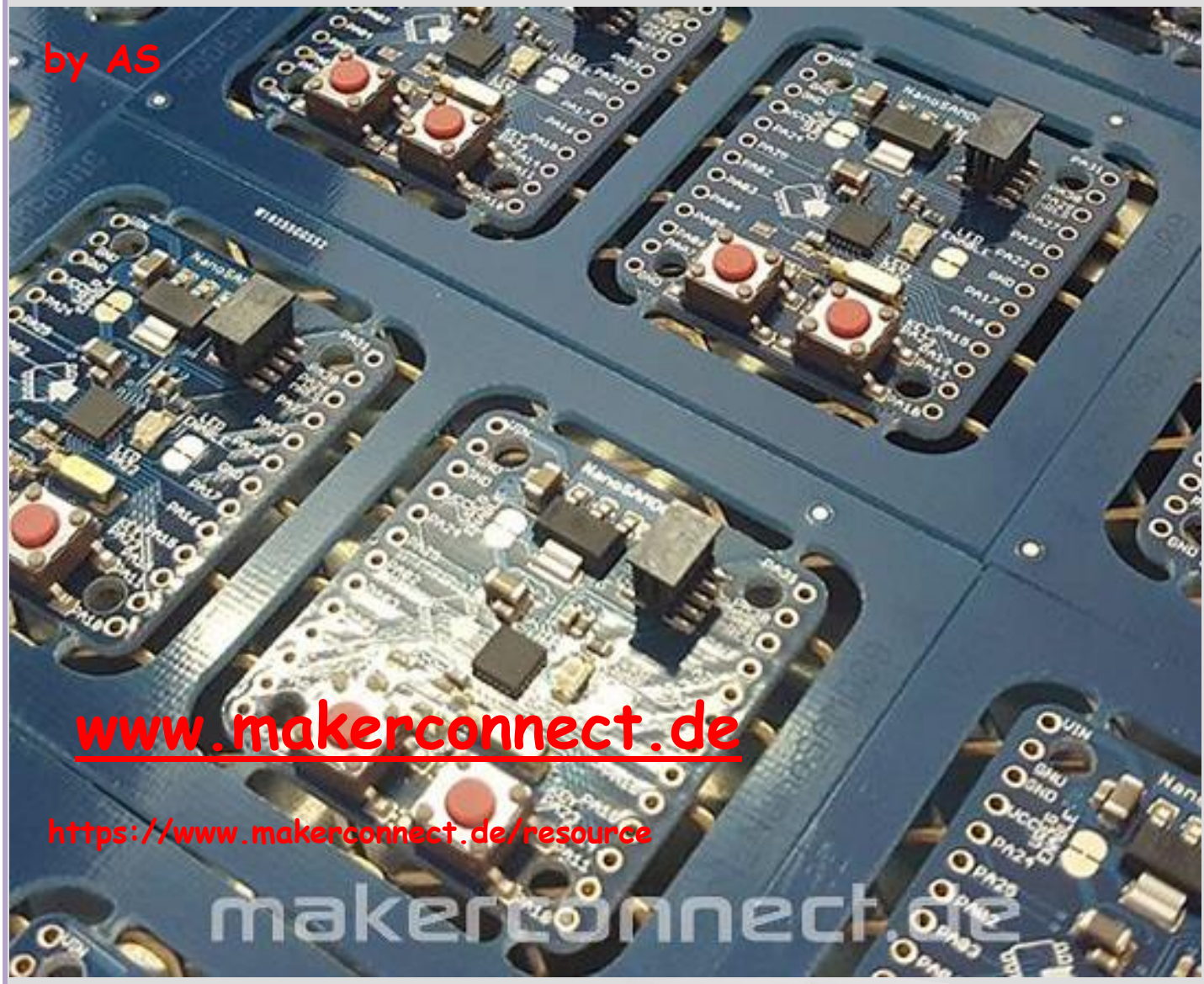
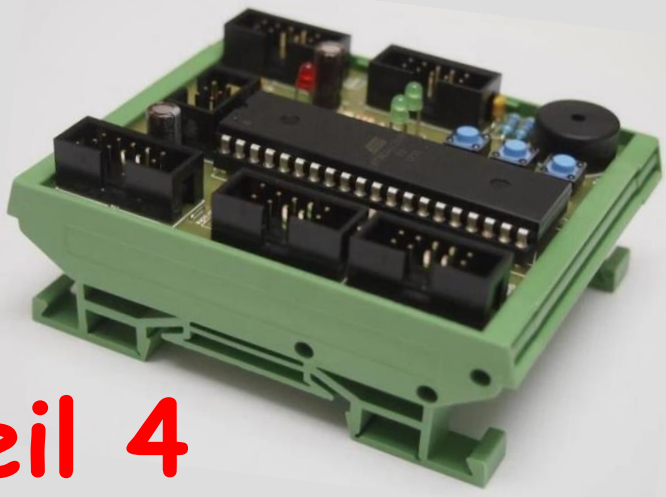
Mikrokontroller & I²C Bus



Prozessor – Board 1 mit dem
AT 1284 P, 3 x Ports,
ISP und 2 x I²C – Bus
= Teil 4 – Ports & Pins =



