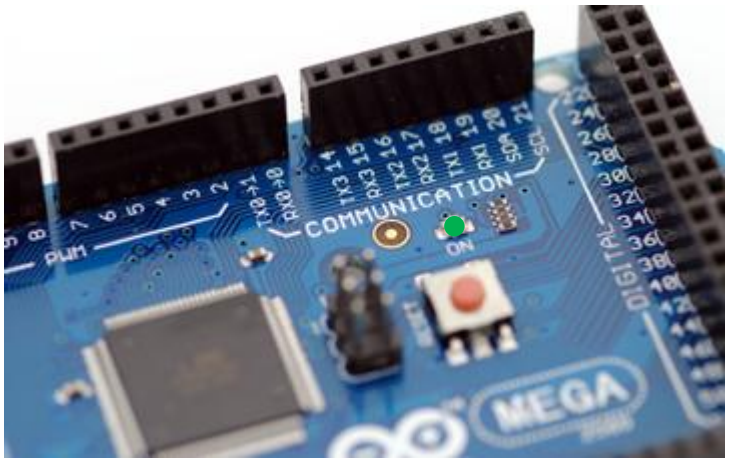
ARDUINO

Basics, Teil 2



Serielle Kommunikation und analoge Schnittstellen

Markus Ulsaß, attraktor Hamburg, 3.6.2013

Attraktor

Attraktor

- **der Makerspace in Hamburg**
- seit 2010 gemeinnütziger Verein
- auf 300 Quadratmetern
- ca. 70 Mitglieder
- Veranstaltungszentrum und offene Werkstatt
- für Technikinteressierte und Kreative
- Geräte wie CNC-Fräsen, 3D-Drucker und Laser-Cutter
- Regelmäßige Events: Basteldonnerstag, Back to Hack, Elektronik Stammtisch
- Hamburger Maker-Treffen, FBCamp u.v.m
- Finanzierung über Mitgliederbeiträge, Events und Spenden
- Führung im Anschluss

you're always welcome

Veranstaltungstermine im Attraktor Blog

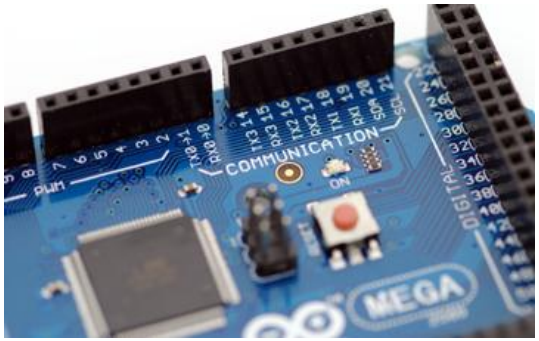
http://blog.attraktor.org/

Agenda

- Erwartungen/ Erfahrungen (C/ C++, Register)?
- Was ist Arduino (kurze Einführung/ Wiederholung)
- Der „Arduino MEGA 2560“
- Grundlagen serielle Kommunikation, RS-232/ U(S)ART
- Arduino & serielle Schnittstelle, Programmierung
- USART – Die Register am Beispiel des ATMEL ATmega328
- Arduino unter der Haube: Wie geht das in C (C++)?
- Der Analog-Digital Wandler (ADC) im Arduino
- Temperatur-Sensor an der analogen Schnittstelle
- Buchtipps und Links
- Dauer: ca. 1 Stunde/ 30 Folien
- Fragen jederzeit – Details nach dem Vortrag

Was ist Arduino?

Arduino steht für einfaches Programmieren sowie die unkomplizierte Anbindung elektronischer Bauteile. Es ist eine Open Source-Plattform, die die Verbindung von Soft- und Hardware auf einfache Weise ermöglicht.
Herz der Arduino-Boards ist ein Mikrocontroller von ATMEL . Beim „Arduino MEGA“ z.B. der „ATmega2560“. Als Programmiersprache – beim Arduino heißt der Quellcode „Sketch“ – dient ein auf C/ C++ basierender Dialekt.



Arduino MEGA 2560

Neben dem „Arduino MEGA“ gibt es noch zahlreiche weitere Varianten (z.B. „UNO“, „Due“, Leonardo“), die auf dem Arduino-Prinzip basieren.

Ein paar Fakten zum „Arduino MEGA“

- Mikrocontroller: ATMEL ATmega2560
- Versorgungsspannung: 5V
- 54 digitale Input/ Output Pins (davon 15 mit PWM). Zum Vergleich „UNO“: 14 (20), 6 PWM)
- **16 analoge Input Pins (10-Bit ADC) – „UNO“: 6**
- Maximale Stromabgabe pro I/ O Pin: 40 mA
- Flash Memory: 256 KB (8 KB Bootloader) - „UNO“: 32/ 0,5 KB
- SRAM: 8 KB , EEPROM: 4 KB - „UNO“: 2/ 1 KB
- **4 Serielle Schnittstellen (USART) – „UNO“: 1**
- 6 Interrupts – „UNO“: 2
- Arbeitstakt 16 MHz

Was kann der Arduino?

Anwendungsbeispiele:

