

Übungen zu Systemnahe Programmierung in C (SPiC)

Moritz Strübe, Rainer Müller
(Lehrstuhl Informatik 4)



Sommersemester 2014

Inhalt

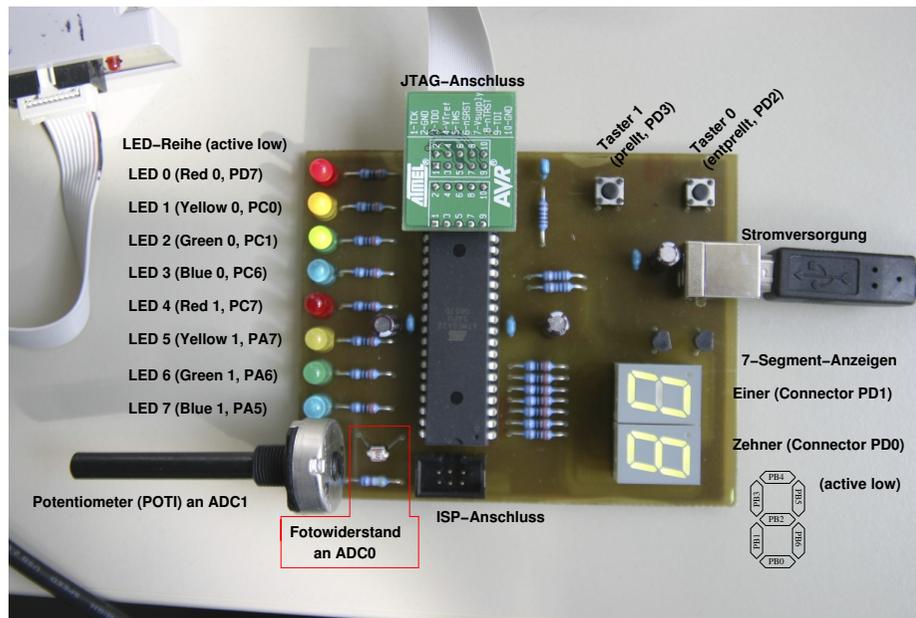
Aufgabe4
Led Modul

Wiederholung
Konfiguration der Pins

Hinweise zur Aufgabe



Led Modul



LED Module - Aufgabe

- Das LED-Modul der SPiCboard-Bibliothek selbst implementieren
 - Gleiches Verhalten wie das Original
 - Beschreibung:
http://www4.cs.fau.de/Lehre/SS14/V_SPIC/Uebung/doc
 - Das eigene Modul dann mit einem Testprogramm linken
 - Andere Teile der Bibliothek können für den Test benutzt werden 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



Konfiguration der Pins

- Jeder I/O-Port des AVR- μ 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



I/O-Port-Register

- DDR x : 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 x : 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 $i = 1 \rightarrow$ 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 x : Bit i gibt den aktuellen Wert des Pin i von Port x an (nur lesbar)



Beispiel: Initialisierung eines Ports

- Pin 3 von Port B (PB3) als Ausgang konfigurieren und auf Vcc schalten:

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

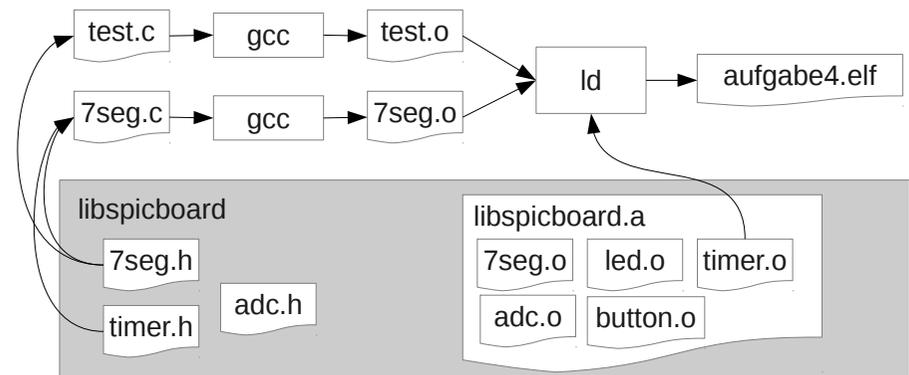
- Pin 2 von Port D (PD2) als Eingang nutzen, pull-up-Widerstand aktivieren und prüfen ob ein low-Pegel anliegt:

```
1 DDRD &= ~(1 << 2); /* PD2 als Eingang nutzen... */
2 PORTD |= (1 << 2); /* pull-up-Widerstand aktivieren */
3 if ( (PIND & (1 << 2)) == 0) { /* den Zustand auslesen */
4     /* ein low Pegel liegt an, der Taster ist gedrückt */
5 }
```

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



Funktionsweise des Linkers



Initialisierung eines Moduls

- Module müssen oft Initialisierung durchführen (z.B. Ports konfigurieren)
 - z.B. in Java mit Klassenconstructoren 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

```
1 static uint8_t initDone = 0;
2 static void init(void) { ... }
3 void mod_func(void) {
4     if(initDone == 0) {
5         initDone = 1;
6         init();
7     }
8     ....
```

■ Initialisierung darf nicht mit anderen Modulen in Konflikt stehen!

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
- Temporäres deaktivieren zum Test der Originalfunktiononen:

```
1 #if 0
2     ....
3 #endif
```