Übungen zu Systemnahe Programmierung in C

Abschnitt 5.4: Aufgabe (led)

25.05.2020

Tim Rheinfels
Benedict Herzog
Bernhard Heinloth

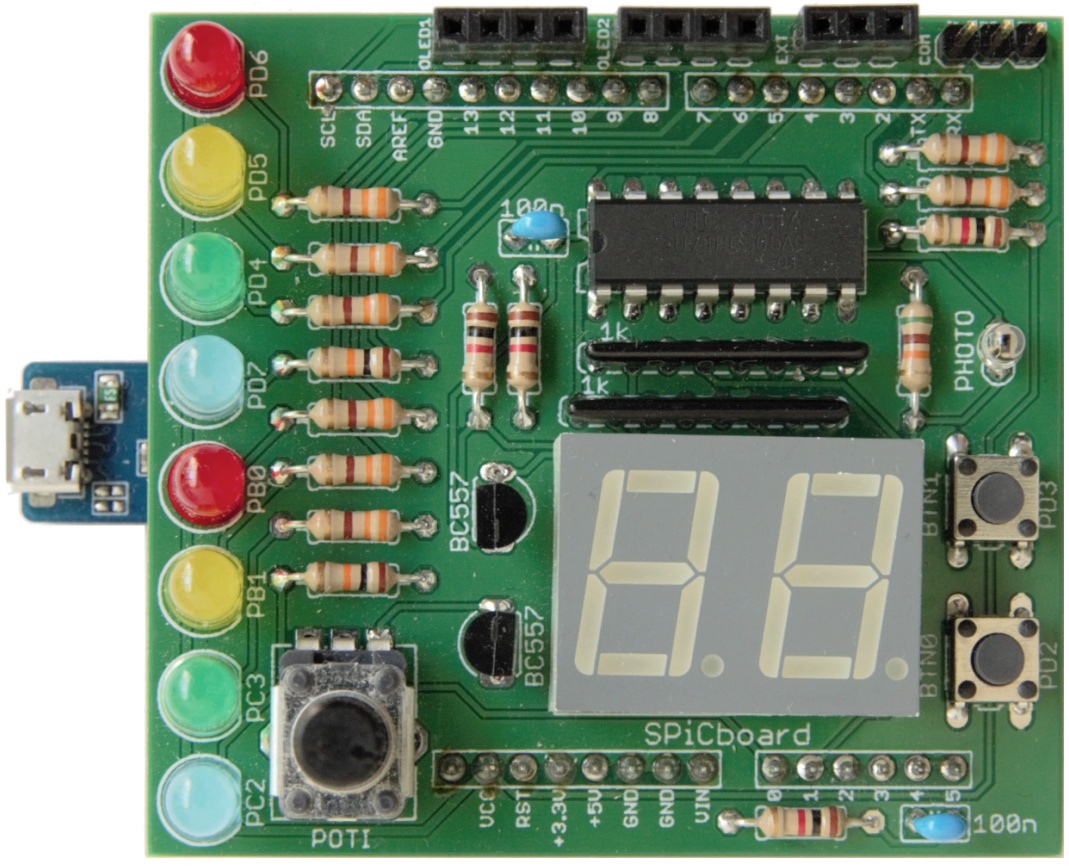
Lehrstuhl für Informatik 4
Friedrich-Alexander-Universität Erlangen-Nürnberg

Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

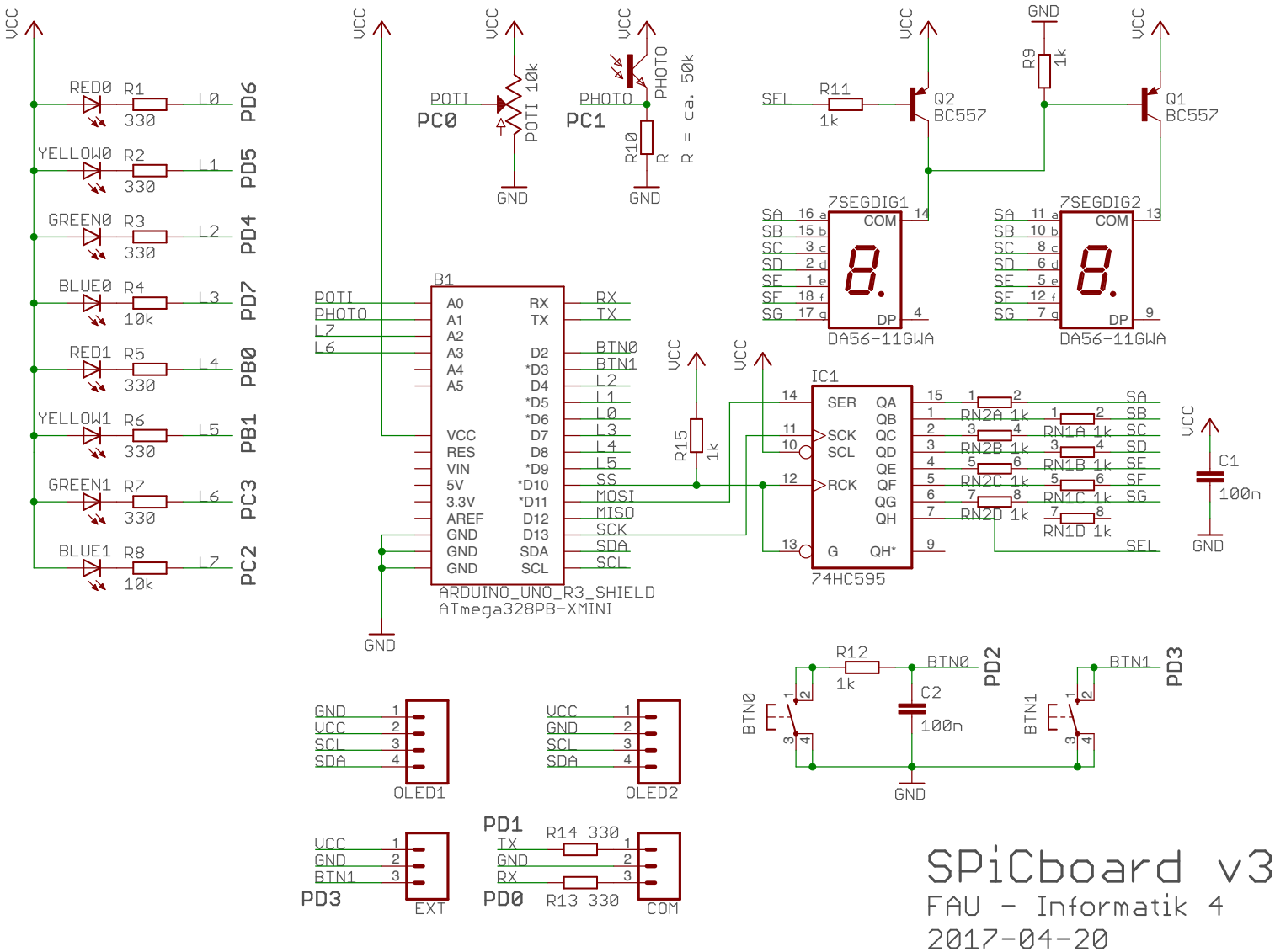
TECHNISCHE FAKULTÄT

LED-Modul – Übersicht



- LED 0 (REDo) ⇒ PD6 ⇒ Port D, Pin 6 ⇒ Bit 6 in PORTD und DDRD
- ...
- LED 7 (BLUE1) ⇒ PC2 ⇒ Port C, Pin 2 ⇒ Bit 2 in PORTC und DDRC