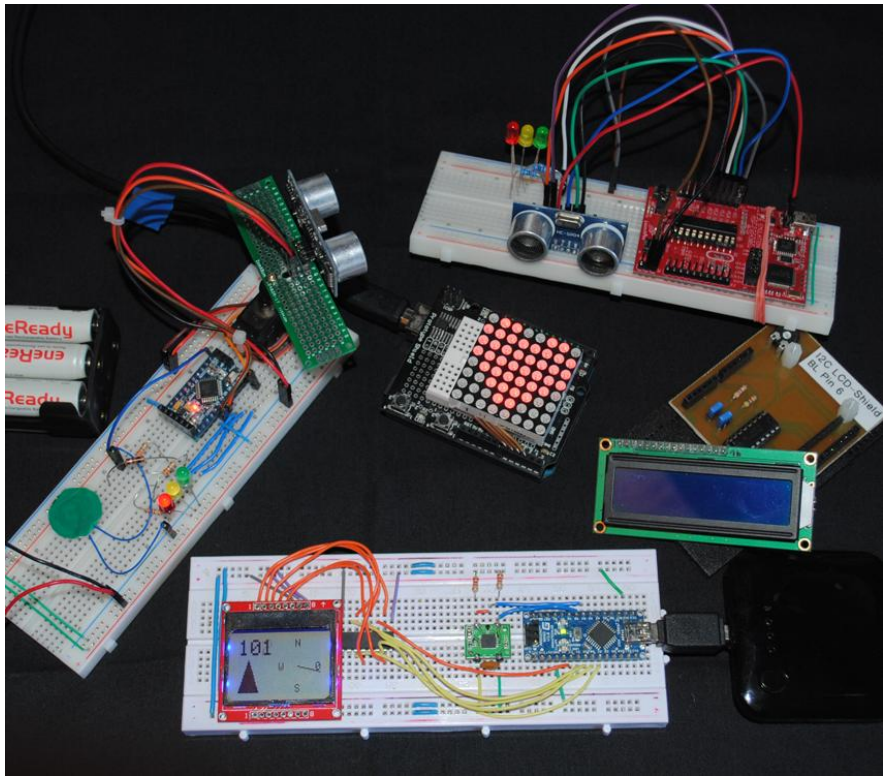
TFT-Touchscreen

ThermalCam

Kompass

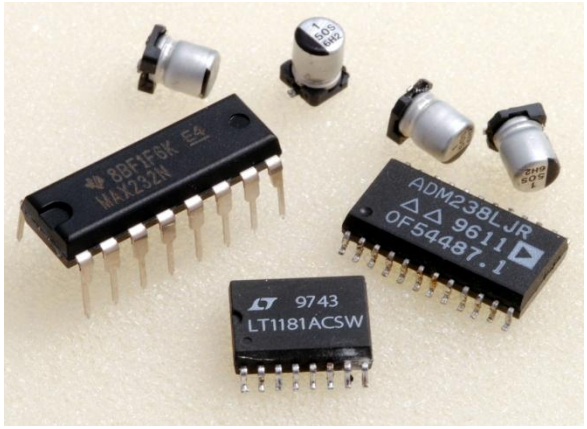
LCDLED-Matrix

Ultraschall-Sensor



Die serielle Schnittstelle – Ursprung

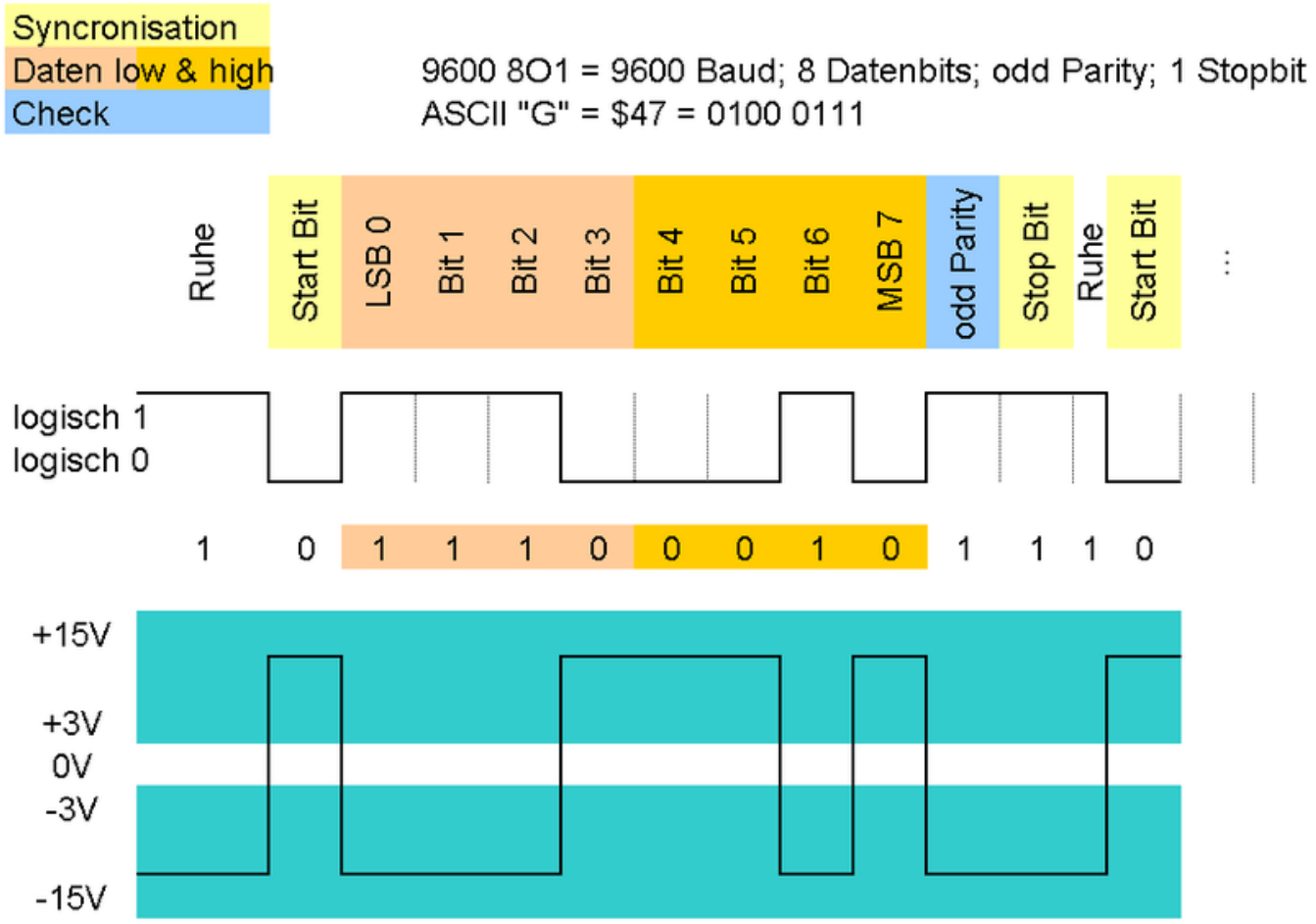
- Serielle Schnittstelle über 100 Jahre alt
- RS-232 Standard seit den 60er Jahren spezifiziert
- Sehr einfaches Protokoll
- Spannungsschnittstelle – digitale, binäre Daten werden durch Spannungspegel definiert
- Kein eigenes Taktsignal (-> asynchron), Synchronisation über Start-/ Stop-Flanke und eingestellte Baudrate, kein CRC, „nur“ Parity-Check



TTL- RS-232 Wandler ICs, Kondensatoren

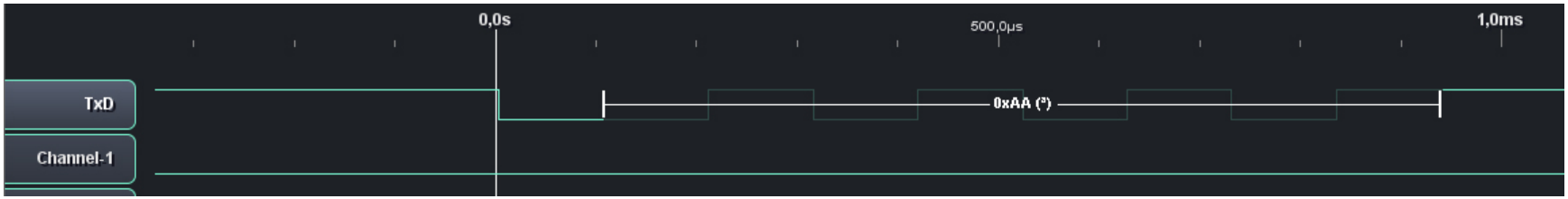
- RS-232 logische „1“ -3 bis -15V, logische „0“ +3 bis +15V. Pegel sind zum Mikrocontroller invertiert („1“/ „HIGH“: Vcc - z.B. 5V, „0“/ „LOW“ 0V)
- PC: D-Sub-Stecker (DB-25/ DB-9)
- Es sind minimal zwei Leitungen notwendig (TX/ RX, GND)
- Am Mikrocontroller: USART/ UART – **U**niversal (**S**ynchronous) **A**synchronous **R**eceiver **T**ransmitter, i.d.R. drei Leitungen (TX, RX, GND)
- Geschwindigkeit wird in BAUD (Bit pro Sekunde - bit/s) gemessen

