Technikschule Esslingen
Kurs: Arduino
Herzlich Willkommen

Arduino Uno

Arduino Uno R3 Back

Arduino Uno R3 Front

Inhalt:

1) Installation Seite 5
2) Eine LED ansteuern (blinken lassen) Seite 8
3) Programmstruktur, Funktionen, Variablen Seite 12
4) Taster steuert eine LED (ein- / ausschalten) Seite 17
5) Programm- und Kontrollstrukturen (if / else, for…., while) Seite 21
6) Funktionen parallel programmieren (LED1 blinkt, Taste steuert LED2 Seite 24
7) Übersicht der Arduino Funktionen Seite 29
8) LED Lauflicht programmieren Seite 30
9) LED Lauflicht per Potentiometer ansteuern Seite 34
10) Lichtsensor steuert LED Lauflicht (Helligkeitsmesser) Seite 39
11) Daten vom Arduino per USB an den PC übertragen und anzeigen Seite 42
12) ASCII Tabelle Seite 38
13) Programm- und Kontrollstrukturen (switch) Seite 47
14) Daten vom PC lesen und wieder an den PC zurückübertragen Seite 48
15) Ansteuerung eines SERVO Motors Seite 49
16) Puls-Weiten-Modulation (PWM) zur Analogausgabe Seite 50
17) Ansteuerung eines Standard Motors Seite 55
18) Motortreiber IC L293D Seite 58
19) Ansteuerung von 2 Standard Motoren Seite 59
20) Ansteuerung eines Schrittmotors Seite 64
21) 7-Segment Anzeigen ansteuern Seite 70

Inhalt: (ii)

22) Ansteuerung eines Transistors	Seite 76
23) Multiplexing	Seite 77
24) Reaktionstester	Seite 80
25) Schieberegister HC595 Porterweiterung	Seite 88
26) XmalY Folientastatur ansteuern	Seite 100
27) LCD Display ansteuern	Seite107
28) Temperatur mit LM35 messen	Seite 115
29) Motor mit Leistungstransistor TIP120 ansteuern	Seite 116
30) Die Hysterese Funktion	Seite 120
31) Töne mit einem Piezo Element erzeugen	Seite 125
32) Melodien über einen Lautsprecher abspielen	Seite 132
33) Der Digital-Analog Wandler (DA)	Seite 133
34) Rechteck, Dreieck und Sinussignale ausgeben	Seite 139
35) Die Register Programmierung für schnelle Programme	Seite 144
36) Projekt „Farbensequenzspiel“	Seite 148
37) Taster Ansteuerung mit „Pull-Down“ oder „Pull-Up“	Seite 150
38) Der BCD Zähler 4026	Seite 153
39) Taster Prellen verhindern	Seite 156
40) Ansteuerung eines 8x8 LED Displays	Seite 170

Inhalt: (iii)

41) Serielle Schnittstellen (One Wire, SPI, I2C) Seite 176
42) Die „Wire“ – Library Seite 180
43) Der I2C 8Bit I/O Expander Seite 184
44) Digitales Potentiometer mit I2C Bus Seite 192
45) Neue Bibliotheken einbinden Seite 196
46) One Wire Feuchte und Temperatur Sensor „DHT11“ Seite 199
47) One Wire Temperatur Sensor „18B20“ Seite 201
48) Die „OneWire“ – Library Seite 203
49) Die SD-Karte Seite 206
50) Die TIME Library Seite 227
51) Das I2C LCD Display Seite 234
52) Grundlagen TRANSISTOR Seite 241
53) Grundlagen DIODE Seite 245
54) Grundlagen WIDERSTAND Seite 250
55) Grundlagen KONDENSATOR Seite 255

Installation:

1) ZIP Datei auf Laptop kopieren, z.B. in Verzeichnis User, Temp, Desktop, etc:

2) ZIP Datei entpacken, z.B. in Verzeichnis „Programme\Arduino" oder den Anweisungen des Installers folgen

3) Treiber installieren:

Startet automatisch nach Anschluß an USB oder über „**Systemsteuerung**" in den „**Gerätemanager**" wechseln. Dort nach unbekannter Hardware suchen. Anclicken und über rechte Maustaste „Treiber aktualisieren". Treiber liegt im **Programmverzeichnis „Arduino\drivers"**

Programmiersoftware (IDE) starten:

1) Anwendung „**arduino.exe**“ doppelclicken“

2) Zum schnelleren Finden und Starten kann der Programm Link auf den Desktop und in die Taskleiste kopiert werden

3) „**Board Typ**“ und „**Schnittstelle**“ auswählen (Orig. Uno R3)

Installation:

- Board Einstellung für „**Iteaduino 2.2**“ von Itead Studio und Seeeduino V3.0 von Seeed
- Treiberinstallation von Hand über Verzeichnis C:\Program Files (x86)\Arduino\drivers (XP & Win7)

Vorgang Zweimal: einmal für USB Port einmal für Seriell Port

- Treiberinstallation übers Internet (Win7)

- Board Einstellung für „**xxx**“ von xxxx
- Treiberinstallation von Hand über Verzeichnis C:\Program Files (x86)\Arduino\drivers (XP & Win7)
- Treiberinstallation übers Internet (Win7)

Unser 1. Programm: Einen digitalen Ausgang und damit eine blinkende Leuchtdiode (LED) ansteuern

Digitale Ein-/Ausgänge:

Pin „0“ und „1“ wird für die USB Schnittstelle verwendet.
Daher stehen diese normalerweise nicht zur Verfügung.
Somit gibt es 12 dig. Ein- und Ausgänge: Pin „2“ bis „13“

Pin „A0“ - „A5“ sind sechs analoge Eingänge.

Diese können mit einem Trick auch als dig. Eingänge verwendet werden.
(s. Programm „Reaktionstester“

Sie können aber auch als dig. Ausgänge verwendet werden.
Dazu im Programm statt Pin „A0“ bis „A5“ → Pin „14“ bis „19“ schreiben

Anode über einen 220 oder 330 Ohm Widerstand mit Digital-Eingang 13 verbinden

Steckboard mit Arduino Plus (5V) und Arduino Masse (Ground verbinden

Kathode mit Masse (Ground) verbinden

Technikschule Esslingen; Kurs: Arduino

Blink_deu_Kurs1 | Arduino 1.0.1

Datei Bearbeiten Sketch Tools Hilfe

Blink_deu_Kurs1

```
/*
  Blink
  Schaltet eine LED zyklisch ein und aus (blinken)

 */

int ledPin = 13;     // An Pin 13 wird die LED angeschlossen
                     // Name "led" vergeben, damit die SW besser lesbar ist

// die folgende "setup" routine muss einmal im Programm stehen und wird nach einem RESET einmal durchlaufen:
void setup()
{
  pinMode(ledPin, OUTPUT);        // mit der Funktion "pinMode" wird der Pin "led" als Ausgang definiert
}

// die "loop" routine muss auch einmal im Programm stehen, wird aber immer wieder von vorne durchlaufen
// eine sogenannte Endlos-Schleife:
void loop()
{
  digitalWrite(ledPin, HIGH);     // schaltet die LED ein (HIGH ist ein Spannungs-Pegel)
  delay(1000);                    // für 1000 ms also 1 sec warten
  digitalWrite(ledPin, LOW);      // schaltet die LED aus indem der Spannungs-Pegel auf LOW gesetzt wird
  delay(600);                     // für 600 ms also 0,6 sec warten
}
```

Jedes Programm:

1) beginnt mit der Deklaration und Initialisierung von **Variablen**

2) gefolgt von der Funktion **setup()**
hier stehen alle Programmteile, die nur einmal ausgeführt werden

3) und dann die Funktion **loop()**
hier stehen alle Programmteile, die immer wieder ausgeführt werden

4) Jede Programmzeile (Befehl) endet mit einem **Semikolon**

;

Funktionen: **Rückgabedatentyp Funktionsname(…)**

1) haben immer eine **Rückgabedatentyp**-Angabe, die den Wert beschreibt, der am Ende zurückgeliefert wird → (**int, byte, char, float oder void**)

2) haben einen eindeutigen **Namen** → (Groß- und Kleinschreibung beachten)

3) Gefolgt von zwei Klammern **(parameter1, parameter2, … parameterX)**
In der Klammer können 1 bis x Parameternamen stehen.
Das sind Werte die beim Funktionsaufruf übergeben werden
und mit denen die Funktion intern arbeitet.

```
void resetPins()    // Funktion setzt 3 Pins auf LOW (Reset)
{
  digitalWrite(taktPin, LOW);
  digitalWrite(speicherPin, LOW);
  digitalWrite(datenPin, LOW);
}
```

```
int Addieren(wert1, wert2)    // Funktion addiert 2
                              // Werte und gibt die Summe zurück
{
  int summe = wert1 + wert2;
  return(summe);
}
```

- *LOW*-Pegel (wird mit *L* oder *0* abgekürzt)
- *HIGH*-Pegel (wird mit *H* oder *1* abgekürzt)

Hier siehst du ein Signal, das digitalen Charakter besitzt:

Im Gegensatz dazu haben analoge Signale eine ganz andere Qualität. Sie können nicht nur im zeitlichen Verlauf zwischen den zwei Pegeln *HIGH* bzw. *LOW* unterscheiden, sondern sie haben die Eigenart, stufenlos zwischen einem minimalen und maximalen Wert zu pendeln.

Variablen und Datentypen

Variablen sind “Behälter” für Daten. Im Laufe eines Programms können diese beschrieben, ausgelesen oder geändert werden.

Um eine Variable nutzen zu können, muss sie zunächst **deklariert** werden. Hierbei wird festgelegt welcher **Datentyp** diese Variable besitzen und evtl. auch welchen **Anfangswert** sie erhalten soll.

Der Typ legt fest, welche Art von Zahlen man in der Variable speichern kann (ganze Zahlen, positive ganze Zahlen, Dezimalzahlen, Ja/Nein Wert, usw).

Der Name einer Variablen muss mit einem Buchstaben beginnen (“_” zählt ebenfalls als Buchstabe) und darf auch Zahlen, aber keine Sonderzeichen enthalten. **Groß- und Kleinschreibung** werden unterschieden → „x“ und „X“ sind verschiedene Variablen.

Dezimalzahlen 0 bis 15 im Dualsystem	
Wertigkeit:	**8 4 2 1**
Null:	0 0 0 0
Eins:	0 0 0 1
Zwei:	0 0 1 0
Drei:	0 0 1 1
Vier:	0 1 0 0
Fünf:	0 1 0 1
Sechs:	0 1 1 0
Sieben:	0 1 1 1
Acht:	1 0 0 0
Neun:	1 0 0 1
Zehn:	1 0 1 0
Elf:	1 0 1 1
Zwölf:	1 1 0 0
Dreizehn:	1 1 0 1
Vierzehn:	1 1 1 0
Fünfzehn:	1 1 1 1

	Stellenwert							
	32	16	8	4	2	1		
Dualzahl	0	0	0	1	0	1	5	Dezimalzahl
	1	0	0	0	1	1	35	
	0	0	1	0	1	0	10	

Vom Dual- ins Dezimalsystem:
Man addiert alle Stellenwerte, die über den Einsen der Dualzahl stehen und erhält die entsprechende grün hinterlegte Dezimalzahl.
Bsp letzte Zeile:
Stellenwerte 8 und 2 addiert. Das Ergebnis ist 10.

Vom Dezimal- ins Dualsystem:
Divisionsmethode (Modulo-Methode) am Beispiel $41_{(10)}$ beschrieben:

$$\begin{array}{rcrcl} 41 & :2 & = & 20 & \text{Rest } \mathbf{1} \\ 20 & :2 & = & 10 & \text{Rest } \mathbf{0} \\ 10 & :2 & = & 5 & \text{Rest } \mathbf{0} \\ 5 & :2 & = & 2 & \text{Rest } \mathbf{1} \\ 2 & :2 & = & 1 & \text{Rest } \mathbf{0} \\ 1 & :2 & = & 0 & \text{Rest } \mathbf{1} \end{array} \uparrow$$

Reste von unten nach oben: $101001_{(2)}$

8-Bit Zweierkomplement

Binärwert	Interpretation als Zweierkomplement	Interpretation als vorzeichenlose Zahl
00000000	0	0
00000001	1	1
...	...	...
01111110	126	126
01111111	127	127
10000000	−128	128
10000001	−127	129
10000010	−126	130
...	...	...
11111110	−2	254
11111111	−1	255

Negative Zahlen werden wie folgt aus einer positiven Zahl kodiert: Sämtliche binären Stellen werden negiert und zu dem Ergebnis der Wert *1* addiert

Dytentyp	Wertebereich	Datenbreite	Beispiel
byte	*0 bis 255*	*1 Byte*	`byte wert = 42;`
unsigned int	*0 bis 65.535*	*2 Bytes*	`unsigned int sekunden = 46547;`
int	*-32.768 bis 32.767*	*2 Bytes*	`int ticks = -325;`
long	-2^{31} *bis* $2^{31}-1$	*4 Bytes*	`long wert = -3457819;`
float	$-3.4 * 10^{38}$ *bis* $3.4 * 10^{38}$	*4 Bytes*	`float messwert = 27.5679;`
double	*siehe float*	*4 Bytes*	`double messwert = 27.5679;`
boolean	*true oder false*	*1 Byte*	`boolean flag = true;`
char	*-128 bis 127*	*1 Byte*	`char mw = 'm';`
String	*variabel*	*variabel*	`String name = "Erik Bartmann";`
Array	*variabel*	*variabel*	`int pinArray[] = {2, 3, 4, 5};`

Unser 2. Programm: zusätzlich einen digitalen Eingang einlesen und damit eine Leuchtdiode (LED) per Taster ein / ausschalten

Technikschule Esslingen; Kurs: Arduino

DIGITAL (PWM~)
UNO
ARDUINO
ON
ICSP
WWW.ARDUINO.CC – MADE IN ITALY
POWER
ANALOG IN
IOREF
RESET
3.3V
5V
GND
GND
Vin
A0
A1
A2
A3
A4
A5

LED_Taster_deu_Kurs2 §

```
/*
  LED und Taster
  Schaltet eine LED ein solange ein Taster gedrückt ist
 */
```

```
// Alternative (Programmierer sind schreibfaul !
void loop()
{   // gibt den Wert HIGH oder LOW aus, der eben eingelesen wurde
  digitalWrite(ledPin, digitalRead(tasterPin));
}
```

```
const int ledPin = 13;      // An Pin 13 wird die LED angeschlossen
                            // Name "ledPin" vergeben, damit die SW besser lesbar ist
const int tasterPin = 2;    // An Pin 2 wird der Taster angeschlossen
int tasterStatus;           // Variable zur Aufnahme des Tasterstatus

// die folgende "setup" routine muss einmal im Programm stehen und wird nach einem RESET einmal durchlaufen:
void setup()
{
  pinMode(ledPin, OUTPUT);      // mit der Funktion "pinMode" wird der Pin "led" als Ausgang definiert
  pinMode(tasterPin, INPUT);    // "TasterPin"  als Eingang definieren
}

// die "loop" routine muss auch einmal im Programm stehen, wird aber immer wieder von vorne durchlaufen
// eine sogenannte Endlos-Schleife:
void loop()
{
  tasterStatus = digitalRead(tasterPin);  // Fkt. liest den Zustand des digitalen Pins ein
  if (tasterStatus == HIGH)               // PULL-DOWN  Variante
    digitalWrite(ledPin, HIGH);           // schaltet die LED ein (HIGH ist ein 5V Spannungs-Pegel)
  else
    digitalWrite(ledPin, LOW);            // schaltet die LED aus (LOW ist ein 0V Spannungs-Pegel)
}
```

Jedes Programm:

1) enthält **Befehlsblöcke**

2) und **Programmschleifen**

3) und diverse **Kontrollstrukturen**

4) Diese werden immer in **geschweifte Klammern** gesetzt

{
}

„While“ Schleife

„For“ Schleife

„if - else“ Kontrollstruktur

„do - while“ Schleife

Eine **Verzweigung** besteht aus einer Bedingung und zwei Codeabschnitten. Zuerst wird die Bedingung ausgewertet: trifft sie zu wird der erste Codeabschnitt ausgeführt, anderenfalls wird der zweite Codeabschnitt ausgeführt

```
if (Bedingung)
{
    Anweisungsblock 1;     // beliebig viele Zeilen
}
else
{
    Anweisungsblock 2;     // beliebig viele Zeilen
}
```

```
while (Bedingung)
{
   Anweisungsblock;
   (beliebig viele Zeilen)
}
```

```
Do
{
  Anweisungsblock;
  (beliebig viele Zeilen)
}
while (Bedingung);
```

Die **For-Schleife** ist eine Kontrollstruktur, mit der man eine Gruppe von Anweisungen (Block) mit einer bestimmten Anzahl von Wiederholungen ausführen kann.

```
for (Zähler=Start-Bedingung;  Zähler = Bedingung;  Zähler ändern)
{
   Anweisungsblock;     // beliebig viele Zeilen
}
```

Unser 3. Programm: digitale Eingänge einlesen, digital Ausgänge ansteuern und damit eine Leuchtdiode (LED) per Taster ein / ausschalten und ein zweite LED blinken lassen

Taster Pin1 mit Digital-Eingang 2 verbinden
Anode über 220 od. 330 Ohm Wid. mit Digital-Eingang 10 verbinden
Anode über einen 220 oder 330 Ohm Widerstand mit Digital-Eingang 13 verbinden
Arduino
www.arduino.cc
Kathode mit Masse (Ground) verbinden
Steckboard mit Arduino Plus (5V) und Arduino Masse (Ground verbinden
Kathode mit Masse (Ground) verbinden
Taster Pin2 mit Plus (5V) verbinden
Taster Pin1 über 10 Kilo-Ohm Wid. mit Masse verbinden

Technikschule Esslingen; Kurs: Arduino

```
/*  LED und Taster
    Schaltet eine LED ein solange ein Taster gedrückt ist, zusätzlich blinkt eine LED  */

int ledPinBlink = 13;       // An Pin 13 wird die blinkende LED angeschlossen
                            // Name "ledPinBlink" vergeben, damit die SW besser lesbar ist
int ledPinTaster = 10;      // An Pin 10 wird die Taster LED angeschlossen
int tasterPin = 2;          // An Pin 2 wird der Taster angeschlossen
int tasterStatus;           // Variable zur Aufnahme des Tasterstatus
int interval = 1000;        // Intervallzeit (1 Sek aus, 1 Sek an == 2 Sekunden)
unsigned long alteZeit;     // Zeit-Variable
int ledStatus = LOW;        // Statusvariable für die Blink-LED

void setup()   // die "setup" Funktion muss einmal in jedem Programm stehen und wird beim Start einmal durchlaufen:
{
  pinMode(ledPinBlink, OUTPUT);       // mit der Funktion "pinMode" wird der Pin "led" als Ausgang definiert
  pinMode(ledPinTaster, OUTPUT);      // mit der Funktion "pinMode" wird der Pin "led" als Ausgang definiert
  pinMode(tasterPin, INPUT);          // "TasterPin"  als Eingang definieren
  alteZeit = millis();                    // jetzigen Zeitstempel merken
}  // fkt setup

// die "loop" routine muss auch einmal im Programm stehen, wird aber immer wieder von vorne durchlaufen
// eine sogenannte Endlos-Schleife:
void loop()
{
  // Blink-LED über Intervallsteuerung blinken lassen
  if((millis() - alteZeit) > interval)
  {
    alteZeit = millis();
    ledStatus = !ledStatus;                 // Toggeln ledStatus, "!" invertiert, LOW --> HIGH; HIGH --> LOW
    digitalWrite(ledPinBlink, ledStatus);   // LED je nach ledStatus ein- od. ausschalten, d.h. "Toggeln" Blink LED
  } // if
  // Taster Status abfragen
  tasterStatus = digitalRead(tasterPin);    // Fkt. liest den Zustand des digitalen Pins ein
  if (tasterStatus == HIGH)                 // Taster mit "Pull-Down" -->  wenn HIGH dann ist Taster gedrückt
    digitalWrite(ledPinTaster, HIGH);       // schaltet die LED ein (HIGH ist ein Spannungs-Pegel)
  else
    digitalWrite(ledPinTaster, LOW);        // schaltet die LED aus, der Spannungs-Pegel wird auf LOW gesetzt
} // fkt loop
```

```
/*  LED und Taster
    Tastendruck schaltet LED ein, nächster Tastendruck schaltet wieder aus --> TOGGELN, zusätzlich blinkt eine LED */

int ledPinBlink = 13;       // An Pin 13 wird die blinkende LED angeschlossen
                            // Name "ledPinBlink" vergeben, damit die SW besser lesbar ist
int ledPinTaster = 10;      // An Pin 10 wird die Taster LED angeschlossen
int tasterPin = 2;          // An Pin 2 wird der Taster angeschlossen
int tasterStatus;           // Variable zur Aufnahme des Tasterstatus
int tasterStatusOld;        // Variable zur Aufnahme des vorigen Tasterstatus
int interval = 1000;        // Intervallzeit (1 Sek aus, 1 Sek an == 2 Sekunden)
unsigned long alteZeit;     // Zeit-Variable
int ledStatusBlink = LOW;   // Statusvariable für die Blink-LED
int ledStatusTaster = LOW;  // Statusvariable für die Blink-LED

void setup()   // die "setup" Funktion muss einmal in jedem Programm stehen und wird beim Start einmal durchlaufen:
{
  pinMode(ledPinBlink, OUTPUT);       // mit der Funktion "pinMode" wird der Pin "led" als Ausgang definiert
  pinMode(ledPinTaster, OUTPUT);      // mit der Funktion "pinMode" wird der Pin "led" als Ausgang definiert
  pinMode(tasterPin, INPUT);          // "TasterPin"  als Eingang definieren
  alteZeit = millis();                    // jetzigen Zeitstempel merken
}  // fkt setup

// die "loop" routine muss auch einmal im Programm stehen, wird aber immer wieder von vorne durchlaufen
// eine sogenannte Endlos-Schleife:
void loop()
{
  // Blink-LED über Intervallsteuerung blinken lassen
  if((millis() - alteZeit) > interval)
  {
    alteZeit = millis();
    ledStatusBlink = !ledStatusBlink;            // Toggeln ledStatus, "!" invertiert, LOW --> HIGH; HIGH --> LOW
    digitalWrite(ledPinBlink, ledStatusBlink);   // LED je nach ledStatus ein- od. ausschalten, d.h. "Toggeln" LED
  } // if

  // Taster Status abfragen
  tasterStatus = digitalRead(tasterPin);     // Fkt. liest den Zustand des digitalen Pins ein
  if ((tasterStatus == HIGH) && (tasterStatusOld == LOW)) // Taster mit "Pull-Down" --> HIGH dann ist Taster gedrückt
  {                                                       // alter Zustand "LOW", aktueller Zustand "HIGH" -->  Flanke
    ledStatusTaster = !ledStatusTaster;                   // Toggeln Taster Status, "!" invert., LOW --> HIGH; HIGH --> LOW
    digitalWrite(ledPinTaster, ledStatusTaster);          // LED je nach Tasterstatus ein- od. ausschalten, d.h. "Toggeln" LED
  }
  tasterStatusOld = tasterStatus;                         // Aktueller tasterStatus speichern bevor wieder neu abgefragt wird
}  // fkt loop
```