SPiCboard Schaltplan

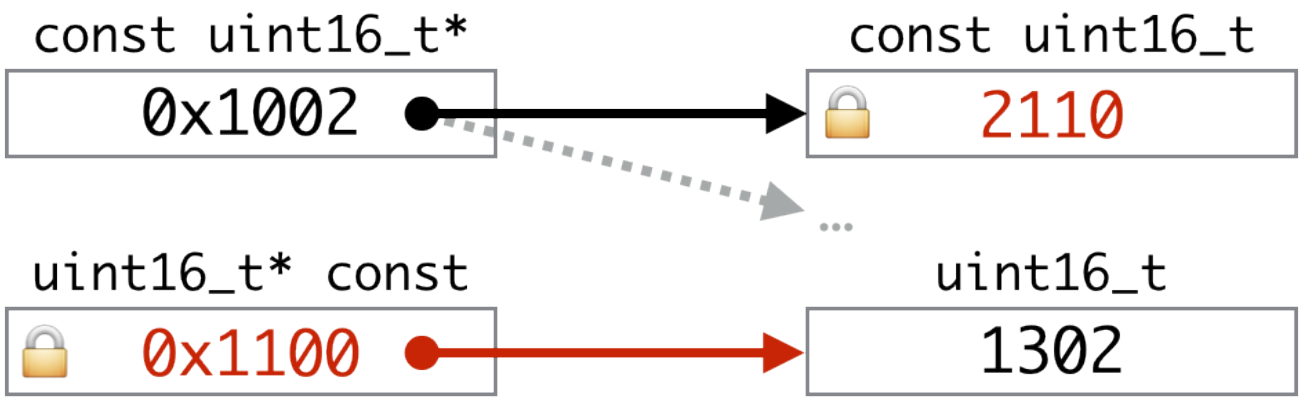


LED-Modul – Aufgabe

- LED-Modul der libspicboard selbst implementieren
  - Gleiches Verhalten wie das Original
  - Beschreibung: `http://www4.cs.fau.de/Lehre/SS20/V_SPIC/SPiCboard/group__LED.shtml`
- Testen des Moduls
  - Eigenes Modul mit einem Testprogramm (`test-led.c`) linken
  - Andere Teile der Bibliothek können für den Test benutzt werden
- LEDs des SPiCboards
  - 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
  - PD6 = Port D, Pin 6

Exkurs: const uint8_t* vs. uint8_t* const

- `const uint8_t*`
  - Ein Zeiger auf einen **konstanten** `uint8_t`-**Wert**
  - **Wert** nicht über den Zeiger veränderbar
- `uint8_t* const`
  - ein **konstanter Zeiger** auf einen (beliebigen) `uint8_t`-Wert
  - **Zeiger** darf nicht mehr auf eine andere Speicheradresse zeigen



Port- und Pin-Array (1)

- Adressoperator: &
- Verweisoperator: *
- Port und Pin Definitionen (in `avr/io.h`)

```
01 #define PORTD (* (volatile uint8_t *) 0x2B)
02 ...
03 #define PD0     0
04 ...
```

- Makro ersetzt `PORTD` durch `(* (volatile uint8_t *) 0x2B)`
  - Nimmt die Zahl `0x2B` (Adresse von PORTD)
  - Castet in `(volatile uint8_t *)` (volatile Zeiger)
  - Dereferenziert Zeiger `*` (schreibt in PORTD)

Port- und Pin-Array (2)

- Port Array:

```
01 static volatile uint8_t * const ports[8] = { &PORTD,
02                                               ...,
03                                               &PORTC };
```

  - Macht Dereferenzierung durch Adressoperator wieder rückgängig

    ⇒ In `ports` stehen Adressen als `uint8_t` Zeiger

- Pin Array:

```
01 static uint8_t const pins[8] = { PD6, ..., PC2 };
```

- Zugriff:

```
01 * (ports[0]) &= ~(1 << pins[0]);
```

Übersetzereinstellungen

- Projekt wie gehabt anlegen
  - Initiale Quelldatei: `test-led.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 Originalfunktionen:

⇒ Sieht der Compiler diese “Kommentare”?

⇒ Wie kann der Code wieder einkommentiert werden?

Testen des Moduls

```
01 void main(void){
02   ...
03   // 1.) Testen bei korrekter LED-ID
04   int8_t result = sb_led_on(RED0);
05   if(result != 0){
06     // Test fehlgeschlagen
07     // Ausgabe z.B. auf 7-Segment-Anzeige
08   }
09   // Einige Sekunden warten
10
11   // 2.) Testen bei ungueltiger LED-ID
12   ...
13 }
```

- Schnittstellenbeschreibung genau beachten (inkl. Rückgabewerte)
- Testen **aller möglichen Rückgabewerte**
- Fehler wenn Rückgabewert nicht der Spezifikation entspricht