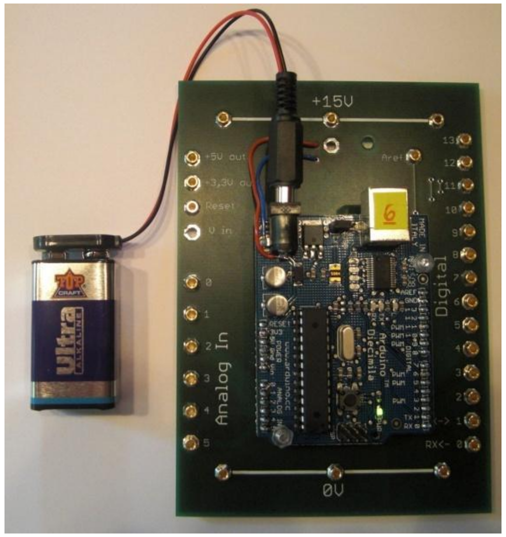


Power Supply

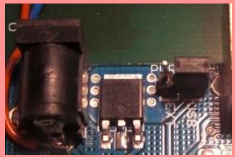
- From USB (Current is limited to 500 mA)
- External power supply (Uno switches automatically) (V_{IN} and GND or power jack)

SMPS

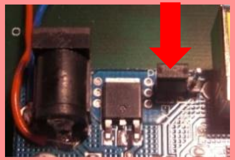
Battery



Diecimila



Jumper to EXT



Rules for the Development

- First draw the circuit
- Program the Arduino before you connect the inputs and outputs!
- If you have different power supplies connect the different GNDs if necessary
- Connect and test the circuit on the solderless board before you connect it to the Arduino
- Connect the power supplies when the circuit is complete and tested

Digital and Analog Input/Output

Digital I/O

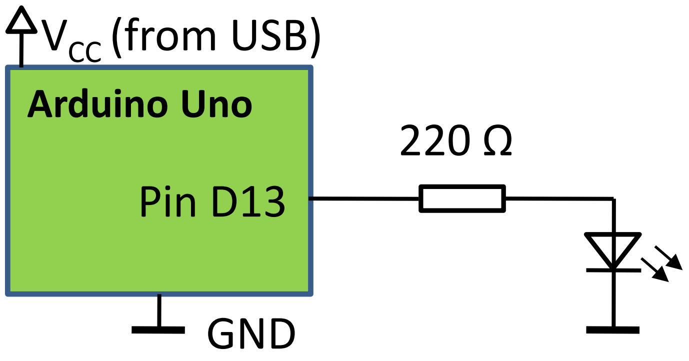
```
pinmode(pin, mode)          - initialization
digitalWrite(pin, value)
int digitalRead(pin)
```

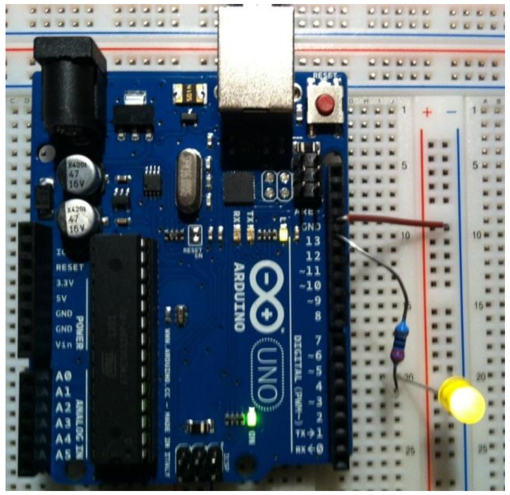
Analog I/O

```
analogReference(type)       - initialization
int analogRead(pin)
analogWrite(pin, value)     - PWM
```

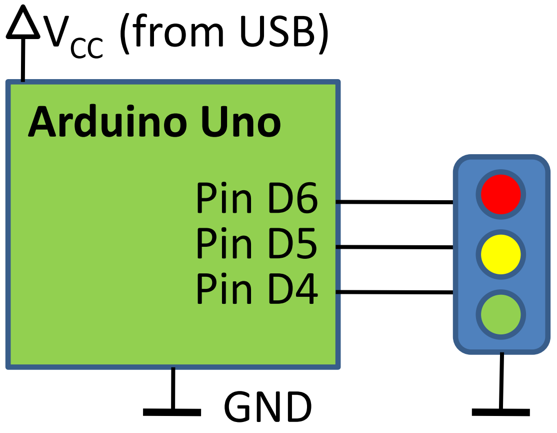
Digital Output

- Make an external LED at pin 13 blinking





- Write a program for a traffic light with 3 LEDs

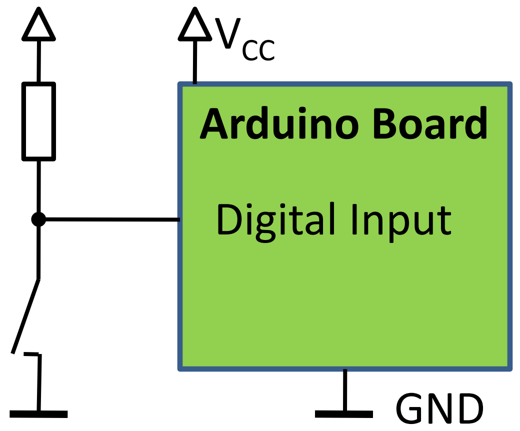


Digital Input

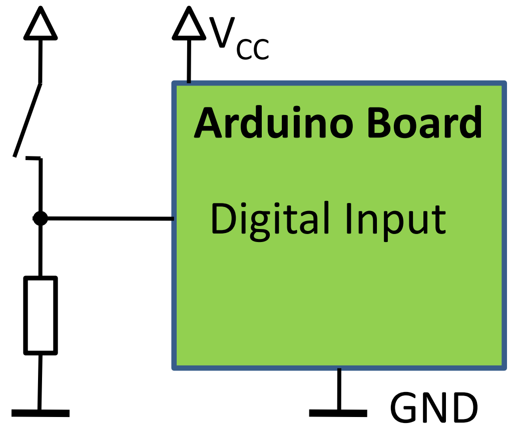
- A digital input floats between 0 and 5 V, if it is not connected
- A resistor pulls an input to 5V (pull up) or to GND (pull down)
- Using a pullup-resistor the switch pushes the input to GND
- Using a pulldown-resistor the switch pushes the input to 5 V

Digital Input

Pullup-resistor



Pulldown-resistor



Tasks for Digital Input

- Connect a switch to digital input 2 of the Arduino
- The switch controls the function of the traffic light:

High:	Normal function
Low:	Yellow light blinking

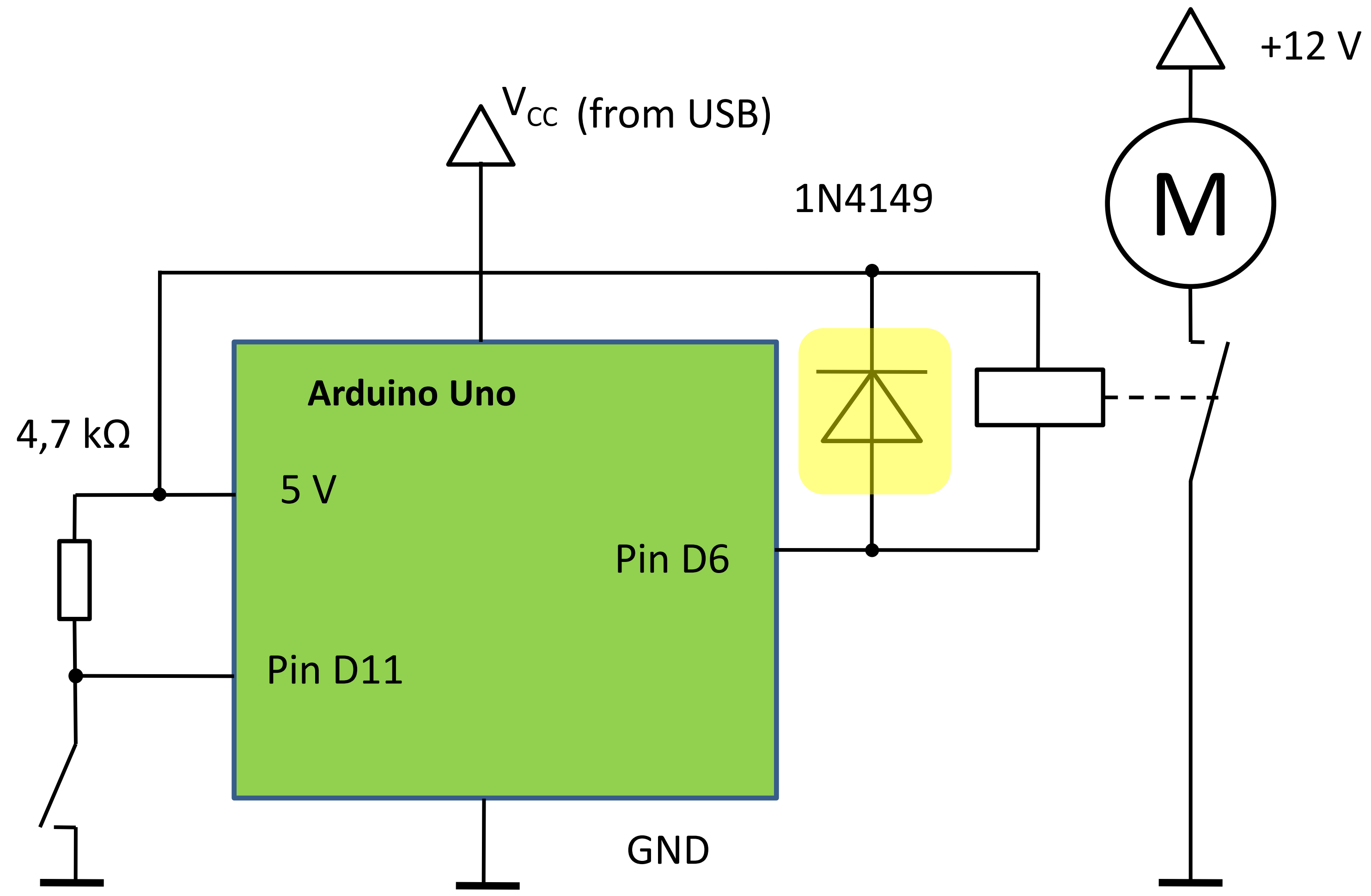
Digital Output expanded

- Maximum of an ATmega8 output:
  5 V and 40 mA
- The output can be expanded by a relay or a transistor:

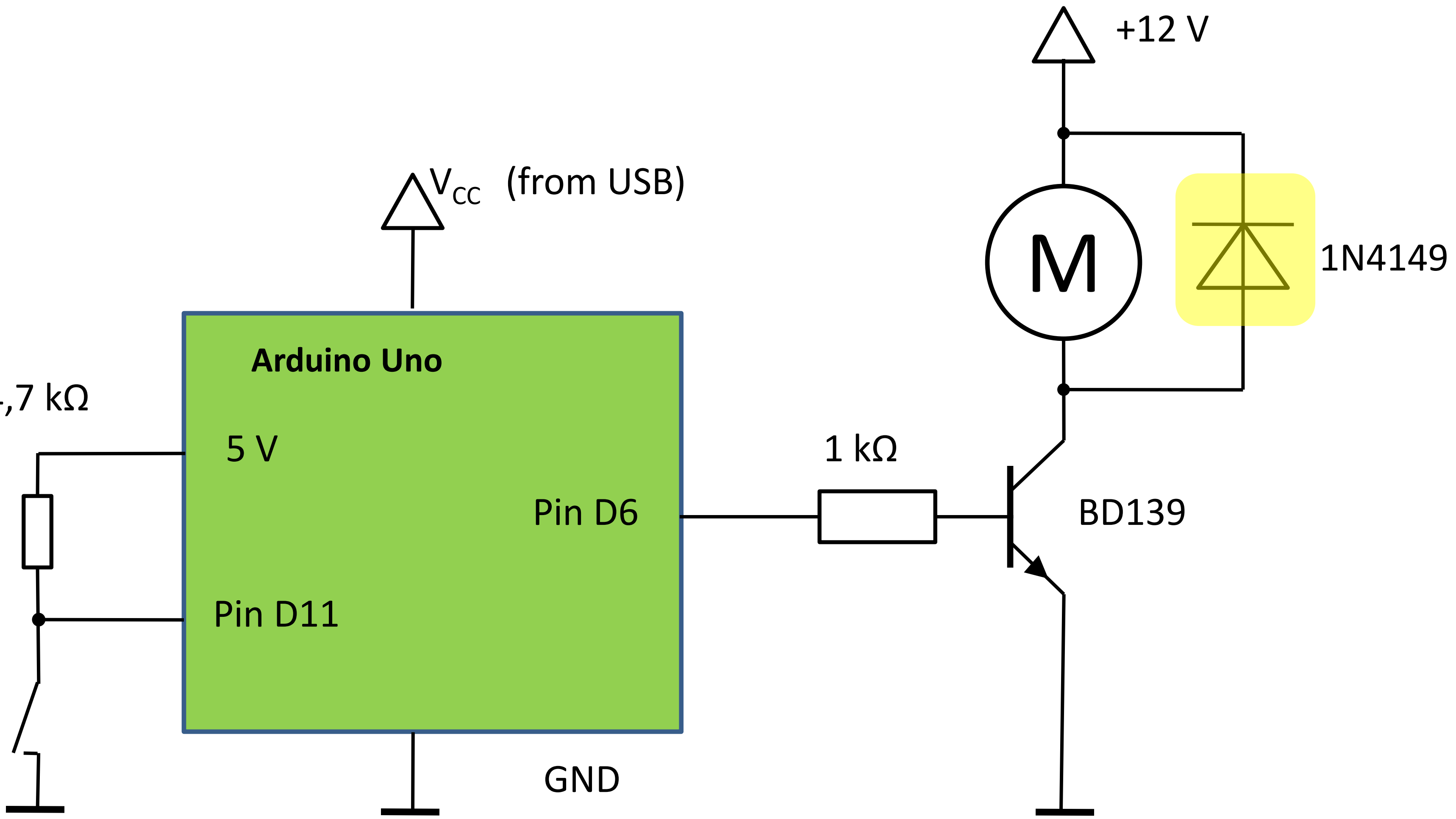
Relay:	5 V type
Transistor:	Emitter to GND
	Base resistor

Tasks for Digital Output expanded

- Connect the 12 V motor to pin 6 of the Arduino first over a relay and then over a npn-transistor (BD 139).
- For the motor use an external supply voltage (don´t forget to connect the different GNDs!).
- Switch the motor on and off by a switch at pin 11 of the Arduino.
- The base resistor of the transistor is 1 kΩ.
- Protect the Arduino and the transistor by a protective diode!