Unsere bisherigen Funktionen:

Arduino besitzt eine Sammlung von Funktionen, eine sogenannte Bibliothek

void setup()	**muss einmal im Programm sein und wir nur einmal durchlaufen**
void loop()	**muss einmal im Programm sein und wird ständig durchlaufen**
pinMode(name, typ)	**legt fest ob ein Pin Ausgang oder Eingang sein soll (name = Name des Pins, typ = OUTPUT oder INPUT)**
digitalWrite(pin, status)	**gibt einen digitalen Wert an einem Pin aus (pin = Name des Pins, status = HIGH oder LOW)**
int digitalRead(pin)	**liest den int-Wert eines digitalen Pins ein (int = Rückgabewert, pin=Name des Pins)**
unsigned long millis()	**gibt die Zeit in Millisekunden zurück (nach Programmstart) (unsigned long = Rückgabewert)**
delay(wert)	**wartet eine vorgegebene Zeit in Millisekunden (wert = Zeit in Millisekunden)**

4. Programm: digitale Ausgänge ansteuern & ein variables LED Lauflicht ansteuern

Kathode mit Widerstand und diesen mit Masse (Ground) verbinden

Seite mit Schrift = Anode
Seite ohne Schrift = Kathode

Hier der Punkt
Seite mit Schrift
RESET
ICSP2
DIGITAL (PWM=~)
UNO
Arduino
ON
ICSP
POWER
ANALOG IN

int analogRead(pin) liest den int-Wert eines analogen Pins ein
(int = Rückgabewert, pin=Name des Pins)

int random(wert) liefert einen Zufallswert zurück
(int = Rückgabewert, wert=oberster Wert (exklusiv))

int random(wert1, wert2) liefert einen Zufallswert zurück
(int = Rückgabewert, wert1=unterster Wert,
wert2=oberster Wert (exklusiv))

int map(wert1, wert2, wert3 , wert4 , wert5) liefert einen Wert zurück, der in einem begrenzten Intervall liegt
wert1 = Eingabewert,
wert2 = min. Wert Eingangsintervall, wert3 = max. Wert Eing.Intervall,
wert4 = min. Wert Ausgangsintervall, wert5 = max. Wert Ausg.Intervall)

LED_Lauflicht_deu_Kurs4 §

```
/*
  LED Lauflicht
  Schaltet eine Reihe von LEDs nach einem bestimmten Muster ein und aus

  Im Schaltkreis sind 10 LEDs mit einem Widerstand gegen Masse geschaltet
 */

// diese Constante ändert sich nicht:
const int ledAnzahl = 10;      // Anzahl der LEDs im bar graph
const int demoDauer = 4;      // Anzahl der Demos, bevor gewechselt wird

int ledPins[] = {2, 3, 4, 5, 6, 7,8,9,10,11};     // ein array mit Pin-Nummmern für LED Anschluss
int wartezeit = 200;                              // Pause zwischen den Wechseln in ms

void setup()
{
  // mit dieser for Schleife werden alle LED pins aus dem Array als Ausgang gesetzt
  for (int i = 0; i < ledAnzahl; i++)
  {
    pinMode(ledPins[i], OUTPUT);                  // alle Pins des Arrays als Ausgang
  }
}

void loop()
{
  for (int i = 0; i < demoDauer; i++)             // Jede Programmvariante mehrfach durchlaufen
  {
    // Schleife, die das LED array durchläuft:
    for (int i = 0; i < ledAnzahl; i++)           // Lauflicht vorwärts
    {
      digitalWrite(ledPins[i], HIGH);             // Array Element "i" auf HIGH Pegel
      delay(wartezeit);                           // kurz leuchten lassen, sonst sieht man nichts
      digitalWrite(ledPins[i], LOW);              // Array Element "i" auf LOW Pegel
    }   // for
  }     // for

  for (int i = 0; i < demoDauer; i++)             // Jede Programmvariante mehrfach durchlaufen
  {
    for (int i = ledAnzahl-1; i > -1; i--)        // Lauflicht rückwärts
    {
      digitalWrite(ledPins[i], HIGH);             // Array Element "i" auf HIGH Pegel
      delay(wartezeit);
      digitalWrite(ledPins[i], LOW);              // Array Element "i" auf LOW Pegel
    }  // for
  }    // for
```

```
  for (int i = 0; i < demoDauer; i++)             // Jede Programmvariante mehrfach durchlaufen
  {
    for (int i = 0; i < ledAnzahl; i++)           // Lauflicht vorwärts und rückwärts
    {
      digitalWrite(ledPins[i], HIGH);             // Array Elemente von vorne auf HIGH Pegel
      digitalWrite(ledPins[(ledAnzahl-1)-i], HIGH); // Array Element von hinten auf HIGH Pegel
      delay(wartezeit);
      digitalWrite(ledPins[i], LOW);              // Array Element von vorne auf LOW Pegel
      digitalWrite(ledPins[(ledAnzahl-1)-i], LOW); // Array Element von hinten auf LOW Pegel
    }  // for
  }    // for

  for (int i = 0; i < (8*demoDauer); i++)
  {                                               // Lauflicht mit Zufallszahl
    int zufall;
    zufall = random(ledAnzahl);                   // liefert Zufallszahl zwischen 0 & < "ledCount"
    digitalWrite(ledPins[zufall], HIGH);          // Array Element "i" auf HIGH Pegel
    delay(wartezeit);
    digitalWrite(ledPins[zufall], LOW);           // Array Element "i" auf LOW Pegel
  }  // for
  delay(1000);          // kurze Pause und von vorne
}    // loop
```

Unser 5. Programm: analoge Eingänge einlesen, digitale Ausgänge ansteuern und damit ein LED Lauflicht ansteuern

Schaltbild eines Potentiometers:

Spannungsteiler aus zwei in Reihe geschalteten ohmschen Widerständen

Kathode mit Widerstand und diesen mit Masse (Ground) verbinden

ARDUINO

Beispiel:

wert = map(wert, 0, 1023, 0, 179)

Wandelt den Poti Wert zwischen 0 und 1023 in einen Wert zwischen 0 und 179, der für den Servo geeignet ist

LED_BarGraph_deu_Kurs5

```
/*  LED bar graph
  Schaltet eine Reihe von LEDs abhängig von der Stellung (Wert) eines
  Potentiometers ein und aus. -->  Balkenanzeige
  Der Schaltkreis:  LEDs von Pin 2 bis Pin 11 gegen Masse   */

// diese Constanten ändern sich nicht:
const int analogPin = A0;      // der Pin mit dem das Potentiometer verbunden ist
const int ledAnzahl = 10;      // Anzahl der LEDs im bar graph

int ledPins[] = {
  2, 3, 4, 5, 6, 7,8,9,10,11 };   // ein array mit Pin-Nummmern für LEDs Anschluss

void setup()
{
  // mit dieser for Schleife werden alle LED pins aus dem Array als Ausgang gesetzt
  for (int i = 0; i < ledAnzahl; i++)
  {
    pinMode(ledPins[i], OUTPUT);
  }
}

void loop()
{
  int sensorWert = analogRead(analogPin);                  // Wert des Potentiometer lesen
  int ledLevel = map(sensorWert, 0, 1023, 0, ledAnzahl);   // map "Wert" im Bereich von 0 bis Anzahl LEDs
  // Schleife durchläuft das LED array:
  for (int i = 0; i < ledAnzahl; i++)                      // Schleife durchläuft das LED array:
  {
    if (i < ledLevel)                        // if the array element's index is less than ledLevel,
    {                                        // turn the pin for this element on:
      digitalWrite(ledPins[i], HIGH);
    }     // if
    else                                     // turn off all pins higher than the ledLevel:
    {
      digitalWrite(ledPins[i], LOW);
    }     // else
  }       // for
}         // loop
```

int analogRead(pin) liest Wert eines analogen Pins ein
(int = Rückgabewert, pin=Pin-Name)

6. Progr.: Ein Lichtsensor „LDR“ steuert das LED Lauflicht

Cadmiumsulfid-Fotowiderstand; die fotoempfindliche Widerstandsschicht befindet sich zwischen den kammartigen Kontaktflächen

LDR: **Viel Licht → kleinerer Widerstand**
Wenig Licht → größerer Widerstand

Arduino
Digital I/O
Analog IN
VCC
GND
13
12
11
10
9
8
7
6
5
4
3
2
1
0
PWM
5 4 3 2 1 0
330
R
LED
LDR
10K
R
GND

LED_BarGraph_LichtSensor_KursX

```
/*   LED bar graph
  Schaltet eine Reihe von LEDs abhängig von der Helligkeit (Wert) eines
  Lichtsensors ein und aus. -->  Balkenanzeige   */

int ledAnzahl = 10;                 // Anzahl der LEDs im bar graph
int analogPin = 0;                  // der Pin mit dem das Potentiometer verbunden ist
int analogLichtWert = 0;
int ledPins[] = {2, 3, 4, 5, 6, 7,8,9,10,11 };  // Array mit Pin-Nummmern für LEDs

void setup()
{       // mit dieser for Schleife werden alle LED pins aus dem Array als Ausgang gesetzt
  for (int i = 0; i < ledAnzahl; i++)
  {
    pinMode(ledPins[i], OUTPUT);
  }
}

void loop()
{
  analogLichtWert = analogRead(analogPin);                  // Wert des Lichtsensors lesen
  steuereLEDs(analogLichtWert);
}       // loop
```

```
// Funktion zum Ansteuern der LED's  KURZFORM
void steuereLEDs2(int wert)
{
  int ledLevel = map(wert, 0, 1023, 0, (ledAnzahl-1));
  for (int i = 0; i < ledAnzahl; i++)
    digitalWrite(ledPins[i], (ledLevel >= i)?HIGH:LOW);
}          // fkt
```

```
void steuereLEDs(int wert)                                  // Funktion zum Ansteuern der LED's
{
  int ledLevel = map(wert, 0, 1023, 0, (ledAnzahl-1));      // map "wert" in einen Bereich von 0 bis Anzahl LEDs
  for (int i = 0; i < ledAnzahl; i++)                       // Schleife durchläuft das LED array:
  {
    if (i < ledLevel)                       // if the array element's index is less than ledLevel,
    {                                       // turn the pin for this element on:
      digitalWrite(ledPins[i], LOW);
    }     // if
    else                                    // turn off all pins higher than the ledLevel:
    {
      digitalWrite(ledPins[i], HIGH);
    }     // else
  }       // for
}         // fkt
```

7. Progr.: Arduino lernt das „reden“. Ein Lichtsensor „LDR“ steuert das LED Lauflicht und sendet seinen „Licht-Wert“ an den PC über die serielle Schnittstelle

Kommunikation mit dem PC:

Arduino besitzt Funktionen, um Daten mit dem PC auszutauschen.
Es können Daten am PC angezeigt werden oder Daten unf Befehle vom PC an den Arduino gesendet werden

Serial.begin(int)	**initialisiert die USB Schnittstelle mit einer Baudrate (int = Größer der Baudrate)**
Serial.print(wert, format)	**sendet Zeichenkette per USB an PC ohne Zeilenvorschub (wert = Name des Zeichenkette, format = Zahlenbasis)**
Serial.println(string)	**sendet Zeichenkette per USB an PC mit Zeilenvorschub (string = Name des Zeichenkette)**
boolean Serial.available()	**prüft ob am USB Daten vom PC abgeholt werden können**
string Serial.read()	**liest Daten vom PC an der USB**

LED_BarGraph_LichtSensor_Seriell_KursX §

```
/*  LED bar graph
  Schaltet eine Reihe von LEDs abhängig von der Helligkeit (Wert) eines
  Lichtsensors ein und aus. -->  Balkenanzeige  und zeigt Wert im seriellen Monitor an  */

int ledAnzahl = 10;            // Anzahl der LEDs im bar graph
int analogPin = A0;             // Pin mit dem Potentiometer verbunden ist
int analogLichtWert = 0;
int interval = 2000;           // Intervallzeit (2 Sekunden)
int zeilenZaehler = 1;
unsigned long prev;            // Zeit-Variable

int ledPins[] = {2,3,4,5,6,7,8,9,10,11 };   // ein array mit Pin-Nummmern für LEDs Anschluss

void setup()
{
  // mit dieser for Schleife werden alle LED pins aus dem Array als Ausgang gesetzt
  for (int i = 0; i < ledAnzahl; i++)
  {
    pinMode(ledPins[i], OUTPUT);     // 10 ledPins als Ausgänge einstellen
    Serial.begin(9600);              // serielle Schnittstelle konfigurieren
    prev = millis();                 // jetzigen Zeitstempel merken
  }  // for
}    // setup()

void loop()
{
  analogLichtWert = analogRead(analogPin);  // Wert des Lichtsensors lesen
  steuereLEDs(analogLichtWert);             // Wert nur alle "interval" ms ausgeben
  if((millis() - prev) > interval)
  {
    prev = millis();                        // jetzigen Zeitstempel wieder merken (quasi reset)
    Serial.print("Zeile  ");                // Print ohne new line (LN)
    Serial.print(zeilenZaehler);
    Serial.print("  : ");
    zeilenZaehler++;
    Serial.println(analogLichtWert);        // zum Schluß der Zeile "print" mit new line (LN)
  }  // if
}    // loop()
```

Serieller Monitor einschalten:

8. Progr.: Arduino lernt das „kommunizieren“. Der ASCII Code
Wir senden Text zum Arduino und wieder zurück zum PC

Dez	Hex	Zch	Dez	Hex	Zch	Dez	Hex	Zch	Dez	Hex	Zch	Dez	Hex	Zch	Dez	Hex	Zch
Genormter ASCII-Code (7-Bit-Code)																	
0	0		22	16	■	44	2C	,	65	41	A	86	56	V	107	6B	k
1	1	☺	23	17	↨	45	2D	-	66	42	B	87	57	W	108	6C	l
2	2	☻	24	18	↑	46	2E	.	67	43	C	88	58	X	109	6D	m
3	3	♥	25	19	↓	47	2F	/	68	44	D	89	59	Y	110	6E	n
4	4	♦	26	1A	→	48	30	0	69	45	E	90	5A	Z	111	6F	o
5	5	♣	27	1B	←	49	31	1	70	46	F	91	5B	[	112	70	p
6	6	♠	28	1C	∟	50	32	2	71	47	G	92	5C	\	113	71	q
7	7	•	29	1D	↔	51	33	3	72	48	H	93	5D	]	114	72	r
8	8	◘	30	1E	▲	52	34	4	73	49	I	94	5E	^	115	73	s
9	9	○	31	1F	▼	53	35	5	74	4A	J	95	5F	_	116	74	t
10	A	◙	32	20		54	36	6	75	4B	K	96	60	`	117	75	u
11	B	♂	33	21	!	55	37	7	76	4C	L	97	61	a	118	76	v
12	C	♀	34	22	“	56	38	8	77	4D	M	98	62	b	119	77	w
13	D	♪	35	23	#	57	39	9	78	4E	N	99	63	c	120	78	x
14	E	♫	36	24	$	58	3A	:	79	4F	O	100	64	d	121	79	y
15	F	☼	37	25	%	59	3B	;	80	50	P	101	65	e	122	7A	z
16	10	►	38	26	&	60	3C	<	81	51	Q	102	66	f	123	7B	{
17	11	◄	39	27	’	61	3D	=	82	52	R	103	67	g	124	7C	|
18	12	↕	40	28	(	62	3E	>	83	53	S	104	68	h	125	7D	}
19	13	‼	41	29	)	63	3F	?	84	54	T	105	69	i	126	7E	~
20	14	¶	42	2A	*	64	40	@	85	55	U	106	6A	j	127	7F	Δ
21	15	§	43	2B	+												

8. Progr.: Arduino lernt das „kommunizieren“. Der ASCII Code

Wir senden Text zum Arduino und wieder zurück zum PC

Dez	Hex	Zch	Dez	Hex	Zch	Dez	Hex	Zch	Dez	Hex	Zch	Dez	Hex	Zch	Dez	Hex	Zch
Erweiterungen im Unicode																	
128	80	€	150	96	–	172	AC	¬	193	C1	Á	214	D6	Ö	235	EB	ë
129	81	□	151	97	—	173	AD		194	C2	Â	215	D7	x	236	EC	ì
130	82	‚	152	98	˜	174	AE	®	195	C3	Ã	216	D8	Ø	237	ED	í
131	83	ƒ	153	99	™	175	AF	¯	196	C4	Ä	217	D9	Ù	238	EE	î
132	84	„	154	9A	š	176	B0	°	197	C5	Å	218	DA	Ú	239	EF	ï
133	85	…	155	9B	›	177	B1	±	198	C6	Æ	219	DB	Û	240	F0	ð
134	86	†	156	9C	œ	178	B2	2	199	C7	Ç	220	DC	Ü	241	F1	ñ
135	87	‡	157	9D	□	179	B3	3	200	C8	È	221	DD	Ý	242	F2	ò
136	88	ˆ	158	9E	ž	180	B4	´	201	C9	É	222	DE	þ	243	F3	ó
137	89	‰	159	9F	Ÿ	181	B5	µ	202	CA	Ê	223	DF	ß	244	F4	ô
138	8A	Š	160	A0		182	B6	¶	203	CB	Ë	224	E0	à	245	F5	õ
139	8B	‹	161	A1	¡	183	B7	·	204	CC	Ì	225	E1	á	246	F6	ö
140	8C	Œ	162	A2	¢	184	B8	¸	205	CD	Í	226	E2	â	247	F7	÷
141	8D	□	163	A3	£	185	B9	1	206	CE	Î	227	E3	ã	248	F8	ø
142	8E	Ž	164	A4	¤	186	BA	0	207	CF	Ï	228	E4	ä	249	F9	ù
143	8F	□	165	A5	¥	187	BB	»	208	D0	Ð	229	E5	å	250	FA	ú
144	90	□	166	A6	¦	188	BC	¼	209	D1	Ñ	230	E6	æ	251	FB	û
145	91	‘	167	A7	§	189	BD	½	210	D2	Ò	231	E7	ç	252	FC	ü
146	92	’	168	A8	¨	190	BE	¾	211	D3	Ó	232	E8	è	253	FD	ý
147	93	“	169	A9	©	191	BF	¿	212	D4	Ô	233	E9	é	254	FE	þ
148	94	”	170	AA	a	192	C0	À	213	D5	Õ	234	EA	ê	255	FF	ÿ
149	95	*	171	AB	«												

Einige der „ASCII Zeichen“ bzw. „Zahlen“ werden als Steuerzeichen für Drucker und Bildschirm verwendet und führen zur angegebenen Funktion:

ASCII-Steuerzeichen (Beispiele)

Dez	Befehl	Bedeutung	Dez	Befehl	Bedeutung
2	STX	Start of Text = Textanfang	13	CR	Carriage Return = Wagenrücklauf
3	ETX	End of Text = Textende	17	DC1	Device Control = Gerätesteuerzeichen
7	BEL	engl. Bell = Klingel			(weitere 18, 19, 20 = DC2, DC3, DC4)
10	LF	Line Feed = Zeilenvorschub	26	SUB	Substitution = Zeichen ersetzen
12	FF	Form Feed = Formularvorschub	27	ESC	Escape = Umschaltung (Taste)

Der Unicode als 16-Bit-Code wurde als internationaler Standard definiert, um die Zeichen aller europäischen Sprachen sowie insbesondere die Zeichen im Arabischen, Chinesischen, Japanischen oder Koreanischen abzubilden. In den ersten 128 Plätzen ist der ASCII-Code (American Standard Code for Information Interchange) untergebracht. Von den möglichen 65536 Zeichen des Unicode sind derzeit etwa 40000 Zeichen vergeben. Die ersten 32 Zeichen sind meist Steuerzeichen, die anderen Zeichen sind auch Bestandteil des ANSI-Codes (American National Standards Institut).

