Mikrokontroller & I²C Bus

by AS

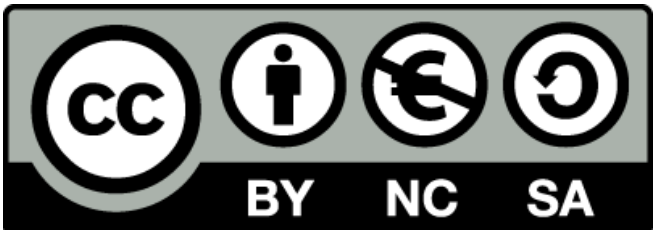
www.makerconnect.de

https://www.makerconnect.de/resource

Der Attiny 841 – ein IC
und verschiedne Module
Teil 3 - Ein- und Ausgänge

I²C Bus und der Attiny 841

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 Versorgungs-spannung 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 Spannungsversor-gung 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 unverzüg-lich 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

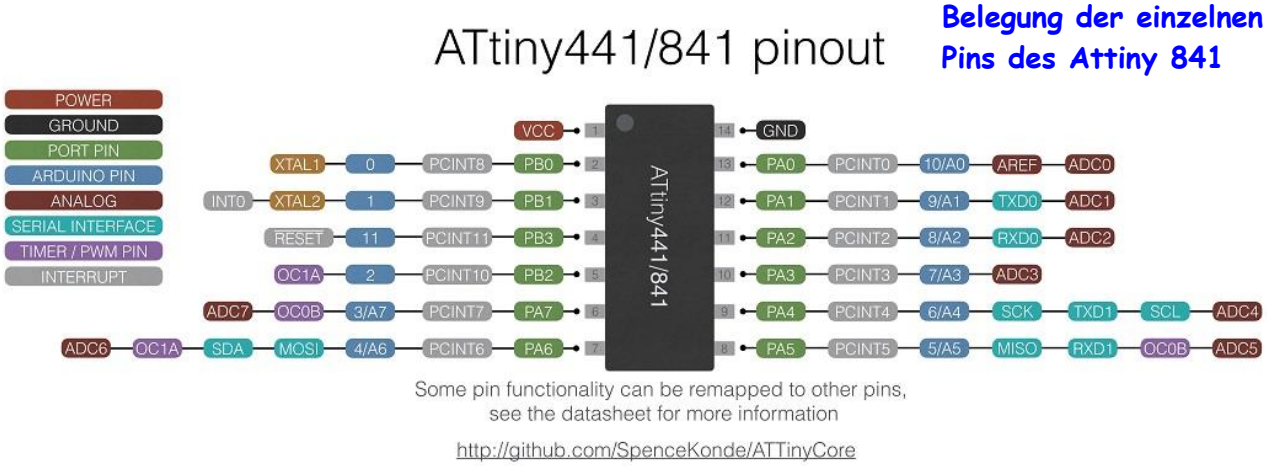
Bestimmungsgemäße Verwendung

- 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 Ge-fahren, 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.

Attiny 841 – Ein- und Ausgänge

Im ersten Teil habe ich verschiedene Module mit Attiny 841 mit Aufbau und Platinen gezeigt. Beim zweiten Teil habe ich die Inbetriebnahme mit den notwendigen Einstellungen des ATtiny841 und die ersten Programme gezeigt. In diesem Teil werde ich konkret auf einzelne Module eingehen und verschiedene Funktionen näher erklären.