+12 V
V_{CC} (from USB)
1N4149
M
Arduino Uno
4,7 kΩ
5 V
Pin D6
Pin D11
GND

Arduino Uno



+12 V
V$_{CC}$ (from USB)
1N4149
M
Arduino Uno
4,7 kΩ
5 V
1 kΩ
Pin D6
BD139
Pin D11
GND

Arduino Uno

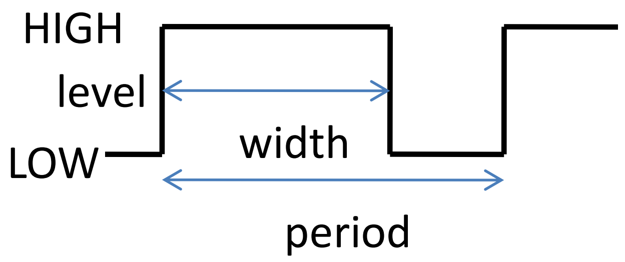
PWM Output

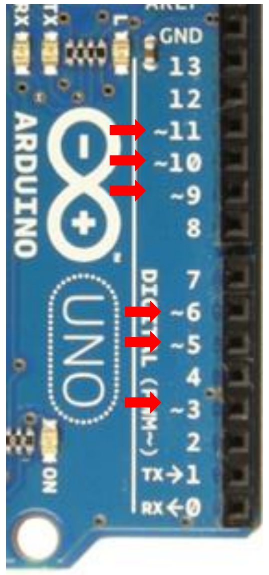
- Pulse Width Modulation
- Characteristics:

  Pulse width range

  Pulse period

  Voltage levels





- Average is like an analog voltage U_{AV}

  U_{AV} = width/period *(HIGH – LOW) + LOW
- For PWM use the **analogWrite()** instruction

Analog Input

- The ATmega 168 has 6 ADC inputs
- The maximum input range is from 0 V to 5 V
- The resolution is 10 bit (1024 values)
- The reference voltage is variable

The ADC of the Arduino

- $0 \leq U_{IN} \leq U_{REF}$
- U_{REF} : 1.1 V, 3.3 V and 5 V

analogReference(type)
Description
Configures the reference voltage used for analog input. The analogRead() function will return 1023 for an input equal to the reference voltage. The options are:
DEFAULT: the default analog reference of 5 volts.
INTERNAL: an built-in reference, equal to 1.1 volts on the ATmega168 and 2.56 volts on the ATmega8.
EXTERNAL: the voltage applied to the AREF pin is used as the reference.
Parameters
type: which type of reference to use (DEFAULT, INTERNAL, or EXTERNAL).

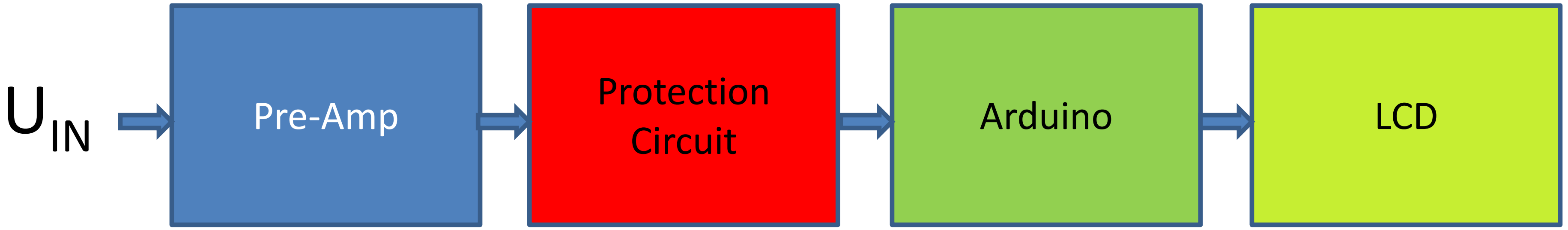
Characteristics of the Voltmeter

- High-impedance input
- Input-range: -5 V to + 5 V
- U_{REF} = 5 V
- Output on LCD :

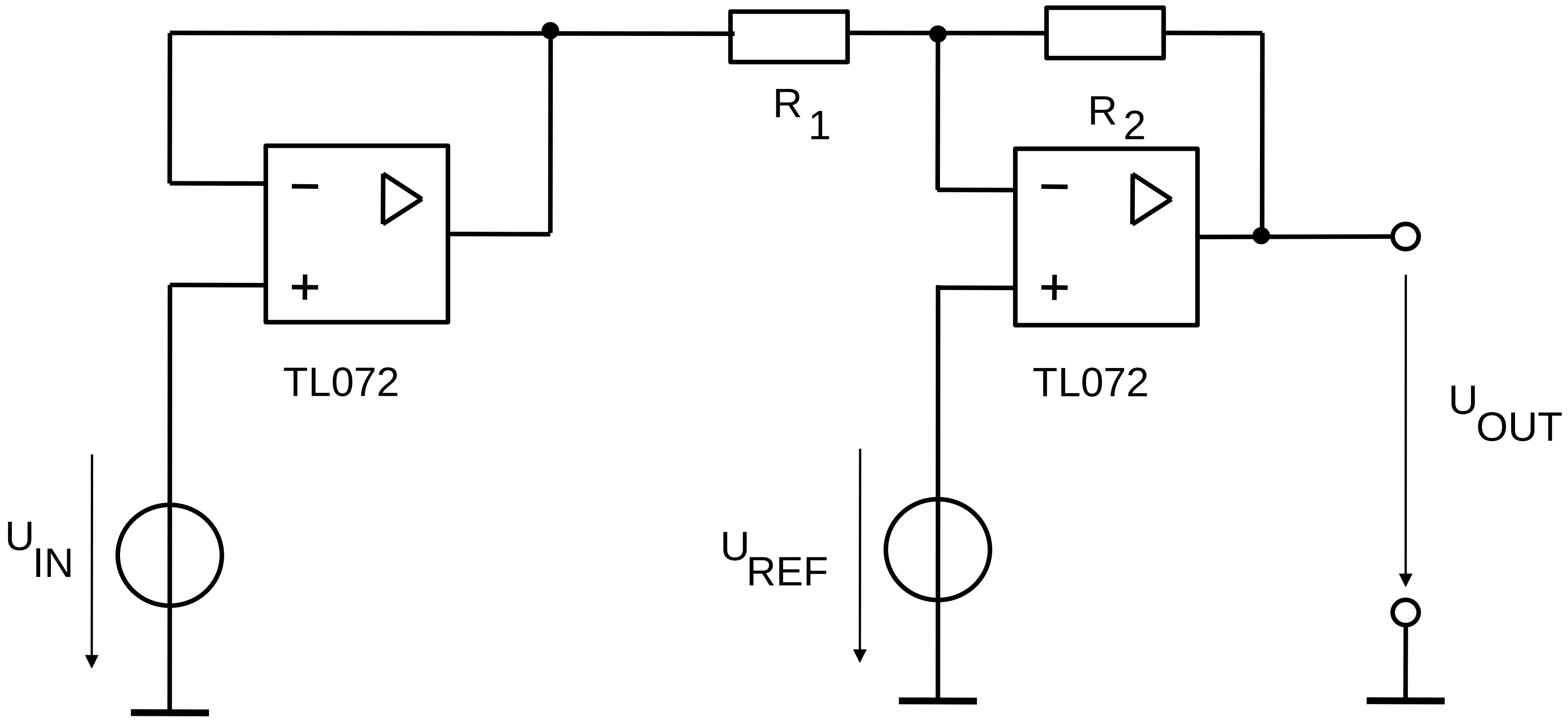
+	2	.	7	3		V	

-	8	7	6		m	V	

Blockdiagram of the Voltmeter



Pre-Amplifier (Level-Shifter)



Arduino Uno

Calculation of the PreAmp

$$U_{REF} = U_{IN} \cdot \frac{R_2}{R_1 + R_2} + U_{OUT} \cdot \frac{R_1}{R_1 + R_2}$$

U_{IN} = 5 V, U_{OUT} = 0 V: $U_{REF} = 5V \cdot \frac{R_2}{R_1 + R_2}$

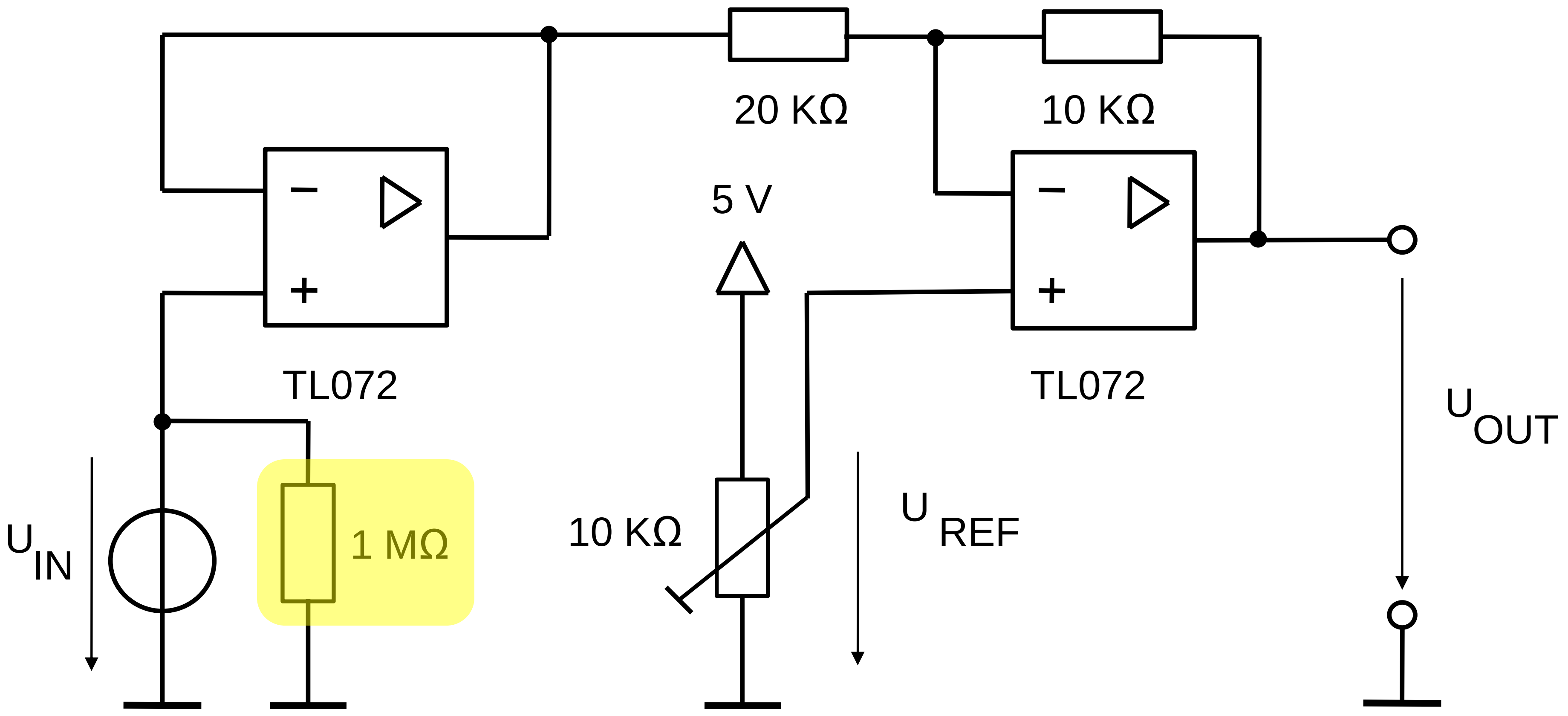
U_{IN} = -5 V, U_{OUT} = 5 V: $U_0 = -5V \cdot \frac{R_2}{R_1 + R_2} + 5V \cdot \frac{R_1}{R_1 + R_2}$