Selbst 255 Zeichen reichen weltweit nicht, daher gibt es den UNICODE.
Dort besteht aber jedes Zeichen aus 2 Byte, Texte sind im Speicher also immer genau doppelt so groß.
Dafür können 65 536 Zeichen dargestellt werden!

```
switch (zahl)
{
 case 0:
 {
   v = 1;
  break;     // 'break' ist hier nötig, damit nicht auch die..
 }
 case 1:     // ..folgenden Anweisungen ausgeführt werden
 {
   v = 2;
   break;
 }
 case 2:
 {
   v = 5;
   break;
 }
 default:
 {
   v = 10;     // der else-Teil
 }
}
```

Hexadezimalziffern, binär und dezimal:

Hex.	Dualsystem				Dez.
0	0	0	0	0	00
1	0	0	0	1	01
2	0	0	1	0	02
3	0	0	1	1	03
4	0	1	0	0	04
5	0	1	0	1	05
6	0	1	1	0	06
7	0	1	1	1	07
8	1	0	0	0	08
9	1	0	0	1	09
A	1	0	1	0	10
B	1	0	1	1	11
C	1	1	0	0	12
D	1	1	0	1	13
E	1	1	1	0	14
F	1	1	1	1	15

Seriell_Read_Write_KursX

```
/*
Daten von der USB lesen und wieder an den PC senden
 */
int ledPin = 13;
int empfangenesByte = 0;        // Speichert die empfangenen Daten
int empfangeneZahl = -1;        // default == -1 == keine Ziffer

void setup()
{
    pinMode(ledPin, OUTPUT);  // Fkt. "pinMode" definiert "ledPin" als Ausgang
    Serial.begin(9600);       // öffnet die serielle Schnittstelle
}                             // und legt die Baudrate auf 9600 bps

void loop()
{                           // sendet daten nur wenn welche empfangen wurden
  if (Serial.available() > 0)                 // gibt es ein Zeichen
  {   // read the incoming byte:
    empfangenesByte = Serial.read();          // Datenbyte lesen
    Serial.print("Arduino hat empfangen: ");  // Text ausgeben
    Serial.print(empfangenesByte);            // Datenbyte wieder ausgeben
    Serial.print(" = ");
    Serial.println((char)empfangenesByte);    // nach ASCII wandeln
```

cast: wandelt Datentypen um

charVar = (char) intVar

Wandelt eine int Wert in einen char Wert

intVar = (int) byteVar

Wandelt einen byte Wert in einen int Wert

Seriell_Read_Write_KursX §

```
    // prüfen ob Zeichen eine Zahl ist
    if ((empfangenesByte < 58) && (empfangenesByte > 47))
      empfZahl = empfangenesByte - '0';  // Zahl ist "ASCII-Wert" - "ASCII-0"
    else
      empfZahl = -1;                     // "-1" == keine Ziffer
    switch(empfZahl) {                   // mehrfache Kontrollstruktur
      case 1:                            //  einmal blinken
        for (int i=0; i<empfZahl; i++)
        {
          digitalWrite(ledPin, HIGH);    // schaltet die LED ein
          delay(1500);
          digitalWrite(ledPin, LOW);     // schaltet die LED ein
          delay(200);
        }
        break;                           // Sprung zum SWITCH Ende
      case 2:                            // 2 mal schneller blinken
        for (int i=0; i<empfZahl; i++)
        {
          digitalWrite(ledPin, HIGH);    // schaltet die LED ein
          delay(1000);
          digitalWrite(ledPin, LOW);     // schaltet die LED ein
          delay(200);
        }
        break;                           // Sprung zum SWITCH Ende
      case 3:                            // 3 mal noch schneller blinken
        for (int i=0; i<empfZahl; i++)
        {
          digitalWrite(ledPin, HIGH);    // schaltet die LED ein
          delay(500);
          digitalWrite(ledPin, LOW);     // schaltet die LED ein
          delay(200);
        }
        break;                           // Sprung zum SWITCH Ende
      default:                           // bei allen anderen Zeichen
         for (int i=0; i<10; i++)
        {                                //  10 mal ganz schnell blinken
          digitalWrite(ledPin, HIGH);    // schaltet die LED ein
          delay(100);
          digitalWrite(ledPin, LOW);     // schaltet die LED ein
          delay(50);
        }  // default
    }      // switch
  }        // if
  delay(2000);
}          // loop
```

Ansteuerung eines SERVO Motors

Servo mit Vorgelegegetriebe. Das unterste Teilfoto zeigt das Innenleben des schwarzen Kunststoffgehäuses.

Positionsregelung **Ansteuerung eines SERVO Motors**

Ein Servo ist ein winkelgeregelter bzw. positionsgeregelter Aktuator. Das bedeutet, dass er anhand seines Eingangssignals eine bestimmte Winkelposition anfährt und diese hält. Der Servo stoppt die Drehung und hält die Position selbstständig

Servos werden über eine Pulsweitenmodulation (PWM) angesteuert. Über die Breite der Pulse wird der Winkel, auf den der Servoarm gestellt werden soll, gesteuert.

Gängig ist ein 50-Hz-Signal (20 ms Periodenlänge), welches zwischen 1 ms (linker Anschlag) und 2 ms (rechter Anschlag) auf High-Pegel und den Rest der Periodenlänge auf Low-Pegel ist.

Umbau eines Servos in einen normalen Getriebemotor
http://www.electronicsplanet.ch/Roboter/Servo/hacking/overview.htm

5V
0V
t on
t periode

20ms
1ms
0⁰

5V
0V
t on
t periode

20ms
1,5ms
90⁰

5V
0V
t on
t periode

20ms
2ms
180⁰

Unser 09. Programm: **Ansteuerung eines SERVO Motors**

Ansteuerung eines SERVO Motors

Befehl | Pin | PWM

```
analogWrite(5, 64);
```

PWM-Wert	Tastgrad
0	0%
64	25%
128	50%
191	75%
255	100%

$$Tastgrad = \frac{t}{T}$$

$$PWM - Wert = \frac{gewünschter\ Tastgrad\ [\%]\ \cdot 255}{100\%}$$

Servo_Motor_KursX

Unser 09. Programm: Ansteuerung eines SERVO Motors

```
/*
Steuert einen SERVO Motor an. Dazu wird das Servo Objekt verwendet
Die Position des Servo wird mit einem Poti eingestellt und der Wert seriell
angezeigt   */

#include <Servo.h>                        // Einbinden der SERVO Bibliothek

Servo meinServo;                          // erzeugt ein Servo object  "meinServo" um den SERVO-Motor zu steuern

int potiPin = 0;                          // analoger Pin für das Potentiometer
int wert;                                 // variable um den Poti Wert vom Analog Pin zu lesen & speichern
int oldWert = 0;                          // variable um den Poti Wert vom Analog Pin zu lesen & speichern

void setup()
{
  meinServo.attach(9);                    // an PWM Pin 9 wird der Servo angeschlossen
  Serial.begin(9600);                     // Serielle Schnittstelle mit 9600 baud initialisieren
}

 void loop()
{
  wert = analogRead(potiPin);            // liest den Poti Wert (Wert zwischen 0 und 1023)
  wert = map(wert, 0, 1023, 0, 179);     // scaliert den Wert auf einen Servo tauglichen Wert zwischen 0 und 180)
  if (wert != oldWert)
  {
    Serial.println(wert);                 // Servo Wert seriell anzeigen
    oldWert = wert;
  }
  meinServo.write(wert);                 // übergibt den scalierten Wert und setzt damit die SERVO Position
  delay(20);                             // wartet eine Weile bis der SERVO eingestellt ist
}
```

Ansteuerung eines Standard Motors

Arduino kann nur kleine Ströme pro DigitalOut → kleine Motoren.
1. Bsp. zeigt einfachster Fall, wie ein Motor angeschlossen wird.
Die **Diode** schützt das Arduino-Board vor **Induktionsströmen**.
Der Motor kann sich nur in eine Richtung und mit einer Geschwindigkeit drehen.
Zum Testen lässt sich der Blink-Code verwenden (**Arduino>File>Examples>Digital>Blink**).

Zum Ändern der Drehrichtung muss Motor umgepolt werden

Ansteuerung eines Standard Motors mit Drehrichtungsumkehr und Geschwindigkeitssteuerung

2. Bsp: Motor an **2 Outputs** angeschlossen.
Damit lässt sich Motor in beide Richtungen drehen.
Geschwindigkeit lässt sich über **PWM-Kanäle** regulieren.
Diese Schaltung benötigt **vier Dioden**.

Die Tabelle zeigt mögliche Schaltzustände:
(LOW bedeutet GND, HIGH bedeutet 5V+)

Digital2	Digital3	Motor
LOW	LOW	AUS (Stop)
LOW	HIGH	LINKS
HIGH	LOW	RECHTS
HIGH	HIGH	AUS (Stop)
LOW	analogWrite(Pin, 122);	Langsam Links
HIGH	analogWrite(Pin, 122);	Langsam Rechts

Motor_PWM_divModi

```
/* SW steuert einen Motor in beide Richtungen über zwei Kanäle
einer davon als PWM, damit wird auch Geschwindigkeit geregelt */

int motorPin1=2;
int motorPin2=3;                    // PWM

void setup(){
  pinMode(motorPin1, OUTPUT);    // Pin als Ausgang
  pinMode(motorPin2, OUTPUT);    // Pin als Ausgang
}

void motorStop(){
  digitalWrite(motorPin1,LOW);  // Beide Pns auf LOW
  digitalWrite(motorPin2,LOW);  // dann ist Motor aus
  delay(1000);
}

void loop(){
  motorStop();                     // Motor Stop
  digitalWrite(motorPin1, HIGH);  // Motor Vor
  digitalWrite(motorPin2, LOW);
  delay(1000);
  motorStop();                     // Motor Stop
  digitalWrite(motorPin1, LOW);   // Motor Rueck
  digitalWrite(motorPin2, HIGH);
  delay(1000);
  motorStop();                     // Motor Stop
  digitalWrite(motorPin1, LOW);   // Motor Vor, langsam zu schnell
  for (int i=0; i<255; i++){
    analogWrite(motorPin2, i);
    delay(50);
  }
  motorStop();                     // Motor Stop
  digitalWrite(motorPin1, HIGH);  // Motor Rueck, langsam zu schnell
  for (int i=255; i>0; i--){
    analogWrite(motorPin2, i);
    delay(50);
  }
  motorStop();                     // Motor Stop
  digitalWrite(motorPin1, LOW);   // Motor Vor, schnell zu langsam
  for (int i=255; i>0; i--){
    analogWrite(motorPin2, i);
    delay(50);
  }
  motorStop();                     // Motor Stop
  digitalWrite(motorPin1, HIGH);  // Motor Rueck, schnell zu langsam
  for (int i=0; i<255; i++){
    analogWrite(motorPin2, i);
    delay(50);
  }
  motorStop();                     // Motor Stop
}
```

Ansteuerung eines High Power Motors mit Relais
→ keine Geschwindigkeitssteuerung möglich

Ansteuerung eines Motors mit PowerTransistor oder FET und Geschwindigkeitssteuerung

Will man die Geschwindigkeit eines High Power Motors regeln, verwendet man die Verstärkung eines PowerTransistors oder FET und externes Netzteil in Kombination mit einem PWM Signal.

Wichtig ist GND des Netzteils mit GND des Arduinos zu verbinden.

Als Transistor kann man z.B. den BD 911 verwenden.

Ansteuerung eines Standard Motors

Die Umpolung erfolgt mittels einer H-Brücke

Schalter S1 und S4 sind geschlossen.
Der Motor dreht rechtsherum.

Schalter S2 und S3 sind geschlossen.
Der Motor dreht linksherum.

Die verschiedenen Möglichkeiten zur Ansteuerung eines Standard Motors mit einem Motor Treiber Baustein L293

Ansteuerung zweier Standard Motoren:
nur Vorwärts und Rückwärts jeweils mit 100% (Full speed)

H-Bridge L293D hält kurzzeitig Strom von 1,2 A und dauerhaft 600 mA pro Kanal aus. Sie hat integrierte Dioden, die Induktionsströme abfangen, die beim Nachdrehen des Motors entstehen.

Ansteuerung zweier Standard Motoren:

Vorwärts und Rückwärts und Steuerung der Geschwindigkeit mittels PWM Signal

Motor_H_Bridge_PWM_divModi

```
/* 2 Motoren mit H-Brücke ansteuern */

int motor1_A=11;
int motor1_B=10;
int motor1_Speed=9;
int motor2_A=5;
int motor2_B=4;
int motor2_Speed=3;

void setup(){
  pinMode(motor1_A,OUTPUT);
  pinMode(motor1_B,OUTPUT);
  pinMode(motor2_A,OUTPUT);
  pinMode(motor2_B,OUTPUT);
}

void loop(){
  // motor1
  for (int i=0; i>256; i+=5){
    digitalWrite(motor1_A,HIGH);      // A = HIGH and B = LOW means the motor will turn right
    digitalWrite(motor1_B,LOW);
    analogWrite(motor1_Speed,i);      // speed counts from 0 to 255
    delay(20);
  }
  for (int i=255; i>0; i-=5){
    digitalWrite(motor1_A,HIGH);      // A = HIGH and B = LOW means the motor will turn right
    digitalWrite(motor1_B,LOW);
    analogWrite(motor1_Speed,i);      // speed counts from 0 to 255
    delay(20);
  }
  // motor2
  for (int i=0; i<256; i+=5){
    digitalWrite(motor2_A,HIGH);      // A = HIGH and B = LOW means the motor will turn right
    digitalWrite(motor2_B,LOW);
    analogWrite(motor2_Speed,i);      // speed counts from 0 to 255
    delay(20);
  }
  for (int i=255; i>0; i-=5){
    digitalWrite(motor2_A,HIGH);      // A = HIGH and B = LOW means the motor will turn right
    digitalWrite(motor2_B,LOW);
    analogWrite(motor2_Speed,i);      // speed counts from 0 to 255
    delay(20);
  }
```

```
  // turn vice versa
  // motor1
  for (int i=0; i<256; i+=5){
    digitalWrite(motor1_A,LOW); // A = LOW and B = HIGH means the motor will turn left
    digitalWrite(motor1_B,HIGH);
    analogWrite(motor1_Speed,i); // speed counts from 0 to 255
    delay(20);
  }
  for (int i=255; i>0; i-=5){
    digitalWrite(motor1_A,LOW); // A = LOW and B = HIGH means the motor will turn left
    digitalWrite(motor1_B,HIGH);
    analogWrite(motor1_Speed,i); // speed counts from 0 to 255
    delay(20);
  }
  // motor2
  for (int i=0; i<256; i+=5){
    digitalWrite(motor2_A,LOW); // A = LOW and B = HIGH means the motor will turn left
    digitalWrite(motor2_B,HIGH);
    analogWrite(motor2_Speed,i); // speed counts from 0 to 255
    delay(20);
  }
  for (int i=255; i>0; i-=5){
    digitalWrite(motor2_A,LOW); // A = LOW and B = HIGH means the motor will turn left
    digitalWrite(motor2_B,HIGH);
    analogWrite(motor2_Speed,i); // speed counts from 0 to 255
    delay(20);
  }
}
```

Zwei_Motoren_ohneBib § | Zwei_Motoren_Fkt

```
/*   Zwei Motoren per Befehl über Schnittstelle ansteuern und auch einen Wert für Power senden
V = Vorwärts; Z = Zurück; L = Links; R = Rechts; S = Stop
plus Zahl zwischen 0 un 255  z.B.  V120   */

// Bewegungsrichtungen + Pause
enum Motion          // Liste mit Konstanten
{
  FORWARD = 1,       // nit "1" beginnend
  BACKWARD,
  RIGHT,
  LEFT,
  PAUSE
};

int motor1_Pin1 = 2;           // Motorpins
int motor1_Pin2 = 3;
int motor2_Pin1 = 4;
int motor2_Pin2 = 5;
int enableM1Pin  = 9;          // Motor1 Enable Pin
int enableM2Pin  = 10;         // Motor2 Enable Pin

byte empfByte = 0;             // Variable für empfangenes BYTE
int moveModus = PAUSE;         // speichert die Richtung bzw. Modus
int oldMoveModus = PAUSE;      // speichert alten Modus
int power = 0;                 // aktuelleGeschwindigkeit
int oldPower = 0;              // alte Geschwindigkeit

void setup()
{
  MotorenSet();          // Ausgänge der Motoren definieren
  Serial.begin(9600);    // öffnet die serielle Schnittstelle und legt die Baudrate auf 9600 bps
}
```

Ansteuerung zweier Standard Motoren

```
void loop()
{
  while (Serial.available() > 0)      // if there's any serial available, read it:
  {
    empfByte = Serial.read();
    switch(empfByte)
    {                          // mehrfache Kontrollstruktur
      case 'V':                // Vorwärts
        moveModus = FORWARD;
        break;                 // Sprung zum SWITCH Ende
      case 'Z':                // rückwärts
        moveModus = BACKWARD;
        break;
      case 'L':                // links
        moveModus = LEFT;
        break;
      case 'R':                // rechts
        moveModus = RIGHT;
        break;
      case 'S':                // Stop
        moveModus = PAUSE;
        power = 0;
        break;
      default:                 // alle anderen Tasten STOP
        moveModus = PAUSE;
        break;
    }       // switch
    // look for the next valid integer in the incoming serial stream:
    power = Serial.parseInt();              // Fkt liest einen ZahlenString und liefert Wert zurück
    power = constrain(power, 0, 255);       // begrenzt "power" zwischen 0 und 255 (PWM)
    Serial.print("Modus:  ");
    Serial.print(moveModus);
    Serial.print("     Power:  ");
    Serial.println(power);
  }  //while
  if ((moveModus != oldMoveModus) || (power != oldPower))
  {   // nur wenn neuer Modus
    MotorenMove(moveModus, power);    // dann neuer Move Befehl
    oldMoveModus = moveModus;         // neuer Zustand speichern
    oldPower = power;                 // damit wird Fkt nur aufgerufen bei Änderung
  }   // if
}    // loop
```