Serielle Schnittstelle - Protokoll



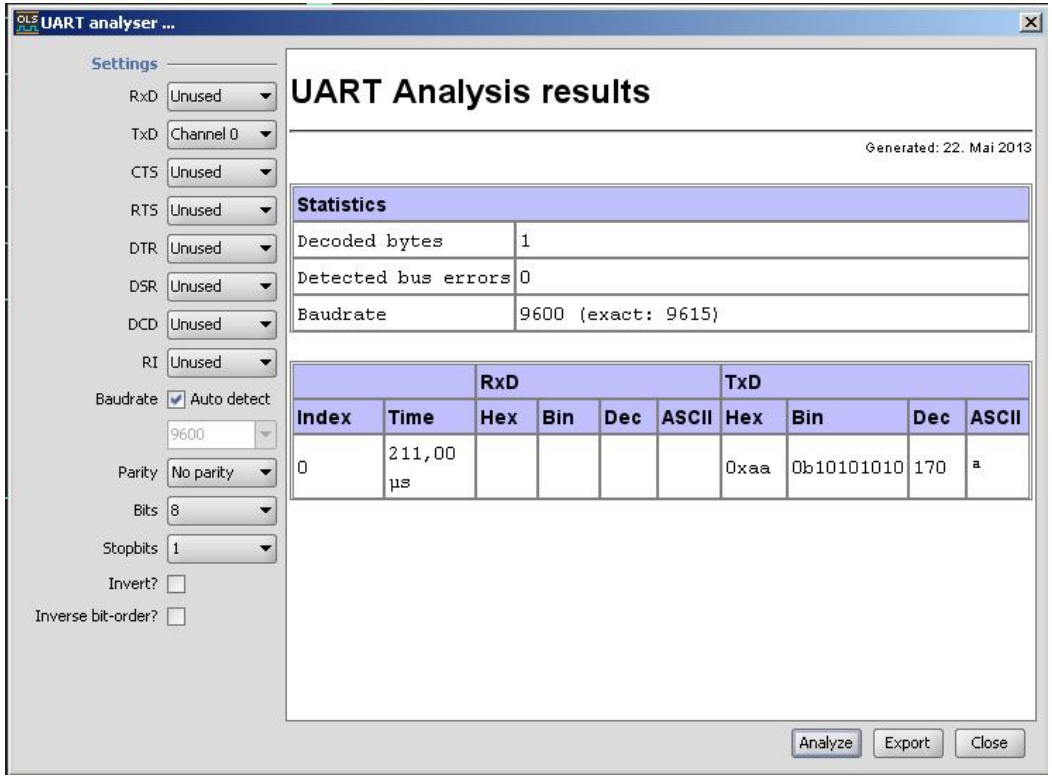
Quelle: Wikipedia, Gerald.deppe

Serielle Schnittstelle – Beispiel



Beispiel für ein serielles Datenpaket (Frame)

- Datenrate 9600 Baud – Brutto-Datenrate
- Bitlänge 104,16µS (1 Sek. / 9600 Bits)
- 1 Start-Bit, 8 Datenbits, keine Paritätsprüfung (Bit), 1 Stop-Bit
- Datenframe 10 Bit
- Payload 8 Bit (Hex 0xAA, Binär 0B10101010)



UART-Protokoll Analyse: Open Bench Logic Sniffer, Software OLS

Serielle Kommunikation – Arduino IDE

Arduino Serial Hardware (Serial-Reference: http://arduino.cc/en/Reference/Serial)

Used for communication between the Arduino board and a computer or other devices. All Arduino boards have at least one serial port (also known as a UART or USART): **Serial**. It communicates on digital pins 0 (RX) and 1 (TX) as well as with the computer via USB. Thus, if you use these functions, you cannot also use pins 0 and 1 for digital input or output.

You can use the Arduino environment's built-in serial monitor to communicate with an Arduino board. Click the serial monitor button in the toolbar and select the same baud rate used in the call to begin().

The **Arduino Mega** has three additional serial ports: **Serial1** on pins 19 (RX) and 18 (TX), **Serial2** on pins 17 (RX) and 16 (TX), **Serial3** on pins 15 (RX) and 14 (TX). To use these pins to communicate with your personal computer, you will need an additional USB-to-serial adaptor, as they are not connected to the Mega's USB-to-serial adaptor. To use them to communicate with an external TTL serial device, connect the TX pin to your device's RX pin, the RX to your device's TX pin, and the ground of your Mega to your device's ground. (Don't connect these pins directly to an RS232 serial port; they operate at +/- 12V and can damage your Arduino board.)

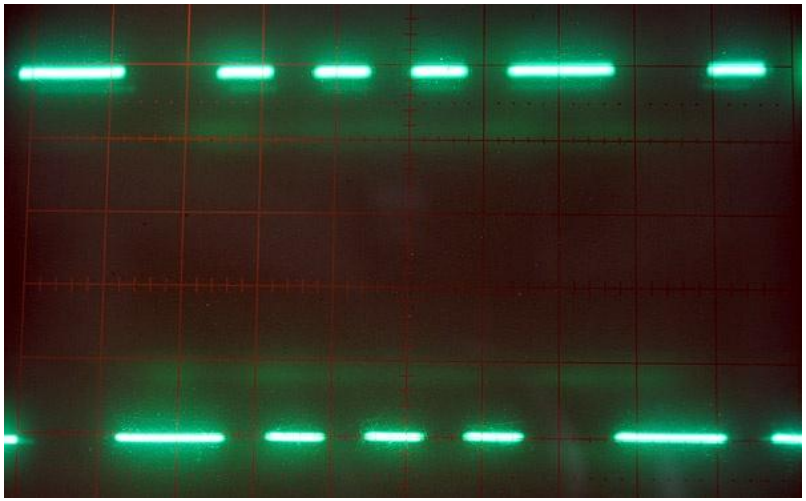
The **Arduino Due** has three additional 3.3V TTL serial ports: **Serial1** on pins 19 (RX) and 18 (TX); **Serial2** on pins 17 (RX) and 16 (TX), **Serial3** on pins 15 (RX) and 14 (TX). Pins 0 and 1 are also connected to the corresponding pins of the ATmega16U2 USB-to-TTL Serial chip, which is connected to the USB debug port. Additionally, there is a native USB-serial port on the SAM3X chip, *SerialUSB*'.