$$-5V \cdot \frac{R_2}{R_1 + R_2} + 5V \cdot \frac{R_1}{R_1 + R_2} = 5V \cdot \frac{R_2}{R_1 + R_2}$$

$$-\frac{R_2}{R_1 + R_2} + \frac{R_1}{R_1 + R_2} = \frac{R_2}{R_1 + R_2}$$

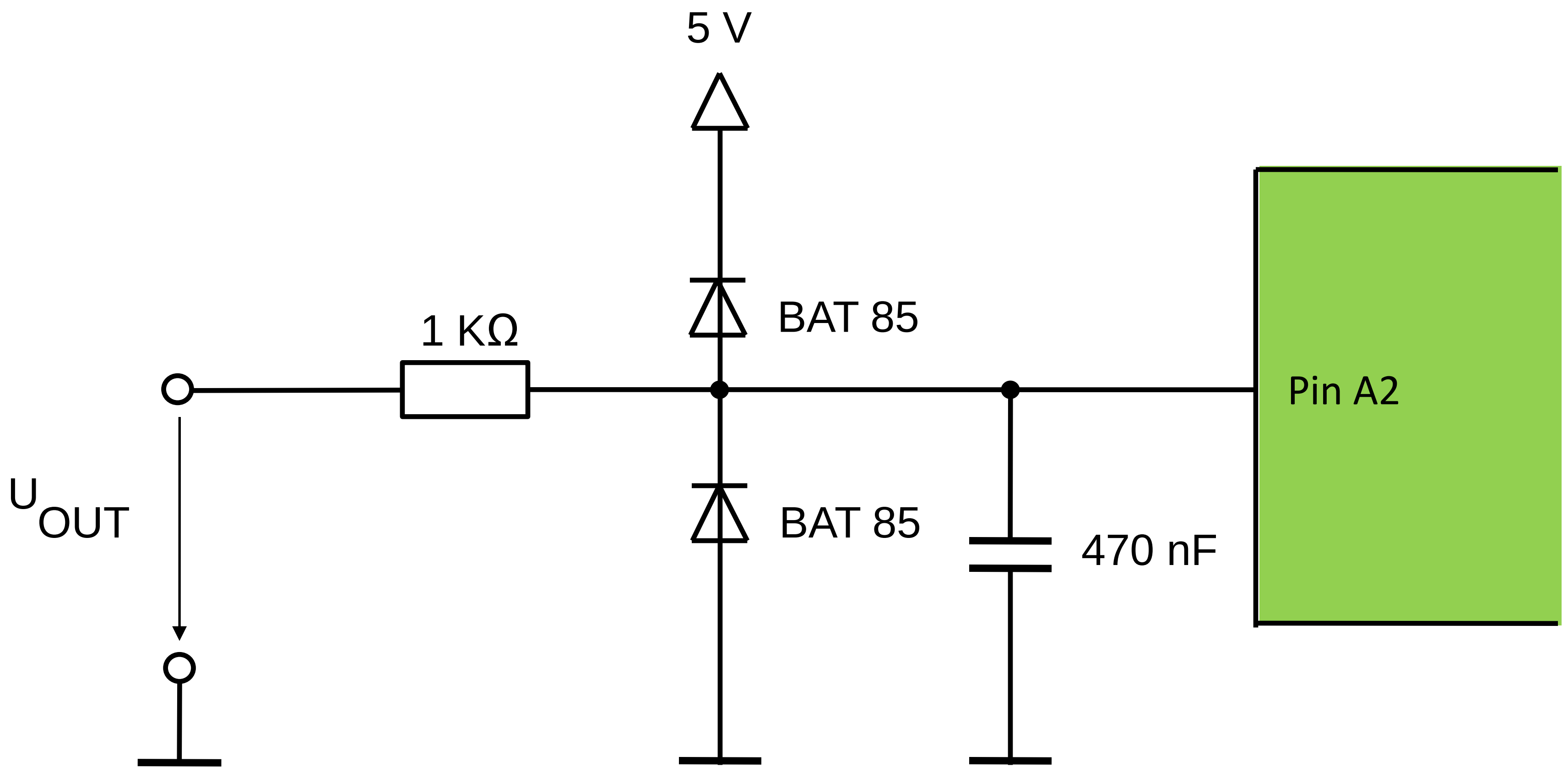
=>: $\underline{\underline{R_1 = 2R_2}}$; =>: $\underline{\underline{U_{REF}}} = 5V \cdot \frac{R_2}{2R_2 + R_2} = \underline{\underline{\frac{5}{3}V}}$

Pre-Amplifier (Level-Shifter)



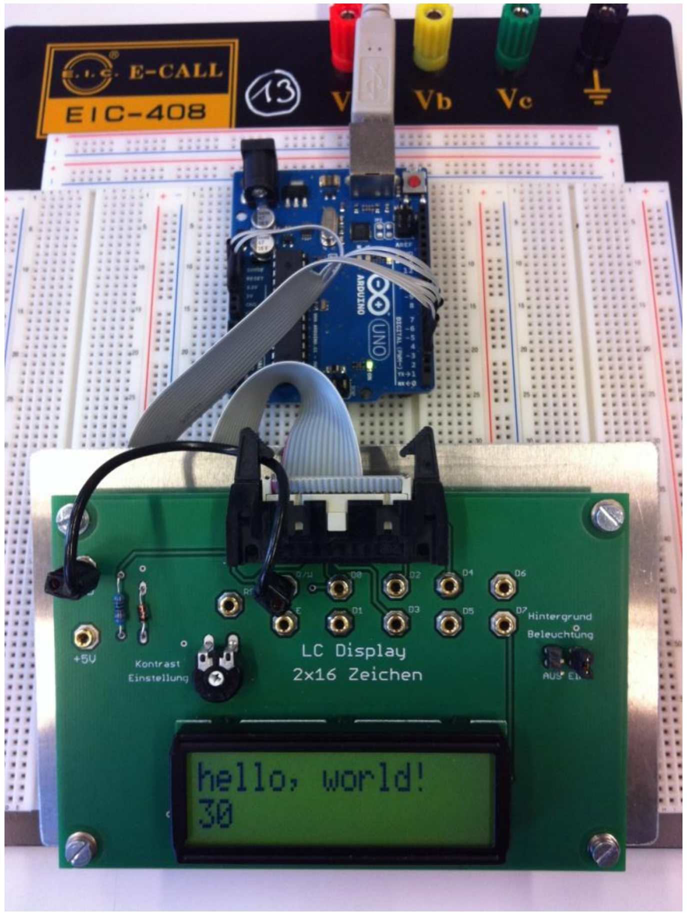
Arduino Uno

Protection Circuit

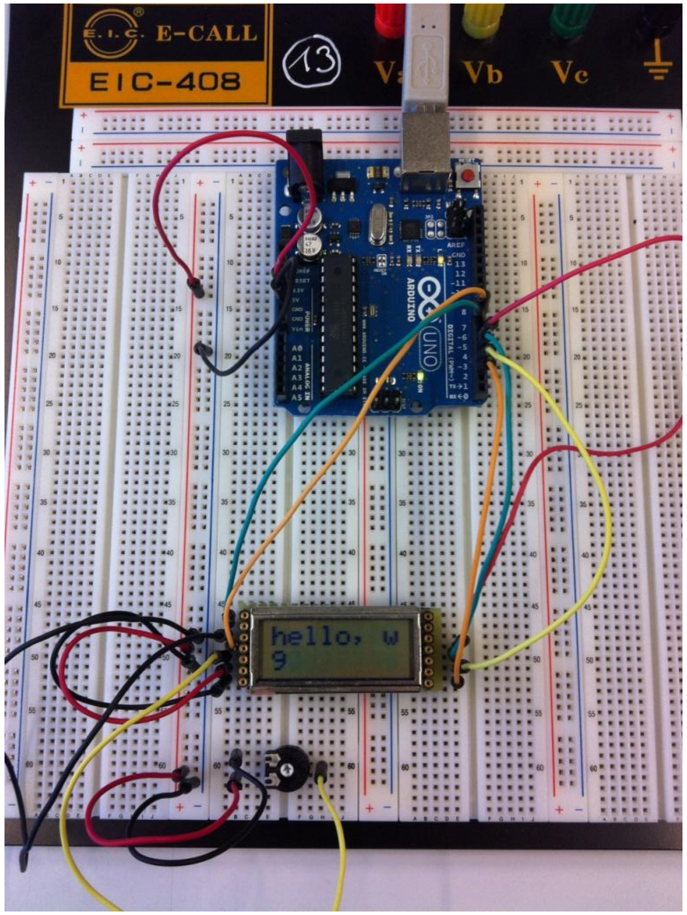


Connection of the LCD-Modules

16 x 2



8 x 2

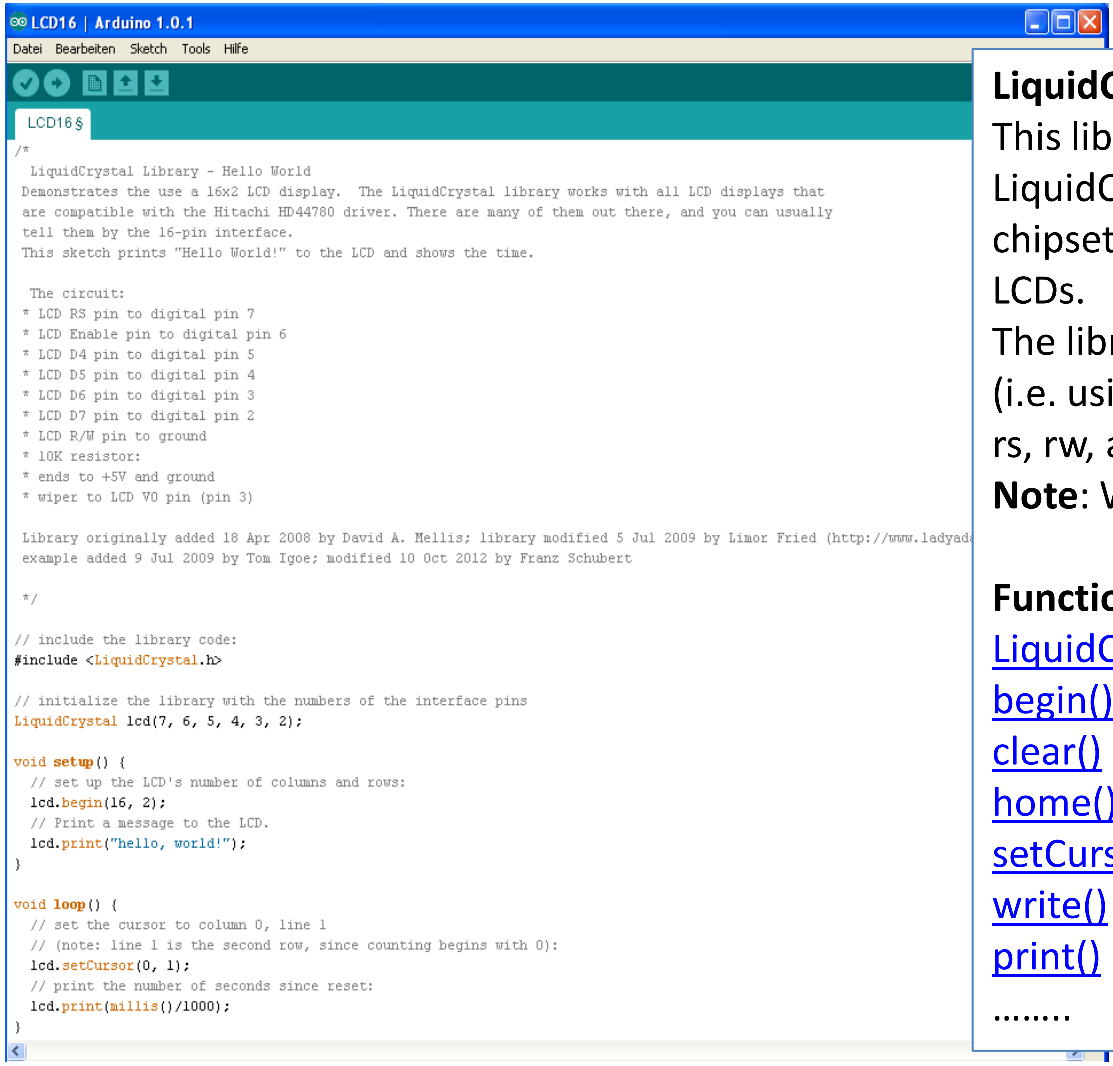


Arduino Uno

Pins of the Adaptor

LCD-Module	Description	Symbol	20-Pin-Adapter	Arduino Uno Digital pin
1	GND	VSS	2	GND
2	+ 5 V	VDD	1	5 V
3	Contrast 0,3 … 1,2 V	VEE	N.C.	N.C.
4	H = Data / L = Command	RS	3	7
5	H = Read / L = Write	R/W	N.C.	N.C.
6	Enable	E	5	6
7	LSB (8 Bit)	D0	N.C.	N.C.
8		D1	N.C.	N.C.
9		D2	N.C.	N.C.
10		D3	N.C.	N.C.
11	LSB (4 Bit)	D4(D0)	12	5
12		D5(D1)	11	4
13		D6(D2)	14	3
14	MSB	D7(D3)	13	2

Program Example for the LCD-Module



```
/*
  LiquidCrystal Library - Hello World
 Demonstrates the use a 16x2 LCD display.  The LiquidCrystal library works with all LCD displays that
 are compatible with the Hitachi HD44780 driver. There are many of them out there, and you can usually
 tell them by the 16-pin interface.
 This sketch prints "Hello World!" to the LCD and shows the time.

  The circuit:
 * LCD RS pin to digital pin 7
 * LCD Enable pin to digital pin 6
 * LCD D4 pin to digital pin 5
 * LCD D5 pin to digital pin 4
 * LCD D6 pin to digital pin 3
 * LCD D7 pin to digital pin 2
 * LCD R/W pin to ground
 * 10K resistor:
 * ends to +5V and ground
 * wiper to LCD V0 pin (pin 3)

 Library originally added 18 Apr 2008 by David A. Mellis; library modified 5 Jul 2009 by Limor Fried (http://www.ladyada
 example added 9 Jul 2009 by Tom Igoe; modified 10 Oct 2012 by Franz Schubert

 */

// include the library code:
#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);

void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.print("hello, world!");
}

void loop() {
  // set the cursor to column 0, line 1
  // (note: line 1 is the second row, since counting begins with 0):
  lcd.setCursor(0, 1);
  // print the number of seconds since reset:
  lcd.print(millis()/1000);
}
```

LiquidCrystal Library
This library allows an Arduino board to control LiquidCrystal displays (LCDs) based on the XXX chipset, which is found on most text-based LCDs.
The library works with in either 4- or 8-bit mode (i.e. using 4 or 8 data lines in addition to the rs, rw, and enable control lines).
Note: We use 4-bit mode.