Board 1 - Teil 4

Copyright

Sicherheitshinweise

Lesen Sie diese Gebrauchsanleitung, bevor Sie diesen Bausatz in Betrieb nehmen und bewahren Sie diese an einem für alle Benutzer jederzeit zugänglichen Platz auf. Bei Schäden, die durch Nichtbeachtung dieser Bedienungsanleitung verursacht werden, erlischt die Gewährleistung / Garantie. Für Folgeschäden übernehmen wir keine Haftung! Bei allen Geräten, die zu ihrem Betrieb eine elektrische Spannung benötigen, müssen die gültigen VDE-Vorschriften beachtet werden. Besonders relevant sind für diesen Bausatz die VDE-Richtlinien VDE 0100, VDE 0550/0551, VDE 0700, VDE 0711 und VDE 0860. Bitte beachten Sie auch nachfolgende Sicherheitshinweise:

- Nehmen Sie diesen Bausatz nur dann in Betrieb, wenn er zuvor berührungssicher in ein Gehäuse eingebaut wurde. Erst danach darf dieser an eine Spannungsversorgung angeschlossen werden.
- Lassen Sie Geräte, die mit einer Versorgungsspannung größer als 24 V- betrieben werden, nur durch eine fachkundige Person anschließen.
- In Schulen, Ausbildungseinrichtungen, Hobby- und Selbsthilfewerkstätten ist das Betreiben dieser Baugruppe durch geschultes Personal verantwortlich zu überwachen.
- In einer Umgebung in der brennbare Gase, Dämpfe oder Stäube vorhanden sind oder vorhanden sein können, darf diese Baugruppe nicht betrieben werden.
- Im Falle eine Reparatur dieser Baugruppe, dürfen nur Original-Ersatzteile verwendet werden! Die Verwendung abweichender Ersatzteile kann zu ernsthaften Sach- und Personenschäden führen. Eine Reparatur des Gerätes darf nur von fachkundigen Personen durchgeführt werden.
- Spannungsführende Teile an dieser Baugruppe dürfen nur dann berührt werden (gilt auch für Werkzeuge, Messinstrumente o.ä.), wenn sichergestellt ist, dass die Baugruppe von der Versorgungsspannung getrennt wurde und elektrische Ladungen, die in den in der Baugruppe befindlichen Bauteilen gespeichert sind, vorher entladen wurden.
- Sind Messungen bei geöffnetem Gehäuse unumgänglich, muss ein Trenntrafo zur Spannungsversorgung verwendet werden
- Spannungsführende Kabel oder Leitungen, mit denen die Baugruppe verbunden ist, müssen immer auf Isolationsfehler oder Bruchstellen kontrolliert werden. Bei einem Fehler muss das Gerät unverzüglich ausser Betrieb genommen werden, bis die defekte Leitung ausgewechselt worden ist.
- Es ist auf die genaue Einhaltung der genannten Kenndaten der Baugruppe und der in der Baugruppe verwendeten Bauteile zu achten. Gehen diese aus der beiliegenden Beschreibung nicht hervor, so ist eine fachkundige Person hinzuzuziehen