Belegung der einzelnen Pins des Attiny 841

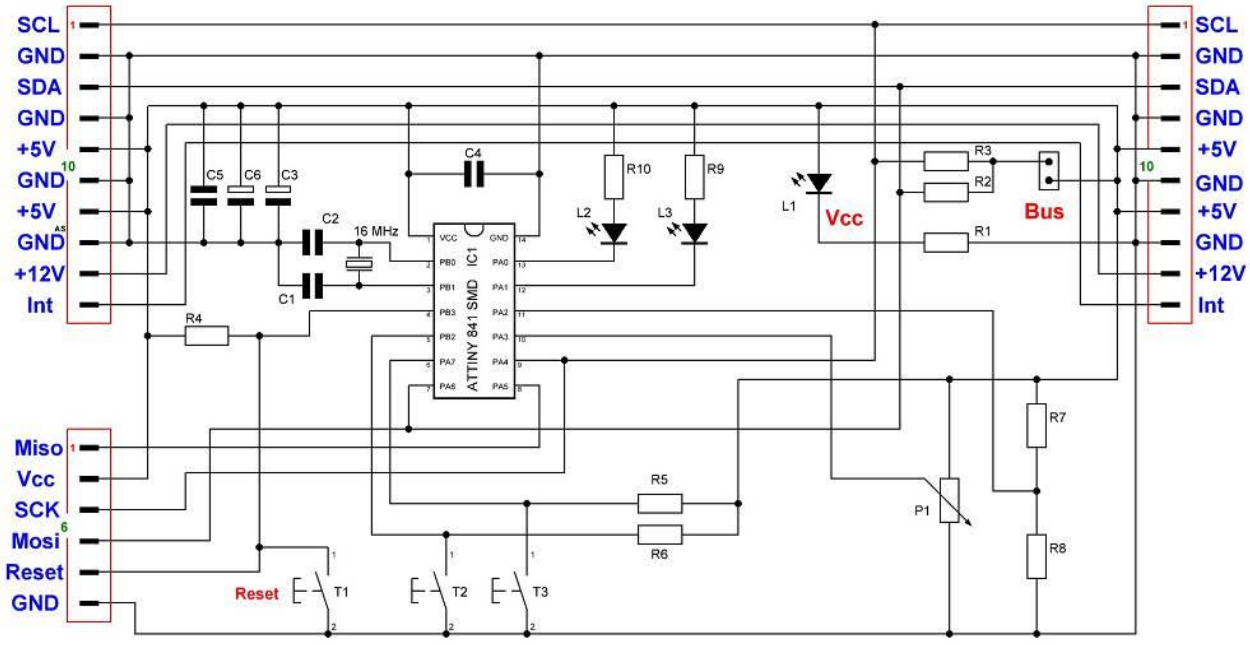


Der Attiny 841 verfügt über 14 Pins, die verschieden genutzt werden können. Jeder Pin hat eine feste Funktion oder kann durch eine entsprechende Programmierung genutzt werden.

Pin 1 – Vcc, Pin 2 – PB0, Pin 3 – PB1, Pin 4 – PB3, Pin 5 – PB2, Pin 6 – PA7, Pin 7 – PA6
Pin 8 – PA5, Pin 9 – PA4, Pin 10 – PA3, Pin 11 – PA2, Pin 12 – PA1, Pin 13 – PA0, Pin 14 – GND

Versorgung	Pin **1** – Vcc, Pin **14** – GND
Port A	PA**0**, PA**1**, PA**2**, PA**3**, PA**4**, PA**5**, PA**6**, PA**7**
Port B	**(**PB**0** und PB**1** für den Quarz**)**, PB**2**, PB**3**

Jedes PIN kann unterschiedliche Aufgaben haben.



Schaltung P173 mit Attiny 841 und Quarz, 2 Taster, 2 LEDs, Poti

Stückliste P173:

