U4 4. Übung

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

U4-1 Register beim AVR-µC

- Beim AVR-µC 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.cs.fau.de/Lehre/SS12/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 µC
    (`#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)
```

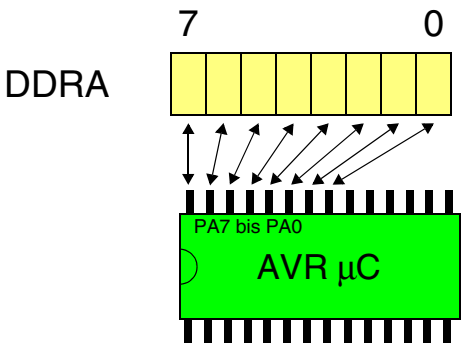
  - Verwenden dieses Registers:

```
volatile uint8_t *portPtr = &PORTA;
PORTA = 0;              /* schreibender Zugriff */
...
if (PORTA == (1 << 3)) /* lesender Zugriff */
    PORTA &= ~(1 << 3);/* lesender und schreibender Zugriff */
*portPtr |= 1;          /* Zugriff über Zeiger */
```

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

U4-2 I/O-Ports des AVR-µC

- Jeder I/O-Port des AVR-µC wird durch drei 8-bit Register gesteuert:
  - Datenrichtungsregister (`DDRx` = data direction register)
  - Datenregister (`PORTx` = port output register)
  - Port Eingabe Register (`PINx` = 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

- **DDRx**: hier konfiguriert man einen Pin *i* von Port *x* als Ein- oder Ausgang
  - Bit *i* = 1→ Pin *i* als **Ausgang** verwenden
  - Bit *i* = 0→ Pin *i* als **Eingang** verwenden
- **PORTx**: Auswirkung abhängig von DDR*x*:
  - ◆ ist Pin *i* als **Ausgang** konfiguriert, so steuert Bit *i* im PORT*x* Register ob am Pin *i* ein high- oder ein low-Pegel erzeugt werden soll
    - Bit *i* = 1→ high-Pegel an Pin *i*
    - Bit *i* = 0→ low-Pegel an Pin *i*
  - ◆ ist Pin *i* als **Eingang** konfiguriert, so kann man einen internen pull-up-Widerstand aktivieren
    - Bit *i* = 1→ pull-up-Widerstand an Pin *i* (Pegel wird auf high gezogen)
    - Bit *i* = 0→ Pin *i* als tri-state konfiguriert
- **PINx**: 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 |= (1 << 3);/* =0x08; PB3 als Ausgang nutzen... */
PORTB |= (1 << 3);/* ...und auf 1 (=high) setzen */
```

