Übungen zu Grundlagen der systemnahen Programmierung in C (GSPiC) im Wintersemester 2018/19

2018-10-22

Alexander von der Haar
Bernhard Heinloth

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

Lehrstuhl für Verteilte Systeme
und Betriebssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

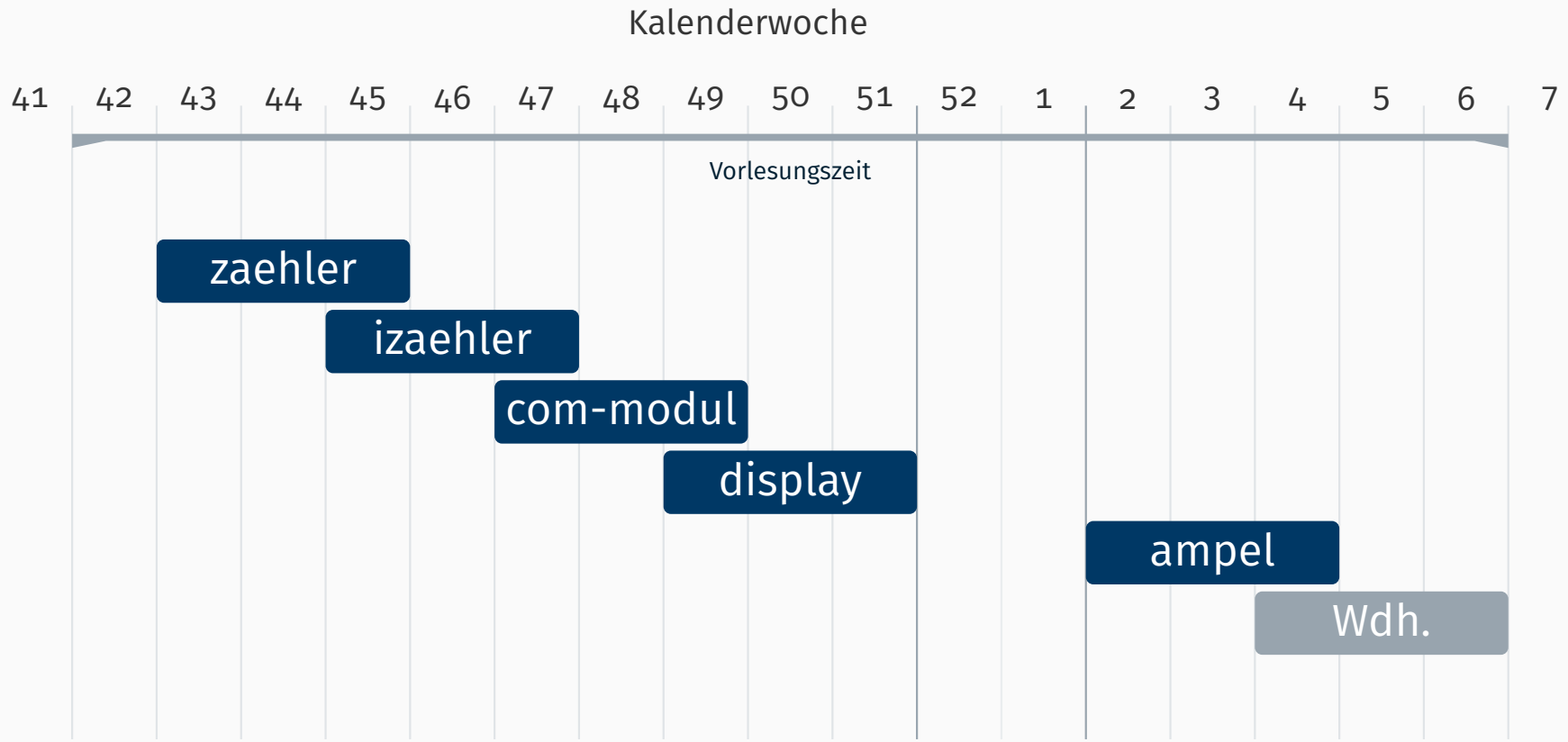
TECHNISCHE FAKULTÄT

Organisatorisches

Tafelübungen

- Tafelübung Do 10:15 – 11:45 (im Raum 00.153-113)
- Ablauf der Tafelübungen:
  1. Besprechung der alten Aufgabe
  2. Praxisnahe Vertiefung des Vorlesungsstoffes
  3. Vorstellung der neuen Aufgabe
  4. ggf. Entwicklung einer Lösungsskizze der neuen Aufgabe
- Folien nicht unbedingt zum Selbststudium geeignet
  → Anwesenheit, Mitschrift
- Übersicht aller GSPiC-Termine:
  `https://www4.cs.fau.de/Lehre/WS18/V_GSPIC/#woch`
- Semesterplan:
  `https://www4.cs.fau.de/Lehre/WS18/V_GSPIC/#sem`

Aufgaben


Kalenderwoche
41
42
43
44
45
46
47
48
49
50
51
52
1
2
3
4
5
6
7
Vorlesungszeit
zaehler
izaehler
com-modul
display
ampel
Wdh.