Ansteuerung zweier Standard Motoren

Zwei_Motoren_ohneBib | Zwei_Motoren_Fkt

Ansteuerung zweier Standard Motoren

```
void MotorenSet()                          // setzen der Motor-Steuer-Ausgänge
{
  pinMode(motor1_Pin1, OUTPUT);
  pinMode(motor1_Pin2, OUTPUT);
  pinMode(motor2_Pin1, OUTPUT);
  pinMode(motor2_Pin2, OUTPUT);
}

void MotorenMove(int mV, int pwm)   // Funktion zum Ansteuern der Motoren
{
  byte m1_1, m1_2, m2_1, m2_2;

  switch(mV)
  {
    case FORWARD:                          // Forwärts
      m1_1 = LOW; m1_2 = HIGH;             // Setzen der 4 H-Brücken Schalter
      m2_1 = LOW; m2_2 = HIGH;
      break;
    case BACKWARD:                         // Rückwärts
      m1_1 = HIGH; m1_2 = LOW;             // Setzen der 4 H-Brücken Schalter
      m2_1 = HIGH; m2_2 = LOW;
      break;
    case RIGHT:                            // Rechts
      m1_1 = HIGH; m1_2 = LOW;             // Setzen der 4 H-Brücken Schalter
      m2_1 = LOW;  m2_2 = HIGH;
      break;
    case LEFT:                             // Links
      m1_1 = LOW;  m1_2 = HIGH;            // Setzen der 4 H-Brücken Schalter
      m2_1 = HIGH; m2_2 = LOW;
      break;
    case PAUSE:                            // Pause
      m1_1 = LOW; m1_2 = LOW;              // Setzen der 4 H-Brücken Schalter
      m2_1 = LOW; m2_2 = LOW;
      break;
    default:                               // bei allen anderen Werten nichts tun
      break;
  }  // switch
  digitalWrite(motor1_Pin1, m1_1);   // Ansteuern der 4 H-Brücken Schalter
  digitalWrite(motor1_Pin2, m1_2);   // Ansteuern der 4 H-Brücken Schalter
  digitalWrite(motor2_Pin1, m2_1);   // Ansteuern der 4 H-Brücken Schalter
  digitalWrite(motor2_Pin2, m2_2);   // Ansteuern der 4 H-Brücken Schalter
  analogWrite(enableM1Pin, pwm);     // Ansteuern PWM Motor 1
  analogWrite(enableM2Pin, pwm);     // Ansteuern PWM Motor 2
}    // fkt
```

Ansteuerung eines Schritt Motors

Ansteuerung eines Schritt Motors

bipolar

unipolar

Ansteuerung eines Schritt Motors

Ansteuerung eines Schritt Motors

1 x bipolarer Schrittmotor (z.B. *Pololu 1208*)

Ansteuerung eines Schritt Motors

Arduino
Digital I/O
Analog IN
Vcc
GND
~PWM
Externe Spannungsversorgung für den Schrittmotor
+ Batt - Batt
Input1
Input2
Input3
Input4
Output1
Output2
Output3
Output4
Vcc1
1,2EN
3,4EN
Vcc2
GND
L293DNE
330 R1
330 R2
R
L
Richtungsanzeigen
A
B
C
D
M
Bipolarer Schrittmotor Pololu 1208
gemeinsame Masseleitung

DerSchrittmotor §

```
/*  Dreht einen Schrizzmotor "x" Schritte im Uhrzeigersinn und danach wieder zurück
Das ganze mit unterschiedlicher Geschwindigkeit   */
#define Stepper_A1 5      // Pin für Stepper A1
#define Stepper_A3 4      // Pin für Stepper A3
#define Stepper_B1 3      // Pin für Stepper B1
#define Stepper_B3 2      // Pin für Stepper B3
byte stepValues[5][4] = {{LOW, LOW, LOW, LOW},     // Stepper aus
                         {LOW, HIGH, HIGH, LOW},   // Step 1
                         {LOW, HIGH, LOW, HIGH},   // Step 2
                         {HIGH, LOW, LOW, HIGH},   // Step 3
                         {HIGH, LOW, HIGH, LOW}}; // Step 4
void setup(){
  pinMode(Stepper_A1, OUTPUT);
  pinMode(Stepper_A3, OUTPUT);
  pinMode(Stepper_B1, OUTPUT);
  pinMode(Stepper_B3, OUTPUT);
  for(int i = 0; i < 10; i++){
    action(30, 2);          // 30 Steps nach rechts mit 2ms Pause
    action(-30, 10);        // 30 Steps nach links mit 10ms Pause
  }
  action(0, 0);             // Stromlos schalten
}

void loop(){/* leer*/}

void action(int count, byte delayValue){
  if(count > 0) // Drehung nach rechts
    for(int i = 0; i < count; i++)
      for(int sequenceStep = 1; sequenceStep <= 4; sequenceStep++)
        moveStepper(sequenceStep, delayValue);
  if(count < 0) // Drehung nach links
    for(int i = 0; i < abs(count); i++)
      for(int sequenceStep = 4; sequenceStep > 0; sequenceStep--)
        moveStepper(sequenceStep, delayValue);
  if(count == 0) // Stromlos schalten
    moveStepper(0, delayValue);
}

void moveStepper(byte s, byte delayValue){
  digitalWrite(Stepper_A1, stepValues[s][0]);
  digitalWrite(Stepper_A3, stepValues[s][1]);
  digitalWrite(Stepper_B1, stepValues[s][2]);
  digitalWrite(Stepper_B3, stepValues[s][3]);
  delay(delayValue); // Pause
}
```

Zeichen	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1
A	1	1	1	0	1	1	1
b	0	0	1	1	1	1	1
C	1	0	0	1	1	1	0
d	0	1	1	1	1	0	1
E	1	0	0	1	1	1	1
F	1	0	0	0	1	1	1

Unser 10. Programm: Ansteuerung einer 7 Segment Anzeige

SC38-11GWA

SA56-11SRWA

SC52-11GWA

SA56-11SRWA, HD1131R LIS546AP

Unser 10. Programm: Ansteuerung einer 7 Segment Anzeige (Klein, bzw. Pins seitlich !!)

Für (Groß, bzw. Pins oben/unten, s. nächste Folie

Unser 10. Programm: Ansteuerung einer 7 Segment Anzeige (Groß, bzw. Pins oben/unten

Pin 1 ist links unten (von oben aus gesehen

Unser 10. Programm: Ansteuerung einer 7 Segment Anzeige (Klein, bzw. Pins seitlich !!)
alle Segmente 3x blinken lassen (Segmenttest) dann 0 – 9 und Punkt nacheinander anzeigen

Single_7LEDSegment_KursX §

```
/*   Steuert eine LED 7 Segment Anzeige an.
Es gibt welche mit gemeinsamer Kathode oder Anode
Auch auf die Anschlüsse achten, sind entweder an der langen oder schmalen Seite */
//                        a,b,c,d,e,f,g,DP     pro Segment 1 Byte mit "1" wenn EIN(aktiv) & "0" wenn "AUS"
int segmente[11][8] = {{1,1,1,1,1,1,0,0},     // Segmente für Ziffer "0"
                       {0,1,1,0,0,0,0,0},     // Segmente für Ziffer "1"
                       {1,1,0,1,1,0,1,0},     // Segmente für Ziffer "2"
                       {1,1,1,1,0,0,1,0},     // Segmente für Ziffer "3"
                       {0,1,1,0,0,1,1,0},     // Segmente für Ziffer "4"
                       {1,0,1,1,0,1,1,0},     // Segmente für Ziffer "5"
                       {1,0,1,1,1,1,1,0},     // Segmente für Ziffer "6"
                       {1,1,1,0,0,0,0,0},     // Segmente für Ziffer "7"
                       {1,1,1,1,1,1,1,0},     // Segmente für Ziffer "8"
                       {1,1,1,1,0,1,1,0},     // Segmente für Ziffer "9"
                       {0,0,0,0,0,0,0,1}};    // Segmente für  "Dezimal Punkt

int pinArray[] = {2,3,4,5,6,7,8,9};           // Array mit allen Segment-Ausgängen
int gemeinsameKathode = 1;                    // Kathode dann = 1, Anode dann 0;

void setup()
{
  for (int i=0; i<8; i++)                     // Schleife über alle Segmente
    pinMode(pinArray[i], OUTPUT);             // jeder Segmentpin als Ausgang
} // setup

void loop()
{
  if (gemeinsameKathode)                     // 7-Segment Typ == gemeinsame Kathode
  {
    for (int i=0; i<3; i++)                  // dreimal Segment-Test, alle LED (Segmente) blinken
    {
      for (int j=0; j<8; j++)                // alle 7 Segmente plus DP ansteuern
        digitalWrite(pinArray[j], HIGH);
      delay(800);                            // kurze Pause
      for (int j=0; j<8; j++)
        digitalWrite(pinArray[j], LOW);      // alle 7 Segmente aus
      delay(200);                            // kurze Pause
    }  // for
    delay(1000);
```

SW Version mit
2 Dimensionalem Array

```
  for (int i=0; i<11; i++)                    // Ziffern 0 - 9 und Dez Punkt
  {                                           // nacheinander ansteuern
    for (int j=0; j<8; j++)                   // Alle Segmente für Ziffer "i" ansteuern
      digitalWrite(pinArray[j], (segmente[i][j]==1)?HIGH:LOW); // Kurzform
    delay(800);
  }  // for
  for (int j=0; j<8; j++)                     // am Ende alles wieder aus
    digitalWrite(pinArray[j], LOW);
  delay(2000);                                // kurz warten, dann von vorne
}  // if gem. Kathode
// keine gemeinsame Kathode also Anode
else                                          // dieses Programm ist fast gleich
{                                             // aber für gemeinsame Anode
  for (int i=0; i<3; i++)                     // dreimal Segment-Test
  {
    for (int j=0; j<8; j++)
      digitalWrite(pinArray[j], LOW);
    delay(800);
    for (int j=0; j<8; j++)
      digitalWrite(pinArray[j], HIGH);
    delay(200);
  }  // for
  delay(1000);

  for (int i=0; i<11; i++)                    // 0 - 9 und Dez Punkt
  {                                           // nacheinander ansteuern
    for (int j=0; j<8; j++)
      digitalWrite(pinArray[j], (segmente[i][j]==1)?LOW:HIGH);
    delay(800);
  }  // for
  for (int j=0; j<8; j++)
    digitalWrite(pinArray[j], HIGH);
  delay(2000);
}  // else gem. Anode
}  // loop
```

SW Version mit 2 Dimensionalem Array (Fortsetzung)

Dieser „ELSE“ Teil wird nur bei gemeinsamer Anode benötigt und kann sonst weggelassen werden

Single_7LEDSegment_1DIM_KursX §

```
//  Segment:          abcdefgDP  wenn Segment aktiv (EIN) dann Bit = "1", sonst Bit = "0"
byte segmente[11] = {B11111100,    // Ziffer "0"
                     B01100000,    // Ziffer "1"
                     B11011010,    // Ziffer "2"
                     B11110010,    // Ziffer "3"
                     B01100110,    // Ziffer "4"
                     B10110110,    // Ziffer "5"
                     B10111110,    // Ziffer "6"
                     B11100000,    // Ziffer "7"
                     B11111110,    // Ziffer "8"
                     B11110110,    // Ziffer "9"
                     B00000001};   // Dezimal-Punkt
int pinArray[] = {2,3,4,5,6,7,8,9};          // Array mit allen Segment-Ausgängen (Pins)
int gemeinsameKathode = 1;                   // Kathode dann = 1, Anode dann 0;

void setup()
{
  for (int i=0; i<8; i++)                    // Schleife über alle 7 Segmente plus Dezimal Punkt, d.h. 8
    pinMode(pinArray[i], OUTPUT);            // jeder Segmentpin als Ausgang, Index von 0 - 7, d.h. i<8
} // setup

void loop()
{
  if (gemeinsameKathode)                    // 7-Segment Typ == gemeinsame Kathode
  {                                         // Alle Segmente und DP ansteuern
    for (int i=0; i<3; i++)                 // dreimal Segmenttest, d.h. 3 mal blinken lassen
    {
      for (int j=0; j<8; j++)               // alle Segmente ansteuern --> HIGH
        digitalWrite(pinArray[j], HIGH);    // --> HIGH
      delay(800);                           // kurze Pause
      for (int j=0; j<8; j++)               // alle Segmente aus
        digitalWrite(pinArray[j], LOW);     // --> LOW
      delay(200);                           // kurze Pause
    }  // for
    delay(1000);
```

SW Version mit 1 Dimensionalem Array und bitread

```
  for (int i=0; i<11; i++)                // alle 10 Ziffern und Dez Punkt nacheinander anzeigen
  {                                       // d.h. 0-9 und DP also 11 nacheinander ansteuern
    for (int j=7; j>=0; j--)              // Abfragen der Bits für die Segmente für Ziffer i
      digitalWrite(pinArray[7-j], bitRead(segmente[i], j) == 1?HIGH:LOW);
    delay(800);                           // EIN Zeit für eine Ziffer
  }  // for
  for (int j=0; j<8; j++)                 // nach letzter Ziffer alles wieder aus
    digitalWrite(pinArray[j], LOW);       // --> LOW
  delay(2000);                            // kurz warten, dann das ganze von vorne
}  // if
else                                      // dieses Programm ist fast gleich
{                                         // aber für gemeinsame Anode
  for (int i=0; i<3; i++)
  {
    for (int j=0; j<8; j++)
      digitalWrite(pinArray[j], LOW);     // Statt HIGH  hier LOW
    delay(800);
    for (int j=0; j<8; j++)
      digitalWrite(pinArray[j], HIGH);    // Statt LOW  hier HIGH
    delay(200);
  }  // for
  delay(1000);

  for (int i=0; i<11; i++)
  {
    for (int j=7; j>=0; j--)              // Abfragen der Bits für die Segmente für Zahl i
      digitalWrite(pinArray[7-j], bitRead(segmente[i], j) == 1?LOW:HIGH); // statt HIGH:LOW hier LOW:HIGH
    delay(800);
  }  // for
  for (int j=0; j<8; j++)
    digitalWrite(pinArray[j], HIGH);
  delay(2000);
}  // else
}  // loop
```

SW Version mit 1 Dimensionalem Array und bitread (Fortsetzung)

Dieser „ELSE“ Teil wird nur bei gemeinsamer Anode benötigt und kann sonst weggelassen werden

Unterschied „2 dimensionales“ vs „1 dimensionales“ Array

```
int segmente[11][8] = {{1,1,1,1,1,1,0,0},    // 0
                       {0,1,1,0,0,0,0,0},    // 1
                       {1,1,0,1,1,0,1,0},    // 2
                       {1,1,1,1,0,0,1,0},    // 3
                       {0,1,1,0,0,1,1,0},    // 4
                       {1,0,1,1,0,1,1,0},    // 5
                       {1,0,1,1,1,1,1,0},    // 6
                       {1,1,1,0,0,0,0,0},    // 7
                       {1,1,1,1,1,1,1,0},    // 8
                       {1,1,1,1,0,1,1,0},    // 9
                       {0,0,0,0,0,0,0,1}};   // Punkt
```

```
byte segmente[11] = {B11111100,    // 0
                     B01100000,    // 1
                     B11011010,    // 2
                     B11110010,    // 3
                     B01100110,    // 4
                     B10110110,    // 5
                     B10111110,    // 6
                     B11100000,    // 7
                     B11111110,    // 8
                     B11110110,    // 9
                     B00000001};   // Punkt
```

- **Jeweils nur Segmentinfo „On“ oder „Off“, d.h. 1 Bit**
- **2 dim Array == 11 x 8 int = 176 byte** **1 dim Array == 11 byte**

```
for (int i=0; i<11; i++)                    // alle 10 Ziffern und Dez Punkt
{                                           // nacheinander ansteuern
  for (int j=0; j<8; j++)                   // Segmente für Zahl i ansteuern
    digitalWrite(pinArray[j], (segmente[i][j]==1)?HIGH:LOW);
  delay(800);
}
```

```
for (int i=0; i<11; i++)                    // alle 10 Ziffern und Dez Punkt
{                                           // nacheinander ansteuern
  for (int j=7; j>=0; j--)                  // Abfragen der Bits für die Segmente für Zahl i
    digitalWrite(pinArray[7-j], bitRead(segmente[i], j) == 1?HIGH:LOW);
  delay(800);
}
```

int bitRead(wert, bitpos) **gibt den Bitwert der Position eines Wertes zurück (int = Rückgabewert, wert=zu lesendes Byte, bitpos= zu lesende Bitposition)**

Unser 11. Programm: Ansteuerung von zwei 7-Segment Anzeigen und Anzeige eines Poti-Wertes

(Klein, bzw. Pins seitlich !!)

BC547

(Klein, bzw. Pins seitlich !!)

Zweimal_7LEDSegmentGC_1DIM_KursX

Teil1

```
/*   Steuert zwei LED 7-Segment Anzeigen per Multiplex an und zeigt den Wert
eines Potis an. Mit eindimensionalen Array & Fkt "bitRead"; gemeinsame Kathode */
//            Segment  abcdefgP
byte segmente[11] = {B11111100,    // 0
                     B01100000,    // 1
                     B11011010,    // 2
                     B11110010,    // 3
                     B01100110,    // 4
                     B10110110,    // 5
                     B10111110,    // 6
                     B11100000,    // 7
                     B11111110,    // 8
                     B11110110,    // 9
                     B00000001};   // Punkt
int pinArray[] = {2,3,4,5,6,7,8,9};   // Array mit allen Segment-Ausgängen
int einerStelle = 12;               // Pin 12 als Ausgang zur Ansteuerung des Transistors "einerStelle"
int zehnerStelle = 13;              // Pin 13 als Ausgang zur Ansteuerung des Transistors "zehnerStelle"
int potiPin = A0;                   // Nummer Analogeingang zum Lesen des Potiwertes
int potiWert;                       // Poti Wert speichern
int zweiStellenZahl;                // umgewandelte 2 stellige Poti Zahl speichern

void setup()
{
//  for (int i=0; i<7; i++)         // Schleife über alle Segmente, außer DP
  for (int i=0; i<8; i++)           // Schleife über alle Segmente, mit DP
    pinMode(pinArray[i], OUTPUT);   // jeder Segmentpin als Ausgang
  pinMode(einerStelle, OUTPUT);     // Ansteuerung Trans 1 als Output
  pinMode(zehnerStelle, OUTPUT);    // Ansteuerung Trans 2 als Output
  test();                           // alle Segmente aufblinken lassen
}   // setup

void loop()
{
  potiWert = analogRead(potiPin);   // Wert des Potentiometer lesen
  zweiStellenZahl = map(potiWert, 0, 1023, 0, 99);   // map 0 - 1023 nach 0 - 99
  anzeige(zweiStellenZahl);         // Fkt. zeigt Wert auf 2 Stellen an
}  // loop
```

Teil2

```
void anzeige(int wert)
{
  int einer, zehner;                    // Variable für Einer und Zehner Stelle
  zehner = wert / 10;                   // Zehnerstelle berechnen
  einer = wert - (zehner * 10);         // Einerstelle berechnen
  digitalWrite(einerStelle, HIGH);      // Transistor für Einerstelle ein = HIGH (NPN)
  digitalWrite(zehnerStelle, LOW);      // Transistor für Zehnerstelle aus = LOW (NPN)
  ansteuerung(einer);                   // Segmente der Einerstelle ansteuern
  digitalWrite(einerStelle, LOW);       // Transistor für Einerstelle aus = LOW (NPN)
  digitalWrite(zehnerStelle, HIGH);     // Transistor für Zehnerstelle ein = HIGH (NPN)
  ansteuerung(zehner);                  // Segmente der Zehnerstelle ansteuern
}    // fkt anzeige

void ansteuerung(int a)
{
  for (int j=7; j>=0; j--)     // Abfragen der Bits für die Segmente für Zahl i
    digitalWrite(pinArray[7-j], bitRead(segmente[a], j) == 1?HIGH:LOW);
  delay(5);
}    // fkt ansteuerung

void test()
{
  digitalWrite(einerStelle, HIGH);          // Transistor für Einerstelle ein = HIGH (NPN)
  digitalWrite(zehnerStelle, HIGH);         // Transistor für Zehnerstelle ein = HIGH (NPN)
  for (int i=0; i<3; i++)                   // dreimal Segmenttest
  {
//    for (int j=0; j<7; j++)                   // alle Segmente außer DP ansteuern
    for (int j=0; j<8; j++)                 // alle Segmente mit DP ansteuern
      digitalWrite(pinArray[j], HIGH);
    delay(800);                             // kurze Pause
//    for (int j=0; j<7; j++)               // alle Segmente außer DP ansteuern
    for (int j=0; j<8; j++)                 // alle Segmente mit DP ansteuern
      digitalWrite(pinArray[j], LOW);       // alle Segmente aus
    delay(200);                             // kurze Pause
  }
  digitalWrite(einerStelle, LOW);           // Transistor für Einerstelle aus = LOW (NPN)
  digitalWrite(zehnerStelle, LOW);          // Transistor für Zehnerstelle aus = LOW (NPN)
}  // fkt test
```

Modulo Operator **„%“**
Syntax result = dividend **%** divisor
Bsp einer = wert % 10

Unser 12. Programm: Reaktionstester
Ansteuerung einer Ampel, von zwei 7-Segment Anzeigen, Messen und Anzeige der Reaktionszeit, mit „Schummel“-überwachung

1) Test aller LEDs und Segmente durch blinken, Test der einzelnen Segmente durch Hochzählen
2) Nach dem Test zum Start „0“ anzeigen
3) Alle LEDs der Ampel sind aus
4) Es wird auf einen Tastendruck gewartet, dann startet die Ampel mit “rot“ → Spielstart
5) Wechsel nach 2 sec auf „gelb“, dann auf „grün“
6) Sobald die Ampel „grün“ anzeigt beginnt der Zähler mit hochzählen
7) Nun so schnell wie möglich „Taster“ drücken, der Zähler zeigt die benötigte Zeit an
8) Schummelfunktion: Wird zu früh gedrückt, also vor „grün“, blinkt die rote LED und zeigt „99“ an und das Spiel beginnt von vorne, d.h. „0“ wird wieder angezeigt
9) Wird nicht gedrückt, d.h. reagiert wird auch eine blinkende LED und „99“ angezeigt und das Spiel kann erneut gestartet werden
10) Am Ende des Spieles kann mit der Taste ein neues Spiel gestartet werden
11) Mit der Reset-Taste des Arduinos startet das komplette Programm erneut, also einschließlich „Testfunktion“

NPN
BC547
C
B
E
NPN
BC547
C
B
E
BC557
PNP
PNP
1K
R2
1K
R1
Rot
Gelb
Grün
Ampel
330
R
GND
Arduino
13
12
11
10
9
8
7
6
5
4
3
2
1
0
PWM
Digital I/O
VCC
GND
Analog IN
5
4
3
2
1
0
10K
R
Taster
+5V
Zehnerstelle
Einerstelle
a
b
c
d
e
f
g
CA
DP
(Kleine Anzeige bzw Pins seitlich !!)

