

# U4 4. Übung

---

- Besprechung Aufgabe 2
- Makros
- Register
- I/O-Ports

# U4-1 Makros

- Makros sind Textersetzungen, welche vom Präprozessor aufgelöst werden. Dies Passiert bevor der Compiler die Dateien verarbeitet.
- Aufbau: `#define Suchwort Ersetzung`
  - ◆ Anweisungsende ist der Zeilenumbruch (kein Strichpunkt!)
- Ersetzung:

```
#define MEINE_KONST 7  
[..]  
a = b + MEINE_KONST; // a = b + 7
```

```
#define MEINE_ERSETZUNG = b + 7  
[..]  
a MEINE_ERSETZUNG; // a = b + 7
```

# U4-1 Makros (2)

## ■ Funktionen:

```
#define POW2(a) (a * a)
[...]  
a = POW2(4); // a = (4 * 4);
```

## ■ Achtung:

```
#define ADD(a, b) a + b  
[...]  
a = ADD(7, 5) * 5; // a = 7 + 5 * 5 = 32
```

### ◆ Berechnungen bei Makros in Klammern setzen

```
#define ADD(a, b) (a + b)  
[...]  
a = ADD(7, 5) * 5; // a = (7 + 5) * 5 = 60
```

## U4-2 Register beim AVR- $\mu$ C

- Beim AVR- $\mu$ C sind die Register
  - ◆ in den Speicher eingebettet
  - ◆ am Anfang des Adressbereichs angeordnet
- Adressen sind der Dokumentation zu entnehmen
- vollständige Dokumentation für "unseren" Mikrokontroller ATmega32:  
[http://www4.informatik.uni-erlangen.de/Lehre/SS10/V\\_SPIC/Uebung/doc/mega32.pdf](http://www4.informatik.uni-erlangen.de/Lehre/SS10/V_SPIC/Uebung/doc/mega32.pdf)
- Für die Aufgaben benötigte Register sind auf den Folien erwähnt
  - ◆ Die Bibliothek (avr-libc), die wir verwenden, definiert bereits sinnvolle Makros für alle Register des AVR  $\mu$ C  
(`#include <avr/io.h>`)

# 1 Makros für Register-Zugriffe

- Makros mit aussagekräftigen Namen können den Umgang mit Registern deutlich vereinfachen
- Beispiel:
  - ◆ Makro für Register an Adresse 0x3b (PORTA beim ATmega32):

```
#define PORTA (*(volatile uint8_t *)0x3b)
```

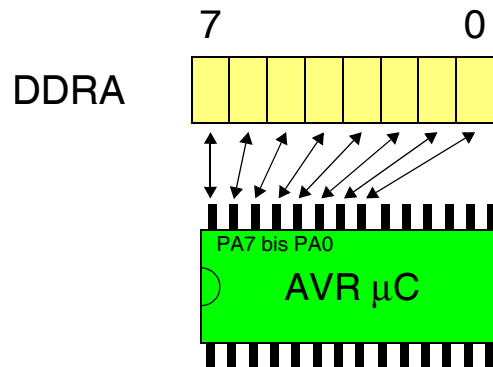
- ◆ Verwenden dieses Registers:

```
volatile uint8_t *portPtr = &PORTA;
PORTA = 0;                /* schreibender Zugriff */
...
if (PORTA == 0x04)       /* lesender Zugriff */
    PORTA &= ~4;         /* lesender und schreibender Zugriff */
*portPtr |= 1;           /* Zugriff über Zeiger */
```

- Das `volatile`-Schlüsselwort verhindert, dass der Compiler Zugriffe auf das Register wegoptimiert.

## U4-3 I/O-Ports des AVR- $\mu$ C

- Jeder I/O-Port des AVR- $\mu$ C wird durch drei 8-bit Register gesteuert:
  - ◆ Datenrichtungsregister ( $DDR_x$  = data direction register)
  - ◆ Datenregister ( $PORT_x$  = port output register)
  - ◆ Port Eingabe Register ( $PIN_x$  = port input register, nur-lesbar)
- Jedem Anschluss-Pin ist ein Bit in jedem der 3 Register zugeordnet
  - Beispiel: DDR von Port A:



# 1 I/O-Port-Register

- **DDR<sub>x</sub>**: hier konfiguriert man einen Pin  $i$  von Port  $x$  als Ein- oder Ausgang
  - Bit  $i = 1 \rightarrow$  Pin  $i$  als **Ausgang** verwenden
  - Bit  $i = 0 \rightarrow$  Pin  $i$  als **Eingang** verwenden
  
- **PORT<sub>x</sub>**: Auswirkung abhängig von DDR<sub>x</sub>:
  - ◆ ist Pin  $i$  als **Ausgang** konfiguriert, so steuert Bit  $i$  im PORT<sub>x</sub> Register ob am Pin  $i$  ein high- oder ein low-Pegel erzeugt werden soll
    - Bit  $i = 1 \rightarrow$  high-Pegel an Pin  $i$
    - Bit  $i = 0 \rightarrow$  low-Pegel an Pin  $i$
  - ◆ ist Pin  $i$  als **Eingang** konfiguriert, so kann man einen internen pull-up-Widerstand aktivieren
    - Bit  $i = 1 \rightarrow$  pull-up-Widerstand an Pin  $i$  (Pegel wird auf high gezogen)
    - Bit  $i = 0 \rightarrow$  Pin  $i$  als tri-state konfiguriert
  