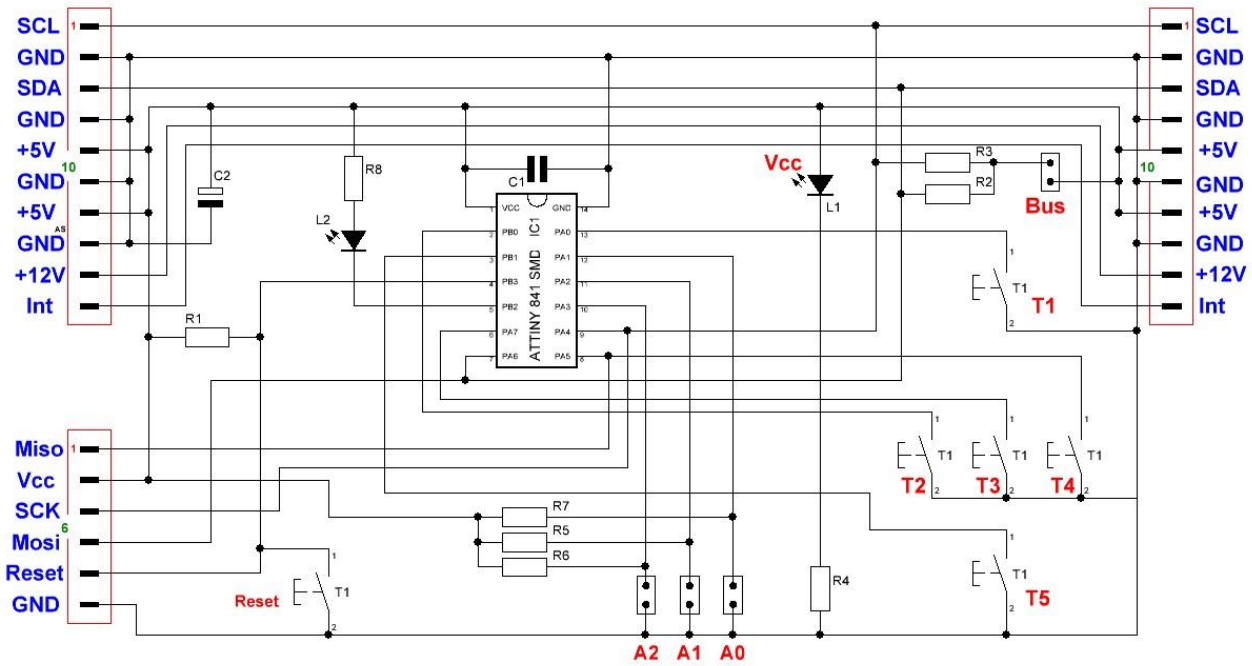
2 x Wannenstecker 2 x 5 RM 2,54
1 x Wannenstecker 2 x 3 RM 2,54
R1 – Widerstand 220 Ohm
R4 – Widerstand 10 Kilo Ohm
R7, R8 – Widerstand 4,7 Kilo Ohm
C1, C2 – Kondensator 15 pF
C4, C5 – Kondensator 100 nF
L1 – LED 3/5 mm 20 mA
1 x Jumper
1 x Platinen Halterung mehrteilig
2 x Taster 2 polig
1 x Poti (Einstellregler) 10 Kilo Ohm
1 x Platine (P173) ca. 42 x 72 mm
1 x Attiny 841 (SMD)
R2, R3 – Widerstand 4,7 Kilo Ohm
R5, R6 – Widerstand 47 Kilo Ohm
R9, R10 – Widerstand 1,7 Kilo Ohm
C3 – Elko 100/16
C6 – Elko 470/16
L2, L3 – LED 3/5 mm 2 mA
1 x Stecker 2 polig
1 x Taster 1 polig
1 x Quarz 16 MHz

Belegung der einzelnen Pins bei der Platine P173:

Versorgung	Quarz	Reset	AC/ADC	Taster	LED	I²C Bus
Pin **1** – Vcc	PB **0**	PB **3**	PA **2**	PA **7**	PA **0**	PA **4**
Pin **14** – GND	PB **1**		PA **3**	PB **2**	PA **1**	PA **6**

Somit können für die Programmierung der Platine P173 nur die I/O Pins PA0, PA1, PA7 und PB2 verwendet werden. Einige Pins werden zur Programmierung mit dem ISP verwendet.
Den ADC werde ich in einem anderen Beitrag beschreiben.

Eine relativ ähnliche Platine habe ich mit 5 Eingängen und ohne Quarz aufgebaut.



Schaltung P177 mit Attiny 841, ohne Quarz, 5 Taster, 2 LEDs

Die Stecker A0, A1 und A2 werden in einem anderen Programm als Adressstecker für den I²C Bus verwendet.

Stückliste P177:

2 x Wannenstecker 2 x 5 RM 2,54
1 x Wannenstecker 2 x 3 RM 2,54
R1 – Widerstand 10 Kilo Ohm
R4 – Widerstand 220 Ohm
R8 – Widerstand 1,7 Kilo Ohm
C2 – Elko 100/16
L2 – LED 3/5 mm 2 mA
4 x Stecker 2 polig
6 x Taster

1 x Platine (P177) ca. 42 x 72 mm
1 x Attiny 841 (SMD)
R2, R3 – Widerstand 4,7 Kilo Ohm
R5, R6, R7 – Widerstand 47 Kilo Ohm
C1 – Kondensator 100nF
L1 – LED 3/5 mm 20 mA
4 x Jumper
1 x Platinen Halterung mehrteilig

Belegung der einzelnen Pins bei der Platine P177:

Versorgung	Reset	Taster	Ausgang	Busadressen	I²C Bus
Pin **1** – Vcc	PB **3**	PA **0** – T1	PB **2** – LED2	PA 1 - A0	PA **4**
Pin **14** – GND		PB **0** – T2		PA 2 - A1	PA **6**
		PA **7** – T3		PA 3 - A2	
		PA **5** – T4			
		PB **1** – T5			

Somit können für die Programmierung der Platine P177 die I/O Pins PA**0**, PB**0**, PA**7**, PA**5**, PB**1** und PB**2** verwendet werden. Einige Pins werden zur Programmierung mit dem ISP bzw. I²C Bus verwendet.