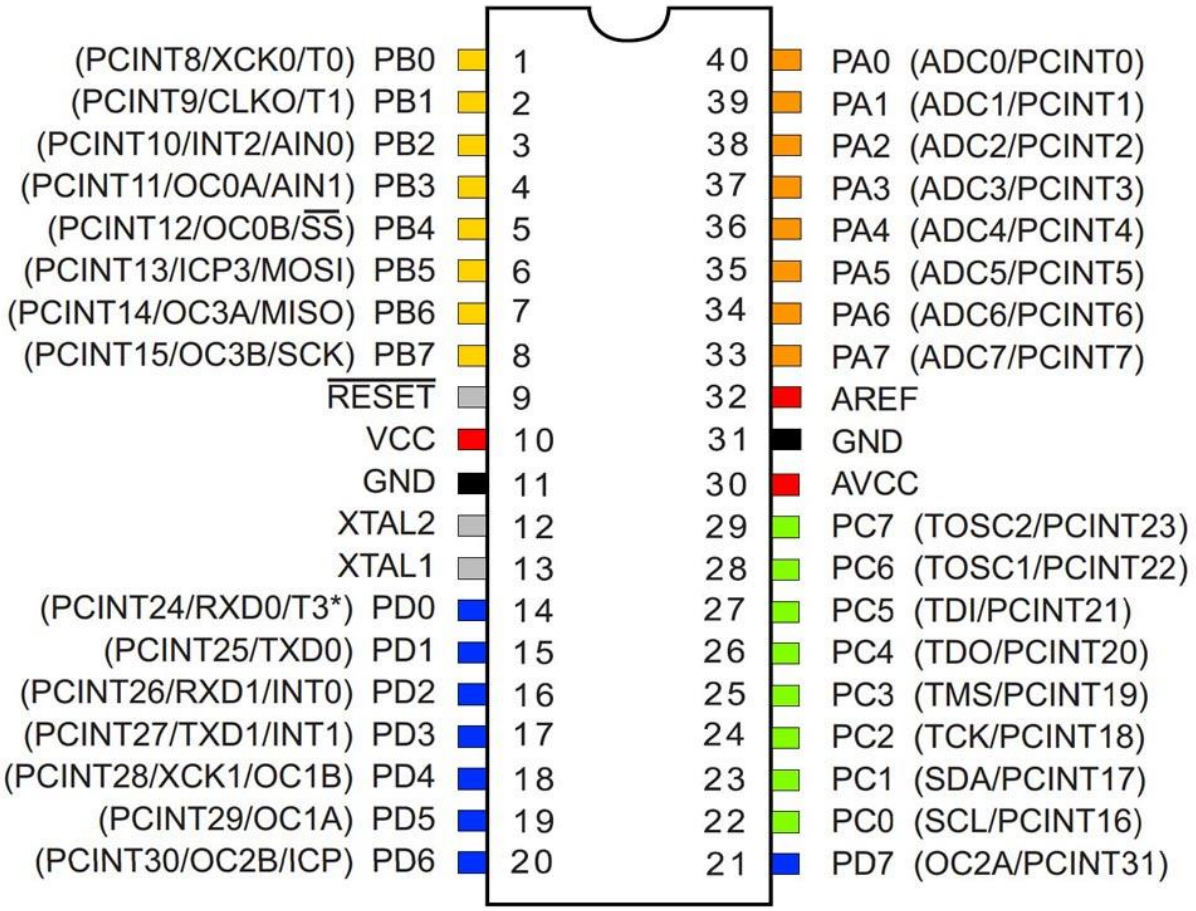
Bestimmungsgemäße Verwendung

- Auf keinen Fall darf 230 V~ Netzspannung angeschlossen werden. Es besteht dann Lebensgefahr!
- Dieser Bausatz ist nur zum Einsatz unter Lern- und Laborbedingungen konzipiert worden. Er ist nicht geeignet, reale Steuerungsaufgaben jeglicher Art zu übernehmen. Ein anderer Einsatz als angegeben ist nicht zulässig!
- Der Bausatz ist nur für den Gebrauch in trockenen und sauberen Räumen bestimmt.
- Wird dieser Bausatz nicht bestimmungsgemäß eingesetzt kann er beschädigt werden, was mit Gefahren, wie z.B. Kurzschluss, Brand, elektrischer Schlag etc. verbunden ist. Der Bausatz darf nicht geändert bzw. umgebaut werden!
- Für alle Personen- und Sachschäden, die aus nicht bestimmungsgemäßer Verwendung entstehen, ist nicht der Hersteller, sondern der Betreiber verantwortlich. Bitte beachten Sie, dass Bedien- und /oder Anschlussfehler außerhalb unseres Einflussbereiches liegen. Verständlicherweise können wir für Schäden, die daraus entstehen, keinerlei Haftung übernehmen.
- Der Autor dieses Tutorials übernimmt keine Haftung für Schäden. Die Nutzung der Hard- und Software erfolgt auf eigenes Risiko.

Board 1 – Teil 4 (Ports & Pins)

Prozessor – Board 1 mit dem ATmega 1284 P, 3 x Ports, ISP, 2 x I^2C – Bus, Taster und LED`s

Auf dem Board 1 habe ich einen Atmega 1284p in der Bauart DIL 40 verwendet. Sehen wir uns nochmal die Anschlüsse des Prozessors an.



Der ATmega 1284P und seine Anschlüsse (Ansicht von oben, auf den IC) (aus WIKI)

Der Prozessor verfügt insgesamt über 4 Ports mit jeweils 8 Pins. Das entspricht 32 möglichen Ein- und Ausgängen. Diese Ports / Pins sind aufgeteilt in

- Port A – PA0 bis PA7 (oben rechts)
- Port B – PB0 bis PB7 (oben links)
- Port C – PC0 bis PC7 (unten rechts)
- Port D – PD0 bis PD7 (unten links)

Die einzelnen Ports habe ich farblich passend dargestellt. Damit ist eine schnelle Zuordnung möglich.
Die nicht genannten Pins 9, 10, 11, 12, 13, 30, 31 und 32 dienen zum Anschluss des Quarzes, Reset-Taster, Betriebsspannung (Vcc), Masse (GND) und AREF.

Was kann ich mit diesen Ports / Pins machen?

Zu jedem Port (je 8 Pins bzw. 8 Bit) gehören drei Register. Diese Register legen die Funktion der entsprechenden Port / Pins fest. Im Einzelnen sind das:

- **DDR** x bestimmt Datenrichtung (Eingang / Ausgang)
- **PORT** x bestimmt Datenregister
- **PIN** x bestimmt Eingangsadresse

Das " **x** " gibt an, zu welchem Port des ATmegas diese Register gehören (Port A, Port B usw.) In den folgenden Beispielen wird Port **C** verwendet.

Schritt 1: Bestimmen der Datenrichtung

DDR x bestimmt dabei die Datenrichtung (Eingang / Ausgang)

Ein Pin des Atmegas kann drei Zustände annehmen:

- Low (0V, Masse)
- High (5V, VCC)
- Tri-State (Eingang hochohmig)

Bei den ersten beiden Möglichkeiten kann man den Zustand des Pins vom Programm aus setzen (High oder Low, also an oder aus), im Tri-State Modus wird von außen ein Pegel angelegt.

Der Pin kann also als ***Ausgang*** oder als ***Eingang*** genutzt werden.

Beim Start des Atmega (Spannung anlegen) sind alle Pins zunächst als **Eingänge** konfiguriert! So können gefahrlos Signale an den Pins anliegen. Wären die Pins als Ausgänge konfiguriert und z.B. auf "**Low**" gesetzt, würden sie alle Eingangssignale kurzschließen und könnten so die Schaltung zerstören. Bei der folgenden Programmierung muss also unbedingt darauf geachtet werden, ob ein Pin wirklich aktiv ein Signal liefern soll.