The **Arduino Leonardo** board uses **Serial1** to communicate via TTL (5V) serial on pins 0 (RX) and 1 (TX). **Serial** is reserved for USB CDC (Connected Device Class) communication. For more information, refer to the Leonardo getting started page and hardware page.

Serielle Kommunikation – Arduino IDE

Funktionen/ Methoden in der Arduino IDE:

- if (Serial)
- available()
- begin()
- end()
- find()
- findUntil()
- flush()
- parseFloat()
- parseInt()
- peek()
- print()
- println()
- read()
- readBytes()
- readBytesUntil()
- setTimeout()
- write()
- serialEvent()



UART-Übertragung am analogen Oszilloskop
1V Y-Div/ 5ms X-Div. , 300 Baud, 1 Bit 3,33ms
Bitmuster: Start, B10101010, 1 Stop, LSB first, invertiert

Beispiele in der Arduino IDE unter “Datei -> Beispiele -> 04.Communication”

Mit “Software Serial” gibt es auch eine serielle Schnittstelle auf Software-Basis – allerdings nur bis 4800 Baud.

„Arduino MEGA“, IDE und Sketch

Das Grundgerüst eines Sketches besteht aus drei Bestandteilen:

1. Präprozessor-Direktiven und Variablendeklaration
2. void **setup**() {

// hier werden Grundeinstellungen gemacht, dieser Teil wird nur einmal zu Beginn des Sketches aufgerufen