Function
LiquidCrystal()
begin()
clear()
home()
setCursor()
write()
print()
....

Characteristics of the Thermometer

- NTC:

Temperature / °C	Resistor / kΩ
0	27,25
50	4,162
100	0,949

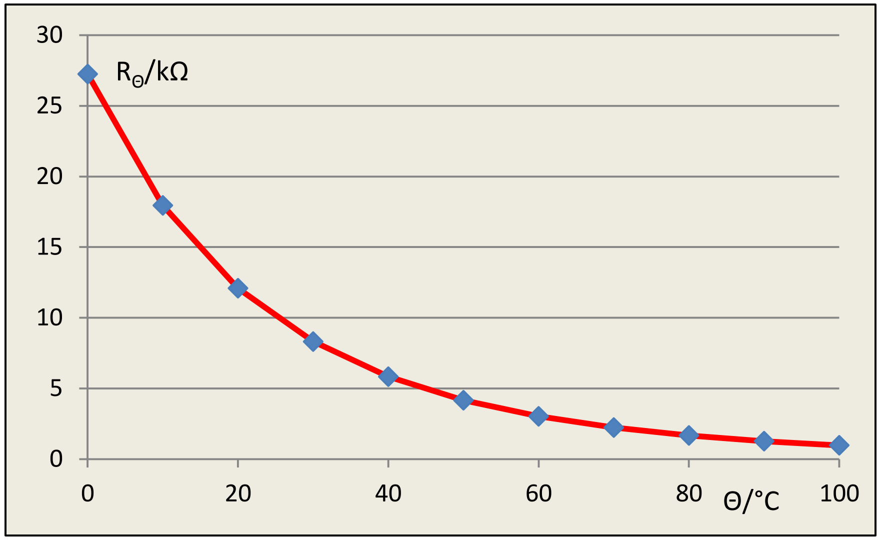
- Input-range: 0 °C to 100 °C
- Buzzer alarm, if temperature encreases 90 °C
- Output on LCD :

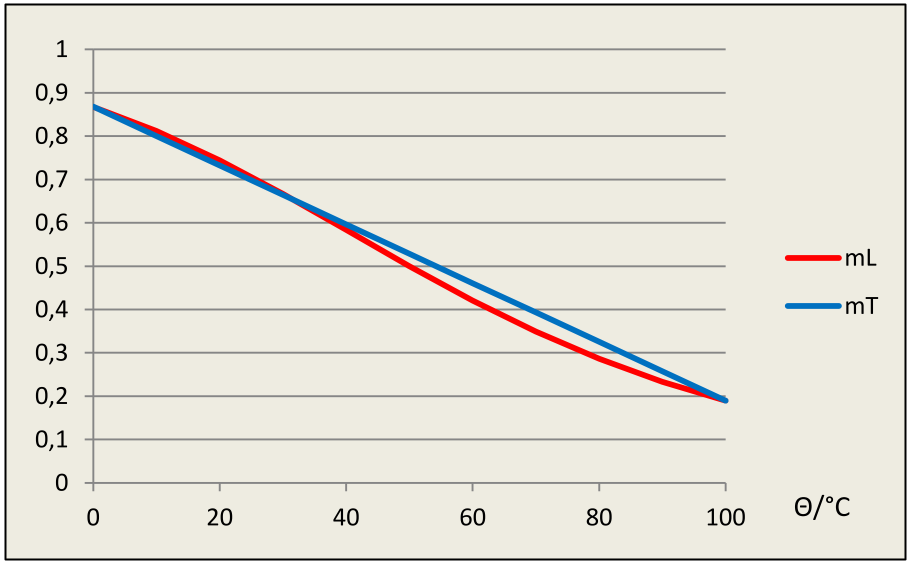
		4	2		°	C	
	1	0	8		°	F	

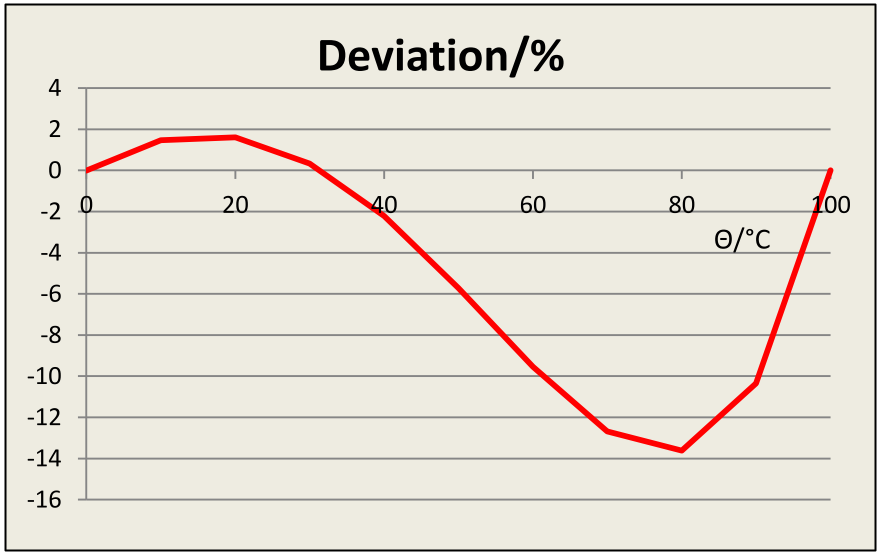
Tasks for the Thermometer

- Download the datasheet of the NTC-resistor
- Linearize the characteristic of the NTC in the range from 0 °C to 100 °C by connecting a serial resistor $R_L = R_{50}$ of the NTC.
- Develop the resulting characteristic $m_L = \frac{R_\Theta}{R_\Theta + R_L}$
- Substitute the resulting characteristic by a straight line $m_T = f(\Theta)$

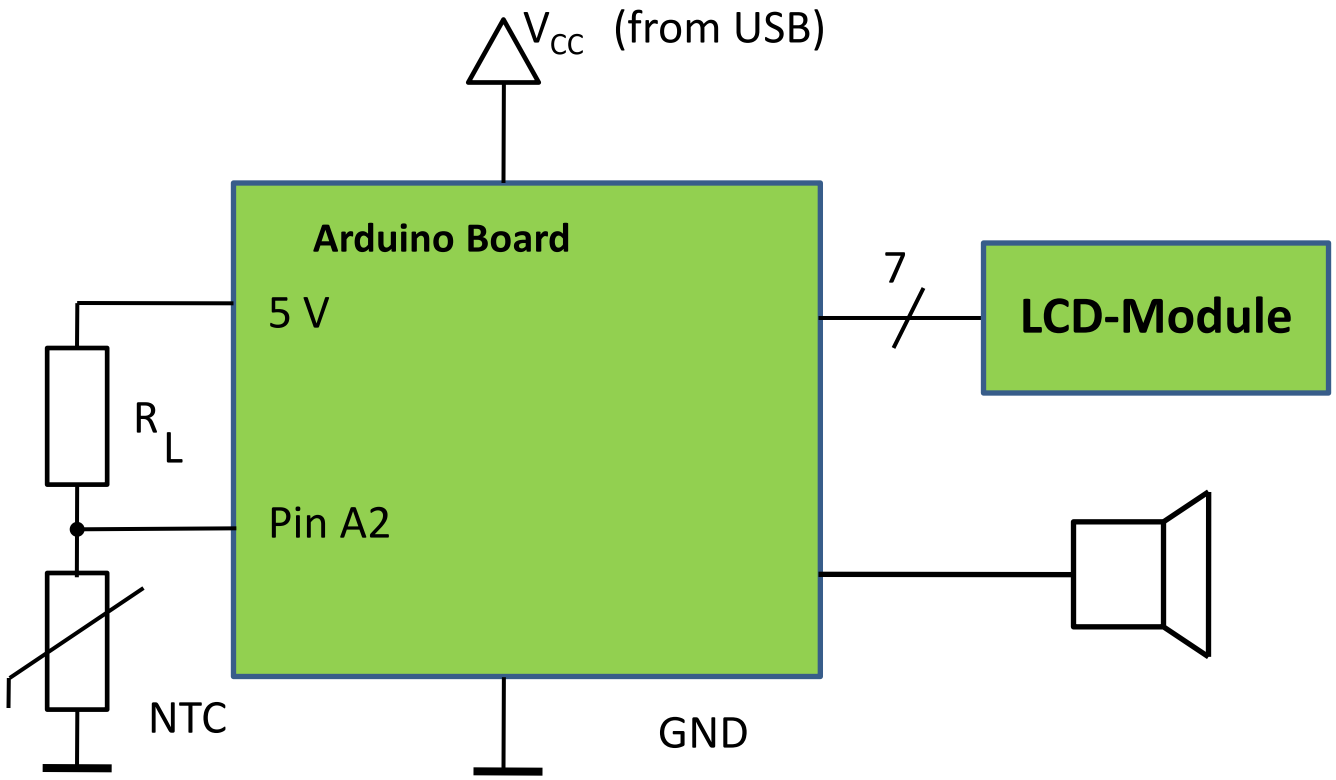
Θ/°C	R_Θ/kΩ	m_L	m_T	Deviation/%
0	27,25	0,86750287	0,86761173	-0,01254866
10	17,95	0,81177641	0,79980778	1,47437473
20	12,09	0,74390844	0,73200384	1,60027727
30	8,314	0,66639949	0,6641999	0,33006975
40	5,829	0,58342508	0,59639596	-2,2232299
50	4,162	0,5	0,52859202	-5,71840461
60	3,022	0,42065702	0,46078808	-9,54009218
70	2,229	0,34877171	0,39298414	-12,6766107
80	1,669	0,28622878	0,3251802	-13,6084935
90	1,266	0,23323508	0,25737626	-10,3505803
100	0,9737	0,18959441	0,18957232	0,01164949







Arduino Uno



Arduino Uno

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 10, 5, 4, 3, 2);
int inputPin = 2;
int readVoltage;
float outputValue;
float m;

void setup() {
  readVoltage = 0;
  outputValue = 0;
  m = 0.143f; //slope of our approximated line function

  lcd.begin(8, 2);
  Serial.begin(9600);
}
void loop() {
  readVoltage = 1023 - analogRead(inputPin);
  //the input value is inverted so that it increases with temperature
  outputValue = m*( (float)readVoltage - 223.0f ) + 11.1f;
  // the line function maps the correct temperatures to our input
  lcd.setCursor(0,0);
  delay(100);
  lcd.print(outputValue,DEC);
  lcd.print(".");
  lcd.print((int)(outputValue*100)%100,DEC);
  lcd.print(" C       ");
  delay(100);
  Serial.println(outputValue);
}
```

Arduino Uno

Tasks for Analog Input and PWM Output

- Dimm an LED with a potentiometer
- Check the function of the multicolour LED
- Write a program for controlling the colour of the multicolour LED with a potentiometer
- Control the rpm of the DC-motor with a potentiometer
- Sense the dark with the photoresistor
- Write a program for the piezo buzzer to play a melody

