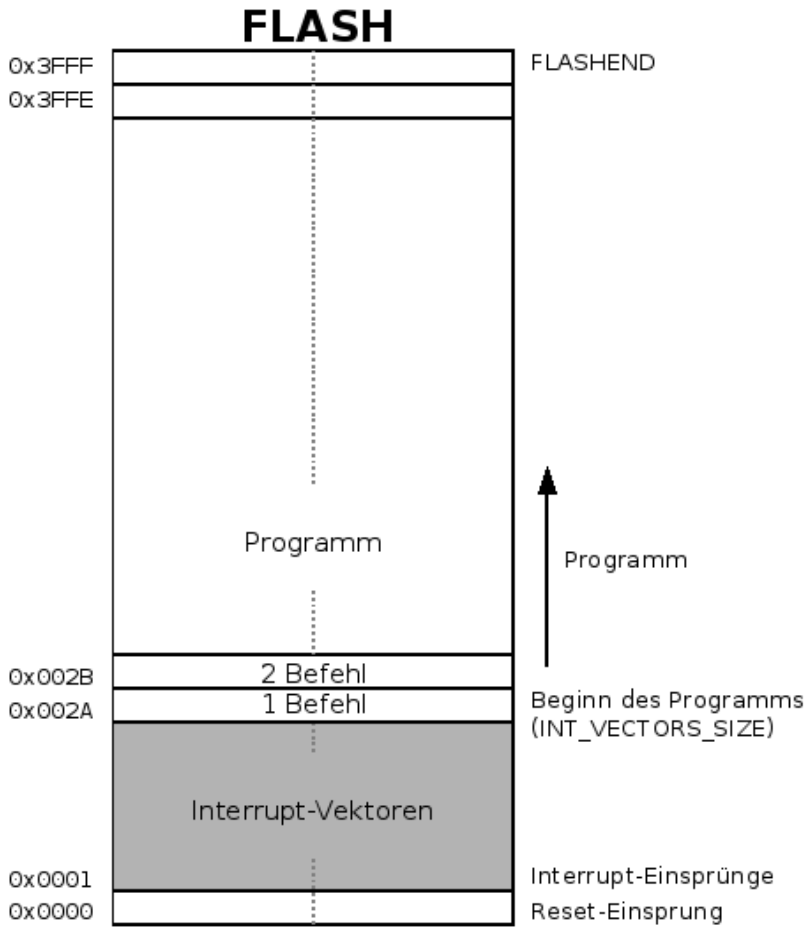
Digitale Ein- und Ausgabe

- **ATmega32:**
  4 Ports A-D mit je 8 Pins (Bit) = 32 individuell ansteuerbare E/A-Pins. Jeder dieser Pins kann als Eingang oder Ausgang konfiguriert werden. Als Eingang kann ein interner Pull-Up-Widerstand dazugeschaltet werden. Achtung: alle Pins sind mindestens doppelt belegt.
- Für die Programmierung eines Pins n stehen drei Bits in drei Registern (x = Portname A-D) zur Verfügung:

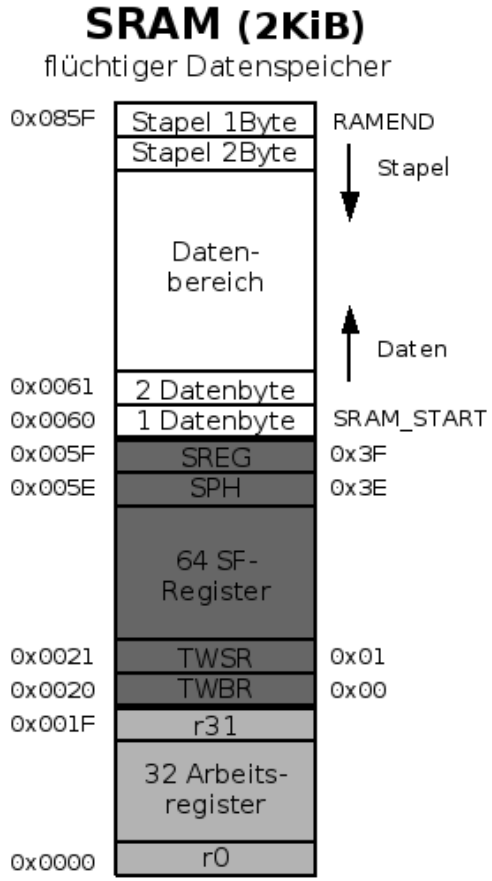
Memory Map

ATMega32

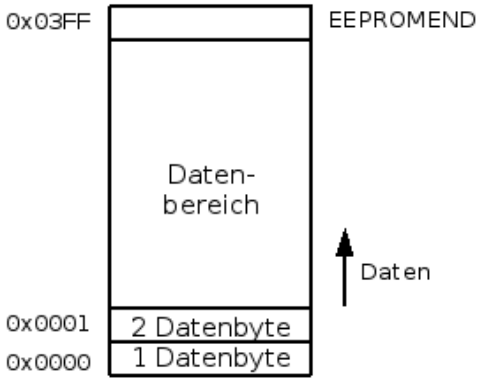
Programmspeicher (32KiB)
nichtflüchtiger Speicher, 16 Bit breit



Datenspeicher
8 Bit breit



EEPROM (1KiB)
nichtflüchtiger Datenspeicher



Digitale Ein- und Ausgabe

Datenrichtungsregister DDRx (Data Direction Reg.)

Legt (meist im Initialisierungsteil) fest ob ein Pin als Ein- oder Ausgang verwendet wird.

0 = Eingang (default)

1 = Ausgang

Unbenutzt sollte ein Pin immer als Eingang deklariert werden.

Beispiele:

```
ldi     r16,0xF0     ;obere 4 Bit Ausgang, untere
out     DDRD,r16     ;4 Bit als Eingang(unbenutzt)

sbi     DDRA,3       ;Pin 3 (PORTA3,PA3) = Ausgang
cbi     DDRA,4       ;Pin 4 (PORTA3,PA3) = Eingang
```

Digitale Ein- und Ausgabe

- **Datenausgaberegister PORTx** (PORTx Data Reg.)
  Über das Datenausgaberegister werden die Daten an die einzelnen Pins ausgegeben (les- und schreibbar).
  Ist ein Pin als Eingang initialisiert dann besitzt dieses Register eine andere Funktion! Mit einer Eins kann dann ein interner Pull-Up-Widerstand zum Eingangspin zugeschaltet werden.

Beispiele:

```
ldi     Tmp1,'A'     ;ASCII-Code von 'A' auf LEDs
out     PORTD,Tmp1   ;
sbi     PORTB,7      ;oberste LED PB7 einschalten

cbi     DDRD,0       ;PD0 = Eingang
sbi     PORTD,0      ;internen Pull-Up-Widerstand
                      zuschalten
```

Digitale Ein- und Ausgabe

- **Dateneingaberegister PINx**
  (PORTx INput Pins Adress)
  Über das Dateneingaberegister PINx werden Daten eingelesen. Es ist nur lesbar (an sich ist PINx kein klassisches Register, denn die Bits folgen augenblicklich dem Zustand der Pins).

Beispiele:

```
        in      Tmp1,PINA    ;PORTA (8 Bit) einlesen
        andi    Tmp1,0x80    ;maskiere oberstes Bit
        breq    SW_ON        ;falls Null dann nach SW_ON
        ....
SW_ON:  ....

        sbis    PINA,7       ;ueberspringe naechste
        ....                 ;Zeile falls PA7=0
SW_OFF: ....
```

Digitale Ein- und Ausgabe

- **Initialisierung:**
  Datenrichtungsregister DDRx
  0 = Eingang (default, unbenutzte Pins immer als Eingang)
  1 = Ausgang
  Falls Eingang kann mit dem **Datenausgaberegister PORTx** ein interner Pull-Up-Widerstand zugeschaltet werden:
  0 = kein interner Pull-Up-Widerstand
  1 = interner Pull-Up-Widerstand

- **Hauptprogramm:**
  Ausgabe über das **Datenausgaberegister PORTx**
  Einlesen über das **Dateneingaberegister PINx**

Erstes Programm: A303_dig_out_8bit.asm

- Das erste Programm soll acht LEDs gleichzeitig einschalten. Dazu sind acht LEDs mit den acht Pins von **PORTD** zu verbinden.
- Erstelle das Programm indem du die Vorlage ergänzt.
- Speichere das Programm mit dem Namen "**A303_dig_out_8bit.asm**" ab.
- Teste das Programm!