Im Datenblatt, auf der Seite 246, in der Tabelle werden 25-13 werden für verschiedene Ausgänge unterschiedliche Belastungen angegeben.

Abgesehen vom Reset-Pin verträgt jedes Bein:

- **10mA bei 5V** Spannungsversorgung (Sink und Source)
  (PA0, PA1, PA2, PA3, PA4, PA6, PB0, PB1, PB2)
- 5mA bei 3V Spannungsversorgung (Sink und Source)
- 2mA bei 1,8V Spannungsversorgung (Sink und Source)

Abweichend davon können A5 und A7 (High Sink Pins) mehr Strom aufnehmen/nach GND ableiten:

- **20mA bei 5V** Spannungsversorgung (High Sink) **(PA5, PA7)**
- 10mA bei 3V Spannungsversorgung (High Sink)
- 4mA bei 1,8V Spannungsversorgung (High Sink)

Im Port High Drive Enable Register (PHDE) können diese beiden Pins sogar (jeder für sich) für extra "große" Ströme aufgeschaltet werden (Extra High Sink), die sie dann aufnehmen können:

- **20mA bei 5V** Spannungsversorgung (Extra High Sink) **(PA5, PA7)**
- 20mA bei 3V Spannungsversorgung (Extra High Sink)
- 8mA bei 1,8V Spannungsversorgung (Extra High Sink)

Für die LEDs benutze ich nur Exemplare mit einem Strom von 2mA mit einem Vorwiderstand von 1,7 Kilo Ohm. Den max. erlaubten Strom bei 5V habe ich gekennzeichnet.
Nach Angabe des Herstellers darf der Summenstrom des Attiny 841 von Port A und Port B 100mA nicht überschreiten.

Jeder Port-Pin besteht aus vier Registerbits: **DDRxn, PORTxn, PUExn und PINxn**.

Das **DDRxn**-Bit im **DDRx**-Register wählt die Richtung dieses Pins aus. Wenn **DDRxn** als logische **Eins** geschrieben wird, wird **PINxn** zum **Ausgangspin**.
Wenn **DDRxn** mit logischer **Null** geschrieben wird, wird **PINxn** als **Eingangspin** konfiguriert.

Wenn **PORTxn** als logisch **Eins** gesetzt wird und **PINxn** als Ausgangspin konfiguriert ist, wird der Port-Pin auf **High** (**Eins**) gesetzt.

Wenn **PORTxn** als logische **Null** gesetzt wird und **PINxn** als Ausgangspin konfiguriert ist, wird der Port-Pin auf **Low** (**Null**) gesetzt.

Wenn ein Pull-up-Freigabebit, **PUExn**, gesetzt wird, wird der **Pull-up**-Widerstand am entsprechenden Port-Pin, **PINxn**, aktiviert.

DDR, PORT, PUE, PIN – Registerbits
x – Angabe der Ports (**A, B**)
n – Angabe der PINS (**0-7**)

Binärlogik: die Bitoperatoren

Logische Bitoperatoren

Die logischen Bitoperatoren sind:

- **&** logisches UND
- **|** logisches ODER
- **^** exklusives ODER (XOR)
- **~** Negation (NICHT)

Ein wesentlicher Unterschied zu den gewohnten Operatoren wie z.B. Plus oder Minus ist, dass die Bitoperatoren bitweise angewendet werden.

Bitoperator UND (&)

Der Bitoperator UND prüft, ob die beiden Operanden 1 (**true**) sind. Ist das der Fall, dann ist das Ergebnis 1, sonst 0 (**false**).

1. 0 & 0 = 0
2. 1 & 0 = 0
3. 0 & 1 = 0
4. 1 & 1 = 1

Anwendung auf ein ganzes Byte:

1. 0b10011100 & 0b01010111 = 0b00010100

Bitoperator ODER (|)

Der Bitoperator ODER prüft, ob *mindestens* einer der beiden Operanden 1 (**true**) ist. Wenn ja, dann ist das Ergebnis 1, sonst 0.