```
int potiPin = 2;
int bluePin1 = 6;
int bluePin2 = 3;
int greenPin = 9;
int redPin = 5;
int readVoltage;
void setup() {
  readVoltage = 0;
  pinMode(bluePin1,OUTPUT);
  pinMode(bluePin2,OUTPUT);
  pinMode(greenPin,OUTPUT);
  pinMode(redPin,OUTPUT);
  Serial.begin(9600);
}
void loop() {
  int i=0;
  readVoltage = analogRead(potiPin);

  if(readVoltage/128 >= 6 ){
    analogWrite(redPin, 255);
    analogWrite(greenPin, (128*7 - readVoltage)*2 );
    analogWrite(bluePin1, 0);
    analogWrite(bluePin2, 0);
  }
  else{
    if(readVoltage/128 >= 5 && readVoltage/128 < 6){
      analogWrite(redPin, (readVoltage - 128*5)*2 );
      analogWrite(greenPin, 255);
      analogWrite(bluePin1, 0);
      analogWrite(bluePin2, 0);
    }
    else{
      if(readVoltage/128 >= 4 && readVoltage/128 < 5){
        analogWrite(redPin, 0 );
        analogWrite(greenPin, 255);
        analogWrite(bluePin1, (128*5 - readVoltage)*2 );
        analogWrite(bluePin2, 0 );
      }
```

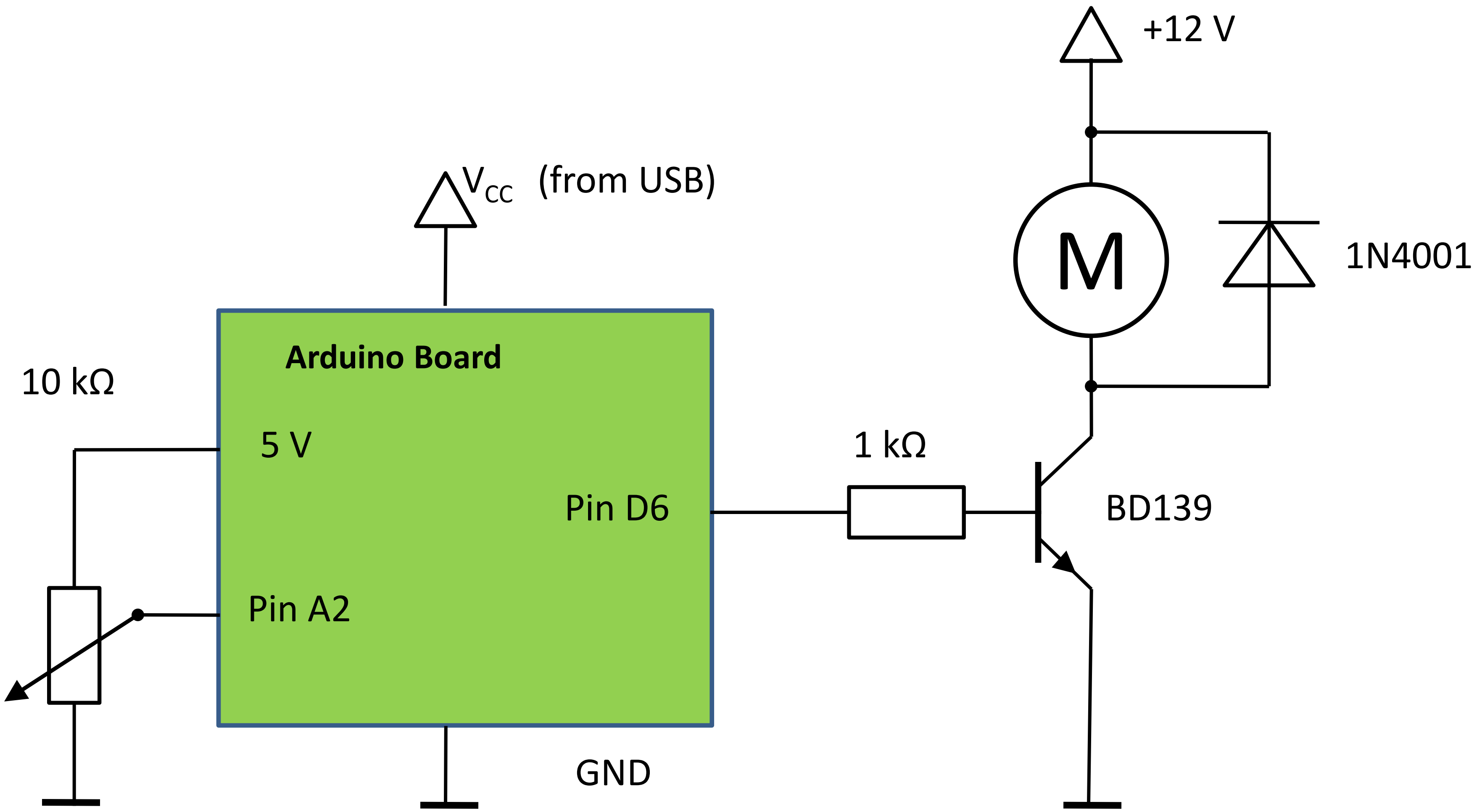
```
else{
        if(readVoltage/128 >= 4 && readVoltage/128 < 5){
          analogWrite(redPin, 0 );
          analogWrite(greenPin, (128*5 - readVoltage)*2);
          analogWrite(bluePin1, 255);
          analogWrite(bluePin2, 0 );
        }
        else{
          if(readVoltage/128 >= 3 && readVoltage/128 < 4){
            analogWrite(redPin, (128*4 - readVoltage)*2 );
            analogWrite(greenPin, 0);
            analogWrite(bluePin1, 255);
            analogWrite(bluePin2, 0 );
          }
          else{
            if(readVoltage/128 >= 2 && readVoltage/128 < 3){
              analogWrite(redPin, 255 );
              analogWrite(greenPin, (128*3 - readVoltage)*2);
              analogWrite(bluePin1, 255);
              analogWrite(bluePin2, 0 );
            }
            else{
              analogWrite(redPin, 255);
              analogWrite(greenPin, 255);
              analogWrite(bluePin1, 255);
              analogWrite(bluePin2, 0);
            }
          }
        }
      }
    }
  }
    if(readVoltage/128 >= 4 && readVoltage/128 < 7){
    analogWrite(greenPin, (7*128 - readVoltage)/4 );
  }
  else{
    analogWrite(greenPin, 0);
  }
  Serial.println(readVoltage/128);
}
```

Arduino Uno

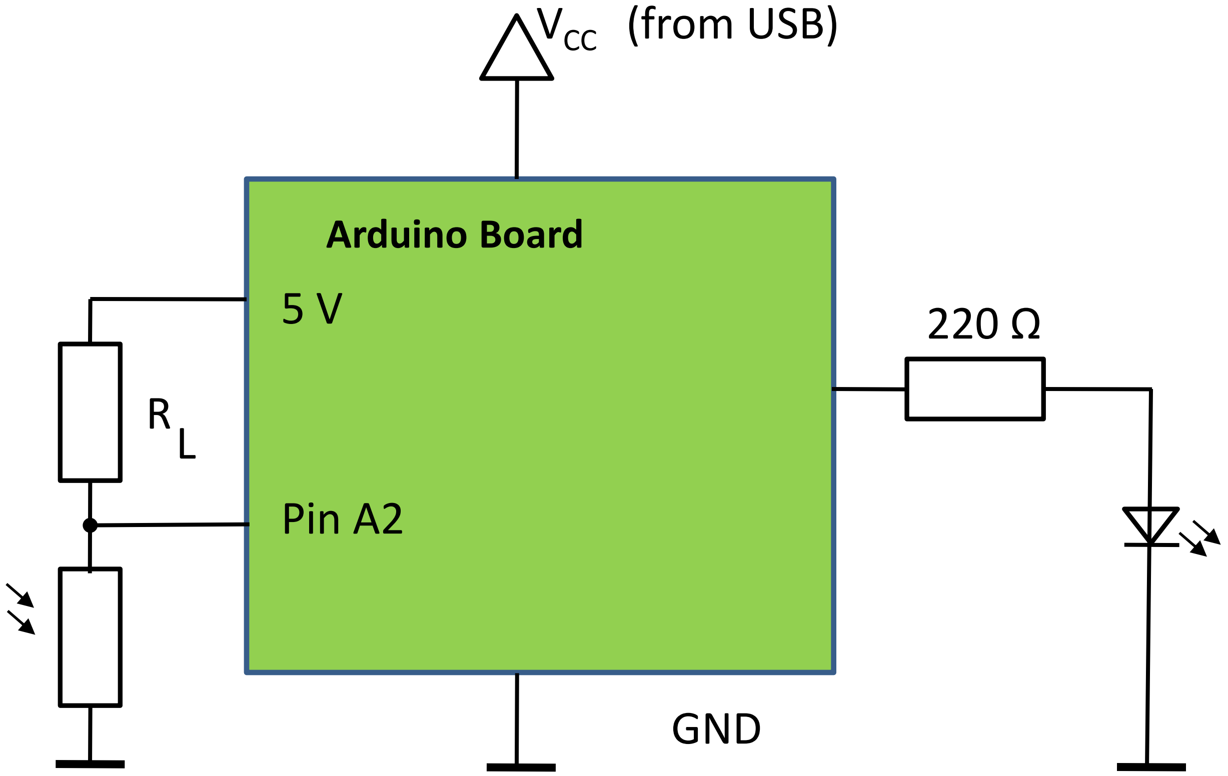
```
int photoDiode = 2;
int buzzerPin = 3;
int readVoltage;
void setup() {
  readVoltage = 0;
  pinMode(buzzerPin,OUTPUT);
  Serial.begin(9600);
}
void loop() {
  readVoltage = analogRead(photoDiode);
  if(readVoltage > 60) {
    analogWrite(buzzerPin,40);
    delay(1000);
    analogWrite(buzzerPin,80);
    delay(200);
    analogWrite(buzzerPin,120);
    delay(500);
    analogWrite(buzzerPin,200);
    delay(100);
  }
  Serial.println(readVoltage);
}
```

Notes

note	frequency/Hz	period/µs
c	261	3830
d	294	3400
e	329	3938
f	349	2864
g	392	2550
a	440	2272
b	493	2038
C	523	1912


+12 V
V$_{CC}$ (from USB)
M
1N4001
Arduino Board
10 kΩ
5 V
Pin D6
1 kΩ
BD139
Pin A2
GND

Arduino Uno



Arduino Uno

Piezo Buzzer as Sensor

Introduction

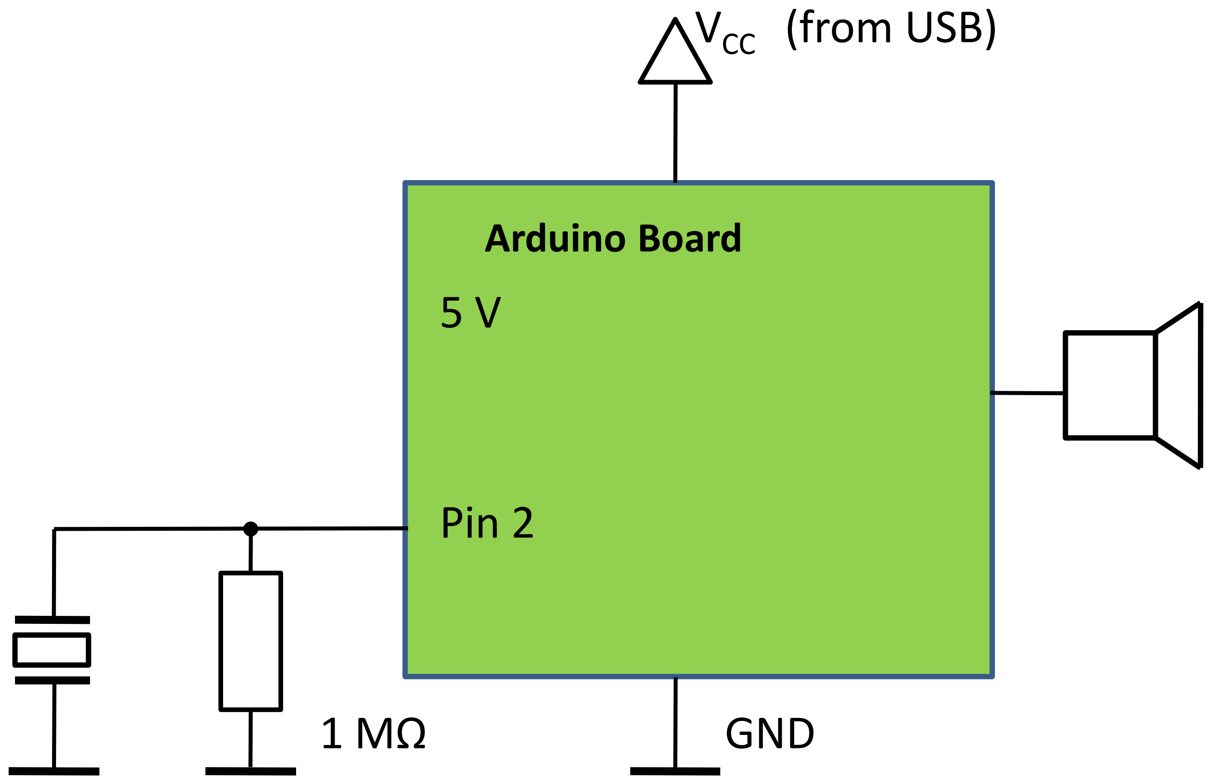
- Piezo buzzers exhibit the *reverse piezoelectric effect*.
- The normal piezoelectric effect is generating electricity from squeezing a crystal.
- Can get several thousand volts, makes a spark

Piezo Knock Sensor

- To read a piezo you can connect it to an analog input, but: - You need to drain off any voltage with a resistor
- The protection diodes inside the AVR chip protect against the high voltage

Tasks

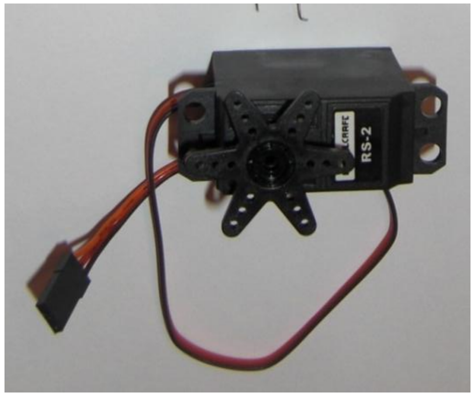
- Piezo-sensor: input value -> serial out
- Piezo-sensor: input value -> buzzer frequency



Arduino Uno

Servo Motor

Servos are DC motors with built in **gearing** and **feedback control loop circuitry**.



Servo Wiring

All servos have three wires:

Black or **Brown** is for ground.

Red is for power (~4.8-6V).

Yellow, **Orange**, or **White** is the **signal** wire (3-5V).

Tasks

- Pot position 0…180° to servo position and LCD
- Railroad crossing barrier
- Railroad crossing sign (blinking, beep)

Library for the Servo Motor 1

Servo library
This library allows an Arduino board to control RC servo motors. Servos have integrated gears and a shaft that can precisely controlled. Standard servos allow the shaft to be positioned at various angles, usually between 0 and 180 degrees. Continuous rotation servos allow the rotation of the shaft to be set to various speeds.

As of Arduino 0017, the Servo library supports up to 12 motors on most Arduino boards and 48 on the Arduino Mega. On boards other than the Mega, use of the library disables analogWrite() (PWM) functionality on pins 9 and 10, whether or not there is a Servo on those pins. On the Mega, up to 12 servos can be used without interfering with PWM functionality; use of 12 to 23 motors will disable PWM on pins 11 and 12.

In Arduino 0016 and earlier, the Servo library uses functionality built in to the hardware, and works only on pins 9 and 10 (and does not work on the Arduino Mega). In this case, if only one servo is used, the other pin cannot be used for normal PWM output with analogWrite(). For example, in Arduino 0016 and earlier, you can't have a servo on pin 9 and PWM output on pin 10.

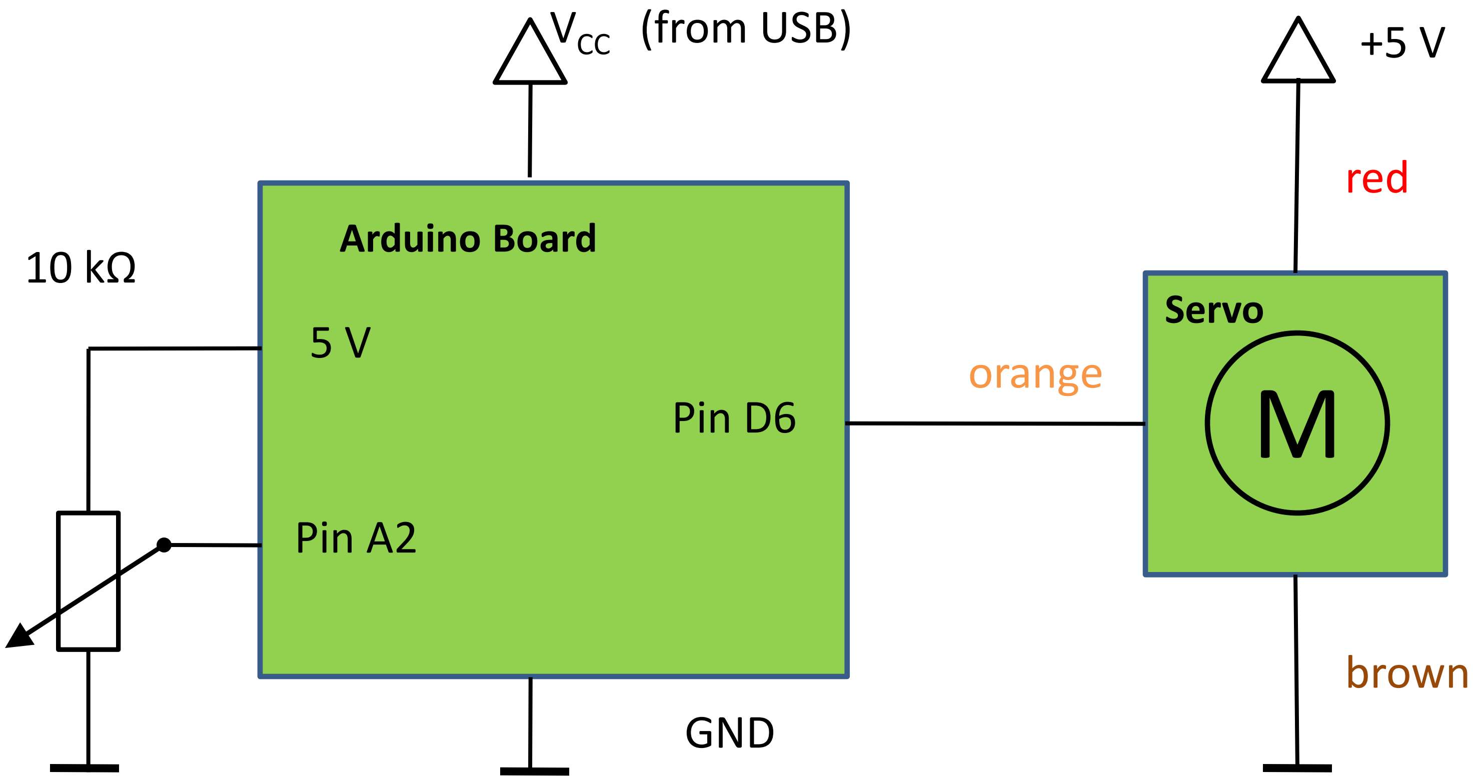
Library for the Servo Motor 2

Circuit
Servo motors have three wires: power, ground, and signal. The power wire is typically red, and should be connected to 5V power supply. The ground wire is typically black or brown and should be connected to a ground pin. The signal pin is typically yellow, orange or white and should be connected to a digital pin on the Arduino board. Note servos draw considerable power, so if you need to drive more than one or two, you need a separate power supply (not the +5V pin on your Arduino!).

Functions
attach()
write()
read()
attached()
detach()

Program Example for the Servo Motor