Erstes Programm: A303

- **A303_dig_out_8bit.asm**

Initialisierung:

```
.DEF  Tmp1 = r16
        ser   Tmp1          ;(oder ldi Tmp1,0xFF)
        out   DDRD,Tmp1     ;8 Pins als Ausgang PORTD
```

Hauptprogramm:

```
MAIN:   out   PORTD,Tmp1    ;8 LEDs einschalten
        rjmp  MAIN          ;Endlosschleife
```

Zweites Programm: A304_dig_out_1bit.asm

- Schreibe das Programm so um, dass nur die LED an Pin 5 aufleuchtet. Es sind sowohl die Initialisierung wie auch die Ansteuerung zu ändern!
- Speichere das Programm mit dem Namen "**`A304_dig_out_1bit.asm`**" ab.
- Teste das Programm!

Zweites Programm: A304

- **A304_dig_out_1bit.asm**

Initialisierung:

```
.DEF  Tmp1 = r16
        ldi   Tmp1,0x20       ;0b00100000
        out   DDRD,Tmp1       ;Pin PD5 als Ausgang
                              ;Rest Eingang!
```

Hauptprogramm:

```
MAIN:   sbi   PORTD,5         ;LED 5 einschalten
        rjmp  MAIN            ;Endlosschleife
```

Maskierung

- Löschen, Setzen und Invertieren eines Bit mittels der Maskierung:

1. SF-Register einlesen: `in Tmp1,PORTA`
2. Arbeitsregister maskieren mit einer logischen UND, ODER bzw. EOR-Funktion: `andi Tmp1,0b11110111`
3. SF-Register zurückspeichern: `out PORTA,Tmp1`

Maskierung

- Löschen eines Bit mit AND
  (0 löscht, 1 hat keinen Einfluss):

```
andi  Tmp1,0b11110111      ;Bit 3 wird gelöscht
```

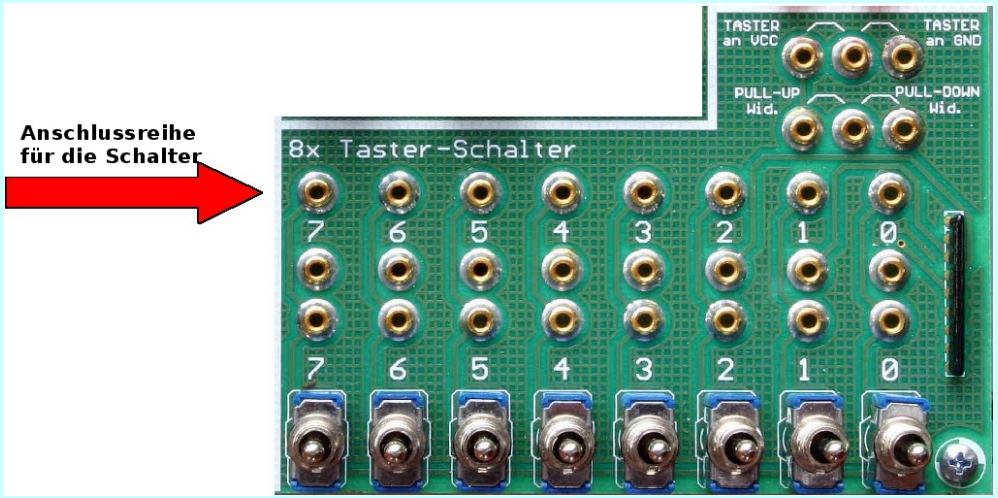
- Setzen eines Bit mit OR
  (1 setzt, 0 hat keinen Einfluss):

