

1.5 Atmega-Programmierung in ASM/Eingabeports und Verzweigung

1.5.1 Aufgabe

In diesem Programm soll LED1 leuchten, solange der Taster T1 (Anschluss PD2, Jumper JP3 ist gesetzt) gedrückt wurde. Um das lösen zu können, muss man zwei Dinge wissen:

- Wie bekommt man die Information, ob T1 gedrückt wurde, ins Programm?
- Wie kann man das Programm dazu bringen, nur dann LED1 einzuschalten, wenn die Information entsprechend ist?

1.5.2 Port-Eingabe

Die Hardware von Taster 1 geschaltet sieht man in Abbildung 1. Wenn der Taster gedrückt wird, liegt an PD2 low-Potential an, also eine Null. Im Datenrichtungs-Register DDRD muss das Bit 2

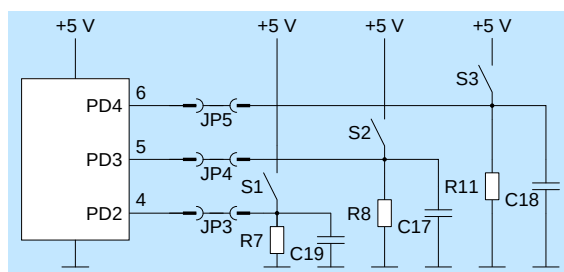


Abbildung 1: Hardware der Eingänge (Auszug)

auf null gesetzt werden (z.B. mit `cbi PORTD, PD2`).

Dann kann man im Register `PIND` (siehe Tabelle 1) an Bit 2 die Information über den Eingang lesen. Der zugehörige Eingabebefehl lautet: in r16, `PIND`. Im Ausgabe-Register `PORTD` kann man

Name	Bit	7	6	5	4	3	2	1	0	Adresse
PIND		PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	SFR 16
R/W		R	R	R	R	R	R	R	R	
Init.		-	-	-	-	-	-	-	-	

Tabelle 1: Ein weiteres Register

diese Information (in PD2) *nicht* erhalten.¹

1.5.3 Sprungbefehle

Bisher kam nur ein Sprungbefehl `rjmp`, der das Programm bedingungslos an einer anderen Stelle fortführen ließ. Hier ist aber ein Sprungbefehl nötig, der je nach Stellung eines Bits ein Sprung veranlasst oder nicht.

Allgemein gibt es dazu Befehle mit Namen wie `brne` (*branch if not equal*) und `breq` (*branch if equal*), die überall in Programmen einsetzbar sind.

Bei AVR-Controllern existiert aber noch eine weitere Familie spezieller Sprungbefehle (Skip-Befehle), die hier genau passen, weil sie ein Bit eines Universal- oder SFR-Registers abfragen und in Abhängigkeit davon *genau einen Befehl* überspringen (Tabelle 2):

¹Bei der Eingabe wird `PORTD` benutzt, um den internen Pull-Up-Widerstand einer Port-Leitung zu aktivieren (durch Setzen des Bits auf null). Das ist jedoch beim Pollin-Board nicht sinnvoll, da dort externe Pull-Down-Widerstände eingesetzt sind.

- a) `sbis=skip-if-bit(i/o)-set` – Überspringe den folgenden Befehl genau dann, falls Bit mit der angegebenen Nummer im angegebenen SFR-Register gesetzt (also auf eins) ist.
- b) `sbic=skip-if-bit(i/o)-clear` – Überspringe den folgenden Befehl genau dann, falls Bit mit der angegebenen Nummer im angegebenen SFR-Register nicht gesetzt (also auf null) ist.
- c) `sbrs=skip-if-bit(reg)-set` – Überspringe den folgenden Befehl genau dann, falls Bit mit der angegebenen Nummer im angegebenen Universal-Register gesetzt (also auf eins) ist.
- d) `sbrc=skip-if-bit(reg)-clear` – Überspringe den folgenden Befehl genau dann, falls Bit mit der angegebenen Nummer im angegebenen Universal-Register nicht gesetzt (also auf null) ist.

Befehl	a	b
<code>in a, b</code>	Univ.-Register	SFR-Nummer
<code>sbic a, b</code>	SFR-Nummer	Bit-Nr. 0–7
<code>sbis a, b</code>	SFR-Nummer	Bit-Nr. 0–7
<code>sbrc a, b</code>	Univ.-Register	Bit-Nr. 0–7
<code>sbrs a, b</code>	Univ.-Register	Bit-Nr. 0–7

Tabelle 2: Eingabe- und Skip-Befehle

Ein Beispiel dazu zeigt das Beispielprogramm `schalt1_led1.asm`

```

1  .include "usr/share/avra/m32def.inc"
2
3      ldi r16, 1<<JTD ; JTAG-Schnittstelle aus
4      out MCUCSR, r16
5      out MCUCSR, r16
6
7      ldi r16, 0xff ; LEDs 1–8 als Ausgang
8      out DDRC, r16
9      ldi r16, 0 ; Taster 0–7 als Eingang
10     out DDRB, r16
11 weiter:
12     in r16, PINB ; PortD-Eingang nach r16
13     out PORTC, r16 ; r16 nach PortD-Ausgang
14     rjmp weiter

```

In diesem Programm wird LED1 eingeschaltet, sobald T1 gedrückt wurde. Von da an bleibt LED1 aber eingeschaltet (bis zu einem Reset). Abbildung 2 zeigt das zugehörige Flussdiagramm.

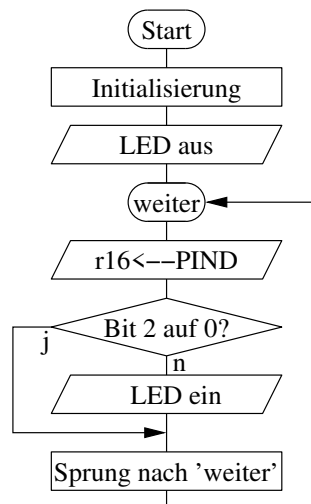


Abbildung 2: Flussdiagramm zu schalt1_led1_pap