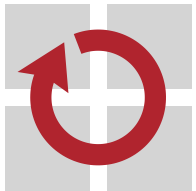
Übungen zu Systemnahe Programmierung in C

Abschnitt 4.1: Ein- und Ausgabe über Pins

18.05.2020

Tim Rheinfels
Benedict Herzog
Bernhard Heinloth

Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg



Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

General Purpose Input/Output (GPIO)

- Mikrocontroller interagieren mit der Außenwelt
- Neben definierten Protokollen auch beliebige (digitale) Signale
- Viele Pins können sowohl als Eingang als auch als Ausgang konfiguriert werden

⇝ General Purpose Input/Output (GPIO)

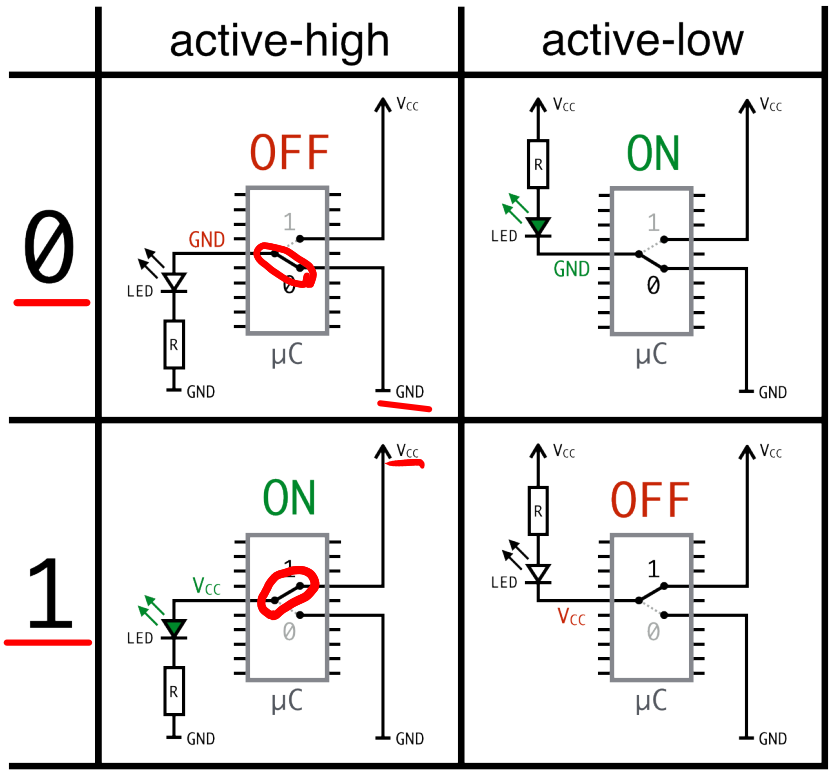
SPI, I^2C, ...

Ausgang: active-high & active-low

Ausgang je nach Beschaltung:

active-high: high-Pegel (logisch 1; V_{cc} am Pin) → LED leuchtet

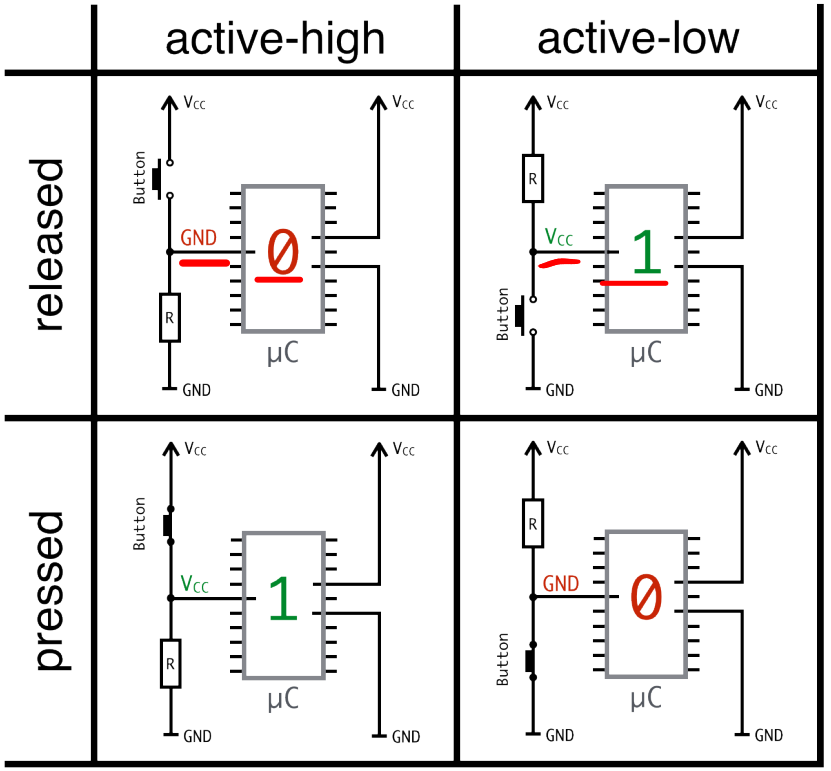
active-low: low-Pegel (logisch 0; *GND* am Pin) → LED leuchtet