Möchte man die Datenrichtung vom Programm aus umschalten, eignen sich dazu folgende Funktionen:

```
// Beispiel: DDR für Port C, Bit 0 und Bit 1 werden gesetzt (→ Ausgänge)
DDRC |= (1<<PC0) | (1<<PC1);

// Beispiel: DDR für Port C, Bit 4 wird gelöscht (→ Eingang)
DDRC &= ~(1<<PC4);

// Beispiel: DDR für Port C, Bit 5 und Bit 7 wird gelöscht (→ Eingang)
DDRC &= ~( (1<<PC5) | (1<<PC7) );
```

Ein Nachteil dabei ist die Anzahl der möglichen Pins. Bei 8 Pins muss ich jeden einzelnen Pin in der notwendigen Schreibweise angeben. Das könnte sehr schnell z.B. so aussehen:

```
DDRC |= (1<<PC0) | (1<<PC1 | (1<<PC2 | (1<<PC3 | (1<<PC4 | (1<<PC5 | (1<<PC6 ...  );
```

Leider ist das etwas unübersichtlich und man kann schnell etwas übersehen. Es gibt aber eine andere Schreibweise. Sieht etwas komisch aus, aber ist kurz und übersichtlich.

```
DDRC=0b00000010                    // Port C, PC 1 auf Ausgang setzen
```

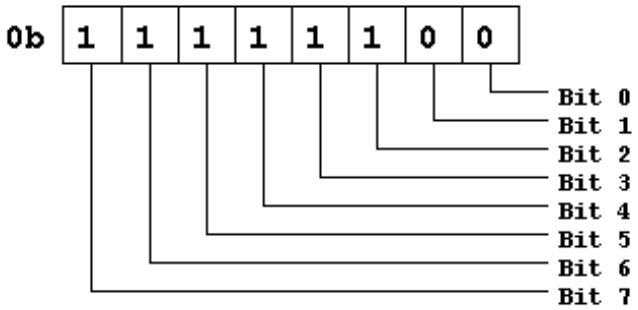
Es wird die Datenrichtung, der Port und jeder einzelne Pin genannt.
Sehen wir es uns genauer an.

DDR C = 0 b 00000010 (Binäre Schreibweise)

- als erstes kommt die Bezeichnung der Datenrichtung - **DDR**
- dann kommt die Angabe des Ports - **C**
- und die Angabe der Pins - **00000010**

Jeder Pin wird dabei da bei nur mit 0 oder 1 dargestellt. Es ist zu beachten, dass von hinten gezählt wird.

Im WIKI habe ich diese Zeichnung gefunden



Man kann sehr gut erkennen, aus welcher Richtung gezählt wird und wo welches Bit / Pin geschaltet wird. Im Beispiel wird Bit 2, 3, 4, 5, 6 und 7 gesetzt.

Schritt 2: Ausgänge schalten

PORT x bestimmt dabei das Datenregister (Ausgang auf Low oder High)

In Abhängigkeit des Programmes können verschiedene Ausgänge auf Low oder High (Aus oder Ein) gesetzt werden. Das kann man z.B.

```
// Port C - Pin 7 auf "High" setzen, das Bit setzen
PORTC |= (1<<PC7);

// Port C - Pin 7 auf "Low" setzen, also das zugehörige Bit löschen
PORTC &= ~(1<<PC7);
```

Im Beispiel ist der Pin 7 als Ausgang konfiguriert. Möchte man diese Ausgangspins jetzt auf High oder Low setzen, muss das entsprechende Bit im Datenregister "PORT" gesetzt bzw. gelöscht werden.

Ein Nachteil dabei ist wieder die Anzahl der möglichen Pins. Bei 8 Pins muss ich jeden einzelne Pin in der notwendigen Schreibweise angeben. Das könnte sehr schnell z.B. so aussehen:

```
PORTC |= (1<<PC0) | (1<<PC1 | (1<<PC2 | (1<<PC3 | (1<<PC4 | (1<<PC5 | (1<<PC6 ...  );
```

Leider ist das etwas unübersichtlich und man kann schnell etwas übersehen. Es gibt aber eine andere Schreibweise. Sieht etwas komisch aus, aber ist kurz und übersichtlich. (Hatten wir schon mal)

```
PORT C =0 b 00000010                    // Port C, PC 1 auf Ausgang setzen
```

Es wird der Port und jeder einzelne Pin genannt. Die genaue Erklärung steht weiter oben. Jeder Port verfügt, in der Regel, über 8 Pins. Diese tragen die Bezeichnung **PC0** bis **PC7**. Die Zählung beginnt dabei mit **0** und geht bis **7**. Dabei ist zu beachten, der **Pin0** ist der **erste** Ein- oder Ausgang.

Bitte beachten:

Für jeden Pin, der als Ausgang verwendet werden soll, muss dabei das entsprechende Bit auf dem Port gesetzt werden. Soll der Pin als Eingang verwendet werden, muss das entsprechende Bit gelöscht sein.

Schritt 3: Eingänge lesen

Der Status (Zustand) eines Ports kann aus dem Register "**Pin**" gelesen werden. Im Beispiel wurde der Pin 4 als Eingang konfiguriert. Dabei ist wieder zu beachten, der Pin kann den Zustand **Low** oder **High**, also 0 oder 1, annehmen.