```
// Sweep
// by BARRAGAN <http:                      //barraganstudio.com>

#include <Servo.h>
Servo myservo;                             // create servo object to control a servo
int pos = 0;                               // variable to store the servo position

void setup()
{
  myservo.attach(9);                       // attaches the servo on pin 9 to the servo object
}

void loop()
{
  for(pos = 0; pos < 180; pos += 1)        // goes from 0 degrees to 180 degrees
                 {                         // in steps of 1 degree
                 myservo.write(pos);       // tell servo to go to position in variable 'pos'
                  delay(15);               // waits 15ms for the servo to reach the position
                 }
  for(pos = 180; pos>=1; pos-=1)           // goes from 180 degrees to 0 degrees
                  {
                 myservo.write(pos);       // tell servo to go to position in variable 'pos'
                 delay(15);                // waits 15ms for the servo to reach the position
                 }
}
```



V_{CC} (from USB)
+5 V
red
Arduino Board
10 kΩ
5 V
Servo
orange
Pin D6
M
Pin A2
brown
GND

Arduino Uno

Communication

- The I2C Bus
- Serial I/O

The I2C Bus

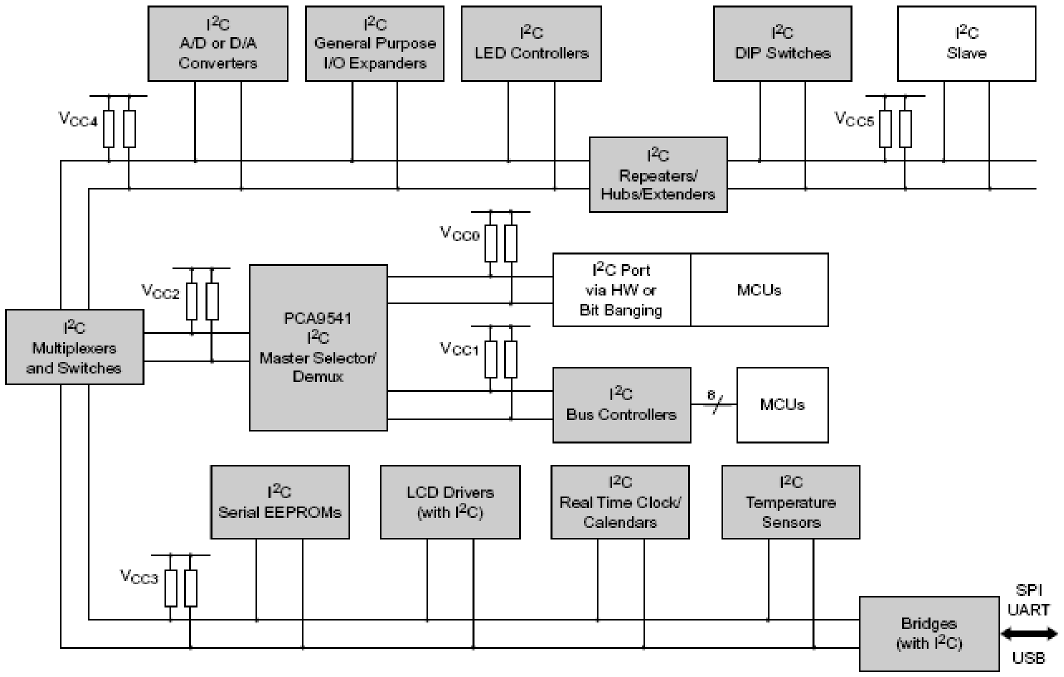
Introduction
The I2C-bus is a de facto world standard that is now implemented in over 1000 different ICs manufactured by more than 50 companies. Additionally, the versatile I2C-bus is used in a variety of control architectures such as System Management Bus (SMBus), Power Management Bus (PMBus), Intelligent Platform Management Interface (IPMI), and Advanced Telecom Computing Architecture (ATCA).

I2C-bus features
In consumer electronics, telecommunications and industrial electronics, there are often many similarities between seemingly unrelated designs. For example, nearly every system includes:

- **Some intelligent control, usually a single-chip microcontroller**
- **General-purpose circuits like LCD and LED drivers, remote I/O ports, RAM, EEPROM, real-time clocks or A/D and D/A converters**
- **Application-oriented circuits such as digital tuning and signal processing circuits for radio and video systems, temperature sensors, and smart cards**

NXP Semiconductors: UM10204 I2C-bus specification and user manual

To exploit these similarities to the benefit of both systems designers and equipment manufacturers, as well as to maximize hardware efficiency and circuit simplicity, Philips Semiconductors (now NXP Semiconductors) developed a simple bidirectional 2-wire bus for efficient inter-IC control. This bus is called the Inter IC or I2C-bus. All I2C-bus compatible devices incorporate an on-chip interface which allows them to communicate directly with each other via the I2C-bus. This design concept solves the many interfacing problems encountered when designing digital control circuits.



I²C A/D or D/A Converters
I²C General Purpose I/O Expanders
I²C LED Controllers
I²C DIP Switches
I²C Slave
VCC4
VCC5
I²C Repeaters/ Hubs/Extenders
VCC0
VCC2
VCC1
I²C Multiplexers and Switches
PCA9541 I²C Master Selector/ Demux
I²C Port via HW or Bit Banging
MCUs
I²C Bus Controllers
8
MCUs
I²C Serial EEPROMs
LCD Drivers (with I²C)
I²C Real Time Clock/ Calendars
I²C Temperature Sensors
VCC3
Bridges (with I²C)
SPI
UART
USB

NXP Semiconductors: UM10204 I2C-bus specification and user manual

Here are some of the features of the I2C-bus:

- Only two bus lines are required; a serial data line (SDA) and a serial clock line (SCL).
- Each device connected to the bus is software addressable by a unique address and simple master/slave relationships exist at all times; masters can operate as master-transmitters or as master-receivers.
- It is a true multi-master bus including collision detection and arbitration to prevent data corruption if two or more masters simultaneously initiate data transfer.
- Serial, 8-bit oriented, bidirectional data transfers can be made at up to 100 kbit/s in the Standard-mode, up to 400 kbit/s in the Fast-mode, up to 1 Mbit/s in Fast-mode Plus, or up to 3.4 Mbit/s in the High-speed mode.
- On-chip filtering rejects spikes on the bus data line to preserve data integrity.
- The number of ICs that can be connected to the same bus is limited only by a maximum bus capacitance. More capacitance may be allowed under some conditions.

NXP Semiconductors: UM10204 I2C-bus specification and user manual

Definition of I2C-bus terminology

Transmitter the device which sends data to the bus

Receiver the device which receives data from the bus

Master the device which initiates a transfer, generates clock signals and terminates a transfer

Slave the device addressed by a master

Multi-master more than one master can attempt to control the bus at the same time without corrupting the message

Arbitration procedure to ensure that, if more than one master simultaneously tries to control the bus, only one is allowed to do so and the winning message is y device addressed is considered a slave.

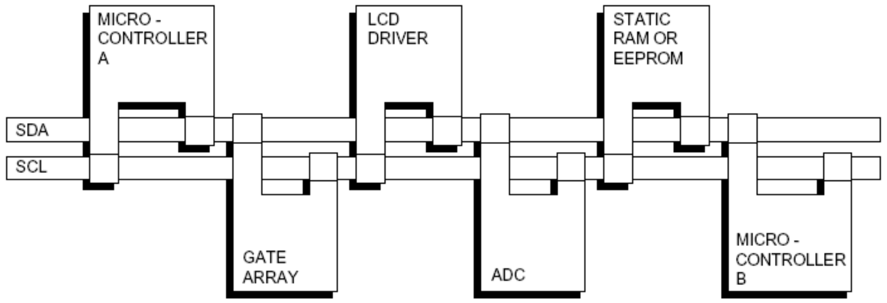
Synchronization procedure to synchronize the clock signals of two or more devices

NXP Semiconductors: UM10204 I2C-bus specification and user manual

The I2C-bus protocol