}

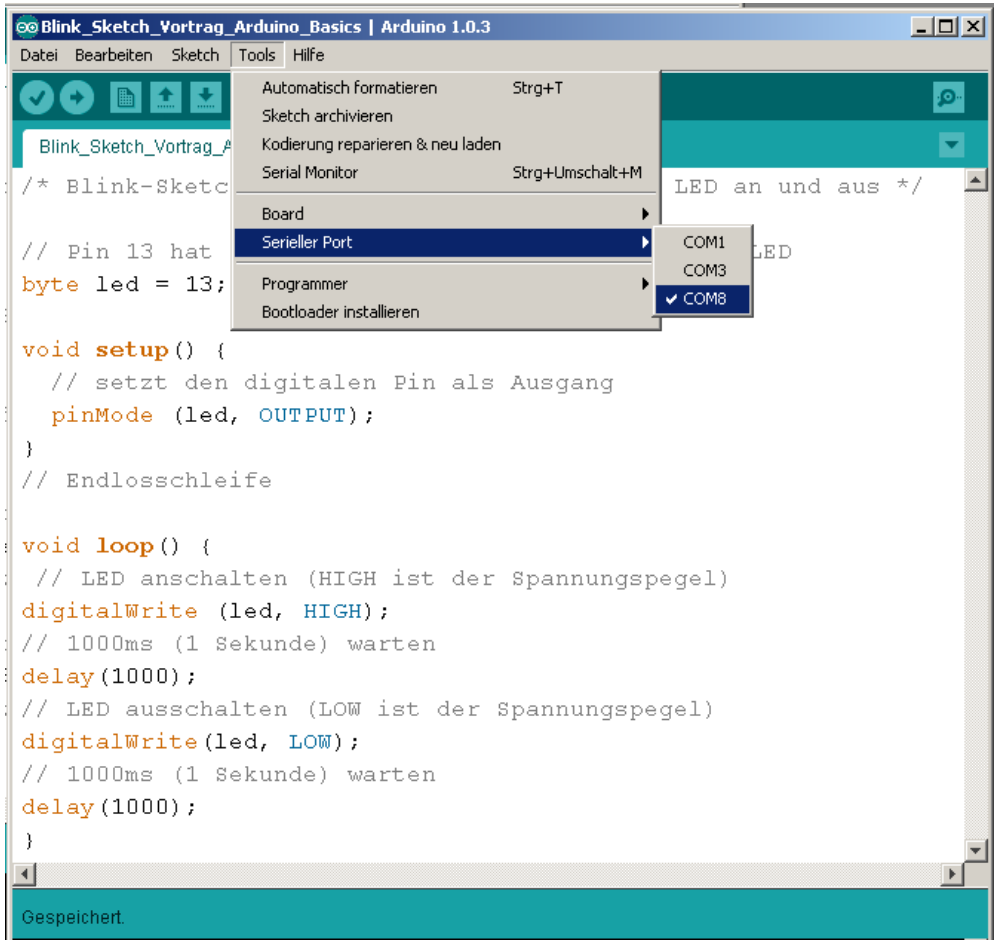
3. void **loop**() {

// Endlosschleife - hier folgt das eigentliche Programm

}

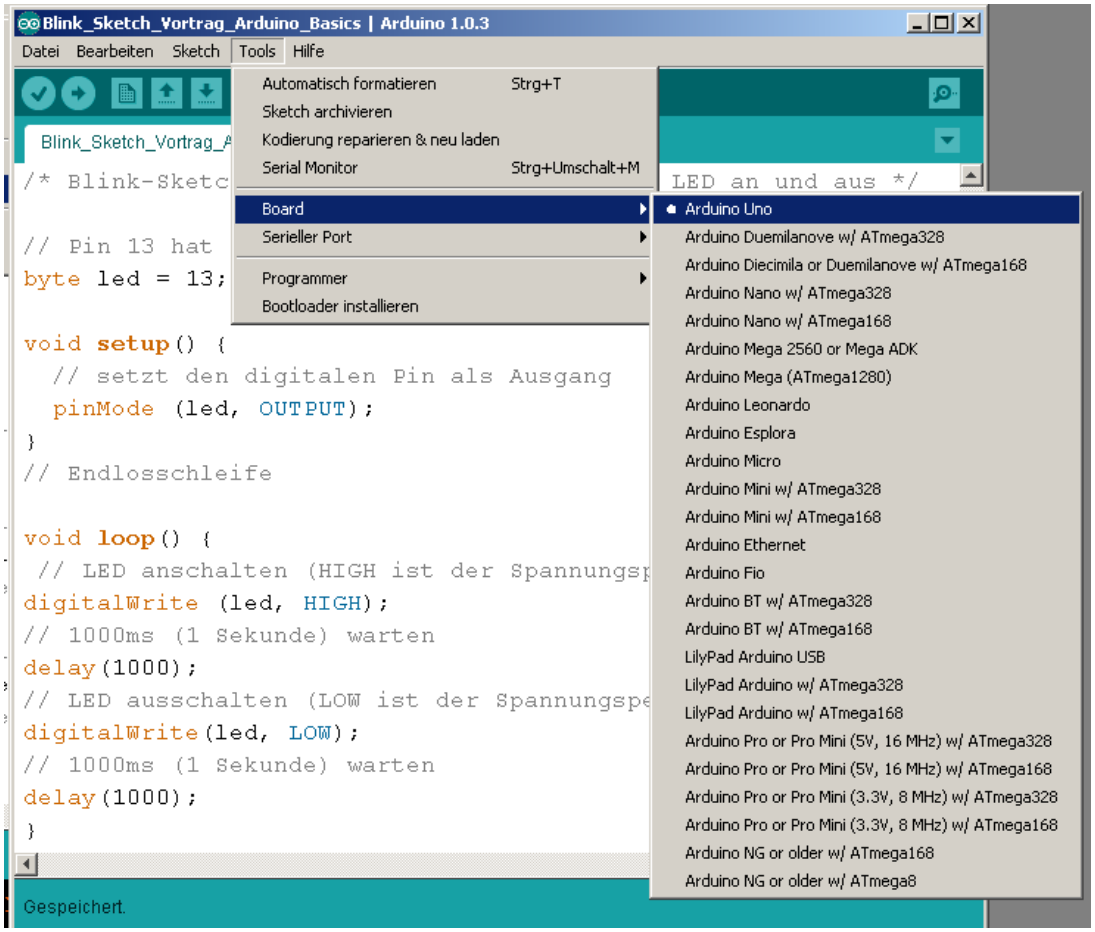
Sketch auf den Arduino bringen

1. Seriellen Port auswählen

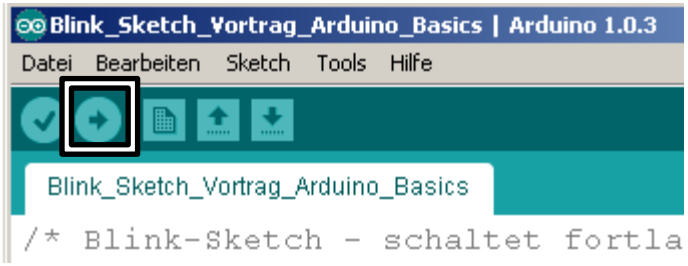


Sketch auf den Arduino bringen

2.
Board/
Mikrocontroller
auswählen

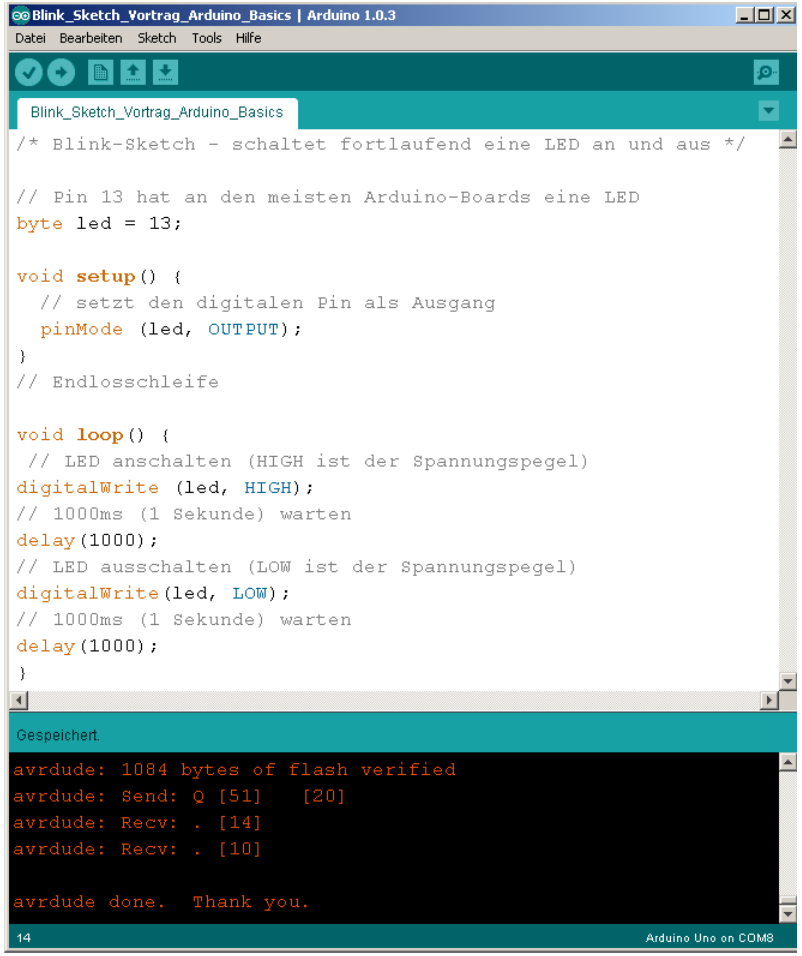


Sketch auf den Arduino bringen



3.

Upload



Serielle Kommunikation – Arduino IDE

Sketch-Beispiel für das Senden über Serial:
(Ausgabe über den seriellen Monitor der Arduino IDE)

```
char c; // Initialisierung der Variable c als Datentyp char

void setup() {
    Serial.begin(9600); // Initialisierung der Schnittstelle mit 9600 Baud, 8 Bit, 1 Stop-Bit
    c = 1;
}

void loop() {
    Serial.print(c); // Ausgabe der Variable
    Serial.println(); // Zeilenumbruch
    Serial.print(c, DEC);  // Ausgabe als Dezimalzahl
    Serial.println(); // mit dem Punktoperator „.“ greift man auf die Methode der Klasse zu
    delay(100); // 100 ms Verzögerung
    c++; // Inkrementieren von c um 1, gleichbedeutend mit c = c+1;
}
```

Serielle Kommunikation – Arduino IDE

Sketch-Beispiel für das Empfangen über Serial:
(Eingabe über den seriellen Monitor der Arduino IDE)

```
char c; // Initialisierung der Variable c als Datentyp char

void setup() {
    Serial.begin(9600); // Initialisierung der Schnittstelle mit 9600 Baud, 8 Bit, 1Stop-Bit
}

void loop() {
    if(Serial.available()){ // Wenn ein Byte empfangen wurde
        c = Serial.read(); // wird das empfangene Byte der Variable c zugewiesen
        Serial.println(c); // und im seriellen Monitor mit Zeilenumbruch ausgegeben
    }
}
```

USART des ATmega328 (Arduino „UNO“)