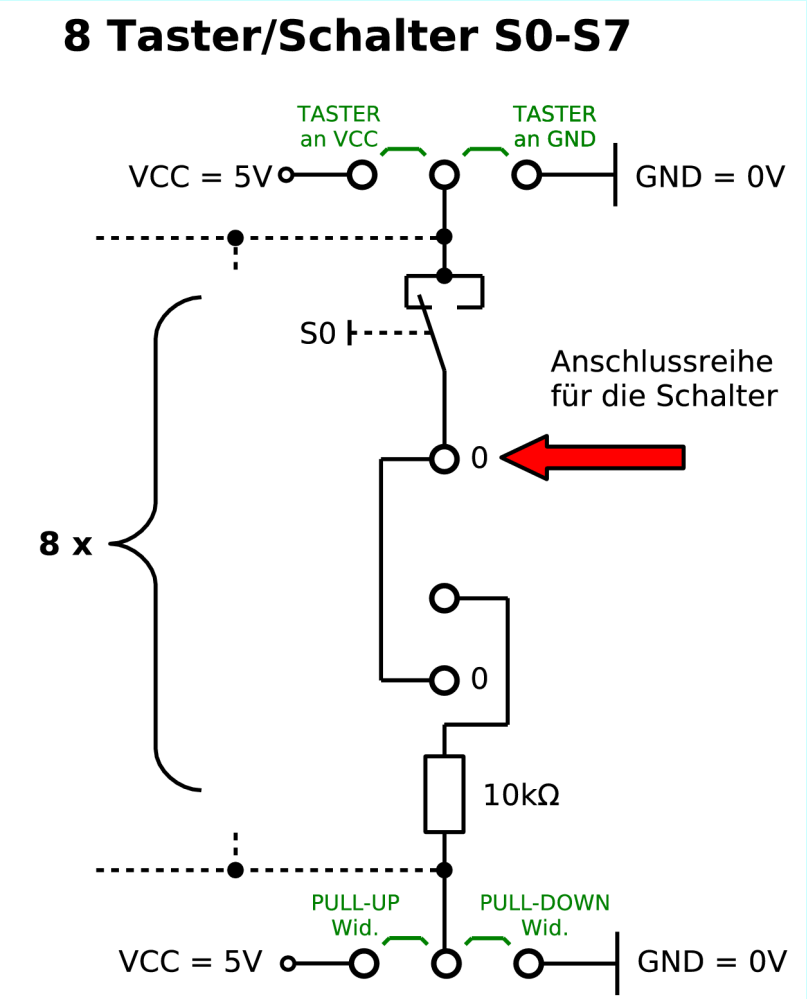
```
ori   Tmp1,0b00010100      ;Bit 2 und 4 werden gesetzt
```

- Toggeln (Invertieren) eines Bit mit EOR
  (1 invertiert, 0 hat keinen Einfluss)

```
ldi   Tmp2,0b00100000      ;Maske setzen
eor   Tmp1,Tmp2            ;Bit 5 wird invertiert
```

Daten mittels Schalter einlesen

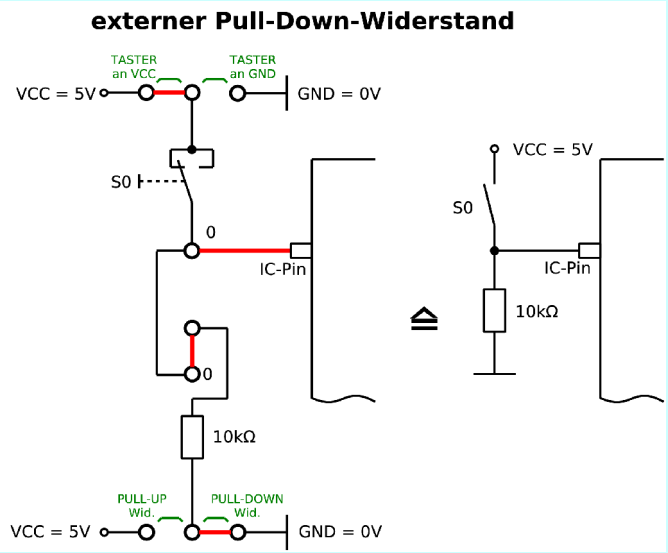


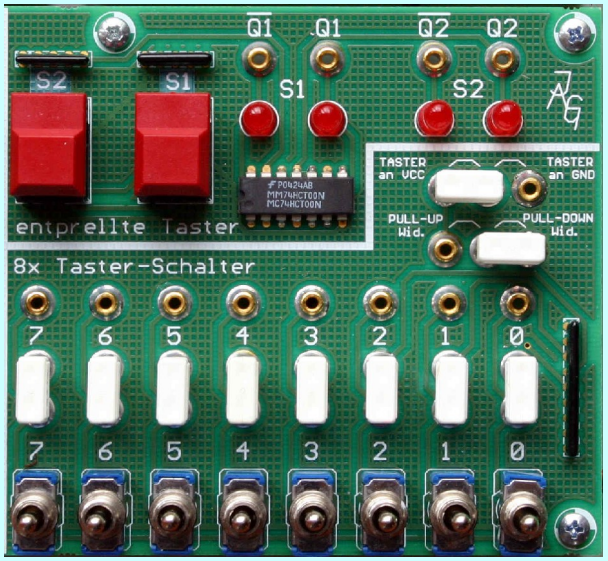


Daten mittels Schalter einlesen

positive Logik:

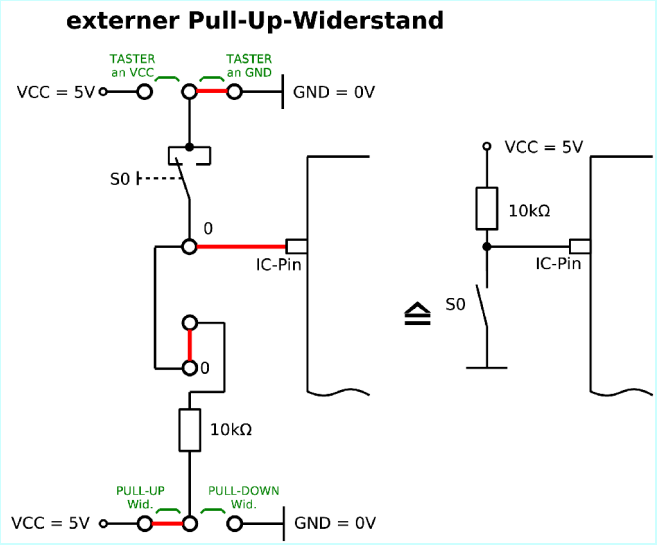
Externer Pull-Down

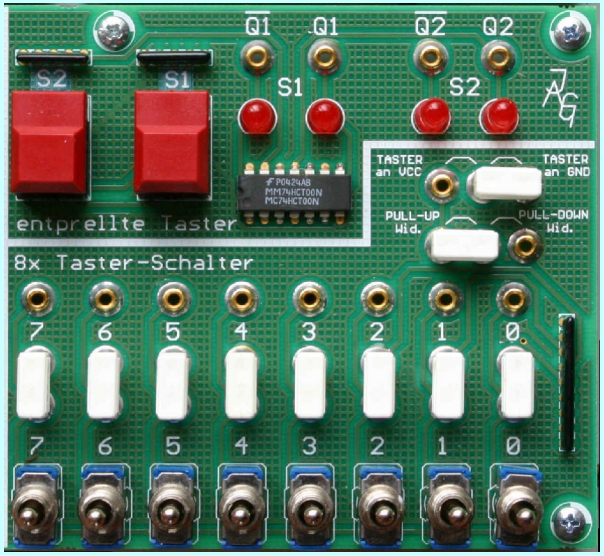




negative Logik:

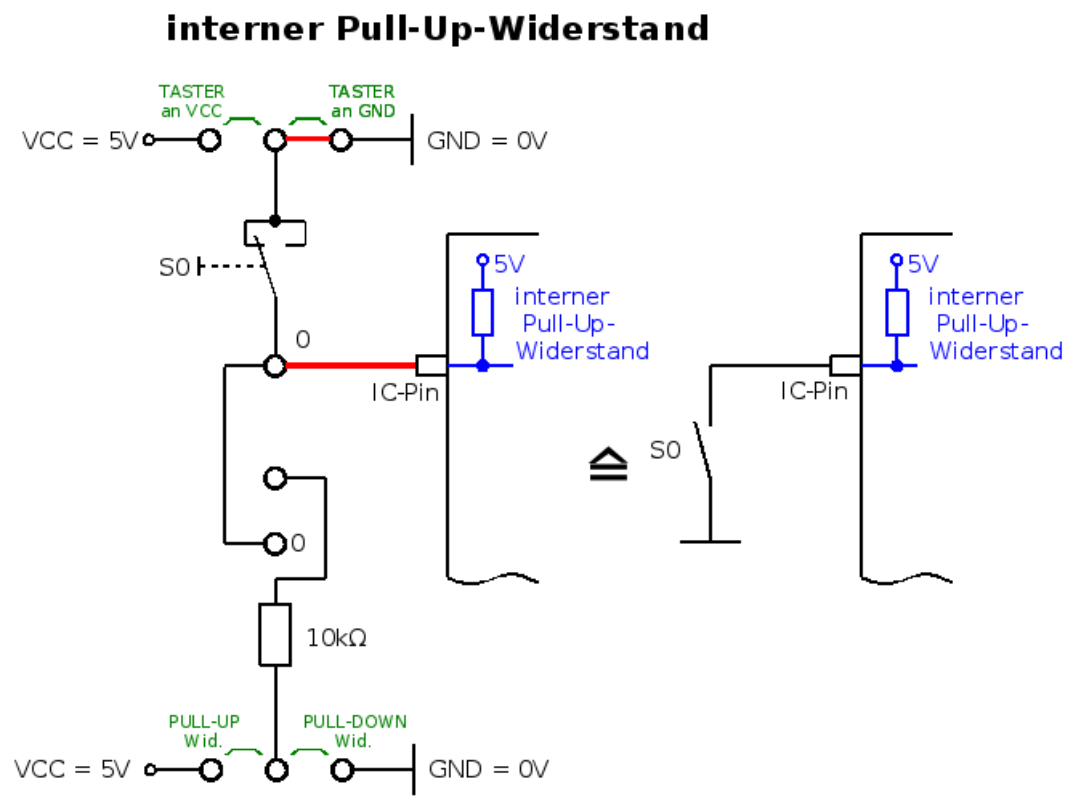
Externer Pull-UP

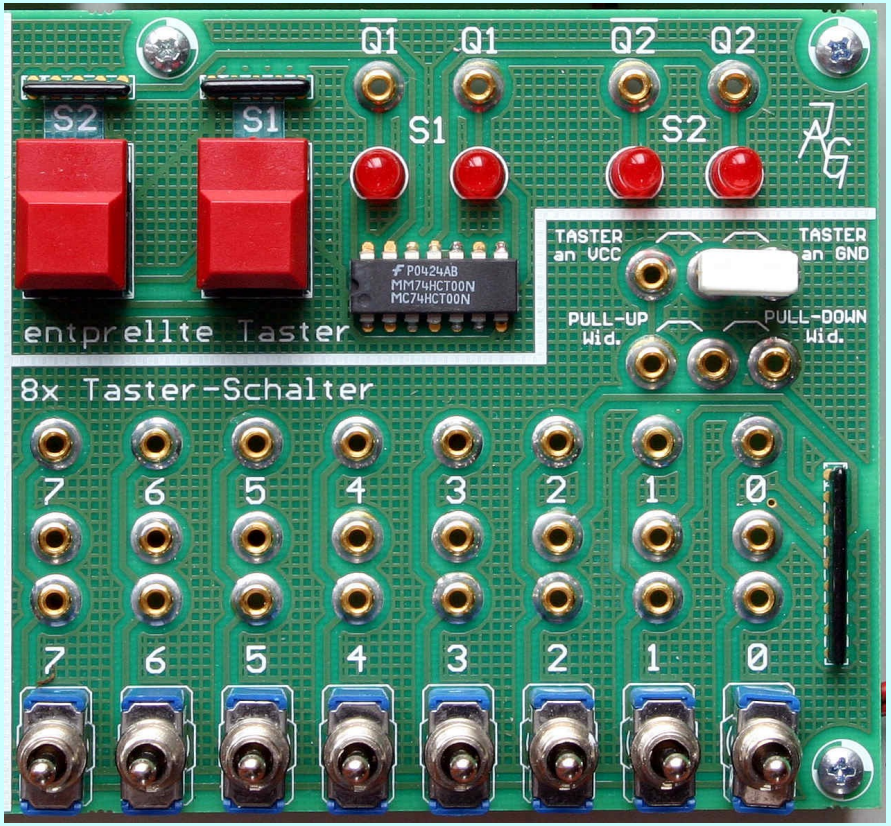




Daten mittels Schalter einlesen

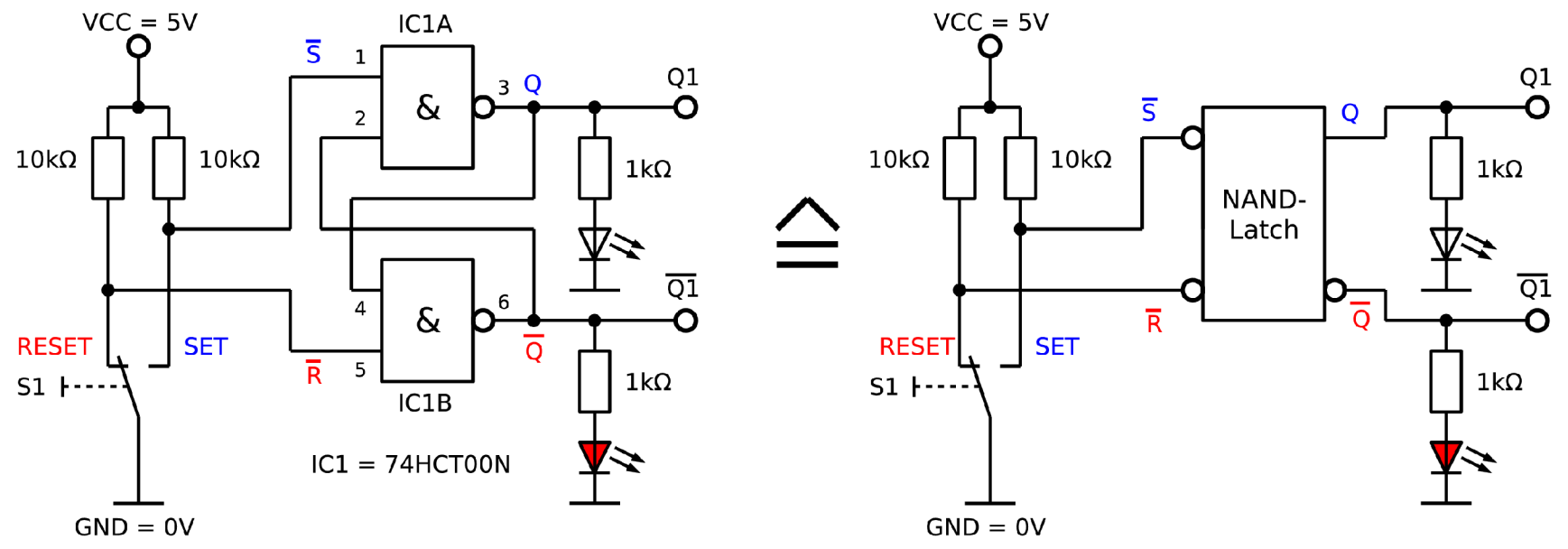
negative Logik: interner Pull-UP

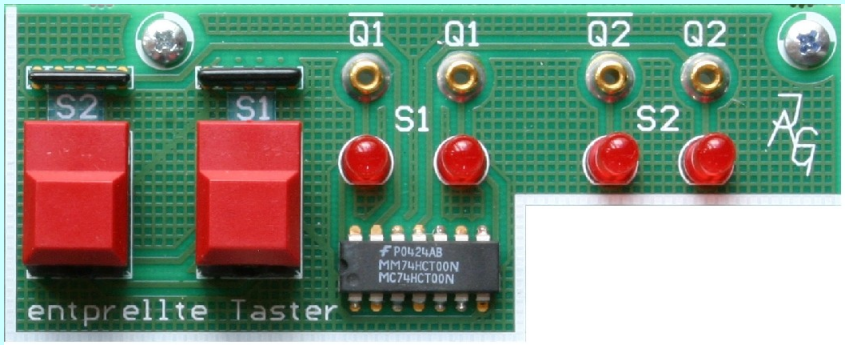




Daten mittels Schalter einlesen

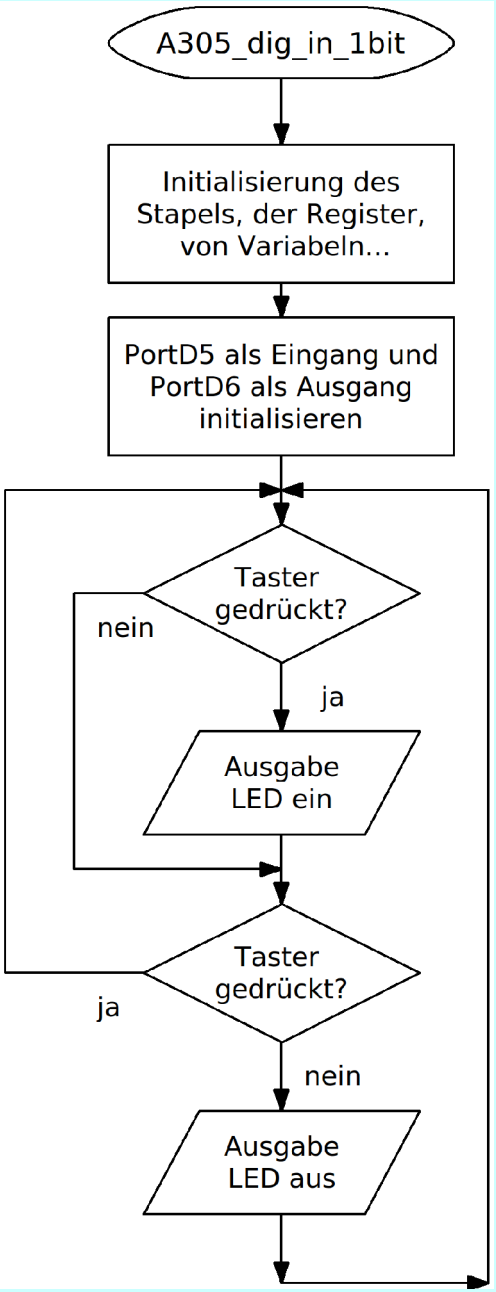
2 entprellte Taster:





Drittes Programm: A305_dig_in_1bit

- Setze das nebenstehende Flussdiagramm um. Es wird ein nicht entprellter Taster verwendet (Befehle: **`sbic, sbis, sbi, cbi`**).
- Erstelle das Programm indem du die Vorlage ergänzt und speichere das Programm mit dem Namen "**`A305_dig_in_1bit.asm`**". ab.
- Teste das Programm einmal ohne Pull-Up-Widerstand, mit dir selbst als Pull-Up-Widerstand, und einmal mit einem internen Pull-Up-Widerstand!
- Simuliere das Programm im Studio 4 (Funktion "Debug").



Drittes Programm A305_dig_in_1bit.asm

```
INIT:
.DEF   Tmp1  = r16          ;Register 16 dient als erster Zwischensp.
.EQU   PButt = 5            ;Taster (Push-Button) an PD5
.EQU   LED   = 6            ;LED an PD6
       ;nur Pin6 von PortD als Ausgang initialisieren
       ldi    Tmp1,0x40     ;DDRD = 0b01000000
       out    DDRD,Tmp1     ;PD6 auf 1 (Ausgang) alle anderen 0 (Eing.)
      (sbi    PORTD,PButt   ;internen Pull-Up-Widerstand einschalten)

;Hauptprogramm

MAIN:  sbis   PIND,PButt    ;Ueberspringe 1 Zeile, wenn Bit gesetzt
                            ;Taster nicht gedrueckt (PD5 = 1)
       sbi    PORTD,LED     ;Schalte LED ein (PD6 = 1)
       sbic   PIND,PButt    ;Ueberspringe 1 Zeile, wenn Bit geloescht
                            ;Taster gedrueckt (PD5 = 0)
       cbi    PORTD,LED     ;Schalte LED aus (PD6 = 0)

       rjmp   MAIN          ;Endlosschleife
```

Digitale Ein- und Ausgabe

- Lesen bzw. Schreiben eines ganzen Byte:

```
ldi   Tmp1,0b00001111   ;untere 4 Bit Ausgang, obere
out   DDRD,Tmp1         ;4 Bit als Eingang

ldi   Tmp1,0b11110000   ;int. Pull-Ups einschalten
out   PORTD,Tmp1        ;

ldi   Tmp1,5            ;dezimal 5 (0b00000101) an PORTB
out   PORTB,Tmp1        ;ausgeben
(Achtung: der gleiche Befehl an PORTD würde
eingeschaltete Pull-Ups wieder ausschalten!)

in    Tmp1,PIND         ;Byte an PORTA einlesen
in    Tmp2,PINA         ;Byte an PORTA einlesen
```

Digitale Ein- und Ausgabe

- Lesen bzw. Schreiben eines Bit:

```
        sbi   DDRA,3      ;PA3 = Ausgang
        cbi   DDRA,4      ;PA4 = Eingang

        sbi   PORTA,3     ;PA3 int. PU-Wid. zuschalten

        sbis  PINA,4      ;ueberspringe naechste
        ....              ;Zeile falls PA4=1
SW_ON1: ....

        sbic  PINA,7      ;ueberspringe naechste
        ....              ;Zeile falls PA7=0
SW_OFF1: ....
```