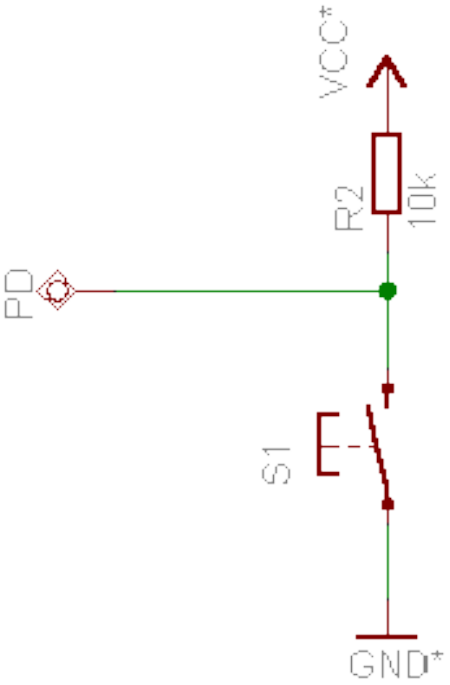
So kann der Status von Pin 4 ausgewertet werden:

```
// Prüfen, ob das Status-Bit für Pin 4 "gelöscht" ist, also 0 ist. Der PIN liegt dann auf Low
        if ( !(PINC & (1<<PINC4)) )
         {
           // Eine Aktion, wenn Pin 4 Low ist
         }
// Prüfen, ob das Status-Bit für Pin 4 "gesetzt" ist, also 1 ist. Der PIN liegt dann auf High
        if (PINC & (1<<PINC4))
         {
           // Eine Aktion, wenn Pin 4 High ist
         }
```

Möchte man Eingänge lesen, muss dazu immer das Register " Pin " verwendet werden.

Pullup-Widerstand

Jeder Eingangspin des Prozessors sollte einen definierten Pegel am Eingang haben. Dazu kann man einen z.B. Widerstand nach Vcc (+5V) legen.
Diesen Widerstand nennt man einen **Pullup**-Widerstand.
Wenn der Taster geöffnet ist, so ist es seine Aufgabe, den Eingangspegel am Pin auf Vcc zu ziehen. Daher auch der Name: 'pull up' (engl. für hochziehen).
Ohne diesen Pullup-Widerstand würde ansonsten der Pin bei geöffnetem Taster in der Luft hängen, also weder mit Vcc noch mit GND verbunden sein. Dieser Zustand ist aber unbedingt zu vermeiden, da bereits elektromagnetische Einstreuungen auf Zuleitungen ausreichen, dem Pin einen Zustand vorzugaukeln, der in Wirklichkeit nicht existiert.
Der Pullup-Widerstand sorgt also für einen definierten High-Pegel bei geöffnetem Taster.
Wird der Taster geschlossen, so stellt dieser eine direkte Verbindung zu GND her und der Pegel am Pin fällt auf GND.
Durch den Pullup-Widerstand fließt ein kleiner Strom von Vcc nach GND. Da Pullup-Widerstände in der Regel aber relativ hochohmig sind, stört dieser kleine Strom meistens nicht weiter.
Der empfohlene Wert liegt bei ca. 4,7 KOhm bis 10 KOhm

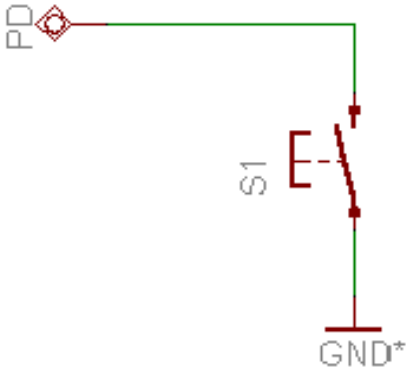


Standard Taster Anschluss

Interne Pull-Up Widerstände

Pin`s für Ein- und Ausgänge eines AVR verfügen über zuschaltbare interne Pullup Wider-

stände (nominal mehrere 10 kOhm, z. B. ATmega16 20-50 kOhm). Diese können in vielen Fällen statt externer Widerstände genutzt werden.
Die internen PullUp Widerstände von Vcc zu den einzelnen Pins werden über das Register **PORTx** aktiviert bzw. deaktiviert, wenn ein Pin als **Eingang** geschaltet ist.
Wird der Wert des entsprechenden Portpins auf 1 gesetzt, so ist der Pull-Up Widerstand aktiviert. Bei einem Wert von 0 ist der Pull-Up Widerstand nicht aktiv. Man sollte jeweils entweder den internen oder einen externen Pull-Up Widerstand verwenden, aber nicht beide zusammen.



Taster bei Benutzung des interen Pullup

Im Beispiel werden alle Pin`s des Ports C als Eingänge geschaltet und alle Pull-Up Widerstände aktiviert. Weiterhin wird Pin PC7 als Eingang geschaltet und dessen interner Pull-Up Widerstand aktiviert, ohne die Einstellungen für die anderen Portpins (PC0-PC6) zu verändern.

```
#include <avr/io.h>
...
DDRC  = 0x00;          /* alle Pins von Port D als Eingang */
PORTC = 0xff;          /* interne Pull-Ups an allen Port-Pins aktivieren */
...
DDRC  &= ~(1<<PC7);    /* Pin PC7 als Eingang */
PORTC |= (1<<PC7);     /* internen Pull-Up an PC7 aktivieren */
```

Bitte beachten:

Ist ein Pin als Eingang konfiguriert, kann über das Register "PORT" ein Pull-Up Widerstand geschaltet werden. Liest man das "PORT"-Register aus, liest man lediglich den Status der Pullup-Widerstände und **nicht** den Status des Pins!
Ein Pull-Up Widerstand sorgt für einen definierten Pegel an einem Pin, wenn keine externen Bauteile angeschlossen sind. "Pull-Up" bedeutet hier "nach oben ziehen". Der unbeschaltete Pin wird auf die Höhe der Betriebsspannung "gezogen", also High-Pegel.