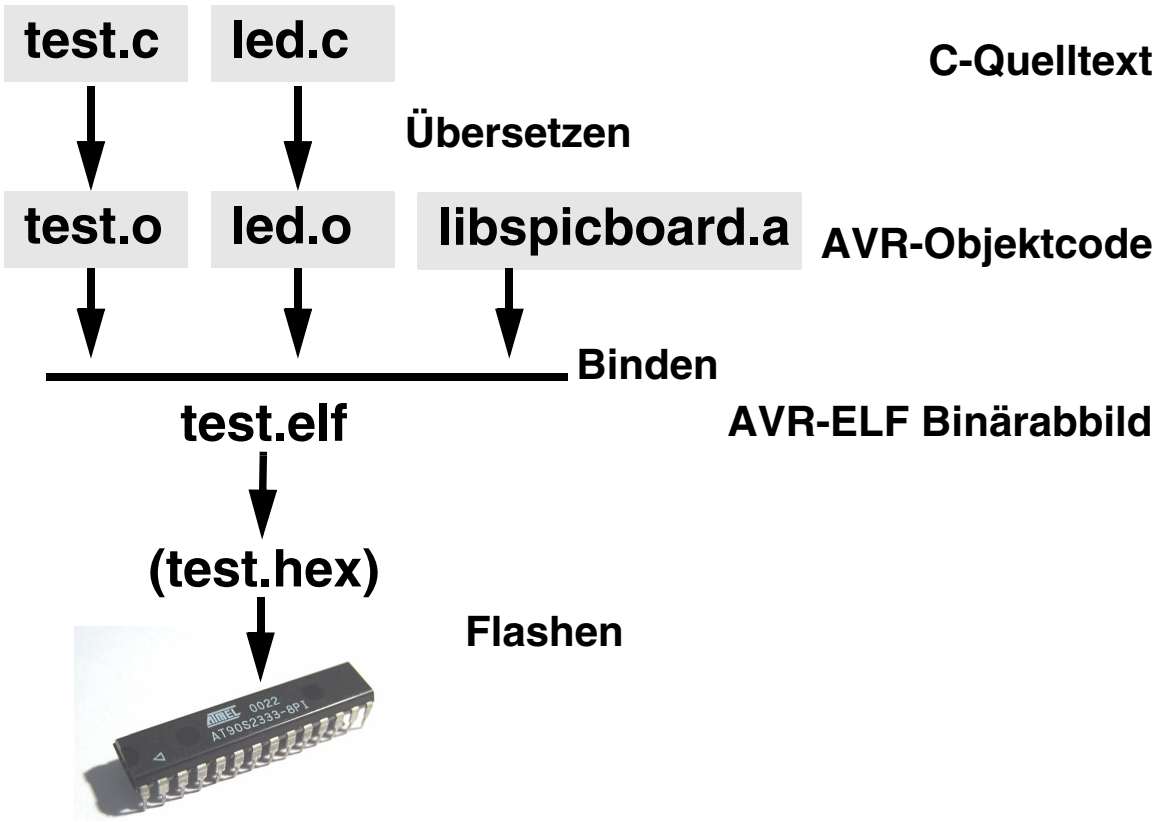
- Pin 2 von Port D (PD2) als Eingang nutzen, pull-up-Widerstand aktivieren und prüfen ob ein low-Pegel anliegt:

```
DDRD &= ~(1 << 2);/* PD2 als Eingang nutzen... */
PORTD|= (1 << 2);/* pull-up-Widerstand aktivieren*/

if ( (PIND & (1 << 2)) == 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

U4-3 Überblick: Modulare Softwareentwicklung



1 Modul-Schnittstelle

- Definiert
  - Funktionsprototypen
  - Typen
  - Daten (globale Variablen, nach Möglichkeit zu vermeiden!)
- Beschreibung der Schnittstelle in einer Header-(**.h**)-Datei
  - verbindliche Vorgabe für Implementierungen
  - darf nicht verändert werden (warum?)
  - Sichtbarkeit von Hilfsdaten/-funktionen auf Modul beschränken (`static`)
- Einbinden der Schnittstellenbeschreibung in anderen Modulen

2 Schnittstellenbeschreibung

- Erstellen einer **.h**-Datei (Konvention: gleicher Name wie **.c**-Datei)

```
#ifndef LED_H
#define LED_H

/* fixed-width Datentypen einbinden (werden im Header verwendet) */
#include <stdint.h>

/* LED-Typ */
typedef enum { RED0=0, YELLOW0=1, GREEN0=2, ... } LED;

/* Funktion zum Aktivieren einer bestimmten LED */
uint8_t sb_led_on(LED led);

/* Irgendeine Variable */
extern uint8_t einevariable;

...
#endif
```


2 Schnittstellenbeschreibung (2)

- Mehrfachinkludierung (evtl. Zyklen!) vermeiden
  - durch Definition und Abfrage eines Präprozessormakros
  - Konvention: das Makro hat den Namen der **.h**-Datei, '**.**' ersetzt durch '_'
  - der Inhalt wird nur eingebunden, wenn das Makro noch nicht definiert ist
- Flacher Namensraum: Wahl möglichst eindeutiger Namen

3 Einbinden von Schnittstellenbeschreibungen

- Einbinden mit `#include` dort, wo diese verwendet werden
  - im Header nur solche Schnittstellen, die auch im Header benötigt werden
    ☞ z.B. `stdint.h` im vorangehenden Beispiel
  - von der Implementierung verwendete Modulschnittstellen sind auch in dieser einzubinden
    ☞ verschiedene Implementierungen verwenden evtl. verschiedene Module

4 Initialisierung eines Moduls

- Module müssen oft Initialisierung durchführen (z.B. Ports konfigurieren)
  - z.B. in Java mit Klassenkonstruktoren möglich
  - C kennt kein solches Konzept
- Workaround: Modul muss bei erstem Aufruf einer seiner Funktionen ggf. die Initialisierung durchführen
  - muss sich merken, ob die Initialisierung schon erfolgt ist
  - Mehrfachinitialisierung vermeiden (Synchronisation!)