1. 0 | 0 = 0
2. 1 | 0 = 1
3. 0 | 1 = 1
4. 1 | 1 = 1

Anwendung auf ein ganzes Byte:

1. 0b10011100 | 0b010101111 = 0b11011111

Bitoperator XOR (^)

Der Bitoperator XOR prüft, ob *genau* einer der beiden Operanden 1 ist. Wenn ja, ist das Ergebnis 1, sonst 0. Im Gegensatz zum ODER liefert dieser Operator also auch dann 0, wenn beide Operanden 1 sind.

1. 0 ^ 0 = 0
2. 1 ^ 0 = 1
3. 0 ^ 1 = 1
4. 1 ^ 1 = 0

Anwendung auf ein ganzes Byte:

1. 0b10011100 ^ 0b010101111 = 0b11010011

Bitoperator NICHT (~)

Der Bitoperator NICHT hat nur einen Operanden. NICHT kehrt den Wert um, aus 1 (true) wird also 0 (false) und umgekehrt.

1. ~0 = 1
2. ~1 = 0

Anwendung auf ein ganzes Byte:

1. ~0b10011100 = 0b01100011

Auf Shiftoperatoren werde ich in diesem Tut nicht eingehen.

Sehen wir und als erstes die Belegung des Ports A noch einmal an:

PA0, PA1, PA2, PA3, PA4, PA5, PA6, PA7

Das ist die « normale » Zählweise. Für die Programmierung wird es anders gezählt.
Die Zählweise erfolgt hierbei von rechts nach links:

PA7, PA6, PA5, PA4, PA3, PA2, PA1, PA0

Als erstes wollen wir PA0 und PA1 im Register DDR als Ausgang setzen und den PIN auf High.

```
DDRA=0b00000011;          // Register DDR A, PA1 und PA0 auf Ausgang schalten
```

In diesem Beispiel wird im Register DDR A PA1 und PA0 auf Ausgang geschaltet. PA2 bis PA7 bleiben als Eingang.

```
PORTA=0b00000011;         // Register Port A, PA1 und PA 0 auf ein High schalten
```

In diesem Beispiel wird im Register PORT A PA1 und PA0 auf **High (1)** geschaltet. PA2 bis PA7 bleiben **Low (0)**.

Auf die einzelnen Register und deren genaue Programmierung werde ich nicht eingehen. Bitte die Daten dazu dem Datenblatt des Herstellers entnehmen.

Durch die Einbindung (#include) der Datei avr/io.h werden gerätespezifische E/A definiert

```
#include <avr/io.h>                    // Einbindung Datei Ausgänge
```

Beispiel für Schreibweisen:

// Setzen der Bits 0, 1, 2, 3 und 4, Binaer 00011111 = Hexadezimal 1F oder direkte Zuweisung

```
DDRA = 0x1F;
```

// Uebersichtliche Alternative – Binaerschreibweise

```
DDRA = 0b00011111;
```

// Ausfuehrliche Schreibweise: identische Funktionalitaet, mehr Tipparbeit
// aber uebersichtlicher und selbsterklaerend:

```
DDRA |= (1 << DDA0) | (1 << DDA1) | (1 << DDA2) | (1 << DDA3) | (1 << DDA4);
```

Die Pins 5 bis 7 werden (da 0) als Eingänge geschaltet.

// Alle Pins des Ports A als Ausgang definieren

```
DDRA = 0xff;
```

// Pin0 wieder auf Eingang und andere im ursprünglichen Zustand belassen:

```
DDRA &= ~(1 << DDA0);
```

// Pin 3 und 4 auf Eingang und andere im ursprünglichen Zustand belassen:

```
DDRA &= ~((1 << DDA3) | (1 << DDA4));
```

// Pin 0 und 3 wieder auf Ausgang und andere im ursprünglichen Zustand belassen:

```
DDRA |= (1 << DDA0) | (1 << DDA3);
```

// Alle Pins auf Eingang:

```
DDRA = 0x00;
```

Es können auch Ports geschaltet werden, z.B. „Einschalten" und „Ausschalten", also Ausgänge auf „**High**" oder „**low**" setzen, erfolgt analog:

```
PORTA |= (1<<PINA2);                    // setzt Bit 2 in PORTA und damit Pin PA2 auf high
PORTA &= ~( (1<<PINA4) | (1<<PINA5) );          // Pin PA4 und Pin PA5 "low"
```

Die Schreibweise `DDRA = 0b00011111;` hat leider einen Nachteil. Es werden gleichzeitig alle Bits verändert, egal ob sie gebraucht werden oder nicht. Mit der Schreibweise
`DDRA |= (1 << DDA0) | (1 << DDA1) | (1 << DDA2) | (1 << DDA3) | (1 << DDA4);`
werden nur die gewünschten Bits geändert. Alle anderen behalten ihren Wert.