Beispiel:

```
/* ATB_B1_Prg_6.c Created: 27.12.2014 19:24:39 Author: AS */

# define F_CPU 16000000UL          // Angabe der Quarzfrequenz, wichtig für die Zeit
# include <avr/io.h>               // Ausgänge
# include <util/delay.h>           // Zusätzlich Datei für Pause
int main (void)                    // Hauptprogramm
 {
// An Port A sind 2 LEDs und ein Taster angeschlossen. Die LEDs sind vom positiven Pol der
// Spannungsquelle geschaltet, der Taster zieht den Eingang bei Betätigung auf Masse
   DDRA |= (1<<PA4) | (1<<PA5);    // LED1+2 auf PA4 und PA5 auf Ausgang
   // Datenrichtung für den Eingang
   DDRA &= ~(1<<PA1);              // Taster 1 auf PA 1 auf Eingang
```

```
  while ( 1 )                          // Programm läuft in Endlosschleife
   {          // Prüfen, ob der Taster betätigt wurde, der Pin liegt dann auf 0 (Low)
    if ( !(PINA & (1<<PINA1)) )
     {                                 // LED an PA 4 aus, PA 5 an
      PORTA &= ~(1<<PA4);
      PORTA |= (1<<PA5);
     }
    else
     {                                 // LED an PA 4 an, Pin 5 aus
      PORTA |= (1<<PA4);
      PORTA &= ~(1<<PA5);
     }
   }
}
```

Unbeschaltete Pins

Meistens werden nicht alle Pins des Controllers genutzt. Unbeschaltete (bzw. unprogrammierte) Pins sind standardmäßig als hochohmige Eingänge definiert und damit anfällig für Störeinflüsse. Solche Störungen führen nicht sofort zum Absturz des Controllers, können aber schwer auffindbare Fehler ("zufällig" kippende Bits) verursachen. Daher sollte man die offenen Pins auf einen definierten Zustand bringen. Dazu bieten sich mehrere Möglichkeiten an:

1. Auf der Platine eine Verbindung zu VCC oder GND herstellen
2. Interne Pullups aktivieren
3. Pin als Ausgang konfigurieren und auf VCC oder GND schalten

Für Testaufbauten ist die zweite Möglichkeit zu empfehlen: Der Pin liegt auf einem definierten Pegel, versehentlich angeschlossene Eingangssignale werden aber nicht kurzgeschlossen. Möchte man sich nicht auf die sehr hochohmigen Pullups verlassen, sollte man die dritte Option wählen oder externe Widerstände anbringen (z.B. 10 kOhm nach Vcc). Nicht zu empfehlen ist die erste Variante: Durch die feste Verbindung ist der Pin später nicht mehr benutzbar, ohne Leiterbahnen zu trennen. An einem offenen Pin kann man leicht Ergänzungen anlöten. Außerdem besteht hier die Gefahr, den Pin bei einem Programmierfehler als Ausgang zu konfigurieren und kurzzuschließen.

LED an einem Pin

Mit einer LED kann ich am besten den Ausgang eines Pins darstellen. Dazu eignen sich die LED`s in den Farben Rot, Grün und Gelb am besten. Nach Angabe im Netz haben die LED folgende Durchlassspannung:

Rot – 1,8 V
Gelb – 2,0 V
Grün – 2,2 V

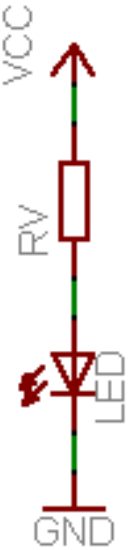


Bild aus WIKI
Anschluss LED an Pin

Bei jeweils 20 mA. Die genauen Daten bitte den Datenblättern der Hersteller entnehmen.

Sehen wir uns die kleine Schaltung genauer an. Sie besteht aus einem Vorwiderstand und einer LED. Der Anschluss erfolgt dabei an Vcc (+5V) und geht nach Masse (GND). Eine LED kann nur mit einem Vorwiderstand betrieben werden um den Strom durch die LED zu begrenzen. In Abhängigkeit der verwendeten Farbe der LED werden unterschiedliche Spannungen benötigt. Bei den Farben Rot, Gelb und Grün liegen sie bei ca. 2V. Die Ausgänge unserer Prozessoren vertragen max. 20 mA.
Wie gross muss der Vorwiderstand für eine LED an einem Pin sein?

$$R_V = \frac{V_{CC} - U_{LED}}{I_{LED}} = \frac{5,0V - 2,0V}{20\,mA} = 150\ Ohm$$

R_V – Vorwiderstand in Ohm
V_{CC} – Betriebsspannung in Volt
U_{LED} – Durchlassspannung in Volt
I_{LED} – Strom durch die LED in mA (A)

Bei einer Betriebsspannung von +5V und einer roten, gelben oder grünen LED mit einer Durchlassspannung von ca. 2V bei einem LED-Strom von max. 20 mA muss der Vorwiderstand von mindestens 150 Ohm haben. Ich habe einen Widerstandswert von **220 Ohm** gewählt.

$$I_{LED} = \frac{V_{CC} - U_{LED}}{R_V} = \frac{5,0V - 2,0V}{220\,Ohm} = 13,6\,mA$$

Bei einem Vorwiderstand von **220 Ohm** fliesst ein Strom von max. **13,6 mA** durch die LED. In dieser Rechnung habe ich den Spannungsabfall über den Transistor im Controller vernachlässigt. Nach den Angaben im Datenblatt, soll dieser max. 0,46V betragen.

