

Übungen zu Systemnahe Programmierung in C (SPiC)

Moritz Strübe, Rainer Müller
(Lehrstuhl Informatik 4)



Sommersemester 2013



Inhalt

Wiederholung
Konfiguration der Pins

Aufgabe4
7Seg Anzeige



- 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:

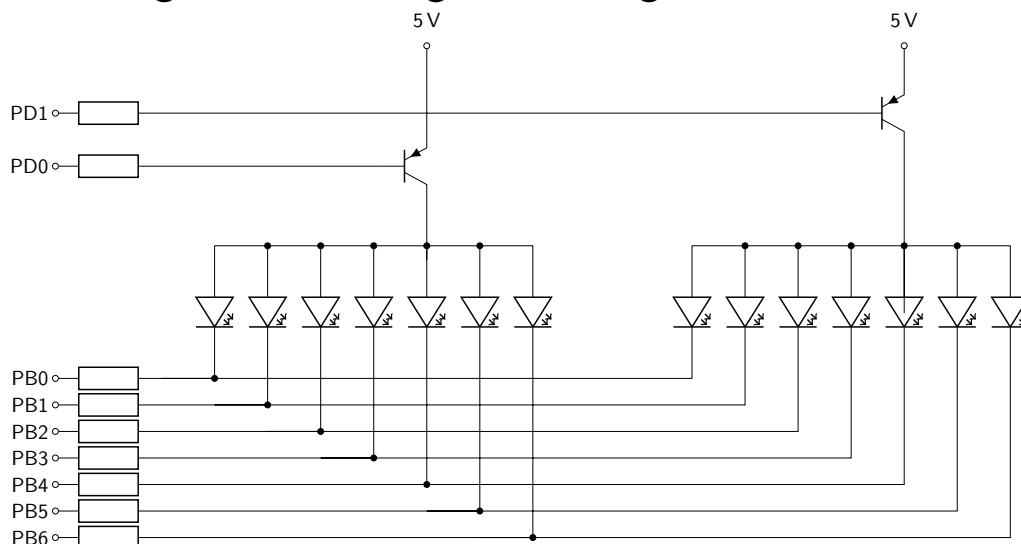
```
1 DDRD &= ~(1 << 2); /* PD2 als Eingang nutzen... */
2 PORTD|= (1 << 2); /* pull-up-Widerstand aktivieren*/
3 if ( (PIND & (1 << 2)) == 0) { /* den Zustand auslesen */
4     /* ein low Pegel liegt an, der Taster ist gedrückt */
5 }
```

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



Funktionsweise der 7Seg Anzeige

- Schaltung der Siebensegmentanzeige

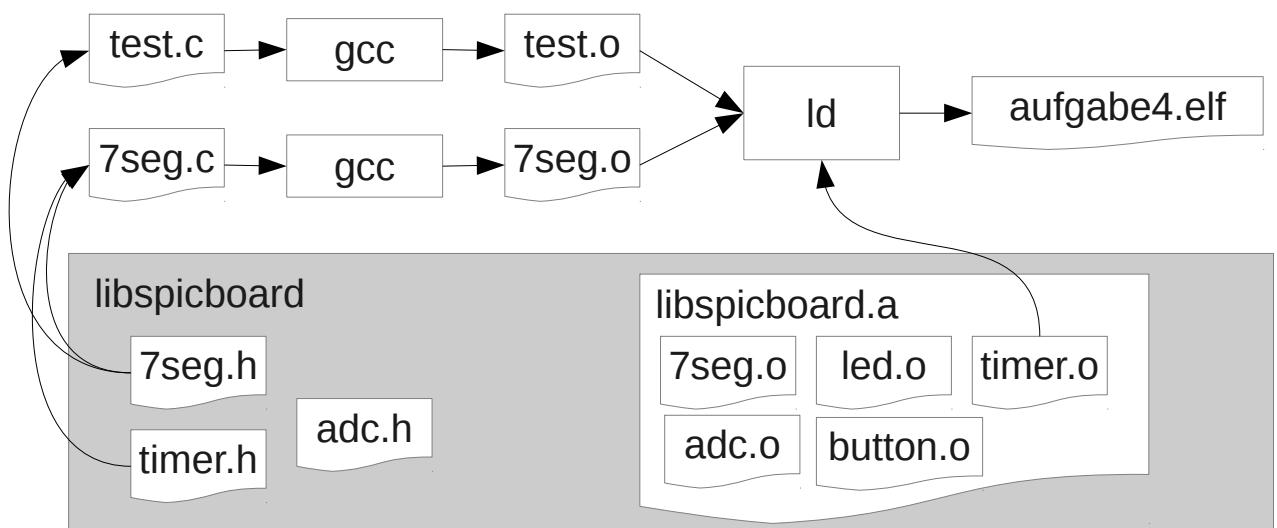


- PD7 geht zum ISP und kann als nicht angeschlossen betrachtet werden.
- Durch alternierende Aktivierung der beiden Anzeigen erscheinen beide aktiv

- Gleiches Verhalten wie das Original
 - Ausnahme: sb_7seg_showStr() muss nicht implementiert werden
 - Beschreibung:
http://www4.cs.fau.de/Lehre/SS13/V_SPIC/Uebung/doc
- Timer-Modul ermöglicht es, dass zu bestimmten Zeitpunkten eine Funktion aufgerufen wird
 - Die Funktion muss dem Modul als Pointer übergeben werden ↗ 13-19
 - Der Aufruf findet im Interrupt-Kontext statt
 - ⇒ Die aufgerufene Funktion sollte möglichst kurz sein



Funktionsweise des Linkers



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

```
1 #ifndef LED_H
2 #define LED_H
3 /* fixed-width Datentypen einbinden (werden im Header verwendet) */
4 #include <stdint.h>
5 /* LED-Typ */
6 typedef enum { RED0=0, YELLOW0=1, GREEN0=2, ... } LED;
7 /* Funktion zum Aktivieren einer bestimmten LED */
8 uint8_t sb_led_on(LED led);
9 /* Irgendeine Variable */
10 extern uint8_t einevariable;
11 ...
12 #endif
```

■ 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



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!)

```
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!



- Projekt wie gehabt anlegen
 - Initiale Quelldatei: test.c
 - Dann weitere Quelldatei 7seg.c hinzufügen
- Wenn nun übersetzt wird, werden die Funktionen aus dem eigenen 7SegModul verwendet
- Andere Teile der Bibliothek werden nach Bedarf hinzugebunden
- Temporäres deaktivieren zum Test der Originalfunktionen:

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