Einsatz der Binärlogik bei der Portmanipulation

Selektives Setzen von Bits

Um nun ein einzelnes oder mehrere Bits *selektiv* zu setzen, verwendet wir das logische ODER. Folgendermaßen könnt ihr zum Beispiel **PA0** und **PA1** auf **HIGH** zu setzen, ohne die restlichen Pins des PORTA zu beeinflussen:

```
PORTA |= 0b00000011
```
es werden alle Bits beeinflusst, besser mit:

```
PORTA |= (1<<PINA2)|(1<<PINA0);
```

Es sind mit #define verschiedene Einstellungen im Programm möglich. Zum Beispiel:

```
#define led1 PINA0
#define led2 PINA1
```

Dadurch ist zwar eine vereinfachte Schreibweise möglich, gut lesbar wäre das natürlich nicht.

PORTA |= (1 << led1) | (1 << led1); da
(1<<PA0) | (1<<PA1) = 0b00000001 | 0b00000010 = 0b00000011

Selektives Löschen von Bits

Auch das selektive Löschen von Bits ist sehr einfach, es erschließt sich aber vielleicht nicht unbedingt auf den ersten Blick. Ich nehme wieder PA1 als Beispiel:

PORTA &= ~(1<<PINA1), gleichbedeutend mit:
PORTA &= ~(0b00000010) bzw. PORTA &=0b11111101

Ihr könnt natürlich auch mehrere Bits gleichzeitig löschen, hier zum Beispiel PA0 und PA1:

PORTA &= ~((1<<PINA1)|(1<<PINA0))

Selektives Invertieren von Bits

Für das Invertieren einzelner Bits eignet sich das logische XOR. Dabei macht man sich zunutze, dass ein ^1 aus einer 0 eine 1 macht und umgekehrt. Ein ^0 hingegen verhält sich neutral. So z. B. invertiert ihr PB5 selektiv:

PORTA ^= (1<<PINA0)

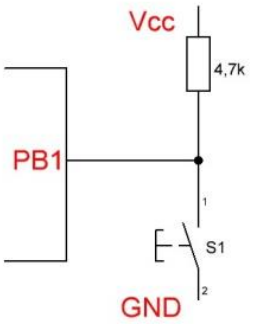
Oder für mehrere Bits, die invertiert werden sollen:

PORTA ^= (1<<PINA0)|(1<<PINA1)

Wenn ihr einen ganzen Port invertieren wollt, kommen zwei Varianten infrage:

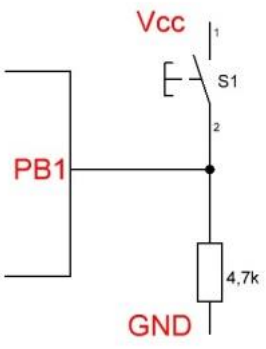
PORTB = ~PORTB oder PORTB ^= 0b11111111

Selektives Abfragen von Pin-Zuständen



Bei der Abfrage von Pins gibt es zwei verschiedene Möglichkeiten:

Active Low: Bei dieser Methode wird der Kontakt zwischen den Eingangspin des Controllers und Masse geschaltet. Damit bei offenem Schalter der Controller kein undefiniertes Signal bekommt, wird zwischen die Versorgungsspannung und den Eingangspin ein sogenannter **Pull-Up** Widerstand geschaltet. Dieser dient dazu, den Pegel bei geöffnetem Schalter auf logisch 1 zu ziehen. (Auf der Platine P173)



Active High: Hier wird der Kontakt zwischen die Versorgungs-spannung und den Eingangspin geschaltet. Damit bei offener Schalterstellung kein undefiniertes Signal am Controller ansteht, wird zwischen de Eingangspin und die Masse ein **Pull-Down** Widerstand geschaltet. Dieser dient dazu, den Pegel bei geöffneter Schalterstellung auf logisch 0 zu halten.