Pins toggeln

So kann ein Pin getoggelt bzw. sein Status invertiert werden:

```
PORTA ^= (1<<PA4);
```

Dabei wird der PA4 beim betätigen des Tasters zwischen H und L bzw. L und H geschaltet. In einem Programm habe ich es dargestellt und mit einer „normalen Abfrage eines Tasters verglichen.

```
// Prüfen, ob der Taster 1 betätigt wurde
if ( !(PINA & (1<<PINA1)) )
 {
   PORTA ^= (1<<PA4);                    // toggelt PA4
 }
```

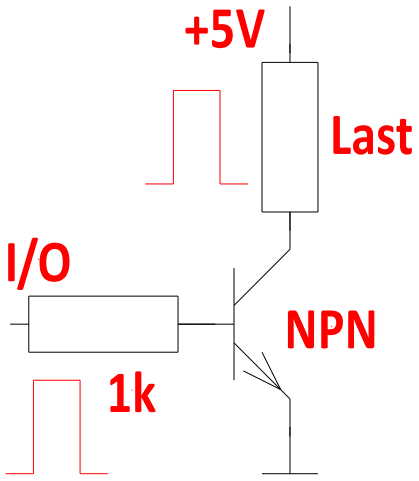
Beim Toggeln drehen wir die Polarität eines Ausgangs immer nur einfach um, ohne eine Sicherheit darüber zu haben, ob sich daraus eine 1 oder eine 0 ergibt. Es hängt immer vom vorherigen Zustand ab. Wenn wir eine klare Abhängigkeit eines Ausgangs haben wollen, dann ist eine klare Zuordnung besser.

Erweiterungen von Pins

Ein Ausgang unseres Prozessors verträgt nach Angabe des Herstellers max. 20 mA. Das reicht für eine „normale" LED aus.
Es gibt auch andere LEDs, die max. 2 mA verbrauchen.
Ich kann den Ausgang auch mit einem NPN-Transistor beschalten und dadurch einen höheren Strom schalten.
Als Transistor kann ich z.B. denn

BC 547 bis zu **200 mA** oder den
BC 337 bis zu **800 mA** verwenden.

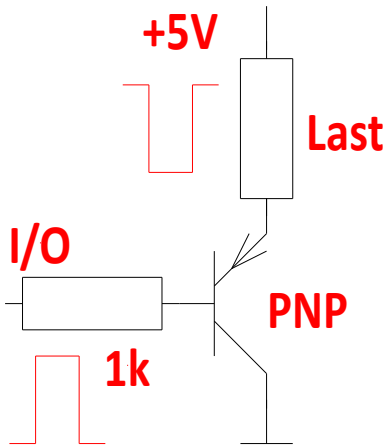
In dieser Schaltung erfolgt die Ausgabe gleichphasig



Bei dieser Schaltung verwende ich einen PNP-Transistor.
Dabei kann ich z.B. die Transistoren

BC 557 bis zu **200 mA** oder den
BC 327 bis zu **800 mA** verwenden.

In dieser Schaltung erfolgt die Ausgabe gegenphasig. Das bedeutet, dass das Signal zwischen Eingang und Ausgang gedreht wird

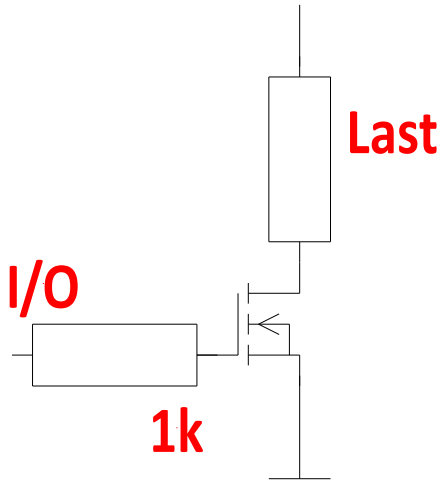


Es sind auch andere Transistoren mit höherer Leistung möglich. Dabei muss aber der Kühlkörper berücksichtigt werden.

Als nächsten Typ kann man einen Mosfet verwenden. Diese eignen sich gut für Anwendungen, bei denen hohe Ströme mit niedrigen Spannungen geschaltet werden müssen. Am besten eignen sich dazu Logic Level Fets. Diese schalten bereits bei 5V Ansteuerung nahezu voll durch.