Two wires, serial data (SDA) and serial clock (SCL), carry information between the devices connected to the bus. Each device is recognized by a unique address (whether it is a microcontroller, LCD driver, memory or keyboard interface) and can operate as either a transmitter or receiver, depending on the function of the device. An LCD driver may be only a receiver, whereas a memory can both receive and transmit data. In addition to transmitters and receivers, devices can also be considered as masters or slaves when performing data transfers. A master is the device which initiates a data transfer on the bus and generates the clock signals to permit that transfer. At that time, any device addressed Is considered a slave.

The I2C-bus is a multi-master bus. This means that more than one device capable of controlling the bus can be connected to it. As masters are usually microcontrollers, let's consider the case of a data transfer between two microcontrollers connected to the I2C-bus.



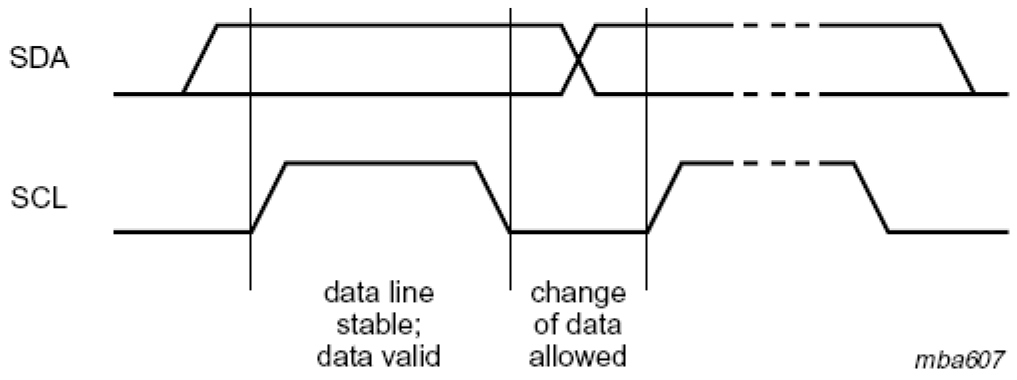
NXP Semiconductors: UM10204 I2C-bus specification and user manual

SDA and SCL logic levels

Due to the variety of different technology devices (CMOS, NMOS, bipolar) that can be connected to the I2C-bus, the levels of the logical ‘0’ (LOW) and ‘1’ (HIGH) are not fixed and depend on the associated level of V_{DD}. Input reference levels are set as 30 % and 70 % of V_{DD}; V_{IL} is $0.3V_{DD}$ and V_{IH} is $0.7V_{DD}$.
Some legacy device input levels were fixed at V_{IL} = 1.5 V and V_{IH} = 3.0 V, but all new devices require this 30 %/70 % specification.

Data validity

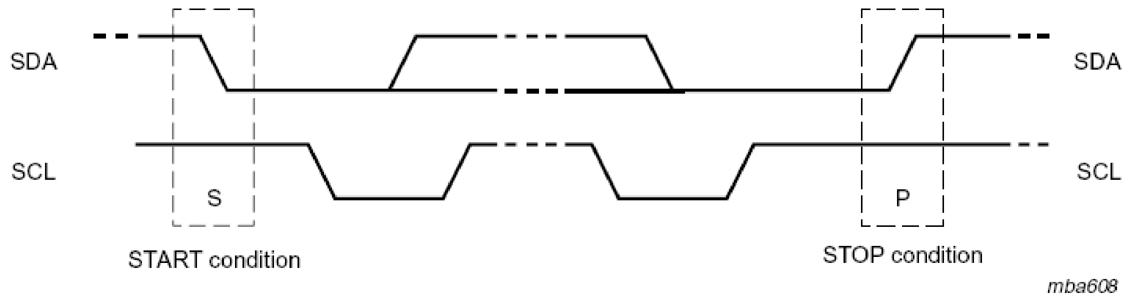
The data on the SDA line must be stable during the HIGH period of the clock. The HIGH or LOW state of the data line can only change when the clock signal on the SCL line is LOW. One clock pulse is generated for each data bit transferred.



NXP Semiconductors: UM10204 I2C-bus specification and user manual

START and STOP conditions

All transactions begin with a START (S) and can be terminated by a STOP (P). A HIGH to LOW transition on the SDA line while SCL is HIGH defines a START condition. A LOW to HIGH transition on the SDA line while SCL is HIGH defines a STOP condition.
START and STOP conditions are always generated by the master. The bus is considered to be busy after the START condition. The bus is considered to be free again a certain time after the STOP condition. The bus stays busy if a repeated START (Sr) is generated instead of a STOP condition. In this respect, the START (S) and repeated START (Sr) conditions are functionally identical.
For the remainder of this document, therefore, the S symbol will be used as a generic term to represent both the START and repeated START conditions, unless Sr is articularly relevant.

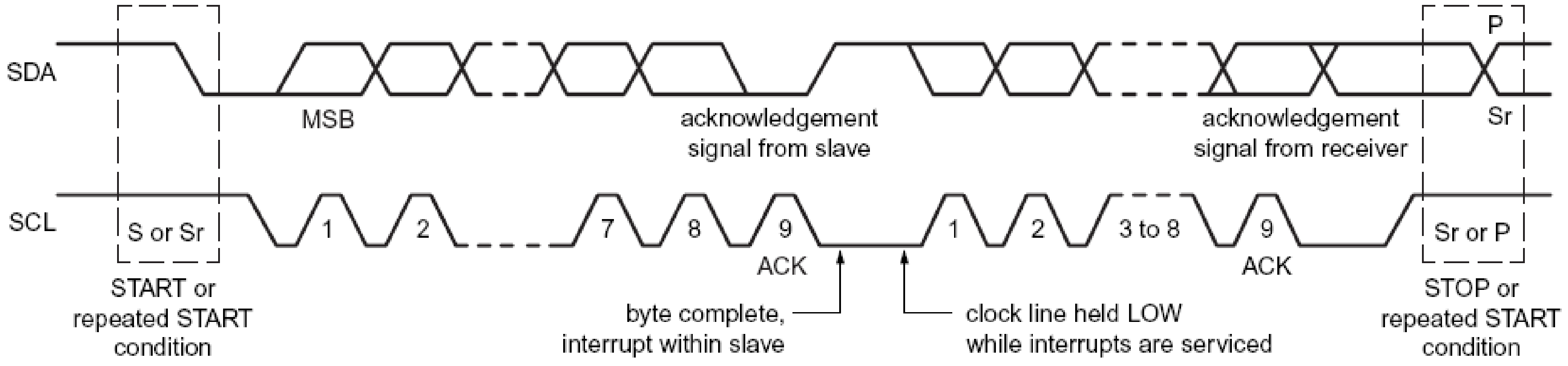


Detection of START and STOP conditions by devices connected to the bus is easy if they incorporate the necessary interfacing hardware. However, microcontrollers with no such interface have to sample the SDA line at least twice per clock period to sense the transition.

NXP Semiconductors: UM10204 I2C-bus specification and user manual

Byte format

Every byte put on the SDA line must be 8 bits long. The number of bytes that can be transmitted per transfer is unrestricted. Each byte has to be followed by an Acknowledge bit. Data is transferred with the Most Significant Bit (MSB) first. If a slave cannot receive or transmit another complete byte of data until it has performed some other function, for example servicing an internal interrupt, it can hold the clock line SCL LOW to force the master into a wait state. Data transfer then continues when the slave is ready for another byte of data and releases clock line SCL.


SDA
SCL
MSB
acknowledgement signal from slave
acknowledgement signal from receiver
P
Sr
S or Sr
1
2
7
8
9
ACK
1
2
3 to 8
9
ACK
Sr or P
START or repeated START condition
byte complete, interrupt within slave
clock line held LOW while interrupts are serviced
STOP or repeated START condition

NXP Semiconductors: UM10204 I2C-bus specification and user manual

Acknowledge (ACK) and Not Acknowledge (NACK)

The acknowledge takes place after every byte. The acknowledge bit allows the receiver to signal the transmitter that the byte was successfully received and another byte may be sent. All clock pulses including the acknowledge 9th clock pulse are generated by the master.

The Acknowledge signal is defined as follows: the transmitter releases the SDA line during the acknowledge clock pulse so the receiver can pull the SDA line LOW and it remains stable LOW during the HIGH period of this clock pulse (see Figure 4). Set-up and hold times (specified in Section 6) must also be taken into account.

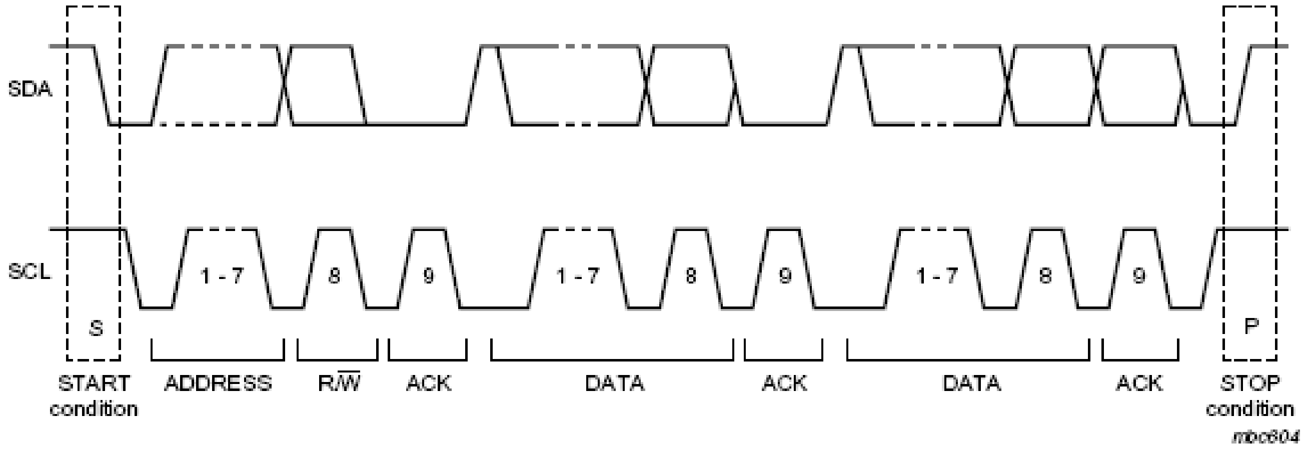
When SDA remains HIGH during this 9th clock pulse, this is defined as the Not Acknowledge signal. The master can then generate either a STOP condition to abort the transfer, or a repeated START condition to start a new transfer. There are five conditions that lead to the generation of a NACK:

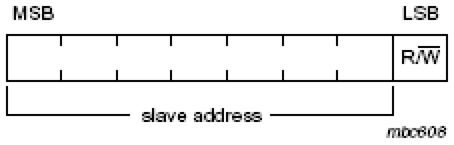
1. No receiver is present on the bus with the transmitted address so there is no device to respond with an acknowledge.
2. The receiver is unable to receive or transmit because it's performing some real-time function and is not ready to start communication with the master.
3. During the transfer the receiver gets data or commands that it does not understand.