Ampel
330
R10
330
R11
330
R14
GND
1K
R8
1K
R9
E
BC557
PNP
B
C
Arduino
Digital I/O
Analog IN
Vcc
GND
~PWM
330
R7
330
R6
330
R5
330
R4
330
R3
330
R2
330
R1
Zehnerstelle
Einerstelle
g
f
C.A
e
d
a
b
c
DP
Taster
10K
R13
+5V

Reaktionstester_KursX | RT_Anzeige | RT_Funktionen

```
/*   Steuert zwei LED 7-Segment Anzeigen per Multiplex an und zeigt die
Reaktionszeit an. Mit eindimensionalen Array & Fkt "bitRead"; gemeinsame Kathode */

const byte segmente[11] = {B11111100,    // 0   Type "const" kann nicht geändert werden,
                           B01100000,    // 1   dafür schneller
                           B11011010,    // 2
                           B11110010,    // 3
                           B01100110,    // 4
                           B10110110,    // 5
                           B10111110,    // 6
                           B11100000,    // 7
                           B11111110,    // 8
                           B11110110,    // 9
                           B00000001};   // Punkt
const byte pinArray[] = {2,3,4,5,6,7,8};        // Array mit Out Pins für 7Segmente, ohne DP
const byte einerStelle = 12,
           zehnerStelle = 13;                   // Transistor Ausgang Pins
const byte ledPinRot = 9,
           ledPinGelb = 10, ledPinGruen = 11;   // LED  Ausgang Pins
const byte tasterPin  = A1;                     // Analog Eingang 1 fuer Taster
byte anzeigeWert = 0;                           // byte, da nur Werte von 0 - 99
byte phase = 0;                                 // phase 0,1,2,3, usw.
long startPunkt = 0;                            // ms Zaehler daher long
long startZeit = 0;                             // ms Startzeit daher long
long endZeit = 0;                               // ms Endzeit daher long
int reakZeit = 0;
boolean taster = false;                         // speichert Zustand des Tasters
boolean tasterNeu = false;                      // speichert Flanke eines Tastendruckes
boolean gestoppt = false;                       // Status ob Spiel korrekt gestoppt wurde

void setup()
{
  for (int i=0; i<7; i++)                 // Schleife über alle Segmente
    pinMode(pinArray[i], OUTPUT);         // jeder Segmentpin als Ausgang
  pinMode(einerStelle, OUTPUT);           // Ansteuerung T1 als Output
  pinMode(zehnerStelle, OUTPUT);          // Ansteuerung T2 als Output
  pinMode(ledPinRot, OUTPUT);             // Ansteuerung LED als Output
  pinMode(ledPinGelb, OUTPUT);            // Ansteuerung LEDals Output
  pinMode(ledPinGruen, OUTPUT);           // Ansteuerung LED als Output
  test();                                 // steuert alle Segmente & LEDs an
  Serial.begin(9600);                     // serielle Schnittstelle starten
}   // setup
```

```
44  void loop()
45  {
46    // permanente Tastenabfrage
47    if (analogRead(tasterPin) > 900)        // Spannung > Schwelle
48    {                                       // dann ist Taste gedrückt
49      if (!taster)                          // war taste noch nicht gedrückt
50      {
51        tasterNeu = true;                   // dann ist taste neu gedrückt (Flanke)
52        taster = true;                      // Status Taste setzen
53      }  // if
54    }  // if
55    else                                    // Spannung < Schwelle
56    {                                       // Taste nicht gedrückt
57      taster = false;                       // Status Taste setzen
58    }  // else
59    // Es muss immer was angezeigt werden, sonst geht 7 Seg. Anzeige aus
60    if (gestoppt) {                         // Spiel gestoppt?
61      anzeige(reakZeit);                    // dann Stopzeit (Reaktionszeit) anzeigen
62    }  // if
63    else {
64      anzeige(0);                           // sonst "0" anzeigen
65    }  // else
66    if ((tasterNeu) && (phase == 0))   // Taste gedrückt und Spiel bereit?
67    {                                  // dann Spiel los
68      tasterNeu = false;               // Tastenaktion ausgeführt daher zurücksetzen
69      phase = 1;                       // Phase "1" der Ampel starten (ROT)
70      startPunkt = millis();           // Timer auf aktuelle (Start)Zeit setzen
71    }
72    if ((phase == 1) && (millis() - startPunkt > 400))  // nach 400ms startet ROT
73      digitalWrite(ledPinRot, HIGH);  // ROT an
74    if ((phase == 1) && (millis() - startPunkt > 2400)) // nach 2 Sec ROT startet GELB
75    {
76      digitalWrite(ledPinGelb, HIGH); // GELB an
77      phase = 2;                       // start Phase2 == GELB & Fehlererkennung
78    }  //if
```

```
79    if ((phase == 2) && (tasterNeu))   // Wird Taste in Phase 2 (Gelb) gedrückt ?
80    {                                  // dann zu früh --> Fehler
81      tasterNeu = false;               // Tastenaktion ausgeführt daher zurücksetzen
82      blinken();                       // "99" anzeigen und blinken
83      reset();                         // neues Spiel
84    }  // if
85    if ((phase == 2) && (millis() - startPunkt > 4400)) // nach 2 Sec
86    {
87      digitalWrite(ledPinRot, LOW);          // ROT aus
88      digitalWrite(ledPinGelb, LOW);         // GELB aus
89      digitalWrite(ledPinGruen, HIGH);       // GRUEN an
90      phase = 3;                             // Start Phase "3"
91      startZeit = millis();                  // Start der Reaktionszeit
92    }  // if
93    if ((phase == 3) && (!gestoppt))         // Spiel in Phase Gruen (3) und läuft
94    {
95      if (tasterNeu)                         // Taste gedrückt?
96      {
97        endZeit = millis();                  // dann Spiel regulaer zu Ende
98        tasterNeu = false;                   // Tastenaktion ausgeführt daher zurücksetzen
99        gestoppt = true;                     // Spiel regulär gestoppt
100       reakZeit = endZeit - startZeit;      // Reaktionszeit berechnen
101       Serial.println(reakZeit);            // Stopzeit einmal seriel ausgeben
102       if (reakZeit > 999) {
103         reakZeit = 99;                     // Reaktionszeit anzeigen
104       }  // if
105       else {
106         reakZeit = reakZeit / 10;          // nur Zehntel und Hundertstel anzeigen
107       }  // else
108     }   // if
109     else {
110       anzeige(anzeigeWert++);          // Anzeigewert hochzählen und anzeigen
111     }  // else
112   }
113   if (anzeigeWert > 98) {              // keine Reaktion
114     blinken();                         // "99" anzeigen und blinken
115     reset();                           // neues Spiel
116   }  // if
117   if ((gestoppt)&&(tasterNeu))         // Spiel gestoppt und Taste gedrückt
118   {
119     tasterNeu = false;                 // Tastenaktion ausgeführt daher zurücksetzen
120     reset();                           // neues Spiel
121   }  // if
122 }  // loop
```

Reaktionstester_KursX § | RT_Anzeige § | RT_Funktionen §

```
void test()                                   // 7 Segment Testroutine
{
  digitalWrite(einerStelle, HIGH);            // T1 == EIN
  digitalWrite(zehnerStelle, HIGH);           // T2 == EIN
  for (byte i=0; i<3; i++)                    // dreimal Segmenttest und LED Test
  {
    digitalWrite(ledPinRot, HIGH);            // Rot EIN
    digitalWrite(ledPinGelb, HIGH);           // Gelb EIN
    digitalWrite(ledPinGruen, HIGH);          // Grün EIN
    for (byte j=0; j<7; j++)                  // alle Segmente ansteuern (ohne DP)
      digitalWrite(pinArray[j], HIGH);        // --> EIN
    delay(800);                               // kurze Pause
    digitalWrite(ledPinRot, LOW);             // Rot AUS
    digitalWrite(ledPinGelb, LOW);            // Gelb AUS
    digitalWrite(ledPinGruen, LOW);           // Grün AUS
    for (byte j=0; j<7; j++)                  // alle Segmente ansteuern (ohne DP)
      digitalWrite(pinArray[j], LOW);         // --> AUS
    delay(200);                               // kurze Pause
  }
  digitalWrite(einerStelle, LOW);             // T1 == AUS
  digitalWrite(zehnerStelle, LOW);            // T2 == AUS
  for (byte k=0; k<100; k++)                  // 0 - 99 anzeigen
  {
    for (byte j=0; j<8; j++)                  // Wert "8" mal anzeigen, entspricht delay
      anzeige(k);                             // Zähler "k" anzeigen
  }
}  // fkt test
```

```
void ansteuerung(int a)                    // 7 Segmente einer Zahl "a" ansteuern
{                                          // Abfragen der Bits für die Segmente für
  for (byte j=7; j>0; j--)                 // Zahl "a", außer "0", da dies DP
    digitalWrite(pinArray[7-j], bitRead(segmente[a], j) == 1?HIGH:LOW);
  delay(5);                     // kurze Pause, sonst zuviele unnötige Wechsel
}    // fkt ansteuerung

void anzeige(byte wert)              // zerlegt eine 2stellige Zahl und zeigt die Ziffern an
{
  byte einer, zehner;                // Variable für die beiden Ziffern
  zehner = (wert / 10);              // Zehnerstelle berechnen
  einer = wert - (zehner * 10);      // Einerstelle berechnen
  digitalWrite(einerStelle, HIGH);   // Transistor für Einerstelle ein = HIGH (NPN)
  digitalWrite(zehnerStelle, LOW);   // Transistor für Zehnerstelle aus = LOW (NPN)
  ansteuerung(einer);                // Segmente der Einerstelle ansteuern
  digitalWrite(einerStelle, LOW);    // Transistor für Einerstelle aus = LOW (NPN)
  digitalWrite(zehnerStelle, HIGH); // Transistor für Zehnerstelle ein = HIGH (NPN
  ansteuerung(zehner);               // Segmente der Zehnerstelle ansteuern
}    // fkt anzeige
```

Reaktionstester_KursX§ | RT_Anzeige§ | RT_Funktionen§

```
// Fkt. blinkt rote LED und zeigt 99 == Fehler an
void blinken()
{
  digitalWrite(ledPinRot, HIGH);     // Rot  EIN
  digitalWrite(ledPinGelb, LOW);     // Gelb AUS
  digitalWrite(ledPinGruen, LOW);    // Grün AUS
  for (byte i=0; i<40; i++)          // Blink Schleife
  {
    digitalWrite(ledPinRot, (i%2 == 0)?HIGH:LOW); // "20 x ein" und "20 x aus"
    for (byte j=0; j<10; j++) // Wert 10 mal anzeigen, als delay Fkt
      anzeige(99);
  }
}

// Auf Anfangswerte zurücksetzen
void reset()
{
  phase = 0, startPunkt = 0, stopZeit = 0;
  anzeigeWert = 0;
  taster = false, gestoppt = false;
  digitalWrite(ledPinRot, LOW);
  digitalWrite(ledPinGelb, LOW);
  digitalWrite(ledPinGruen, LOW);
}
```

MAX7221 Serial Interfaced, 8-Digit LED Display Driver von MAXIM

Ein Schieberegister ansteuern, die sogenannte Porterweiterung

Pin description

Symbol	Pin	Description
Q1	1	parallel data output 1
Q2	2	parallel data output 2
Q3	3	parallel data output 3
Q4	4	parallel data output 4
Q5	5	parallel data output 5
Q6	6	parallel data output 6
Q7	7	parallel data output 7
GND	8	ground (0 V)
Q7S	9	serial data output
$\overline{MR}$	10	master reset (active LOW)
SHCP	11	shift register clock input
STCP	12	storage register clock input
$\overline{OE}$	13	output enable input (active LOW)
DS	14	serial data input
Q0	15	parallel data output 0
V_{CC}	16	supply voltage

Functional diagram

interne Register
DS
1
Serieller Eingang
0 1 2 3 4 5 6 7
0 0 0 0 0 0 0 0
Q_A Q_B Q_C Q_D Q_E Q_F Q_G Q_H
Ausgänge

interne Register
DS
1
Serieller Eingang
SH_CP
ST_CP
0 1 2 3 4 5 6 7
0 0 0 0 0 0 0 0
Q_A Q_B Q_C Q_D Q_E Q_F Q_G Q_H
Ausgänge

interne Register
DS
0
Serieller Eingang
SH_CP
ST_CP
0 1 2 3 4 5 6 7
1 0 0 0 0 0 0 0
Q_A Q_B Q_C Q_D Q_E Q_F Q_G Q_H
Ausgänge

interne Register
DS
0
Serieller Eingang
SH_CP
ST_CP
0 1 2 3 4 5 6 7
1 0 1 1 0 1 0 1
1 0 1 1 0 1 0 1
Q_A Q_B Q_C Q_D Q_E Q_F Q_G Q_H
Ausgänge

Unser 13. Programm: digitale Ausgänge über ein Schieberegister ansteuern, die sogenannte Porterweiterung und damit eine LED einschalten

Technikschule Esslingen; Kurs: Arduino

LED_mit_HC595_Kurs6 §

```
/*
  LED Lauflicht mit HC595
  Schaltet eine Reihe von LEDs nach einem bestimmten Muster ein und aus
  und nutzt dabei ein Schieberegister HC595

  Im Schaltkreis sind 8 LEDs mit einem Widerstand gegen Masse geschaltet
  und mit den Ausgängen des HC595 verbunden
 */

int taktPin = 8;          // SH_CP = Schieberegister Takteingang (Shiftregister Clock Input)
int speicherPin = 9;      // ST_CP = Speicherregister Takteingang (Storageregister Clock Input)
int datenPin = 10;        // DS = Serieller Eingang (Serial data Input)

void setup()
{
  pinMode(taktPin,OUTPUT);            // "taktPin" als Ausgang definieren
  pinMode(speicherPin,OUTPUT);        // "speicherPin" als Ausgang definieren
  pinMode(datenPin,OUTPUT);           // "datenPin" als Ausgang definieren
  resetPins();                        // Alle Pins auf LOW setzen. Diese Funktion ist unten definiert
  digitalWrite(datenPin, HIGH);       // DS für spätere Übernahme durch SH_CP auf HIGH setzen
  delay(20);                          // kurze Verarbeitungs-Pause
  digitalWrite(taktPin, HIGH);        // SH_CP; --> DS-Pegel-Eingang in interne Speicherregister übertragen
  delay(20);                          // kurze Verarbeitungs-Pause
  digitalWrite(speicherPin, HIGH);    // ST_CP;  --> interner Speicherregisterinhalt an Ausgänge übertragen
  delay(20);                          // kurze Verarbeitungs-Pause
  resetPins();                        // Alle Pins wieder auf LOW setzen
}

void loop()
{
     //   leere Haupt-Schleife//
}    // loop

void resetPins()          // Funktion setzt alle Pins auf LOW (Reset)
{
  digitalWrite(taktPin, LOW);
  digitalWrite(speicherPin, LOW);
  digitalWrite(datenPin, LOW);
}
```

LED_mit_HC595_Kurs7

```
/*  LED Lauflicht mit HC595
    Schaltet eine Reihe von LEDs nach einem bestimmten Muster ein und aus
    und nutzt dabei ein Schieberegister HC595
    Im Schaltkreis sind 8 LEDs mit einem Widerstand gegen Masse geschaltet
    und mit den Ausgängen des HC595 verbunden  */

int taktPin = 8;        // SH_CP = Schieberegister Takteingang (Shiftregister Clock Input)
int speicherPin = 9;    // ST_CP = Speicherregister Takteingang (Storageregister Clock Input)
int datenPin = 10;      // DS = Serieller Eingang (Serial data Input)
int datenArray1[] = {1,0,1,0,1,0,1,0};
int datenArray2[] = {0,1,0,1,0,1,0,1};

void setup()
{
  pinMode(taktPin,OUTPUT);          // "taktPin" als Ausgang definieren
  pinMode(speicherPin,OUTPUT);      // "speicherPin" als Ausgang definieren
  pinMode(datenPin,OUTPUT);         // "datenPin" als Ausgang definieren
  resetPins();                      // Alle Pins auf LOW setzen. Diese Funktion ist unten definiert
  delay(20);                        // kurze Verarbeitungs-Pause
}

void loop()
{
  setzePins(datenArray1);           // Setze Pins mit Fkt. über Daten-Array1
  digitalWrite(speicherPin, HIGH);  // ST_CP;  --> Speicherregisterinhalt an Ausgänge übertragen
  delay(800);
  setzePins(datenArray2);           // Setze Pins über Daten-Array2
  digitalWrite(speicherPin, HIGH);  // ST_CP;  --> Speicherregisterinhalt an Ausgänge übertragen
  delay(800);
}    // loop

void resetPins()          // Funktion setzt alle Pins auf LOW (Reset)
{
  digitalWrite(taktPin, LOW);
  digitalWrite(speicherPin, LOW);
  digitalWrite(datenPin, LOW);
}
```

Unser 14. Programm: digitale Ausgänge über ein Schieberegister ansteuern, die sogenannte Porterweiterung und damit ein LED Lauflicht realisieren

```
void setzePins(int daten[])
{  // Fkt. schiebt alle 8 Bits aus dem Array nacheinander in das Schieberegister
  for(int i=0; i<8; i++)
  {
    resetPins();
    digitalWrite(datenPin, daten[i]);  // Datenelement[i] anlegen
    delay(20);
    digitalWrite(taktPin, HIGH);       // Daten ins Register schieben
    delay(20);
  }  // for
}    // fkt
```

„shiftOut“ **sendet ein BYTE seriell über „datenPin“, getaktet wird mit „taktPin“**
„datenPin“ == Pin-Nummer, die mit Eingang „DS“ verbunden ist
„takPin“ == Pin-Nummer, die mit Eingang „SH_CP“ verbunden ist
„MSBFIRST“ oder „LSBFIRST“ == Konstante die angibt ob Byte von links oder rechts geschoben werden soll)
„wert“ == das Byte das übertragen werden soll

LED_mit_HC595_ShiftOut_Kurs8

```
/*  LED Lauflicht mit HC595
    Schaltet eine Reihe von LEDs nach einem bestimmten Muster ein und aus
    und nutzt dabei ein Schieberegister HC595
    Hier alle digitalen Werte von 0 bis 255 */

int taktPin = 8;          // SH_CP = Schieberegister Takteingang (Shiftregister Clock Input)
int speicherPin = 9;      // ST_CP = Speicherregister Takteingang (Storageregister Clock Input)
int datenPin = 10;        // DS = Serieller Eingang (Serial data Input)
byte wert = 0;

void setup()
{
  pinMode(taktPin,OUTPUT);            // "taktPin" als Ausgang definieren
  pinMode(speicherPin,OUTPUT);        // "speicherPin" als Ausgang definieren
  pinMode(datenPin,OUTPUT);           // "datenPin" als Ausgang definieren
  delay(20);                          // kurze Verarbeitungs-Pause
}

void loop()
{
  wert = 0;
  for (int i=0; i<256; i++)
  {
    digitalWrite(speicherPin, LOW);   //
    delay(20);
    shiftOut(datenPin, taktPin, MSBFIRST, wert);
    digitalWrite(speicherPin, HIGH);  // ST_CP;  --> Speicherregisterinhalt an Ausgang
    wert++;                           // "wert" plus "1"
    delay(500);
  }  // for
  delay(1000);
}    // loop
```

15. Progr.: dig. Ausgänge über Schieberegister ansteuern, sogenannte Porterweiterung und mit Hilfe der „shiftout“ Fkt. beliebige LED-Kombinationen realisieren

1 NIPPLE hat 4 Bit → 0 – 15

0/1	0/1	0/1	0/1

1 BYTE hat 8 Bit (2 Nipple) → 0 – 255

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1

MSB **LSB**

1 WORD hat 16 Bit (2 Byte) → 0 – 65.536

0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	**HB**
0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	**LB**

Funktion Shift „<<„ oder „>>“:
dient zur Bit Manipulation;
„Operand2“ gibt die Anzahl der Stellen an

shiftOut(datenPin, taktPin, MSBFIRST, wert >> 8

Filtern mit der UND Funktion:
„A“ entspricht „MASKE“

A	B	Y = A ∧ B
0	0	0
0	1	0
1	0	0
1	1	1

shiftOut(datenPin, taktPin, MSBFIRST, wert & 255);

Schieberegister 2 (Höherwertiges Byte)
interne Register
2^{15} 2^{14} 2^{13} 2^{12} 2^{11} 2^{10} 2^{9} 2^{8}
0 0 0 0 0 0 0 0
DS
1
Serieller Eingang
Q_H'' Q_H Q_G Q_F Q_E Q_D Q_C Q_B Q_A
Ausgänge
Schieberegister 1 (Niederwertiges Byte)
interne Register
2^{7} 2^{6} 2^{5} 2^{4} 2^{3} 2^{2} 2^{1} 2^{0}
0 0 0 0 0 0 0 0
DS
1
Serieller Eingang
Q_H'' Q_H Q_G Q_F Q_E Q_D Q_C Q_B Q_A
Ausgänge

HIGH
LOW
Schieberegister
Schieberegister
Bitte bis zum Ende durchfahren!
0 1 1 0 0 1 1 0
HIGH
0 1 0 0 0 1 0 1
LOW

Schieberegister 74HC595 ist beliebig kaskadierbar. Hier ein Bsp. Für 16 Ausgänge, also 2 „BYTE“ bzw. ein „WORD“.
Der „QH“ Ausgang, also der Übertrag des „LOW“ Byte wird mit dem Dateneingang des „HIGH“ Byte verbunden. Die Daten werden damit durchgeschoben!