- **PIN<sub>x</sub>**: Bit  $i$  gibt den aktuellen Wert des Pin  $i$  von Port  $x$  an (nur lesbar)

## 2 Beispiel: Initialisierung eines Ports

- Pin 3 von Port B (PB3) als Ausgang konfigurieren und auf  $V_{CC}$  schalten:

```
DDRB |= 0x08; /* =(1 << 3); PB3 als Ausgang nutzen... */
PORTB |= 0x08; /* ...und auf 1 (=high) setzen */
```

- Pin 2 von Port D (PD2) als Eingang nutzen, pull-up-Widerstand aktivieren und prüfen ob ein low-Pegel anliegt:

```
DDRD &= ~0x04; /* PD2 als Eingang nutzen... */
PORTD |= 0x04; /*...und den pull-up-Widerstand aktivieren*/

if ( (PIND & 0x04) == 0) { /* den Zustand auslesen */
    /* ein low Pegel liegt an, der Taster ist gedrückt */
}
```

- Die Initialisierung der Hardware wird in der Regel **einmalig** zum Programmstart durchgeführt



# U4-4 Aufgabe 4: LED-Modul

---

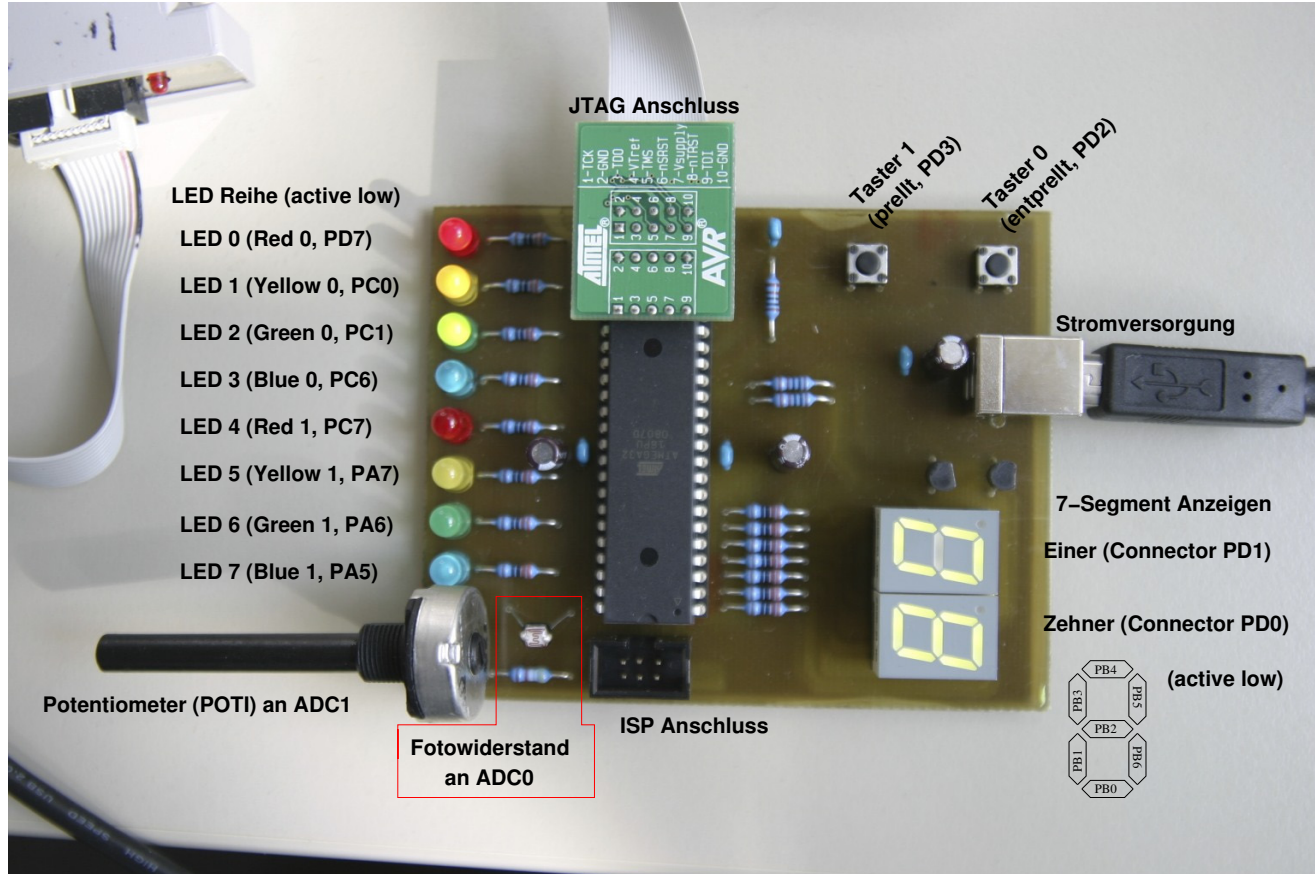
- Das LED-Modul der SPiCboard-Bibliothek selbst implementieren
- Das eigene Modul dann mit einem Testprogramm linken
- Andere Teile der Bibliothek können für den Test benutzt werden