4. During the transfer, the receiver cannot receive any more data bytes.
5. A master-receiver needs to signal the end of the transfer to the slave transmitter.

NXP Semiconductors: UM10204 I2C-bus specification and user manual

The slave address and R/W bit

Data transfers follow the format shown in the left figure. After the START condition (S), a slave address is sent. This address is 7 bits long followed by an eighth bit which is a data direction bit (R/W)—a ‘zero’ indicates a transmission (WRITE), a ‘one’ indicates a request for data (READ) (refer to the right figure). A data transfer is always terminated by a STOP condition (P) generated by the master. However, if a master still wishes to communicate on the bus, it can generate a repeated START condition (Sr) and address another slave without first generating a STOP condition. Various combinations of read/write formats are then possible within such a transfer.



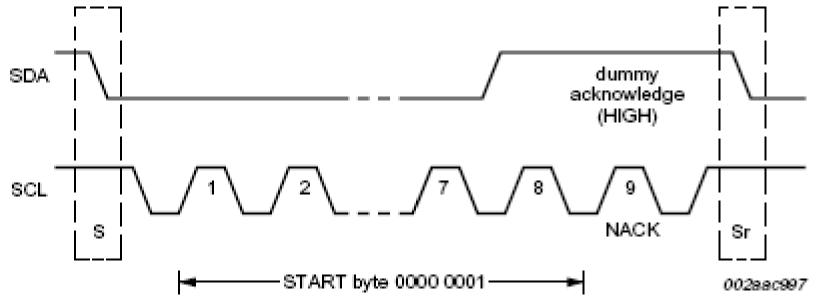


NXP Semiconductors: UM10204 I2C-bus specification and user manual

START byte

Microcontrollers can be connected to the I2C-bus in two ways. A microcontroller with an on-chip hardware I2C-bus interface can be programmed to be only interrupted by requests from the bus. When the device does not have such an interface, it must constantly monitor the bus via software. Obviously, the more times the microcontroller monitors, or polls the bus, the less time it can spend carrying out its intended function.
There is therefore a speed difference between fast hardware devices and a relatively slow microcontroller which relies on software polling.
In this case, data transfer can be preceded by a start procedure which is much longer than normal. The start procedure consists of:

- **A START condition (S)**
- **A START byte (0000 0001)**
- **An acknowledge clock pulse (ACK)**
- **A repeated START condition (Sr).**



NXP Semiconductors: UM10204 I2C-bus specification and user manual

After the START condition S has been transmitted by a master which requires bus access, the START byte (0000 0001) is transmitted. Another microcontroller can therefore sample the SDA line at a low sampling rate until one of the seven zeros in the START byte is detected. After detection of this LOW level on the SDA line, the microcontroller can switch to a higher sampling rate to find the repeated START condition Sr which is then used for synchronization.
A hardware receiver will reset on receipt of the repeated START condition Sr and will therefore ignore the START byte.
An acknowledge-related clock pulse is generated after the START byte. This is present only to conform with the byte handling format used on the bus. No device is allowed to acknowledge the START byte.

Bus clear
In the unlikely event where the clock (SCL) is stuck LOW, the preferential procedure is to reset the bus using the HW reset signal if your I2C devices have HW reset inputs. If the I2C devices do not have HW reset inputs, cycle power to the devices to activate the mandatory internal Power-On Reset (POR) circuit.
If the data line (SDA) is stuck LOW, the master should send 9 clock pulses. The device that held the bus LOW should release it sometime within those 9 clocks. If not, then use the HW reset or cycle power to clear the bus.

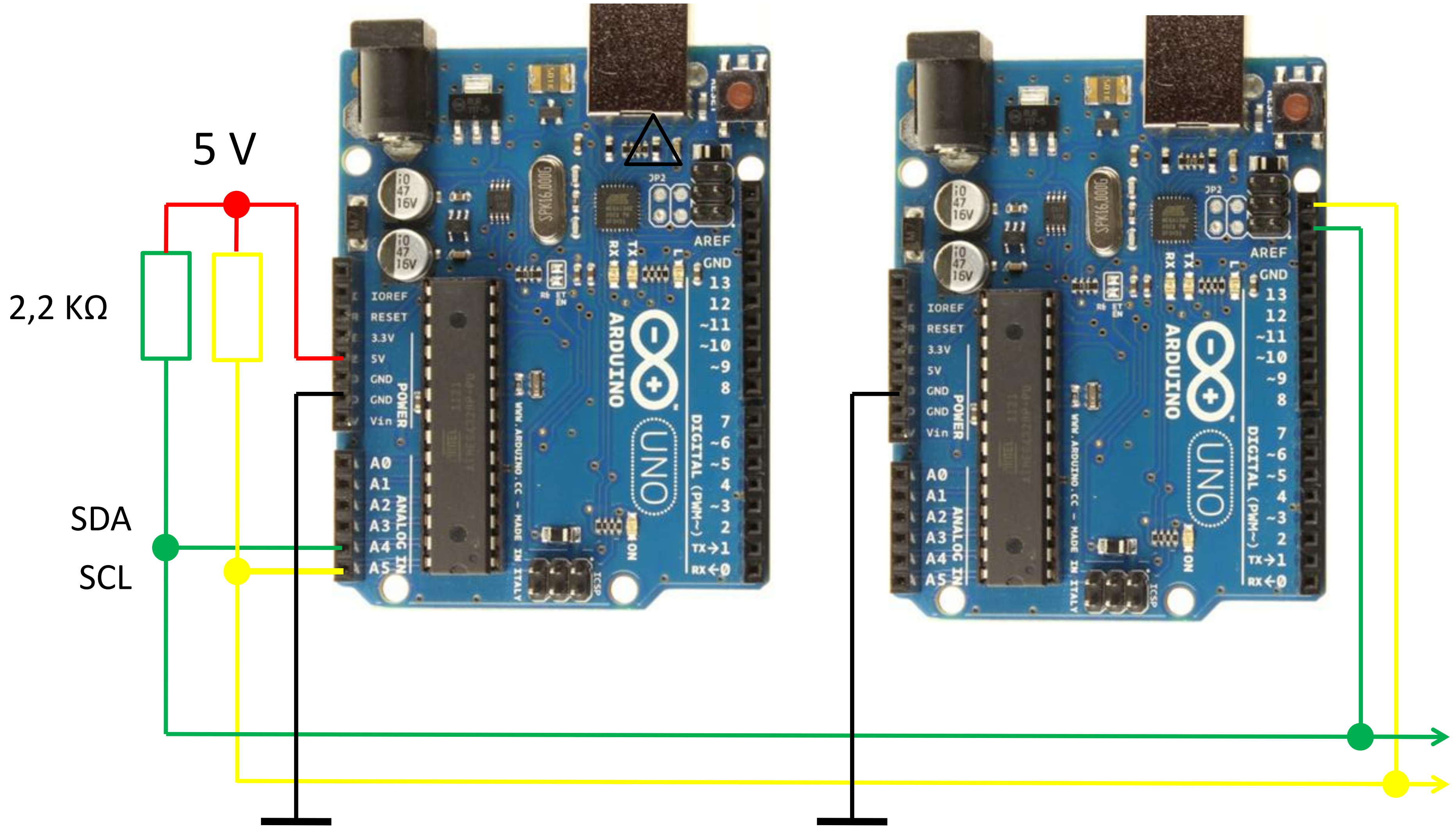
NXP Semiconductors: UM10204 I2C-bus specification and user manual

The I2C Bus Library

Wire Library
This library allows you to communicate with I2C / TWI devices. On the Arduino, SDA (data line) is on analog input pin 4, and SCL (clock line) is on analog input pin 5.

Functions
begin()
begin(address)
requestFrom(address, count)
beginTransmission(address)
endTransmission()
send()
byte available()
byte receive()
onReceive(handler)
onRequest(handler)

Connection of I2C Master and Slave



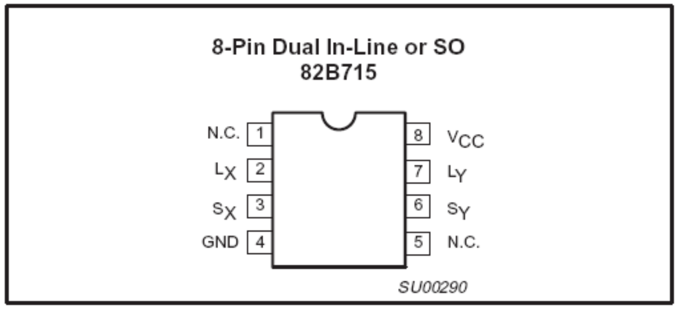
Arduino Uno

I2C Bus Extender 82B715

DESCRIPTION
The 82B715 is a bipolar integrated circuit intended for application in I2C bus systems. While retaining all the operating modes and features of the I2C system it permits extension of the practical separation distance between components on the I2C bus by buffering both the data (SDA) and the clock (SCL) lines.

The I2C bus capacitance limit of 400pF restricts practical communication distances to a few meters. Using one 82B715 at each end of longer cables reduces the cable loading capacitance on the I2C bus by a factor of 10 times and may allow the use of low cost general purpose wiring to extend bus lengths.

PIN CONFIGURATIONS

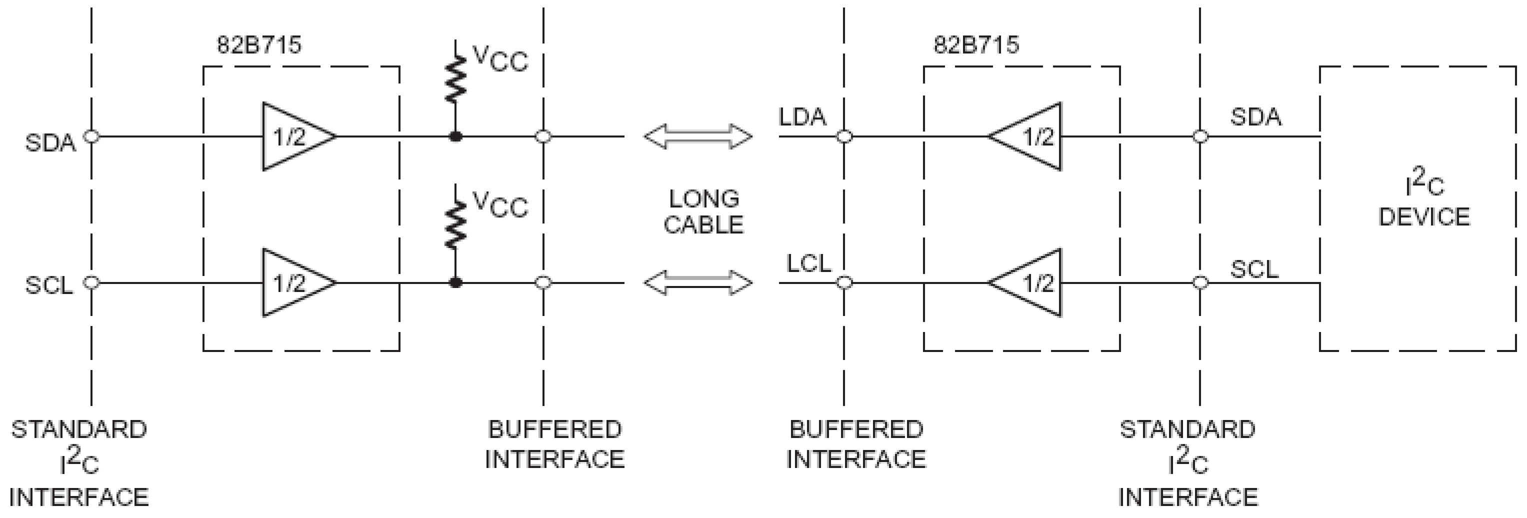


PINNING

PIN	SYMBOL	FUNCTION
1	N.C.	
2	L_X	Buffered Bus, LDA or LCL
3	S_X	I^2C Bus, SDA or SCL
4	GND	Negative Supply
5	N.C.	
6	S_Y	I^2C Bus, SCL or SDA
7	L_Y	Buffered Bus, LCL or LDA
8	V_{CC}	Positive Supply

PHILIPS Data Sheet

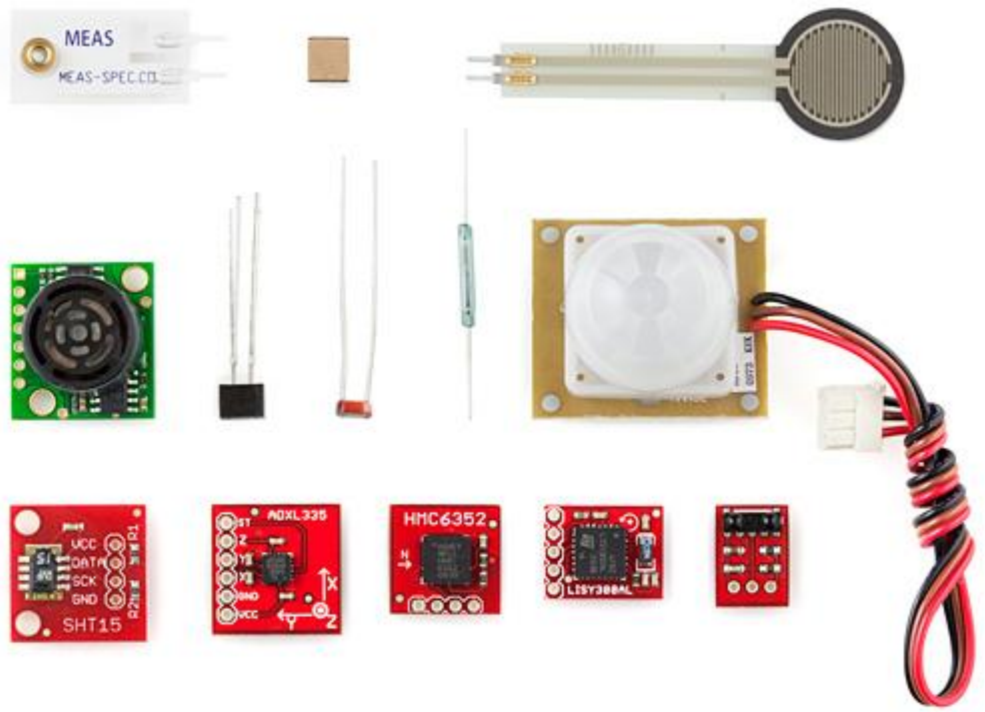
Minimum Sub-System with 82B715



PHILIPS Data Sheet

Tasks for I2C

- The master sends a sign to the slave
- The slave answers with the next ASCII sign
- Input of the sign via keyboard of the PC
- Output to LCD (first row: sign of the master; second row: sign from the slave)
- Watch the signals on SDA and SCL on a scope (sending a repeating a signal continuously)


MEAS
MEAS-SPEC.COM
VCC
DATA
SCK
GND
SHT15
ADXL335
ST
Z
Y
X
GND
VCC
HMC6352
N
LISY300AL

Arduino Uno