Taktpin „SH_CP“ und Datenübernahmepin „ST_CP“ werden mit einander verbunden

HC595_mal2_PortErweiterung

```
/* Zwei kaskadierte HC595 also 16 Ausgänge. Der Überlauf des 1. HC595 wird an den Datenpin
des 2. HC595 angeschlossen.
Ausgegeben werden Muster und ein 16 Bit Zähler  */

int taktPin = 8;          // SH_CP (11) = Schieberegister Takteingang (Shiftregister Clock Input)
int speicherPin = 9;      // ST_CP (12) = Speicherregister Takteingang (Storageregister Clock Input)
int datenPin = 10;        // DS (14) = Serieller Eingang (Serial data Input)

void setup(){
  pinMode(taktPin,OUTPUT);          // "taktPin" als Ausgang definieren
  pinMode(speicherPin,OUTPUT);      // "speicherPin" als Ausgang definieren
  pinMode(datenPin,OUTPUT);         // "datenPin" als Ausgang definieren
}  // setup

void loop(){
  sendeBytes(0B1111111111111111);       // Zu übertragene Binärzahl
  delay(1000);
  sendeBytes(0B0000000000000000);       // Zu übertragene Binärzahl
  delay(1000);
  sendeBytes(0B1010101010101010);       // Zu übertragene Binärzahl
  delay(1000);
  sendeBytes(0B0101010101010101);       // Zu übertragene Binärzahl
  delay(1000);
  for (int i=0; i<65535; i++)
  {
    sendeBytes(i);                      // eigene Fkt
    delay(100);
  }  // for
}  // loop

// Funktion zum Übertragen der beiden BYTES an den HC595
void sendeBytes(int wert)
{
  digitalWrite(speicherPin, LOW);                   // HC595 Pin ST_CP auf Low
  shiftOut(datenPin, taktPin, MSBFIRST, wert >> 8);  // High Byte 8 Bit nach rechts, dann ausgeben
  shiftOut(datenPin, taktPin, MSBFIRST, wert & 255);  // High Byte löschen und Low Byte ausgeben
  digitalWrite(speicherPin, HIGH);                  // Pin ST_CP auf HIGH (Flanke) --> Wert übernehmen& ausgeben
}  // fct
```

Porterweiterung mal2 (2x HC 595 kaskadiert)

Z1, Z2, Z3, Z4; S1,S2,S3,S4

Spalten
2
1
0
3
2
1
0
Zeilen
1
2
3
4
5
6
7
8
9
*
0
#

Spalten
2
1
0
3
2
1
0
Zeilen
1
2
3
4
5
6
7
8
9
*
0
#

High-Pegel an Draht in Reihe 0

- Abfragen des Pegels an Spalte *0*
- Abfragen des Pegels an Spalte *1*
- Abfragen des Pegels an Spalte *2*

High-Pegel an Draht in Reihe 1

- Abfragen des Pegels an Spalte *0*
- Abfragen des Pegels an Spalte *1*
- Abfragen des Pegels an Spalte *2*

Arduino
13
12
11
10
9
8
7
6
5
4
3
2
1
0
PWM
PWM
PWM
PWM
PWM
PWM
Digital I/O
VCC
GND
Analog IN
5
4
3
2
1
0
10K
R
GND
1
2
3
4
5
6
7
8
A
B
C
D

DasKeyPad | MyKeyPad_Fkts

```
/*
Scope:          Das KeyPad (Digital)
Date:           30.03.2013
Description:    Dieser Sketch zeigt Dir die Ansteuerung eines KeyPads (Digital)
                Die Anzeige des gedrückten Tasters erfolgt im Serial-Monitor
Referenz:       http://arduino.cc/en/Reference/Serial
                http://arduino.cc/en/Reference/If
*/

#define KEY_NOT_PRESSED '-'   // Wird benötigt, wenn keine Taste gedrückt wird

// int row[] = {2, 3, 4, 5};     // Array mit Zeilen Pin-Nummern initialisieren
// int col[] = {6, 7, 8, 9};     // Array mit Spalten Pin-Nummern initialisieren


void setup()
{
  Serial.begin(9600);                 // Serielle Ausgabe vorbereiten
  setKeyPad();
}

void loop()
{
  char myKey = readKey();             // Abfragen des gedrückten Tasters
  if(myKey != KEY_NOT_PRESSED)        // Abfrage, ob irgendein Taster gedrückt
    Serial.println(myKey);            // Ausgabe des Tastenzeichens
}
```

Unser 16. Programm: Ein Tastenfeld ansteuern und die gedrückte Taste auf dem seriellen Monitor anzeigen

DasKeyPad | MyKeyPad_Fkts §

```
// definition aller Tasten
#define KEY_1 '1'
#define KEY_2 '2'
#define KEY_3 '3'
#define KEY_4 '4'
#define KEY_5 '5'
#define KEY_6 '6'
#define KEY_7 '7'
#define KEY_8 '8'
#define KEY_9 '9'
#define KEY_0 '0'
#define KEY_STAR '*'
#define KEY_HASH '#'
#define KEY_A 'A'
#define KEY_B 'B'
#define KEY_C 'C'
#define KEY_D 'D'

long lastValue;                 // Letzte Zeit der millis-Funktion
long debounceTime = 500;        // Prellzeit auf 500ms setzen

int row[] = {2, 3, 4, 5};       // Array mit Zeilen Pin-Nummern initialisieren
int col[] = {6, 7, 8, 9};       // Array mit Spalten Pin-Nummern initialisieren

void setKeyPad()                // Ein- Ausgänge des KeyPad initialisieren
{
  for(int r = 0; r < 4; r++)
    pinMode(row[r], OUTPUT);
  for(int c = 0; c < 4; c++)
    pinMode(col[c], INPUT);
}
```

```
char readKey()          // Funktion zum Ermitteln des gedrückten Tasters auf dem KeyPad
{
  char key = KEY_NOT_PRESSED;     // Standard ist keine Taste ist gedrückt
  for(int r = 0; r < 4; r++)      // Zeilen nacheinander auf HIGH legen
  {
    digitalWrite(row[r], HIGH);   // Zeile auf HIGH
    delay(1);                     // kurz warten
    for(int c = 0; c < 4; c++)    // Spalten nacheinander lesen
    {
      if((digitalRead(col[c]) == HIGH)&&(millis() - lastValue) >= debounceTime)
      {       // wenn eine Taste gedrückt also HIGH und Tastenwiederholzeit abgelaufen
        if((c==3)&&(r==3)) key = KEY_1;     // dann Code der gedrückten Taste
        if((c==2)&&(r==3)) key = KEY_2;     // abhängig von Zeilen- und Spaltennummer
        if((c==1)&&(r==3)) key = KEY_3;     // in Variable setzen
        if((c==0)&&(r==3)) key = KEY_A;
        if((c==3)&&(r==2)) key = KEY_4;
        if((c==2)&&(r==2)) key = KEY_5;
        if((c==1)&&(r==2)) key = KEY_6;
        if((c==0)&&(r==2)) key = KEY_B;
        if((c==3)&&(r==1)) key = KEY_7;
        if((c==2)&&(r==1)) key = KEY_8;
        if((c==1)&&(r==1)) key = KEY_9;
        if((c==0)&&(r==1)) key = KEY_C;
        if((c==3)&&(r==0)) key = KEY_STAR;
        if((c==2)&&(r==0)) key = KEY_0;
        if((c==1)&&(r==0)) key = KEY_HASH;
        if((c==0)&&(r==0)) key = KEY_D;
        lastValue = millis();       // Intervall zurück setzen
      }
    }
    digitalWrite(row[r], LOW);      // Zurücksetzten der Zeile auf Ursprungspegel LOW
    delay(1);                       // kurz warten
  }
  return key;                       // Tastencode zurückgeben
}
```

LCD Hitachi HD44780 (or compatible) has a **parallel interface**, meaning that the microcontroller (ARDUINO) has to manipulate several interface pins at once to control the display. It has the following pins:

Register select (RS) pin: controls where in the LCD's memory you're writing data to. You can select either the **data register**, which holds what goes on the screen, or an **instruction register**, which is where the LCD's controller looks for instructions on what to do next.

Read/Write (R/W) pin: that selects **reading mode** or **writing mode**

Enable pin: that enables writing to the registers, kind of clock

Data pins (D0 -D7): The states of these pins (high or low) are the bits that you are writing to a register when you write, or the values you're reading when you read.

Display constrast pin (Vo), **power supply pins (+5V & GND)** and **LED Backlight (Bklt+ and BKlt-)** pins that you use to power the LCD, control display contrast, and turn on / off the LED backlight.

The process of controlling the display involves putting the data that form the image of what you want to display into the data registers, then putting instructions in the instruction register.

The **LiquidCrystal Library** simplifies this for you so you don't need to know the low-level instructions. The Hitachi-compatible LCDs can be controlled in two modes: **4-bit or 8-bit**. The 4-bit mode requires seven I/O pins from the Arduino, while the 8-bit mode requires 11 pins. For displaying text on the screen, you can do most everything in 4-bit mode, so example shows how to control a 2x16 LCD in 4-bit mode.

LCD Hitachi HD44780

- LCD RS pin to digital pin 12
- LCD Enable pin to digital pin 11
- 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 Hitachi HD44780

Anschluß zusammen mit Keypad,
→ daher geänderte Pinbelegung

LiquidCrystal Library

This library allows an Arduino board to control LiquidCrystal displays (LCDs) based on the Hitachi HD44780 (or compatible) 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, enable, and, optionally, the rw control lines).

Library – Functions:

LiquidCrystal()
begin()
clear()
home()
setCursor()
write()
print()
cursor()
noCursor()
blink()
noBlink()
display()
noDisplay()
scrollDisplayLeft()
scrollDisplayRight()
autoscroll()
noAutoscroll()
leftToRight()
rightToLeft()
createChar()

Beschaltung der meisten HD44780-kompatiblen Anzeigemodule

Pin Nr.	Symbol	Art	Funktion
1	VSS	Stromversorgung	GND (Masse)
2	VCC	Stromversorgung	+ 5V
3	VEE	Stromversorgung	Kontrastspannung
4	RS	Datenleitung	0 = Befehl, 1 = Daten
5	R/W	Datenleitung	0 = schreiben, 1 = lesen
6	E	Datenleitung	Enable
7	D0	Datenleitung	Datenleitung 0
8	D1	Datenleitung	Datenleitung 1
9	D2	Datenleitung	Datenleitung 2
10	D3	Datenleitung	Datenleitung 3
11	D4	Datenleitung	Datenleitung 4
12	D5	Datenleitung	Datenleitung 5
13	D6	Datenleitung	Datenleitung 6
14	D7	Datenleitung	Datenleitung 7
15	A	Stromversorgung	Hintergrundbeleuchtung Anode (+)[B 1]
16	K	Stromversorgung	Hintergrundbeleuchtung Kathode (-)[B 1]

LCD_HelloWorld

```
/*  LCD - Hello World: Demonstrates the use a 16x2 or 20x4 LCD display.
The LiquidCrystal library works with all LCD displays that are compatible with the
Hitachi HD44780 driver. This sketch prints "TEXT" to the LCD and shows the time. */

const int LCD_RS = 16;              // == Analog A2; LCD RS Register Select Pin (4)
const int LCD_Enable = 15;          // == Analog A1; LCD ENABLE Pin (6)
const int LCD_D4 = 10;              // LCD Data4 Pin (11)
const int LCD_D5 = 11;              // LCD Data5 Pin (12)
const int LCD_D6 = 12;              // LCD Data6 Pin (13)
const int LCD_D7 = 13;              // LCD Data7 Pin (14)
#define  ROWS  4                    // Anzahl Zeilen 2 oder 4
#define  COLUMNS 20                 // Anzahl Spalten 16 oder 20

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

LiquidCrystal lcd(LCD_RS, LCD_Enable, LCD_D4, LCD_D5, LCD_D6, LCD_D7);  // initialize library
                                                                        // with nr. of interface pins
void setup()
{
  lcd.begin(COLUMNS, ROWS);          // set up the LCD's number of columns and rows:
  lcd.setCursor(0, 0);               // Cursor to column 1, line 1;
  lcd.print("hello, world!");        // Print a message to the LCD.
  lcd.setCursor(0, 1);               // Cursor to column 1, line 2;
  lcd.print("hello, Esslingen!");    // Print a message to the LCD.
  if (ROWS == 4)                     // 4 Zeilen verfügbar ?
  {
    lcd.setCursor(0, 2);              // Cursor to column 0, line 3;
    lcd.print("hello, ARDUINO's");    // Print a message to the LCD.
  }  // if
}  // setup

void loop()
{
  // (note: line 1 is the second row, since counting begins with 0):
  if (ROWS == 4)                       // grosses LCD?
  {
    lcd.setCursor(0, 3);               // Cursor to column 1, line 4;
    lcd.print(millis()/1000);          // print the number of seconds since reset:
  }  // if
  else {                               // kleines Display
    lcd.setCursor(14, 0);              // Cursor to column 15, line 1;
    lcd.print(millis()/1000);          // print the number of seconds since reset:
  }  // else
  delay(200);                        // kurze Verzögerung
}  // loop
```

Unser 17. Programm: Text und Zeit auf einem LCD Display anzeigen

KeyPad_und_LCD | MyKeyPad_Fkts

```
/* KeyPad (Digital) und  LCD Display.  Dieser Sketch zeigt die Ansteuerung eines
KeyPads (Digital). Die Anzeige der gedrückten Taste erfolgt im LCD Display    */

#include <LiquidCrystal.h>          // include the library code:
#define KEY_NOT_PRESSED '-'         // Wird benötigt, wenn keine Taste gedrückt wird
#define  ROWS  4                    // Anzahl Zeilen 2 oder 4
#define  COLUMNS 20                 // Anzahl Spalten 16 oder 20

const byte LCD_RS = 16;             // == Analog A2; LCD RS Register Select Pin (4)
const byte LCD_Enable = 15;         // == Analog A1; LCD ENABLE Pin (6)
const byte LCD_D4 = 10;             // LCD Data4 Pin (11)
const byte LCD_D5 = 11;             // LCD Data5 Pin (12)
const byte LCD_D6 = 12;             // LCD Data6 Pin (13)
const byte LCD_D7 = 13;             // LCD Data7 Pin (14)

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(LCD_RS, LCD_Enable, LCD_D4, LCD_D5, LCD_D6, LCD_D7);

void setup() {
  Serial.begin(9600);               // Serielle Ausgabe vorbereiten
  setKeyPad();
  lcd.begin(COLUMNS, ROWS);         // set up the LCD's columns & rows, here 20x4 oder 16x2
  lcd.setCursor(0, 0);              // Cursor to column 1, line 1; --> counting begins with 0!
  lcd.print("hello, world!");       // Print a message to the LCD.
  lcd.setCursor(0, 1);              // Cursor to column 1, line 2;
  lcd.print("hello, Esslingen!");   // Print a message to the LCD.
  lcd.setCursor(0, 2);              // Cursor to column 1, line 3;
  lcd.print("hello, ARDUINO's");    // Print a message to the LCD.
  lcd.setCursor(0, 3);              // Cursor to column 1, line 4;
  lcd.print("Sek.: ");              // Print a message to the LCD.
  lcd.setCursor(10, 3);             // Cursor to column 11, line 4;
  lcd.print("Taste: ");             // Print a message to the LCD.
}  // setup

void loop() {
  char myKey = readKey();           // Abfragen des gedrückten Tasters
  if(myKey != KEY_NOT_PRESSED)      // Abfrage, ob irgendein Taster gedrückt
    Serial.println(myKey);          // Ausgabe des Tastenzeichens
  lcd.setCursor(6, 3);              // Cursor to column 7, line 4; --> counting begins with 0)
  lcd.print(millis()/1000);         // print the number of seconds since reset:
  if(myKey != KEY_NOT_PRESSED) {    // Abfrage, ob irgendein Taster gedrückt
    lcd.setCursor(17, 3);           // Cursor to column 18, line 4;
    lcd.print(myKey);               // print the number of seconds since reset:
  }  // if
}  // loop
```

Unser 18. Programm: Ein Tastenfeld und LCD ansteuern und die verstrichene Zeit und die gedrückte Taste auf einem LCD Display anzeigen

KeyPad_und_LCD | MyKeyPad_Fkts

```
// definition aller Tasten
#define KEY_1 '1'
#define KEY_2 '2'
#define KEY_3 '3'
#define KEY_4 '4'
#define KEY_5 '5'
#define KEY_6 '6'
#define KEY_7 '7'
#define KEY_8 '8'
#define KEY_9 '9'
#define KEY_0 '0'
#define KEY_STAR '*'
#define KEY_HASH '#'
#define KEY_A 'A'
#define KEY_B 'B'
#define KEY_C 'C'
#define KEY_D 'D'

long lastValue;                 // Letzte Zeit der millis-Funktion
long debounceTime = 400;        // Tastenwiederholzeit in ms setzen

const byte row[] = {2, 3, 4, 5};        // Array mit Zeilen Pin-Nummern initialisieren
const byte col[] = {6, 7, 8, 9};        // Array mit Spalten Pin-Nummern initialisieren

void setKeyPad()                // Ein- Ausgänge des KeyPad initialisieren
{
  for(int r = 0; r < 4; r++)
    pinMode(row[r], OUTPUT);
  for(int c = 0; c < 4; c++)
    pinMode(col[c], INPUT);
}  // fct
```

```
char readKey()          // Funktion zum Ermitteln des gedrückten Tasters auf dem KeyPad
{
  char key = KEY_NOT_PRESSED;     // Standard ist keine Taste ist gedrückt
  for(int r = 0; r < 4; r++)      // Zeilen nacheinander auf HIGH legen
  {
    digitalWrite(row[r], HIGH);   // Zeile auf HIGH
    delay(1);                     // kurz warten
    for(int c = 0; c < 4; c++)    // Spalten nacheinander lesen
    {
      if((digitalRead(col[c]) == HIGH)&&(millis() - lastValue) >= debounceTime)
      {       // wenn eine Taste gedrückt also HIGH und Tastenwiederholzeit abgelaufen
        if((c==3)&&(r==3)) key = KEY_1;     // dann Code der gedrückten Taste
        if((c==2)&&(r==3)) key = KEY_2;     // abhängig von Zeilen- und Spaltennummer
        if((c==1)&&(r==3)) key = KEY_3;     // in Variable setzen
        if((c==0)&&(r==3)) key = KEY_A;
        if((c==3)&&(r==2)) key = KEY_4;
        if((c==2)&&(r==2)) key = KEY_5;
        if((c==1)&&(r==2)) key = KEY_6;
        if((c==0)&&(r==2)) key = KEY_B;
        if((c==3)&&(r==1)) key = KEY_7;
        if((c==2)&&(r==1)) key = KEY_8;
        if((c==1)&&(r==1)) key = KEY_9;
        if((c==0)&&(r==1)) key = KEY_C;
        if((c==3)&&(r==0)) key = KEY_STAR;
        if((c==2)&&(r==0)) key = KEY_0;
        if((c==1)&&(r==0)) key = KEY_HASH;
        if((c==0)&&(r==0)) key = KEY_D;
        lastValue = millis();       // Intervall zurück setzen
      }  // if
    }  // for
    digitalWrite(row[r], LOW);      // Zurücksetzten der Zeile auf Ursprungspegel LOW
    delay(1);                       // kurz warten
  }  // for
  return key;                       // Tastencode zurückgeben
}  // fct
```

Unser 19. Aufbau: Ein Temperatur gesteuerter Lüfter mit Anzeige

Der **Temperatur Sensor** ist ein **LM35**. Er sieht aus wie ein Transistor und hat auch 3 Beinchen. Die Funktion der Anschlüsse zeigt das Bild.
Der Sensor wandelt die gemessene Temperatur in einen analogen Spannungswert um der proportional zur Temperatur ist.
Der **Meßbereich** ist: 0 – 100°C,
die **Empfindlichkeit** ist: 10mV/C°, d.h. 1V bei 100°C

Ein Lüfter-Motor:

Freilauf-Diode

Ein Leistungs-Transistor

Da die Ausgänge eines Arduino nicht genug Strom liefern können um einen Motor direkt antreiben zu können, muss man ein **Relais** oder einen **Transistor** ansteuern. Der Arduino steuert quasi einen **Schalter**. Hier im Bsp. ein **TIP120**, ein **Darlington Leistungstransistor**. Der kann 5A Strom steuern.