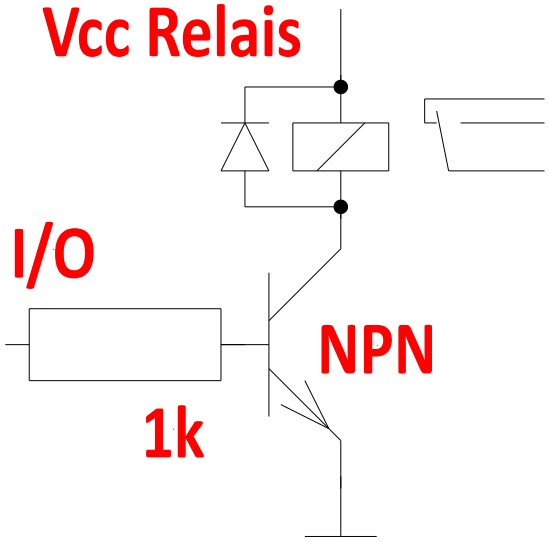
Als Beispiel dazu:

Typ:	IRL 3705N (TO220)	NDS 355 (SMD)
Spannung:	55 V	30 V
Strom:	77 A	1,7A

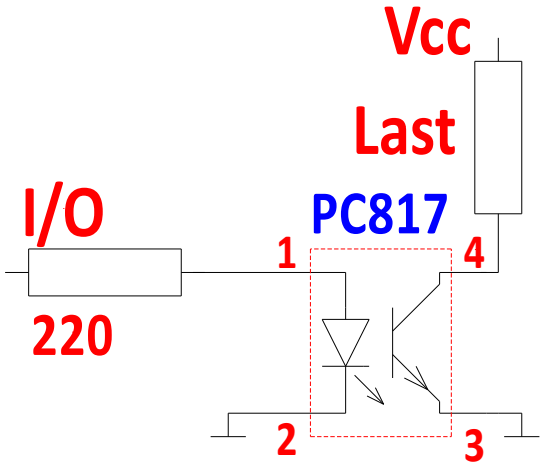


Es gibt noch viele andere Typen

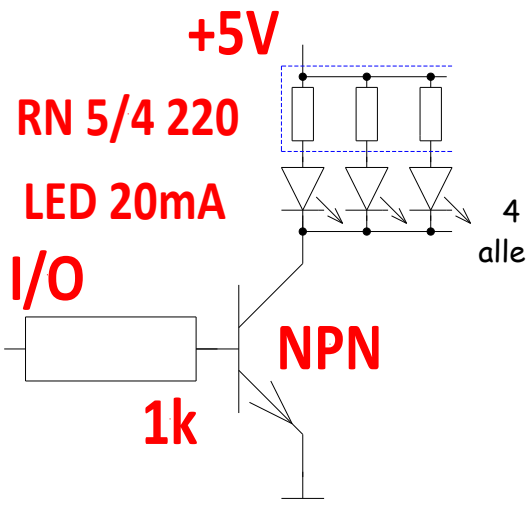
Eine besondere Herausforderung stellt eine galvanische Trennung dar. Dabei darf der Steuerkreis und der Lastkreis nicht mit einander verbunden sein. Weder GND (Masse) oder Vcc dürfen verbunden sein.
Ein Relais eignet sich sehr gut dazu. Dabei ist aber die Leistung und Steuerspannung unbedingt zu beachten. Weiterhin ist eine Freilaufdiode vorzusehen. Das ist die kleine Diode links neben dem Relais. Eine Anzeige der Funktion kann ebenfalls vorgesehen werden. Dabei ist wieder die Betriebsspannung des Relais zu beachten.



Eine Trennung zwischen Steuer- und Lastkreis kann ich auch mit einem Optokoppler machen.
Ein Optokoppler ist eine kleiner IC mit ca. 4 oder 6 Beinen. In seinem inneren befindet sich eine LED und ein lichtempfindlicher Transistor. Da zwischen befindet sich eine spannungsfeste Isolierung. Leider können Optokoppler keine hohen Ströme direkt schalten. Sie haben eine sehr kleine Bauform und können sich mehrfach in einem Gehäuse befinden. Bitte die genauen Daten den entsprechenden Datenblättern entnehmen.



Mit einem Transistor kann ich auch mehrere LED schalten. Dabei sollte jede LED einen extra Vorwiderstand haben. Sehr gut machen sich Widerstandsnetzwerke (RN). Diese gibt es in den Bauarten 5/4 und 9/8. Das bedeutet, das sich jeweils oder 8 Widerstände in einem Gehäuse befinden und eine gemeinsame Verbindung
haben und dieser nach aussen geführt ist.
Dadurch kann ich den gemeinsamen Kontakt auf Vcc (+5V) legen und 4 bzw. 8 LED anschalten und gemeinsam mit einen Transistor schalten.
Dabei ist der Strom zu beachten.



Zum besseren Verständnis habe ich noch eine Tabelle mit den Zahlen in Dezimal, Hexadezimal und Binär eingefügt.

Vergleistabelle zwischen Dezimal, Hexadezimal und Binär

Dezimal	Hexadezimal	Binär
0	0x00	0b00000000
1	0x01	0b00000001
2	0x02	0b00000010
3	0x03	0b00000011
4	0x04	0b00000100
5	0x05	0b00000101
6	0x06	0b00000110
7	0x07	0b00000111
8	0x08	0b00001000
9	0x09	0b00001001
10	0x0A	0b00001010
11	0x0B	0b00001011
12	0x0C	0b00001100
13	0x0D	0b00001101
14	0x0E	0b00001110
15	0x0F	0b00001111
100	0x64	0b01100100
255	0xFF	0b11111111

Einige Teile des Textes wurden zur besseren Übersicht farblich gestaltet.
Die Nutzung erfolgt auf eigenes Risiko.
Ich wünsche viel Spaß beim Bauen und programmieren
Achim

myroboter@web.de