Eingang: active-high & active-low

Eingang je nach Beschaltung:

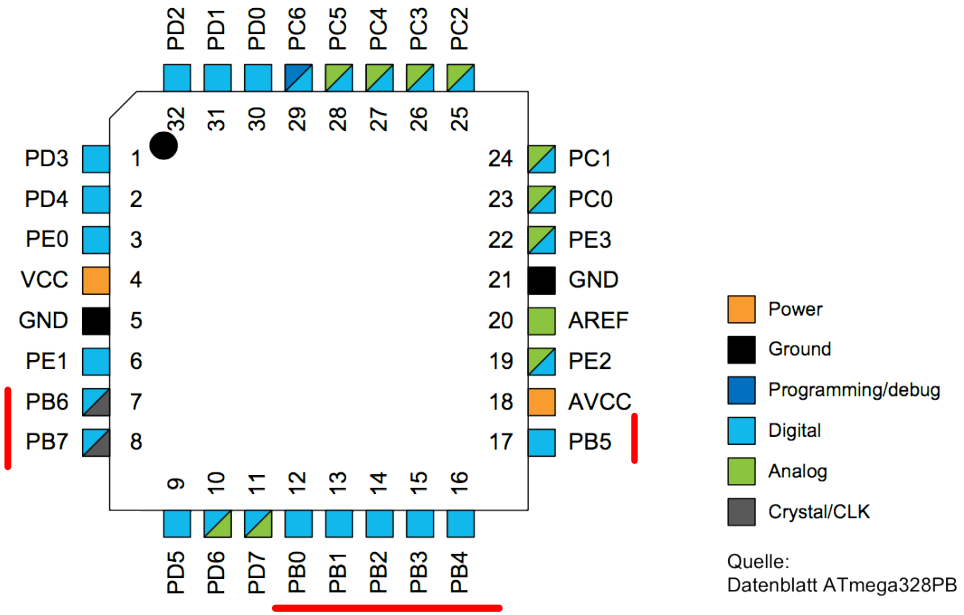
active-high: Button gedrückt → high-Pegel (logisch 1; V_{cc} am Pin)
active-low: Button gedrückt → low-Pegel (logisch 0; *GND* am Pin)



Eingänge sind hochohming, es muss ein definierter Pegel anliegen

⇝ Pull-down oder (interne) Pull-up Widerstände verwenden

Konfiguration der Pins



- Jeweils acht Pins am AVR sind zu einem I/O Port zusammengefasst
- Jeder I/O-Port des AVR wird durch drei 8-bit Register gesteuert:

  `DDRx` Datenrichtungsregister (Data Direction Register)
  `PORTx` Portausgaberegister (Port Output Register)
  `PINx` Porteingaberegister (Port Input Register)

- Jedem Pin eines Ports ist jeweils ein Bit in den drei Register zugeordnet

I/O-Port-Register (1)

DDRx: Data Direction Register konfiguriert Pin `i` als Ein- oder Ausgang

- `Bit i = 1` → Pin `i` als Ausgang verwenden
- `Bit i = 0` → Pin `i` als Eingang verwenden

Beispiel:

```
01 DDRC |= (1 << PC3);  // PC3 als Ausgang (Pin 3 an Port C)
02 DDRD &= ~(1 << PD2); // PD2 als Eingang (Pin 2 an Port D)
```

I/O-Port-Register (2)

PORTx: Port Output Register abhängig von DDRx Register

- Wenn **Ausgang**: Legt high- oder low-Pegel an Pin `i` an
  - `Bit i = 1` → high-Pegel an Pin `i`
  - `Bit i = 0` → low-Pegel an Pin `i`
- Wenn **Eingang**: Konfiguriert internen Pull-Up Widerstand an Pin `i`
  - `Bit i = 1` → aktiviert Pull-Up Widerstand für Pin `i`
  - `Bit i = 0` → deaktiviert Pull-Up Widerstand für Pin `i`

Beispiel:

```
01 PORTC |= (1 << PC3);  // Zieht PC3 auf high (LED aus)
02 PORTC &= ~(1 << PC3); // Zieht PC3 auf low (LED an)
03
04 PORTD |= (1 << PD2);  // Aktiviert internen Pull-Up für PD2
05 PORTD &= ~(1 << PD2); // Deaktiviert internen Pull-Up für PD2
```

I/O-Port-Register (3)

PINx: Port Input Register (nur lesbar) aktuellen Wert von Pin `i`

- Wenn **Eingang**: Abrufen was von extern anliegt
- Wenn **Ausgang**: Abrufen ob high oder low ausgegeben wird

Beispiel:

```
01 if((PIND & (1 << PD2)) == 0) { // Testen ob Pin PD2 low ist
02   // low-Pegel --> Button ist gedrückt
03   [...]
04 }
05
06 if((PIND & (1 << PD2)) != 0) { // Testen ob Pin PD2 high ist
07   // high-Pegel --> Button ist nicht gedrückt
08   [...]
09 }
```

→ Datenblatt, 4.2