19. Programm: Wir messen die Temperatur und steuern einen Lüfter an, sobald die Temperatur eine **einstellbare Schwelle**, z.B. 25°C übersteigt. Wir zeigen Temperatur und Zustand des Motors (ein/aus) entweder im **LCD Display**, oder über den **seriellen Monitor** oder per „**Processing**“ an.

Die Schaltung:

Motoren bzw. Spulen generell verursachen beim Aus- bzw. Umschalten kurze hohe Stromspitzen. Damit diese hohen Ströme Schalter, Relais oder Transistoren nicht zerstören müssen sie abgeleitet (gelöscht) werden. Diese Funktion übernimmt die „Freilauf-Diode“.

Die Formel zur Temperatur-Berechnung:

- analogPin: Wert vom analog Wandler des Arduino. Liegt zwischen 0 – 1024

$$Temperatur\,[°C] = \frac{5.0 \cdot 100.0 \cdot analogPin}{1024.0}$$

```
int analogValue = analogRead(sensorPin);                    // Sensor einlesen & Analogwert speichern
float temperature = (5.0 * 100.0 * analogValue) / 1024;     // Wert in eine Temperatur umwandeln
```

Das LCD Display

Die drei Anzeige Möglichkeiten:

Der Seriell - Monitor

Anzeige mit dem Programm „Processing“

LM35_Temperatur §

```
/*
Dieser Sketch zeigt Dir die Auswertung der Temperatur über
den Temperatursensor LM 35 und den seriellen Monitor
*/

#define sensorPin 0                    // Verbunden mit LM35 Ausgang
#define DELAY 50                       // kurze Wartezeit
const int cycles = 10;                 // Anzahl der Messungen

void setup()
{
  Serial.begin(9600);                  // Die Schnittstelle auf 9600 Baud einstellen
}  // setup()

void loop()
{
  float resultTemp = 0.0;              // Variable für die berechnete Temperatur
  for(int i = 0; i < cycles; i++)      // mehrmals messen, damit Ergebnis stabiler
  {
    int analogValue = analogRead(sensorPin);         // Sensor einlesen und Analogwert speichern
    float temperature = (5.0 * 100.0 * analogValue) / 1024;  // Wert in eine Temperatur umwandeln
    resultTemp += temperature;                       // Aufsummieren der Messwerte
    delay(DELAY);
  } // for
  resultTemp /= cycles;                // Berechnung des Durchschnittes oder Mittelwerts
  Serial.print("Temperatur:  ");       // Ausgabe an die serielle Schnittstelle
  Serial.print(resultTemp);            // Ausgabe an die serielle Schnittstelle
  Serial.write(0xB0);                  // Ausgabe ASCII Grad Zeichen "°" direkt als HEX Wert
  Serial.println(" C");                // Ausgabe an die serielle Schnittstelle mit neue Zeile
} // loop()
```

Die Hysterese-Funktion

Schaltet man ein Licht, Motor oder Heizung abhängig vom Wert eines Sensors ein und aus, so kann dies unter Umständen sehr häufig und schnell erfolgen, wenn der Wert des Sensors um den eingestellten Wert der Schwelle herum schwankt.
Würde man z.B. eine Heizung ständig aus- und wieder einschalten wäre sie schnell kaputt.

Für dieses Phänomen gibt es die **Hysterese Funktion**

```
if (resultTemp > (threshold + hysterese))        // Temp. Wert größer Schwelle + Hysterese-Wert ?
  digitalWrite(motorPin, HIGH);                  // erst dann einschalten
else if (resultTemp < (threshold - hysterese))   // Temp. Wert kleiner Schwelle - Hysterese-Wert ?
  digitalWrite(motorPin, LOW);                   // erst dann ausschalten
```

Ohne Hysterese:

Mit Hysterese:

DieLueftersteuerung_Kurs §

```
/* Description:   Dieser Sketch zeigt die Ansteuerung eines Lüfters, der
über den Temperatursensor LM35 geregelt wird. Zudem werden Status-Informationen
im angeschlossenen LC-Display oder Monitor angezeigt.
Referenz:       http://www.arduino.cc/en/Reference/Wire
                http://www.arduino.cc/en/Reference/LiquidCrystal
                http://arduino.cc/en/Reference/PinMode
                http://arduino.cc/en/Reference/Constants
                http://arduino.cc/en/Reference/For
                http://arduino.cc/en/Reference/Delay
                http://arduino.cc/en/Reference/If
                http://arduino.cc/en/Reference/Else        */
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#define sensorPin 0                     // Verbunden mit LM35 Ausgang
#define DELAY1 10                       // kurze Wartezeit beim Messen
#define DELAY2 1000                     // kurze Wartezeit beim Anzeigen
#define motorPin 9                      // Lüfter-Pin
#define threshold 25                    // Schalt-Temperatur für Lüfter (25 °C)
#define hysterese 1.0                   // Hysterese-Wert (1.0 °C)
#define LCD FALSE                       // LCD vorhanden dann TRUE, sonst FASLE
const int cycles = 10;                  // Anzahl der Messungen
LiquidCrystal_I2C lcd(0x27, 16, 2);     // Adresse auf 0x27 für 16 Zeichen/2 Zeilen

void setup()
{
  pinMode(motorPin, OUTPUT);            // Motor Pin als Ausgang definieren
  #if (LCD)                             // bedingte Compilierung
    lcd.init();                         // LCD initialisieren
    lcd.backlight();                    // Hintergrundbeleuchtung aktivieren
  #else                                 // bedingte Compilierung
    Serial.begin(9600);
  #endif                                // Ende der bedingte Compilierung
}  // setup()
```

```
void loop()
{
  float resultTemp = 0.0;                          // Float Variable für Temperaturberechnung
  for(int i = 0; i < cycles; i++)                  // Temperatur "cycles" mal messen
  {
    int analogValue = analogRead(sensorPin);       // Sensor LM35 lesen
    float temperature = (5.0 * 100.0 * analogValue) / 1024;  // Sensorwert in Temperaturwert wandeln
    resultTemp += temperature;                     // Aufsummieren der Messwerte
    delay(DELAY1);                                 // kurze Messpause
  }  // for
  resultTemp /= cycles;                            // Berechnung des Durchschnitts (Mittelwert)

  #if (LCD)                      // bedingte Compilierung, Ausgabe an LCD
    lcd.clear();                                   // clear-Methode löscht LCD Inhalt
    lcd.print("Temp: ");                           // print-Methode schreibt LCD Inhalt
    lcd.print(resultTemp);
    lcd.write(0xB0);                               // Grad-Zeichen direkt als Hex-Wert
    lcd.print(" C");
    lcd.setCursor(0, 1);                           // setCorsor-Methode positioniert LCD-Cursor
    lcd.print("Motor: ");
  #else                          // bedingte Compilierung, Ausgabe an Monitor
    Serial.print("Temperatur: ");                  // print-Methode schreibt an Serial
    Serial.print(resultTemp);
    Serial.write(0xB0);                            // Ausgabe ASCII Grad Zeichen "°" direkt als HEX Wert
    Serial.println(" C");
    Serial.print("Motor: ");
  #endif  // Ende der bedingte Compilierung

  if(resultTemp > (threshold + hysterese))         // Temp. Wert größer Schwelle + Hysterese-Wert ?
    digitalWrite(motorPin, HIGH);                  // erst dann einschalten
  else if(resultTemp < (threshold - hysterese))    // Temp. Wert kleiner Schwelle - Hysterese-Wert ?
    digitalWrite(motorPin, LOW);                   // erst dann ausschalten

  #if (LCD)                                        // bedingte Compilierung
    lcd.print(digitalRead(motorPin) == HIGH?"an":"aus");        // LCD vorhanden
  #else
    Serial.println(digitalRead(motorPin) == HIGH?"an":"aus");   // Monitor
  #endif                                           // Ende der bedingte Compilierung

  delay(DELAY2);
}  // loop()
```

Definition von Konstanten

```
int        cycles = 10;        // als Variable, kann im Programm vom Rechner geändert werden, braucht Speicher
#define    cycles = 10;        // Präprozessor Direktive, wird vom Compiler ersetzt,
                                   kann im Programm daher nicht geändert werden, braucht kein Speicher
const int  cycles = 10;        // als Konstante mit TYP Angabe wie eine Variable,
                                   kann im Programm aber nicht geändert werden, braucht keinen Speicher
```

Bedingte Compilierung (abh. von Konstante)

```
#define LCD FALSE                    // LCD vorhanden dann TRUE, sonst FASLE

#if (LCD)                            // bedingte Compilierung
   lcd.print(digitalRead(motorPin) == HIGH?"an":"aus");      // LCD vorhanden
   ...
 #else
   Serial.println(digitalRead(motorPin) == HIGH?"an":"aus");   // Monitor
   ...
 #endif                              // Ende der bedingte Compilierung
```

Ausgabe von Variablen oder ASCII

```
Serial.print(resultTemp);            // Ausgabe abh. vom Typ der Variablen
Serial.write(0xB0);                  // Ausgabe ASCII Grad Zeichen "°" direkt als HEX Wert
```

Unser 20. Aufbau: Töne erzeugen mit einem Piezo Element

Ein **Piezoelement** ist ein Bauteil, das den Piezoeffekt ausnutzt, um entweder durch Anlegen einer elektrischen Spannung eine mechanische Bewegung auszuführen oder bei Einwirkung einer mechanischen Kraft eine elektrische Spannung zu erzeugen

Ein Piezo Element:

Mechanische Kraft auf ein Piezoelement bewirkt eine elektrische Spannung

Piezostapel in Sandwichbauweise in einer Schnittdarstellung. Aufgebaut aus Piezokeramikschichten zwischen jeweils zwei Elektroden (+ und −), zwei Endplatten (oben und unten), einer isolierenden Ummantelung und die Anschlüsse, die nach links (+) und rechts (−) aus der Ummantelung herausragen um über einen Mäander angeschlossen zu werden

Ein Piezo Stapel:

Stapelanordnungen werden als Aktor oder als Generator verwendet.
Als Piezoaktor sind möglichst große Bewegungsamplituden bei kleinen Spannungen erwünscht. Da die relative Längsausdehnung proportional zur elektrischen Feldstärke ist, kann das durch einen kleinen Abstand der Elektroden erreicht werden (Feldstärke = Spannung/Elektrodenabstand). Dies wird bei mehrlagigen Piezoelementen (auch „Multilayer-Elemente“, „Piezostack“ oder „Piezostapel“ genannt) durch Kaskadierung erreicht. Dadurch ergibt sich eine mechanische Reihenanordnung mit Ansteuerung durch eine elektrische Parallelschaltung. Bei einer Schichtdicke des Piezomaterials von 0,2 mm kann man ungefähr eine maximale Spannung von 100–150 V anlegen. Die maximale Auslenkung beträgt ungefähr 0,2 %

delayMicroseconds()

Hält das Programm für die angegebene Zeit in Microsekunden an. Tausend Microsekunden sind eine Millisekunde und 1Million Microsekunden sind 1 Sekunde.

1 Sek = 1000 Milli-Sek = 1.000.000 Micro-Sek.

Aktuell ist der größte Wert 16383. Das kann sich in neueren SW Versionen ändern. Für größere Zeitverzögerungen ist die Fkt. delay() zu verwenden.

Syntax	**delayMicroseconds(us)**	
Parameters	us:	the number of microseconds to pause (*unsigned int*)
Returns	None	

PiezoTon_Kurs

```
/*  Referenz:     http://arduino.cc/en/Reference/PinMode
                  http://arduino.cc/en/Reference/Constants
                  http://arduino.cc/en/Reference/DigitalWrite
                  http://arduino.cc/en/Reference/DelayMicroseconds  */

#define piezoPin 13                // Piezo-Element an Pin 13
#define DELAY 1000

void setup()
{
  pinMode(piezoPin, OUTPUT);       // Piezo Pin als Ausgang programmieren
}

void loop()
{
  digitalWrite(piezoPin, HIGH);    // Piezo Pin einschalten
  delayMicroseconds(DELAY);        // "DELAY" Microsekunden warten
  digitalWrite(piezoPin, LOW);     // Piezo Pin ausschalten
  delayMicroseconds(DELAY);        // "DELAY" Microsekunden warten
}
```

tone()

Erzeugt ein Rechteck-Signal der angegebenen Frequenz (50% Pulsverhältnis) am spezifizierten Pin. Eine Dauer kann angegeben werden, wenn nicht wird der Ton gespielt bis zum Aufruf von noTone().

Der Pin kann mit einem Piezo oder Lautsprecher verbunden werden.

Nur ein Ton kann gleichzeitig erzeugt werden. Falls ein Ton bereits gespielt wird hat der Aufruf von tone() mit einem anderen Pin keine Wirkung. Falls ein Ton bereits spielt kann mit einem erneuten Aufruf aber die Frequenz geändert werden.
Use of the tone() function will interfere with PWM output on pins 3 and 11 (on boards other than the Mega).
Achtung: if you want to play different pitches on multiple pins, you need to call noTone() on one pin before calling tone() on the next pin.

Syntax	**tone(pin, frequency)**	
	tone(pin, frequency, dauer)	
Parameters	pin:	pin an welchem der Ton generiert wird
	frequency:	frequenz in hertz - *unsigned int*
	duration:	Dauer in milliseconds (optional) - *unsigned long*
Returns	nichts	

noTone()

Stops the generation of a square wave triggered by tone().

NOTE: if you want to play different pitches on multiple pins,
you need to call noTone() on one pin before calling tone() on the next pin.

Syntax **noTone(pin)**
Parameters pin: the pin on which to stop generating the tone
Returns nothing

Sizeof()

Der sizeof operator gibt die Anzahl an Bytes zurück, die eine beliebige Variable oder ein Array benötigt.

Syntax **sizeof(variable)**
Parameters variable: any variable type or array (e.g. int, float, byte)
Returns Anzahl von Bytes

Ton bezeichnet ein Schallereignis , das von Musikinstrumenten, der menschlichen Stimme oder elektrisch erzeugt wird und dem eine mehr oder weniger exakte Tonhöhe zugeordnet werden kann. Die physikalischen Eigenschaften eines Tones können als Kombinationen von Sinustönen analysiert und beschrieben werden.

Die Parameter eines Tones:

- die Tonhöhe; sie lässt sich als Frequenz angeben oder in Tonsymbolen oder Noten ausdrücken.
- die Lautstärke ; sie lässt sich akustisch als Amplitude oder als Schalldruck bestimmen, aber auch pragmatisch durch Lautstärkeangaben.
- die Tondauer; sie lässt sich physikalisch-akustisch in Sekunden messen oder praktisch durch Notenwerte angeben.
- die Klangfarbe; sie lässt sich akustisch durch die Obertonzusammensetzung bzw. allgemeiner durch das Frequenzspektrum charakterisieren oder praktisch durch Instrumentangaben und Spielweisen bestimmen.

PiezoTonfolge_Kurs | pitches.h

```
/*  Description:  Dieser Sketch lässt ein angeschlossenes Piezo-Element piepsen
                  Es werden unterschiedliche Töne über ein Tone-Array abgerufen.
Referenz:         http://arduino.cc/en/Reference/Array
                  http://arduino.cc/en/Reference/Tone
                  http://arduino.cc/en/Reference/NoTone
                  http://arduino.cc/en/Reference/Delay  */

#include "pitches.h"
#define piezoPin 13                 // Piezo-Element an Pin 13
#define toneDuration 800            // Ton-Dauer
#define tonePause 1200              // Pausenlänge zwischen den Tönen
#define versuche 3                  // Anzahl der Wiederholungen

// Angabe der Töne mit Namen oder direkt als Frequenz
int tones[] = {NOTE_C2, NOTE_C3, 523, 659, 587, 698, 659, 784, 698, 880, NOTE_C4, NOTE_C5};
int elements = sizeof(tones) / sizeof(tones[0]);

void setup()
{
  noTone(piezoPin);                                 // Piezo stumm schalten
  for(int n = 0; n < versuche; n++)                 // Schleife mit Array-Anzahl "elements"
  {
    for(int i = 0; i < elements; i++)                 // Schleife mit Array-Anzahl "elements"
    {
      tone(piezoPin, tones[i], toneDuration);         // Ton spielen
      delay(tonePause);                               // Pause zwischen den Tönen
    }  // for
  } // for
}  // setup()

void loop()
{  /* leer */
}
```

PiezoTonfolge_Kurs

```
/************************
 * Public Constants
 ************************

#define NOTE_B0  31
#define NOTE_C1  33
#define NOTE_CS1 35
#define NOTE_D1  37
#define NOTE_DS1 39
#define NOTE_E1  41
#define NOTE_F1  44
#define NOTE_FS1 46
#define NOTE_G1  49
#define NOTE_GS1 52
#define NOTE_A1  55
#define NOTE_AS1 58
#define NOTE_B1  62
#define NOTE_C2  65
#define NOTE_CS2 69
#define NOTE_D2  73
#define NOTE_DS2 78
#define NOTE_E2  82
#define NOTE_F2  87
#define NOTE_FS2 93
#define NOTE_G2  98
#define NOTE_GS2 104
#define NOTE_A2  110
#define NOTE_AS2 117
#define NOTE_B2  123
#define NOTE_C3  131
#define NOTE_CS3 139
#define NOTE_D3  147
#define NOTE_DS3 156
#define NOTE_E3  165
#define NOTE_F3  175
#define NOTE_FS3 185
#define NOTE_G3  196
#define NOTE_GS3 208
#define NOTE_A3  220
#define NOTE_AS3 233
#define NOTE_B3  247
#define NOTE_C4  262
#define NOTE_CS4 277
#define NOTE_D4  294
#define NOTE_DS4 311
#define NOTE_G4  392
#define NOTE_GS4 415
#define NOTE_A4  440
#define NOTE_AS4 466
#define NOTE_B4  494
#define NOTE_C5  523
#define NOTE_CS5 554
#define NOTE_D5  587
#define NOTE_DS5 622
#define NOTE_E5  659
#define NOTE_F5  698
#define NOTE_FS5 740
#define NOTE_G5  784
#define NOTE_GS5 831
#define NOTE_A5  880
#define NOTE_AS5 932
#define NOTE_B5  988
#define NOTE_C6  1047
#define NOTE_CS6 1109
#define NOTE_D6  1175
#define NOTE_DS6 1245
#define NOTE_E6  1319
#define NOTE_F6  1397
#define NOTE_FS6 1480
#define NOTE_G6  1568
#define NOTE_GS6 1661
#define NOTE_A6  1760
#define NOTE_AS6 1865
#define NOTE_B6  1976
#define NOTE_C7  2093
#define NOTE_CS7 2217
#define NOTE_D7  2349
#define NOTE_DS7 2489
#define NOTE_E7  2637
#define NOTE_F7  2794
#define NOTE_FS7 2960
#define NOTE_G7  3136
#define NOTE_GS7 3322
#define NOTE_A7  3520
#define NOTE_AS7 3729
#define NOTE_B7  3951
#define NOTE_C8  4186
#define NOTE_CS8 4435
#define NOTE_D8  4699
#define NOTE_DS8 4978
```

Unser 21. Aufbau: Der Digital-Analog Wandler

Das **R2R-Netzwerk** setzt digitale Werte in eine Analogspannung um (Digital-Analog-Umsetzung). Meist billiger und einfacher als eine integrierte Schaltung ist ein R/2R-Widerstandsnetzwerk, evtl. gefolgt von einem Operationsverstärker.