Der Widerstandswert von **Pull-Up**- und **Pull-Down**-Widerständen ist an sich nicht kritisch. Wird er allerdings zu hoch gewählt, ist die Wirkung eventuell nicht gegeben. Als üblicher Wert haben sich 10 kOhm eingebürgert. Die AVRs verfügen an den meisten Pins über zuschaltbare interne **Pull-Up** Widerstände, welche insbesondere wie hier bei Tastern und ähnlichen Bauteilen (z. B. Drehgebern) statt externer Bauteile verwendet werden können.

Im ersten Beispiel (**Pull-Up**) wird der PIN PB1 durch den Taster auf GND gelegt. Dadurch ergibt sich bei dieser Abfrage:

```
PINB & (1<<PINB2)                    // Abfrage PB2 wahr
```

Wird der Taster betätigt, liefert der Ausdruck 1, also **true** (wahr), wenn PB1 LOW ist. Ist PB1 HIGH, dann ist der Ausdruck 0, also **false** (falsch).

Auszug aus meinem Programm zur Platine 173:

```
if (PINB & (1<<PINB2))       // Taster T2 abfragen
 {                           // Wenn T2 gedrückt...
   PORTA |=(1<<led1);        // LED 1  PINA0 ein
 }
else
 {                           // wenn nicht
   PORTA &=~(1<<led1);       // LED 1  PINA0 aus
 }
```

Der PIN B2 ist in der Schaltung P173 über den Widerstand R6 auf Vcc gelegt. Solange der Taster T2 nicht schaltet liegt am PB2 Vcc (5V) an. Wird der Taster T2 geschaltet wird PB2 auf GND (Masse) gelegt.

Im zweiten Beispiel (**Pull-Down**) wird der PIN PB1 durch den Taster auf Vcc gelegt. Dadurch ergibt sich bei dieser Abfrage:

```
(!(PINB & (1<<PINB2))                // Abfrage PB2 nicht wahr
```

Wird der Taster betätigt, liefert der Ausdruck 0, also **false** (falsch), wenn PB1 LOW ist. Ist PB1 HIGH, dann ist der Ausdruck 1, also **true** (wahr).

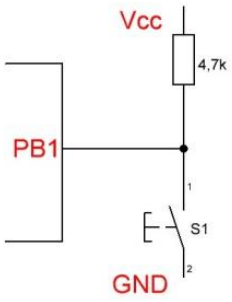
```
if (!(PINB & (1<<PINB2)))    // Taster T2 abfragen
 {                           // Wenn T2 nicht gedrückt...
   PORTA |=(1<<led1);        // LED 1  PINA0 ein
 }
else
 {                           // wenn nicht
   PORTA &=~(1<<led1);       // LED 1  PINA0 aus
 }
```

Der PIN B2 wird in der Schaltung P177 über einen **interne** Widerstand auf Vcc gelegt. Solange der Taster T2 nicht schaltet liegt am PB2 Vcc (5V) an. Wird der Taster T2 geschaltet wird PB2 auf GND (Masse) gelegt. Im Grund erfolgt damit die gleiche Funktion wie auf der P173.

Je nach verwendeter Hardware und Funktion des Tasters kann eine Version genutzt werden.

Werden die Widerstände nicht durch **PUEx** eingeschaltet haben die Taster keine Funktion.

Das PUExn Register



Mit dem **PUExn** Register werden für jedes Port bzw. Pin die internen Widerstände eingeschaltet.
Damit bei offenem Schalter der Controller kein undefiniertes Signal bekommt, wird zwischen die Versorgungsspannung und den Eingangspin ein sogenannter **Pull-Up** Widerstand intern geschaltet.
Wenn ein **Pull-Up**-Freigabebit, **PUExn**, gesetzt wird, wird der **Pull-Up**-Widerstand am entsprechenden Port-Pin, **PINxn**, aktiviert.

In diesem Beispiel werden die entsprechend die **Pull-Ups** von PortA, PA7 und PortB PB2 aktiviert, alle anderen deaktiviert.

PUEA=0b10000000 // R auf PA7 (**PUE** – Registerbit)
PUEB=0b00000100 // R auf PB2 (**x** – Angabe der Ports (**A, B**))
(**n** – Angabe der PINS (**0-7**))

Der **Pull-Up**-Widerstand wird aktiviert, wenn die **PUExn** logisch Eins geschrieben wird. Um den **Pull-Up**-Widerstand auszuschalten, muss PUExn logisch Null geschriebene werden.