```
static uint8_t initDone = 0;
static void init(void) { ... }

void mod_func(void) {
    if(initDone == 0) {
        initDone = 1;
        init();
    }
```

- Initialisierung darf nicht mit anderen Modulen in Konflikt stehen!

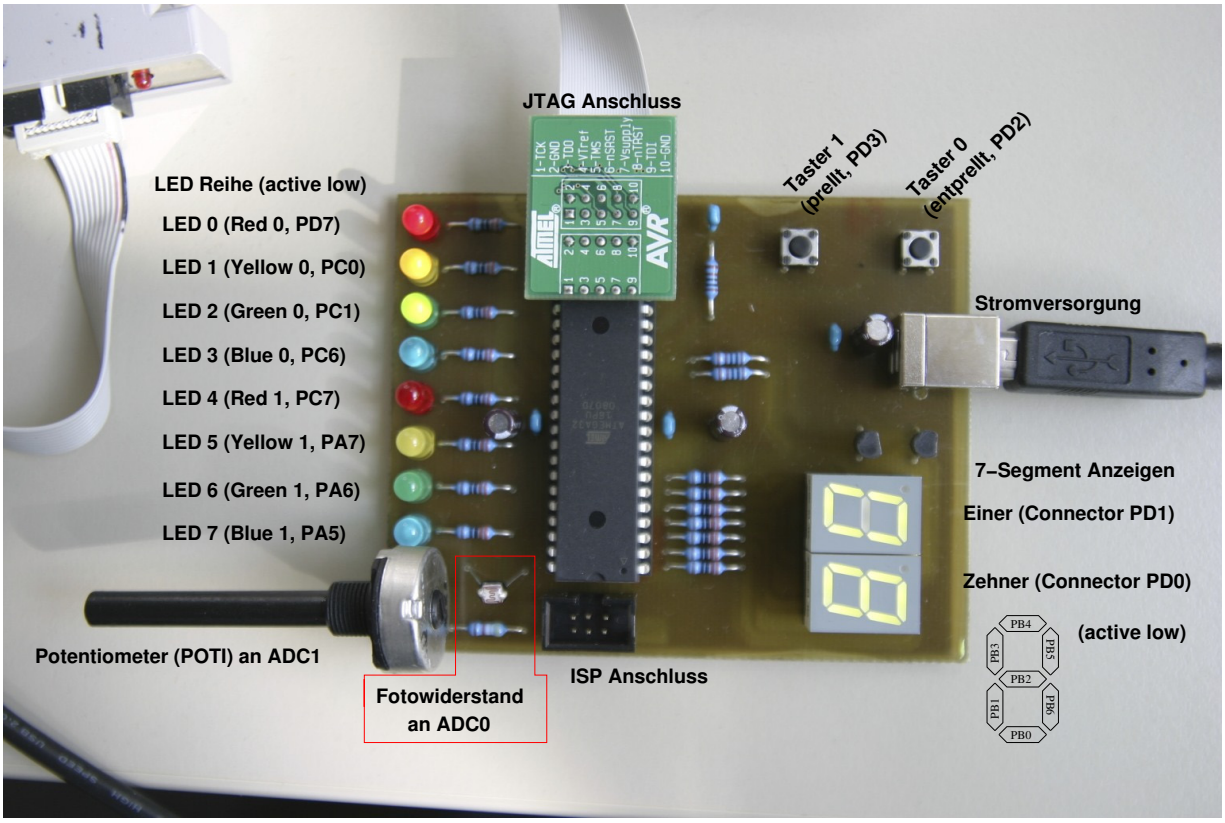
U4-4 Aufgabe 4: LED-Modul

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

1 LEDs des SPiCboard

- Die Anschlüsse und Namen der einzelnen LEDs können dem Übersichtsbild 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)



2 AVR-Studio Projekteinstellungen

- Projekt wie gehabt anlegen
  - Initiale Quelldatei: `test.c`
- Dann weitere Quelldatei `led.c` hinzufügen
- Wenn nun übersetzt wird, werden die Funktionen aus dem eigenen LED-Modul verwendet
- Andere Teile der Bibliothek werden nach Bedarf hinzugebunden