Ein R/2R-Netzwerk ist aus Widerständen mit den Werten R und 2R aufgebaut. Die Eingangsbits liegen entweder auf Masse oder auf der Referenzspannung und speisen über doppelt so große Widerstände (2R) ein wie der horizontale Teil (R) des Netzwerks. Jedes Bit trägt so seinen spezifischen Teil zur resultierenden Ausgangsspannung bei.

Kommerzielle Digital-Analog-Umsetzer haben solche R/2R-Netzwerke in ihrem IC integriert.

Vorteile

- sehr hohe Geschwindigkeit
- einfach zu verstehen
- besteht aus gleichartigen Bauteilen
- ohne Geschwindigkeitsverlust erweiterbar

Nachteile

- die Widerstände müssen möglichst genau gleich sein, insbesondere die für die höherwertigen Bits
- viele Bauteile erforderlich
- das höchste Bit muss (wenn =1) einen hohen Anteil der Ausgangslast treiben

R2R-Netzwerk Prinzipieller Aufbau

MSB
E5
2R
Pin
U
Ausgang
R
E4
E3
E2
E1
LSB
E0
GND

Trick:
Die Hälfte eines Widerstandswertes erreicht man durch „Parallel schalten“

$$U_{Ausgang} = \frac{U_{e5}}{2} + \frac{U_{e4}}{4} + \frac{U_{e3}}{8} + \frac{U_{e2}}{16} + \frac{U_{e1}}{32} + \frac{U_{e0}}{64}$$

$$U_{Auflösung} = \frac{U_{Ref}}{64}$$

Binärwert	Ausgangsspannung
000000	0V
000001	78,13 mV
000010	156,26 mV
000011	234,39 mV
...	...
111111	5V

$$U_{Auflösung} = \frac{U_{Ref}}{64} = \frac{5V}{64} = 78{,}13\ mV$$

DigitalRuftAnalog_Kurs

```
/*  Digital ruft analog
Dieser Sketch zeigt die Digital/Analog-Wandlung über eine Widerstandsleiter
Referenz:       http://arduino.cc/en/Reference/PinMode
                http://arduino.cc/en/Reference/Constants
                http://arduino.cc/en/Reference/For
                http://arduino.cc/en/Reference/DigitalWrite
                http://arduino.cc/en/Reference/BitRead  */


int pinArray[] = {8, 9, 10, 11, 12, 13};     // die 6 Pins des 6BIT DA Wandler
byte R2RPattern;                             // Bit Wert der gewandelt werden soll

void setup()
{
  for(int i = 0; i < 6; i++)
  {
    pinMode(pinArray[i], OUTPUT);             // alle Pins als Ausgang programmieren
  }  // for
  R2RPattern = B000001;                       // Bitmuster zur Ansteuerung der digitalen Ausgänge
} // setup()

void loop(){
  for(int i = 0; i < 6; i++)
  {                                           // wenn Bit = 1 dann Ausgang == HIGH, sonst LOW
    digitalWrite(pinArray[i], bitRead(R2RPattern, i) == 1?HIGH:LOW);
  } // for
} // loop
```

OWON
Stop
M Pos: 0.000ns

OWON
Trig'd
M Pos: 0.000ns

DigitalRuftAnalogRampe_Kurs

```
/*  Digital ruft analog
Dieser Sketch zeigt die Digital/Analog-Wandlung über eine Widerstandsleiter
Referenz:       http://arduino.cc/en/Reference/PinMode
                http://arduino.cc/en/Reference/Constants
                http://arduino.cc/en/Reference/For
                http://arduino.cc/en/Reference/DigitalWrite
                http://arduino.cc/en/Reference/BitRead  */

#define VERSUCHE 5                          // Anzahl der Wiederholungen
#define BITANZAHL 6                         // 6 BIT Wandler
#define RESOLUTION 64                       // 6 Bit Wandler macht 64 Schritte
#define WARTEN 200                          // Zeit zwischen 2 Werten
#define WARTEN2 1000                        // Zeit zwischen 2 Versuchen

int pinArray[] = {8, 9, 10, 11, 12, 13};    // die 6 Pins des 6BIT DA Wandler
byte R2RPattern;                            // Bit Wert der gewandelt werden soll

void setup()
{
  for(int i = 0; i < BITANZAHL; i++)
  {
    pinMode(pinArray[i], OUTPUT);            // alle Pins als Ausgang programmieren
  }  // for
  R2RPattern = B000000;                      // Bitmuster zur Ansteuerung der digitalen Ausgänge
} // setup()

void loop()
{
  for(int m = 0; m < VERSUCHE; m++)
  {
    R2RPattern = B000000;                       // Startwert "0"
    for(int n = 0; n < RESOLUTION; n++)
    {
      for(int i = 0; i < BITANZAHL; i++)
      {                                         // wenn Bit = 1 dann Ausgang == HIGH, sonst LOW
        digitalWrite(pinArray[i], bitRead(R2RPattern, i) == 1?HIGH:LOW);
      } // for
      delay(WARTEN);
      R2RPattern++;                             // Bitwert um eins erhöhen
    } // for
    delay(WARTEN2);
  }  // for
} // loop
```

OWON
Trig'd
M Pos: 0.000ns

DigitalRuftAnalogDreieck_Kurs §

```
/*  Digital ruft analog  Dreieckskurve */

#define VERSUCHE 5                          // Anzahl der Wiederholungen
#define BITANZAHL 6                         // 6 BIT Wandler
#define RESOLUTION 64                       // 6 Bit Wandler macht 64 Schritte
#define WARTEN 200                          // Zeit zwischen 2 Werten

int pinArray[] = {8, 9, 10, 11, 12, 13};    // die 6 Pins des 6BIT DA Wandler
byte R2RPattern;                            // Bit Wert der gewandelt werden soll

void setup()
{
  for(int i = 0; i < BITANZAHL; i++)
  {
    pinMode(pinArray[i], OUTPUT);               // alle Pins als Ausgang programmieren
  }  // for
  R2RPattern = B000000;                         // Bitmuster zur Ansteuerung der digitalen Ausgänge
} // setup()

void loop()
{
  for(int m = 0; m < VERSUCHE; m++)
  {
    R2RPattern = B000000;                       // Startwert "0"
    for(int n = 0; n < RESOLUTION; n++)         // steigende Flanke
    {
      for(int i = 0; i < BITANZAHL; i++)
      {                                         // wenn Bit = 1 dann Ausgang == HIGH, sonst LOW
        digitalWrite(pinArray[i], bitRead(R2RPattern, i) == 1?HIGH:LOW);
      } // for
      delay(WARTEN);
      R2RPattern++;
    } // for
    for(int n = 63; n > 0; n--)                 // Fallende Flanke
    {
      for(int i = 0; i < BITANZAHL; i++)
      {                                         // wenn Bit = 1 dann Ausgang == HIGH, sonst LOW
        digitalWrite(pinArray[i], bitRead(R2RPattern, i) == 1?HIGH:LOW);
      } // for
      delay(WARTEN);
      R2RPattern--;
    } // for
    delay(1000);
  }  // for
} // loop
```

DigitalRuftAnalogSinusKurve_Kurs

```
/*  Dieser Sketch zeigt die Digital/Analog-Wandlung über eine Widerstandsleiter.
 Mit diesem Sketch wird eine Sinuskurve über eine LUT generiert.    */

byte LUT[] =
{
  31, 32, 32, 33, 33, 34, 34, 35, 35, 36, 36, 37, 38, 38, 39, 39, 40, 40,
  41, 41, 42, 42, 43, 43, 44, 44, 45, 45, 46, 46, 47, 47, 48, 48, 49, 49,
  50, 50, 50, 51, 51, 52, 52, 52, 53, 53, 54, 54, 54, 55, 55, 55, 56, 56,
  56, 57, 57, 57, 58, 58, 58, 59, 59, 59, 59, 60, 60, 60, 60, 60, 61, 61,
  61, 61, 61, 61, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62,
  62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 62, 61, 61, 61,
  61, 61, 61, 60, 60, 60, 60, 60, 59, 59, 59, 59, 58, 58, 58, 57, 57, 57,
  56, 56, 56, 55, 55, 55, 54, 54, 54, 53, 53, 52, 52, 52, 51, 51, 50, 50,
  50, 49, 49, 48, 48, 47, 47, 46, 46, 45, 45, 44, 44, 43, 43, 42, 42, 41,
  41, 40, 40, 39, 39, 38, 38, 37, 36, 36, 35, 35, 34, 34, 33, 33, 32, 32,
  31, 30, 30, 29, 29, 28, 28, 27, 27, 26, 26, 25, 24, 24, 23, 23, 22, 22,
  21, 21, 20, 20, 19, 19, 18, 18, 17, 17, 16, 16, 15, 15, 14, 14, 13, 13,
  12, 12, 12, 11, 11, 10, 10, 10, 9, 9, 8, 8, 8, 7, 7, 7, 6, 6, 6, 5, 5,
  5, 4, 4, 4, 3, 3, 3, 3, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6,
  7, 7, 7, 8, 8, 8, 9, 9, 10, 10, 10, 11, 11, 12, 12, 12, 13, 13, 14, 14,
  15, 15, 16, 16, 17, 17, 18, 18, 19, 19, 20, 20, 21, 21, 22, 22, 23, 23,
  24, 24, 25, 26, 26, 27, 27, 28, 28, 29, 29, 30, 30, 31};

void setup()
{
  DDRB = 0b11111111;            // Alle Pins als Ausgang programmiert
}

void loop()
{
  for(int i = 0; i <= 360; i++)
    PORTB = LUT[i];             // Ansteuerung des Port-B Registers
}
```

Wenn es schnell gehen muss. ARDUINO Full Speed nutzen

REGISTER Programmierung ist erforderlich, wenn man die volle Geschwindigkeit des Arduinos ausnutzen will bzw. muss, z.B bei der Ausgabe von schnelleren SINUS-Signalen. Der Arduino hat 3 Register Port B, Port C, Port D

- PORTB
- PORTC
- PORTD

Die Datenrichtung eines jeden **Registers (PORTx)**, also ob Eingang oder Ausgang wird mit einem **Data Direction Register (DDRx)** eingestellt.

Wert	Arbeitsweise
0	Pin arbeitet als Eingang – vergleichbar mit pinMode(pin, INPUT);
1	Pin arbeitet als Ausgang – vergleichbar mit pinMode(pin, OUTPUT);

Gegenüberstellung bisherige Programmierung vs PORT Programmierung macht Vorteil sichtbar

```
digitalWrite(11, HIGH);
digitalWrite(12, HIGH);
digitalWrite(13, HIGH);
```

```
PORTB = 0b00111000;
```

Beispiel

```
void setup(){
  DDRB =  0b11111000; // Pin 8, 9, 10 als INPUT. Pin 11, 12, 13 als
                      // OUTPUT.
  PORTB = 0b00111000; // Pin 11, 12, 13 auf HIGH-Pegel setzen
}
```

ACHTUNG

Es besteht ein nicht unerhebliches Risiko, deinen Mikrocontroller so zu programmieren, dass er anschließend nicht mehr reagiert. Wenn du dir Port-D anschaust, wirst du sicherlich bemerken, dass an *Pin 0* bzw. *Pin 1* die Steuersignale für *RX* bzw. *TX* liegen. *RX* ist für das Empfangen, *TX* für das Senden der Daten verantwortlich. Die Datenflussrichtung ist also folgende: *RX = INPUT, TX = OUTPUT*. Wenn du durch unachtsames Programmieren über *DDRD* diese Werte änderst, kannst du mit Sicherheit keinen Sketch mehr auf dein Arduino-Board übertragen. Du musst dir also ganz sicher, was du hier tust. Überprüfe deinen Sketch lieber dreimal, bevor du ihn an den Mikrocontroller schickst. Nähere Informationen findest du unter *http://www.arduino.cc/en/Reference/PortManipulation*.

Arduino
Digital I/O
Analog IN
VCC
GND
13
12
11
10
9
8
7
6
5
4
3
2
1
0
PWM
5 4 3 2 1 0
330
R
Status LED rot
Status LED gelb
Status LED grün
LED weiß
LED gelb
LED grün
LED rot
Taster weiß
Taster gelb
Taster grün
Taster rot
Piezo-Element
1
2

Arduino
Digital I/O
Analog IN
Vcc
GND
13
12
11
10
9
8
7
6
5
4
3
2
1
0
~PWM
330
R1
R2
R3
R4
R5
R6
R7
LED
Piezo-
Element
Taster weiß
Taster gelb
Taster grün
Taster rot

1. Teil: LEDs und Taster ansteuern.
a) Taster: Vertiefen von „Pull-Down“ und „Pull-Up“

Pull-Down

```
void loop()
{
  tasterStatus = digitalRead(tasterPin);  // Fkt. liest den Zustand des digitalen Pins ein
  if (tasterStatus == HIGH)        // PULL-DOWN  Variante
    digitalWrite(ledPin, HIGH);    // schaltet die LED ein (HIGH ist ein 5V Spannungs-Pegel)
  else
    digitalWrite(ledPin, LOW);     // schaltet die LED aus (LOW ist ein 0V Spannungs-Pegel)
}
```

Arduino „Eingänge“ dürfen nicht offen sein. Man kann einen Taster nicht direkt anschließen, da der Pegel im nicht gedrückten Zustand undefiniert wäre (HIGH oder LOW)

Pull Up

```
void setup()
{
    pinMode(ledPin, OUTPUT);          // LED-Pin als OUTPUT programmieren
    pinMode(tasterPin, INPUT);        // Taster-Pin als INPUT programmieren
    digitalWrite(tasterPin, HIGH);    // Pull-Up Widerstand einschalten
}
void loop()
{
  tasterStatus = digitalRead(tasterPin);  // Fkt. liest den Zustand des digitalen Pins ein
  if (tasterStatus == LOW)         // PULL-UP  Variante
    digitalWrite(ledPin, HIGH);    // schaltet die LED ein (HIGH ist ein 5V Spannungs-Pegel)
  else
    digitalWrite(ledPin, LOW);     // schaltet die LED aus (LOW ist ein 0V Spannungs-Pegel)
}
```

Arduino „Eingänge“ haben einen eingebauten „Pull-Up“ Widerstand v. 20KOhm.

FSS_Tasten_LED_Test§

```
/* Test der 4 Tasten und LEDs
  Jede LED kann per Taster jederzeit ein- und ausgeschaltet werden
*/

#define MAXLED 4                          // Anzahl der LEDs und Taster
int ledPin[] = {2, 3, 4, 5};              // LED-Array mit Pin-Werten
int tasterPin[] = {6, 7, 8, 9};           // Taster-Array mit Pin-Werten
int tasterStatus[] = {LOW, LOW, LOW, LOW};  // Taster-Array mit Status
int tasterStatusOld[] = {LOW, LOW, LOW, LOW};  // Taster-Array mit altem Status
int tasterSumChange;

void setup()
{
  Serial.begin(9600);                     // Serielle Schnittstelle initialisieren (einschalten)
  for(int i = 0; i < MAXLED; i++)
  {
    pinMode(ledPin[i], OUTPUT);           // LED-Pins als OUTPUT programmieren
    pinMode(tasterPin[i], INPUT);         // Taster-Pins als INPUT programmieren
    digitalWrite(tasterPin[i], HIGH);     // Pull-Up Widerstände einschalten
  } // for
}  // fkt
```

```
void loop()
{
  tasterSumChange = LOW;                  // Initialwert keine Taste hat sich geändert
  for(int i = 0; i < MAXLED; i++)         // Schleife über Anzahl MAXLED
  {
    if(digitalRead(tasterPin[i]))         // Array tasterPin[i] lesen
    {                                     // Wenn Pin == HIGH, dann Taste nicht gedrückt, da Taster auf pull down
      tasterStatus[i] = LOW;              // Tastenstatus in einem Array speichern
      digitalWrite(ledPin[i], LOW);       // Array ledPin[i] LOW, d.h. aus
    }  // if
    else                                  // else --> d.h. Pin == LOW, dann Taste gedrückt, da Taster auf pull down
    {
      tasterStatus[i] = HIGH;             // Tastenstatus in einem Array speichern
      digitalWrite(ledPin[i], HIGH);      // Array ledPin[i] HIGH, d.h. ein
    }  // else
    if (tasterStatusOld[i] != tasterStatus[i])  // hat sich Tastenstatus geändert?
    {
      tasterStatusOld[i] = tasterStatus[i];     // dann neuen Status wieder als "Alt" merken
      tasterSumChange = HIGH;                   // es hat sich an 1 - x Taster was geändert
    }  //if
  }  // for

  if (tasterSumChange)                        // nur wenn sich was an einer der Tasten ändert, Status aller Tasten ausgeben
  {
    for(int i = 0; i < MAXLED; i++)           // Status aller MAXLED Tasten ausgeben
    {
      Serial.print("Taste ");                 // Einen Text in einer Zeile zusammenbauen
      Serial.print(i);                        // Nummer der Taste [i]
      Serial.print(" = Status: ");            // Text Status
      Serial.print(tasterStatus[i]);          // Status der Taste [i]
      Serial.print(";    ");                  // Trennzeichen zur nächsten Taste
    }  // for
    Serial.println("");                       // new line für nächste Status Ausgabe
    tasterSumChange = LOW;                    // Ausgabe erledigt, also Änderungsvariable wieder rücksetzen
  } // if
}  // loop
```

FSS_AnalogTasten_LED_Test

```
1  /* Test der 4 Tasten an analog Eingängen und LEDs an digitalen Ausgängen
2    Jede LED kann per Taster jederzeit ein- und ausgeschaltet werden
3    Zustand der Tasten werden SERIAL angezeigt */
4
5  #define MAXLED 4                          // Anzahl der LEDs und Taster
6  #define TST_SCHWELLE 800
7
8  #define LED_PIN_ROT 2
9  #define LED_PIN_GRUEN 3
10 #define LED_PIN_GELB 4
11 #define LED_PIN_BLAU 5
12
13 #define TST_PIN_ROT A1
14 #define TST_PIN_GRUEN A2
15 #define TST_PIN_GELB A3
16 #define TST_PIN_BLAU A4
17
18 int ledPin[] = {LED_PIN_ROT, LED_PIN_BLAU, LED_PIN_GRUEN, LED_PIN_GELB};     // LED-Array mit Pin-Werten
19 int tasterPin[] = {TST_PIN_ROT, TST_PIN_BLAU, TST_PIN_GRUEN, TST_PIN_GELB};  // Taster-Array mit Pin-Werten
20 int tasterStatus[] = {LOW, LOW, LOW, LOW};  // Taster-Array mit Status
21 int tasterStatusOld[] = {LOW, LOW, LOW, LOW};  // Taster-Array mit altem Status
22 int tasterSumChange;
23
24 void setup()
25 {
26   Serial.begin(9600);                     // Serielle Schnittstelle initialisieren (einschalten)
27   for(int i = 0; i < MAXLED; i++)
28   {
29     pinMode(ledPin[i], OUTPUT);           // LED-Pins als OUTPUT programmieren
30   } // for
31 }  // fkt
32
```