## 1 LEDs des SPiCboard

---

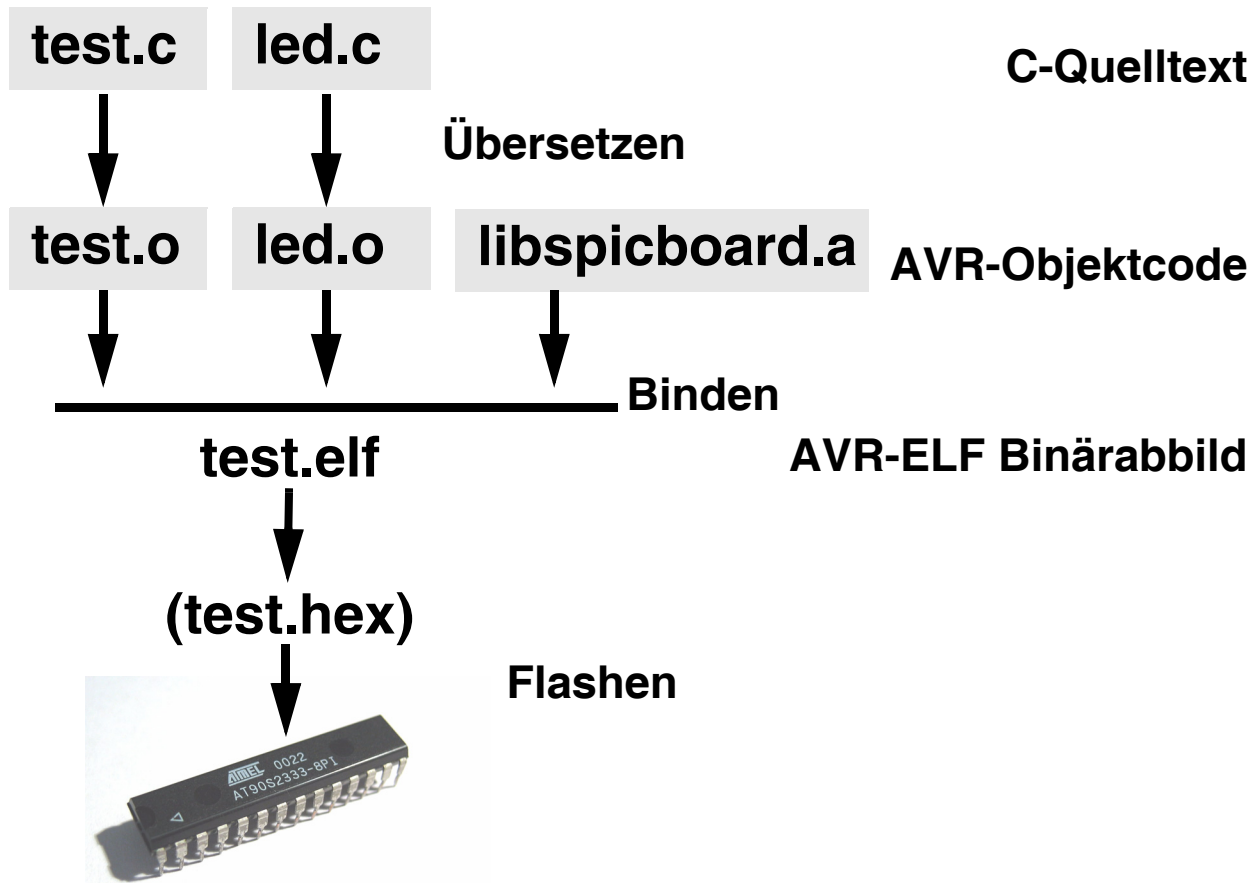
- Die Anschlüsse und Namen der einzelnen LEDs können dem Übersichtsbildchen entnommen werden
- Alle LEDs sind *active low*, d.h. leuchten wenn ein Low-Pegel auf dem Pin angelegt wird.
- PD7 = Port D, Pin 7

# U4-4 Aufgabe 4: LED-Modul (2)



SPIC-Ü

## 2 Toolchain



### 3 AVR-Studio Projekteinstellungen

---

- Projekt wie gehabt anlegen
  - ◆ Initiale Quelldatei: `test.c`
  - ◆ Optimierung: Os (zum Debuggen gegebenenfalls auf O0 setzen)
  - ◆ alle sonstigen Einstellungen wie bisher
  
- Dann weitere Quelldatei `led.c` hinzufügen
  
- Wenn nun übersetzt wird, wird das eigene LED-Modul verwendet
  
- Andere Teile der Bibliothek werden nach Bedarf hinzugebunden
  
- Zum Test mit der Referenzimplementierung `led.c` aus der Liste der Quelldateien löschen