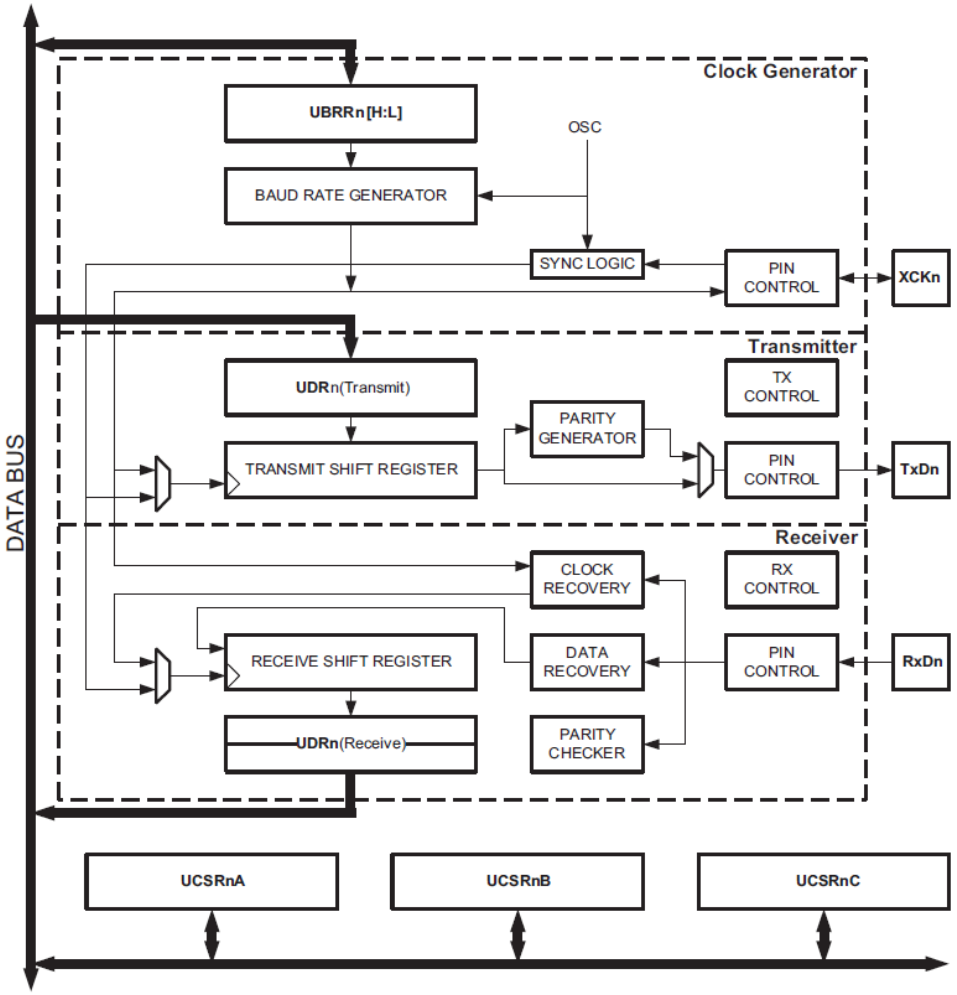
Features des USART0 am ATmega328:

- Voll-Duplex-Betrieb (Unabhängige serielle Empfangs- und Senderegister)
- Asynchroner und synchroner Betrieb
- Unterstützt Datenpakete mit 5, 6, 7, 8, oder 9 Daten-Bits und 1 oder 2 Stop-Bits (30 Kombinationen)
- Odd oder Even Parity-Generierung und Parity Check durch Hardware unterstützt
- Data OverRun Erkennung
- Framing Error Erkennung
- Drei gesonderte Interrupts für TX Complete, TX Data Register Empty und RX Complete
- Kann auch im Master SPI Mode betrieben werden
- Maximale Datenrate bei 16 MHz: 1 (2) Mbps
- Übertragung: LSB first

USART des ATmega328 (Arduino „UNO“)

In einem Mikrocontroller werden so genannte „Register“ beschrieben, um z.B. bestimmte Einstellungen zu setzen

Die USART0 Schnittstelle des ATmega328 hat fünf Register: UDR0, UCSR0A, UCSR0B, UCSR0C, und UBBR0 (L und H)



Blockdiagram USART Schnittstelle, ATmega328 (complete manual)

USART des ATmega328 (Arduino „UNO“)

Beispiel: Ersetzen von „Serial.begin(9600)“ durch das **bitweise Manipulieren** der Register

```
#define BAUD_RATE 9600
#define BAUD_RATE_DIVISOR (F_CPU / 16 / BAUD_RATE -1)

void usart_init(){
  // Initialisieren der USART Status and Control Register 0A, 0B and 0C
  // USART Control and Status Register 0 A auf 0 gesetzt (Voreinstellung)
  // man könnte dies auch auslassen
  UCSR0A = 0;
  // UCSR0B – USART Control and Status Register 0 B - TXEN0 (TX enable) und RXEN0 (RX enable) auf 1 gesetzt
  // schaltet den Empfänger und Sender ein
  UCSR0B  |= 1 << TXEN0 | 1 << RXEN0;
  // UCSR0C – USART Control and Status Register 0 C - UCSZ01 and UCSZ00
  // beide auf 1  gesetzt bedeuten 8 data bits
  UCSR0C = 1 << UCSZ01 | 1 << UCSZ00;
  // Baud Rate auf 9600
  UBRR0 = BAUD_RATE_DIVISOR;
}
```

USART des ATmega328 (Arduino „UNO“)

20.11.3 UCSRnB – USART Control and Status Register n B

Bit	7	6	5	4	3	2	1	0	
	RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 4 – RXENn: Receiver Enable n**

Writing this bit to one enables the USART Receiver. The Receiver will override normal port operation for the RxDn pin when enabled. Disabling the Receiver will flush the receive buffer invalidating the FEn, DORn, and UPEn Flags.

- **Bit 3 – TXENn: Transmitter Enable n**

Writing this bit to one enables the USART Transmitter. The Transmitter will override normal port operation for the TxDn pin when enabled. The disabling of the Transmitter (writing TXENn to zero) will not become effective until ongoing and pending transmissions are completed, i.e., when the Transmit Shift Register and Transmit Buffer Register do not contain data to be transmitted. When disabled, the Transmitter will no longer override the TxDn port.