Im Datenblatt des Herstellers wird diese Tabelle zur Funktion der DDR, PORT und PUE Register angegeben.

Und Versuch einer Übersetzung:

Table 10-1. Port Pin Configurations

DDxn	PORTxn	PUExn	I/O	Pull-up	Comment
0	X	0	Input	No	Tri-state (hi-Z)
0	X	1	Input	Yes	Sources current if pulled low externally
1	0	0	Output	No	Output low (sink)
1	0	1	Output	Yes	NOT RECOMMENDED. Output low (sink) and internal pull-up active. Sources current through internal pull-up resistor and consumes power constantly
1	1	0	Output	No	Output high (source)
1	1	1	Output	Yes	Output high (source) and internal pull-up active

DDRxn	PORTxn	PUExn	I/O	Pull-Up	Kommentar
0	X	0	Eingang	Nein	**Tri-State** Ausgang (0 und 1) oder hochohmig, mit PINxn zu lesen
0	X	1	Eingang	Ja	**Pull-Up** liegt an Vcc, geringer Strom gegen GND
1	0	0	Ausgang	Nein	Ausgang zugeschaltet und auf low, fliesst Strom bis zu einigen mA
1	0	1	Ausgang	Ja	**NICHT EMPFOHLEN** nicht empfohlen, da unnötiger Stromverbrauch
1	1	0	Ausgang	Nein	Ausgang hoch, liefert Strom
1	1	1	Ausgang	Ja	Ausgang hoch mit internem **Pull-Up** Widerstand

Als nächstes werde ich das Programm ATB_Ati_841_Prg_3.c noch einmal nutzen um die Funktion der einzelnen Register noch mal darzustellen.

```
/* ATB_Ati_841_Prg_3.c  * Author : hjsee */

// Programm für Attiny 841 ohne Quarz mit Platine 177
// 5 x Taster als Bedienkreuz ohne Widerstand nach Vcc und Kontroll LED
// 3 x Wahl für Busadresse A0, A1, A2

// Platine P177 Attiny 841 ohne Quarz
// PA0 --> T1
// PA1 --> A0
// PA2 --> A1
// PA3 --> A2
// PB0 --> T2
// PA7 --> T3
// PA5 --> T4
// PB1 --> T5
// PB2 --> LED 2

#define F_CPU 8000000UL                 // Angabe der Frequenz, wichtig für die Zeit
#include <util/delay.h>                 // Einbindung Datei Pause
#include <avr/io.h>                     // Einbindung Datei Ausgänge

int main(void)
 {
   DDRA=0b00000000;                     // DDRA schalten
   DDRB=0b00000100;                     // DDRB schalten
   PORTA=0b00000000;                    // Port A ein/aus 0=Eingang
   PORTB=0b00000100;                    // Port B ein/aus 1=Ausgang

   // PUEA=0b00000001;                  // R auf Taster 1 PA0
   // PUEB=0b00000001;                  // R auf Taster 2 PB0
   // PUEA=0b10000000;                  // R auf Taster 3 PA7
   // PUEA=0b00100000;                  // R auf Taster 4 PA5
   PUEB=0b00000010;                     // R auf Taster 5 PB1

   while(1)                             // Programmschleife
    {
      // if (PINA & (1<<PINA0))         // Taster T1 PA0
      // if (PINB & (1<<PINB0))         // Taster T2 PB0
      // if (PINA & (1<<PINA7))         // Taster T3 PA7
      // if (PINA & (1<<PINA5))         // Taster T4 PA5
      if (PINB & (1<<PINB1))            // Taster T5 PB1
       {                                // Wenn T1 gedrückt...
         PORTB |=(1<<PINB2);            // LED 2 ein
       }
      else
       {                                // wenn nicht
         PORTB &=~(1<<PINB2);           // LED 2 aus
       }
    }
 }
```

In diesem Programm nutze ich den Taster T5 mit Anschluss am PB1. Die Anzeige erfolgt an der LED2 am PB2.
Die Eingänge PA0, PA1 und PA2 nutze ich in diesem Programm nicht.
Wird ein anderer Taster verwendet sind die entsprechenden Zeilen aus zu kommentieren bzw. die Kommentierung zu löschen.

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