/* UCSR0B – USART Control and Status Register 0 B - TXEN0 (TX enable) und RXEN0 (RX enable) auf 1 gesetzt, schaltet den Empfänger und Sender ein */
UCSR0B |= 1 << TXEN0 | 1 << RXEN0; // “OR”-Manipulation der Register-Bits

```
// UCSR0B |= 1 << 4| 1 <<  3;
// UCSR0B |= 0x18;
// UCSR0B |= B00011000;
// UCSR0B |= _BV(TXEN0) | _BV(RXEN0); // <avr/sfr_defs.h>: Special function registers, #define _BV(bit)   (1 << (bit))
// bitSet (UCSR0B, TXEN0); bitSet (UCSR0B, TXEN0);
// sbi(UCSR0B, TXEN0); sbi(UCSR0B, RXEN0);  // veraltet
```

Serielle Kommunikation in „C“

Funktion zum Senden eines char

```
void usart_putc(char c) {
  // Warten bis der Transmitbuffer (UDR0) leer ist, das wird angezeigt,
  // wenn das UDRE0-Bit gesetzt ist
  loop_until_bit_is_set(UCSR0A, UDRE0);  // verzögernde while-Schleife, für
     schnelle Operationen empfiehlt sich die Nutzung der Interrupts
  // UDR0 – USART I/O Data Register 0 ist der Datenbuffer für Lese- und
     Schreiboperationen über USART
  UDR0 = c; // Übertragung eines char
}
```

Serielle Kommunikation in „C“

Funktion zum Senden eines Strings

```
void usart_puts(char *s) { // als Argument wird ein Pointer des char übergeben
  while(*s) { // Solange ein Pointer auf char vorhanden ist
    usart_putc(*s); // wird die Funktion usart_putc() genutzt und ein char übertragen
    s++; // Pointer wird inkrementiert
  }
}
```

Serielle Kommunikation in „C“

Funktion um einen Integer zu senden:

```
void usart_puti(int i) {
  char s[3]; // ein Array für eine dreistellige Zahl wird initialisiert
  itoa(i,s,10); // die C-Funktion itoa() (Integer to ASCII – weder C noch C++,
     aber von Compilern unterstützt) wird genutzt, um den übergebenen
     Integer-Wert in einen Null-terminierten String mit Basis 10 (Zehnersystem)
     zu wandeln
  usart_puts(s); // Der „Integer-String“ wird mit Hilfe der usart_puts()-
     Funktion über die serielle Schnittstelle gesendet
}
```

Analog-Digital-Wandler (ADC) am Arduino

Mit dem Analog-Digital-Wandler (ADC) beim Arduino kann man analoge Spannungspegel in digitale Signale wandeln. Der Arduino „UNO" hat 6 Kanäle („Mini" und „Nano" 8) – der „MEGA" 16.

Die Auflösung des ADC-Wandlers beträgt 10 Bit, kann also die Werte 0 bis 1023 für Spannungen von 0 bis zur Versorgungsspannung (z.B. „UNO: 5 Volt) annehmen. Damit erreicht man auf die Versorgungsspannung vom „UNO" bezogen eine Auflösung von 0,00488 Volt oder 4,88mV. Der Eingangsbereich und damit die Auflösung kann mit der Funktion analogReference() geändert werden.

Das Auslesen eines analogen Eingangs dauert etwa 104 Mikrosekunden (ADC-Prescaler 128, 13 Taktzyklen zum Auslesen, Arduino IDE), es ist also eine maximal Samplerate von 9,6 kS/s möglich.

Die Funktion zum Wandeln eines analogen Spannungswertes in digitale Werte lautet:

analogRead(pin)

Analog-Digital-Wandler (ADC) am Arduino

Der Temperatursensor TMP36GZ (Analog Devices):

- Low voltage operation (2.7 V to 5.5 V)
- Calibrated directly in °C
- **10 mV/°C scale factor**
- ±0.5°C linearity (typ.)
- Specified −40°C to +125°C, operation to +150°C
- Less than 50 µA supply current
- Shutdown current 0.5 µA max (only SOT/ SOIC)
- Low self-heating
- **Output 750mV @ 25°C**
- **Accuracy Ta 25°C: typ. +-1°C, max. +-3°C**

TMP36

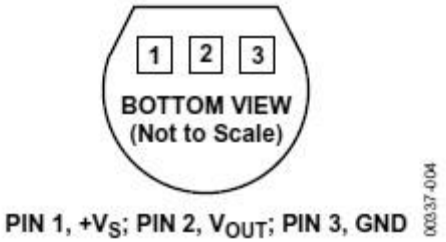


Figure 4. TO-92

TMP36 Pinout, Quelle: Datenblatt

Analog-Digital-Wandler (ADC) am Arduino

Sketch-Beispiel mit einem analogen Temperatursensor (TMP36GZ) mit Ausgabe auf der seriellen Schnittstelle:

```
const byte tmp36 = 0;
int tempRaw = 0;
float temp;

void setup () {
        Serial.begin(9600);
}
void loop () {
        tempRaw = analogRead(tmp36);
        temp = ((4.8828125*tempRaw)-500)/10;
        Serial.println(tempRaw);
        Serial.println(temp);
        delay(1000);
}
```

Analog-Digital-Wandler (ADC) am Arduino

ADC Auslesen – Setzen der Register (3,3 Volt ATmega328P)

```
int var;

// ADC Multiplexer Selection Register (ADMUX) - use Vcc (3V3) as reference and read channel "0"
// you could skip the MUX-bits - it's just mentioned for more clarity
 ADMUX |= 1 << REFS0 | 0 << MUX3 | 0 << MUX2 | 0 << MUX1 | 0 << MUX0;

// ADC Control and Status Register A (ADCSRA)
// enable AD conversion (ADEN) and start conversion (ADSC) with
// prescaler 128, which means 16 MHz/ 128 = 125kHz clock (ADPS2:ADPS0)
ADCSRA |= 1 << ADEN | 1 << ADSC | 1 << ADPS2 | 1 << ADPS1 | 1 << ADPS0;

// ADCSRB – ADC Control and Status Register B - free running mode is 0
// you also could skip this, as ADCSRB is 0 by default
ADCSRB |= 0 << ADTS2 | 0 << ADTS1 | 0 << ADTS0;

// start new conversion and give it a little bit of time
 bitSet(ADCSRA, ADSC);
delay(1);
var = ADC;
```

Arduino Programming Cheat Sheet

Primary source: Arduino Language Reference
http://arduino.cc/en/Reference/

Structure & Flow

Basic Program Structure

```
void setup() {
  // runs once when sketch starts
}
void loop() {
  // runs repeatedly
}
```

Control Structures

```
if (x < 5) { ... } else { ... }
while (x < 5) { ... }
do { ... } while ( x < 5);
for (int i = 0; i < 10; i++) { ... }
break;  // exit a loop immediately
continue;  // go to next iteration
switch (myVar) {
   case 1:
     ...
     break;
   case 2:
     ...
     break;
   default:
     ...
}
return x; // just return; for voids
```

Operators

General Operators

```
=  (assignment operator)
+  (add)          -  (subtract)
*  (multiply)     /  (divide)
%  (modulo)
== (equal to)     != (not equal to)
<  (less than)    >  (greater than)
<= (less than or equal to)
>= (greater than or equal to)
&& (and)    || (or)    ! (not)
```

Compound Operators

```
++ (increment)
-- (decrement)
+= (compound addition)
-= (compound substraction)
*= (compound multiplication)
/= (compound division)
&= (compound bitwise and)
|= (compound bitwise or)
```

Bitwise Operators

```
&  (bitwise and)    |  (bitwise or)
^  (bitwise xor)    ~  (bitwise not)
<< (shift left)     >> (shift right)
```

Built-in Functions

Pin Input/Output

```
Digital I/O (pins: 0-13 A0-A5)
  pinMode(pin,[INPUT, OUTPUT])
  int digitalread(pin)
  digitalWrite(pin, value)
    // Write HIGH to an input to
    // enable pull-up resistors
Analog In (pins: 0-5)
  int analogRead(pin)
  analogReference(
    [DEFAULT, INTERNAL, EXTERNAL])
PWM Out (pins: 3 5 6 9 10 11)
  analogWrite(pin, value)
```

Advanced I/O

```
tone(pin, freqhz)
tone(pin, freqhz, duration_ms)
noTone(pin)
shiftOut(dataPin, clockPin,
  [MSBFIRST,LSBFIRST], value)
unsigned long pulseIn(pin,
  [HIGH,LOW])
```

Time

```
unsigned long millis()
  // overflows at 50 days
unsigned long micros()
  // overflows at 70 minutes
delay(msec)
delayMicroseconds(usec)
```

Math

```
min(x, y)    max(x, y)    abs(x)
sin(rad)     cos(rad)     tan(rad)
sqrt(x)      pow(base, exponent)
constrain(x, minval, maxval)
map(val, fromL, fromH, toL, toH)
```

Random Numbers

```
randomSeed(seed) // long or int
long random(max)
long random(min, max)
```

Bits and Bytes

```
lowByte(x)    highByte(x)
bitRead(x, bitn)
bitWrite(x, bitn, bit)
bitSet(x, bitn)
bitClear(x, bitn)
bit(bitn)  // bitn: 0=LSB 7=MSB
```

Type Conversions

```
char()        byte()
int()         word()
long()        float()
```

External Interrupts

```
attachInterrupt(interrupt, func,
  [LOW, CHANGE, RISING, FALLING])
detachInterrupt(interrupt)
interrupts()
noInterrupts()
```

Variables, Arrays, and Data

Data types

```
void
boolean   (0, 1, true, false)
char      (e.g. 'a'  -128 to 127)
int       (-32768 to 32767)
long      (-2147483648 to 2147483647)
unsigned char  (0 to 255)
byte           (0 to 255)
unsigned int   (0 to 65535)
word           (0 to 65535)
unsigned long  (0 to 4294967295)
float     (-3.4028e+38 to 3.4028e+38)
double    (currently same as float)
```

Qualifiers

```
static    (persists between calls)
volatile  (in RAM (nice for ISR))
const     (make read only)
PROGMEM   (in flash)
```

Arrays

```
int myInts[6]; // array of 6 ints
int myPins[]={2, 4, 8, 3, 6};
int mySensVals[6]={2, 4, -8, 3, 2};
myInts[0]=42;  // assigning first
               // index of myInts
myInts[6]=12;  // ERROR! Indexes
               // are 0 though 5
```

Constants

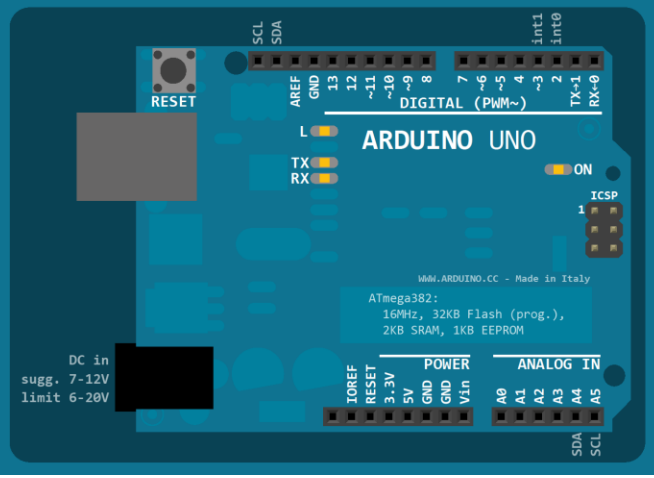
```
HIGH  | LOW
INPUT | OUTPUT
true  | false
143       (Decimal)
0173      (Octal - base 8)
0b11011111 (Binary)
0x7B      (Hexadecimal - base 16)
7U        (force unsigned)
10L       (force long)
15UL      (force long unsigned)
10.0      (force floating point)
2.4e5     (2.4*10^5 = 240000)
```

Pointer Access

```
&  (reference: get a pointer)
*  (dereference: follow a pointer)
```

Strings

```
char S1[8] =
 {'A','r','d','u','i','n','o'};
 // unterminated string; may crash
char S2[8] =
 {'A','r','d','u','i','n','o','\0'};
 // includes \0 null termination
char S3[]="Arduino";
char S4[8]="Arduino";
```



Libraries

Serial (communicate with PC or via RX/TX)

```
begin(long Speed) // up to 115200
end()
int available() // #bytes available
byte read() // -1 if none available
byte peek()
flush()
print(myData)
println(myData)
write(myBytes)
SerialEvent() // called if data rdy
```

SoftwareSerial (serial comm. on any pins)

```
  (#include <softwareSerial.h>)
SoftwareSerial(rxPin, txPin)
begin(long Speed) // up to 115200
listen()       // Only 1 can listen
isListening() // at a time.
read, peek, print, println, write
  // all like in Serial library
```

EEPROM (#include <EEPROM.h>)

```
byte read(intAddr)
write(intAddr, myByte)
```

Servo (#include <Servo.h>)

```
attach(pin, [min_uS, max_uS])
write(angle)  // 0 to 180
writeMicroseconds(uS)
   // 1000-2000; 1500 is midpoint
int read()    // 0 to 180
bool attached()
detach()
```

Wire (I²C comm.) (#include <Wire.h>)

```
begin()      // join a master
begin(addr) // join a slave @ addr
requestFrom(address, count)
beginTransmission(addr) // Step 1
send(myByte)            // Step 2
send(char * mystring)
send(byte * data, size)
endTransmission()       // Step 3
int available() // #bytes available
byte receive()  // get next byte
onReceive(handler)
onRequest(handler)
```


Quelle: https://github.com/liffiton/Arduino-Cheat-Sheet/blob/master/Arduino%20Cheat%20Sheet.pdf

Buchtipps & Links

Links:

Arduino-Homepage: http://arduino.cc

Tutorials: http://tronixstuff.wordpress.com/tutorials/

Arduino UNO Pinout: http://www.pighixxx.com/downloads/the-definitive-arduino-pinout-diagram/

Bücher:

Die elektronische Welt mit Arduino entdecken, Erik Bartmann, O'Reilly, ISBN: 978-3897213197

Beginning Arduino, Michael McRoberts, Apress, ISBN: 978-1-4302-3240-7