Lösungen

- Abgabe unter Linux
- automatische Plagiatsprüfung
  - Vergleich mit allen anderen (auch älteren) Lösungen
  - “abgeschriebene” Lösungen bekommen 0 Punkte

  $\Rightarrow$ Im Zweifelsfall beim Übungsleiter melden
- Punktabzug
  - -1 Punkte je Compilerwarnung
  - -50% der möglichen Punkte falls nicht übersetzbar
- (hilfreiche) Kommentare im Code helfen euch und dem Korrektor

Bonuspunkte

- abgegebene Aufgaben werden mit Übungspunkten bewertet
- ab 50% der erreichbaren Übungspunkte gibt es Bonuspunkte für die Klausur
- Umrechnung der Übungspunkte in Bonuspunkte für die Klausur (bis zu 10% der Punkte)
  - ⟶ Beispiel: 100% der Übungspunkte führen bei 90 möglichen Klausurpunkten zu 9 Bonuspunkten
- Bestehen der Klausur durch Bonuspunkte *nicht möglich*
- Bonuspunkte nicht in nächste Semester übertragbar

Rechnerübungen

- Rechnerübung Do 08:00 – 10:00 (im Raum 00.153-113)
- Unterstützung durch Übungsleiter bei der Aufgabenbearbeitung
- Falls 30 Minuten nach Beginn der Rechnerübung (also um 08:30) niemand anwesend ist, kann der Übungsleiter gehen

CipMap


CipMap
CIP2
Bib-CIP
CIP1
CIP1-N
Win-CIP
CIP3
CIP4
Huber-CIP
Tutorlogin
Lecture Mode
Opt-In
FAQ
Settings
Legal Notice
Privacy Policy
Collapse sidebar
0da
0de
0dd
0dc
0db
0di
0dh
0dg
0df

Anfragen via CipMap stellen

1. besuche die Seite cipmap.cs.fau.de
2. wähle an der obigen Bildschirmleiste den Raum der Rechnerübung aus (00.153-113 bzw. CIP3)
3. klicke links auf *Lecture Mode*. Daraufhin werden viele Rechner grau und einige farbig. Das sind Rechner, an denen bereits ein Request gestellt wurde
4. durch einen Klick auf *Request Tutor* wird eine Anfrage gestellt und in die Warteschlange eingereiht, dein Rechner färbt sich
5. nachdem deine Frage beantwortet wurde, klicke erneut auf die Schaltfläche, um die Anfrage zurückzuziehen

Bitte beachte

- Anfragen können nur zu den Zeiten gestellt werden, in denen die Übung offiziell stattfindet
- Loggst du dich aus, so werden all deine Requests gelöscht

Bei Problemen

- diese Folien konsultieren
- häufig gestellte Fragen (FAQ) und Antworten:

  `https://www4.cs.fau.de/Lehre/WS18/V_GSPIC/SPiCboard/faq.shtml`
- Fragen zu Übungsaufgaben im EEI-Forum posten (darf auch von anderen Studienrichtungen verwendet werden)

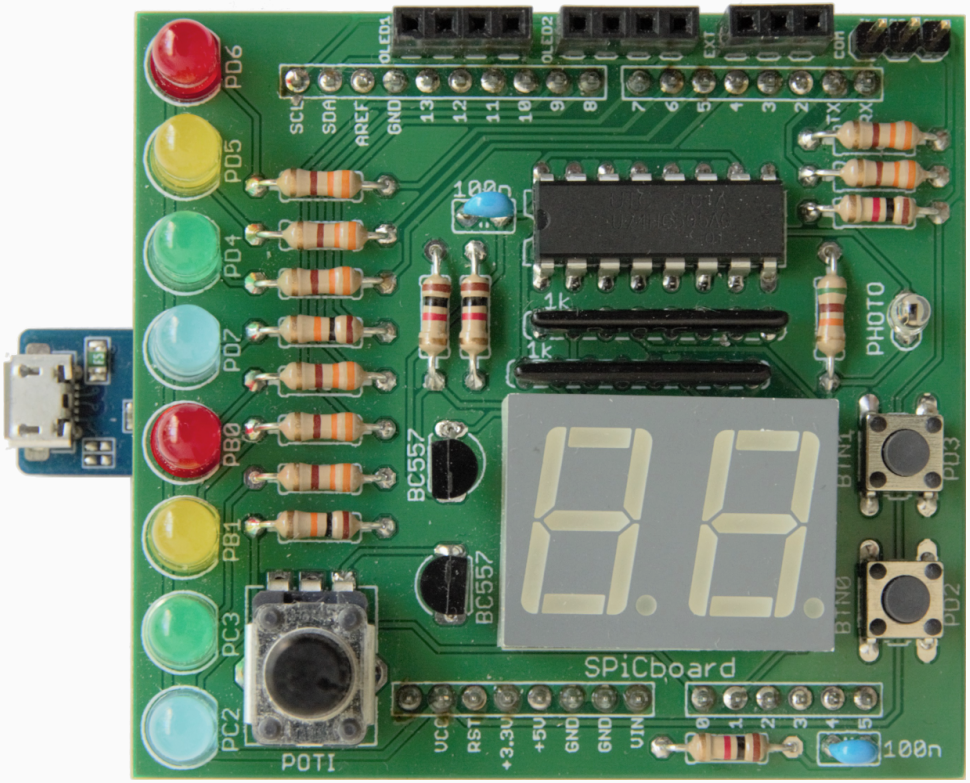
  `https://eei.fsi.uni-erlangen.de/forum/forum/16`
- bei speziellen Fragen Mail an Mailingliste, die alle Übungsleiter erreicht: i4spic@cs.fau.de

  $\Rightarrow$ zum Beispiel auch, wenn kein Übungsleiter auftauchen sollte

Entwicklungsumgebung

Hardware: SPiCboard

- **ATmega328PB Xplained Mini:**
  Mikrocontroller-Board mit integriertem Programmer/Debugger
- Speziell für (G)SPiC angefertigte **SPiCboard**s als Erweiterungsplatine



Aufgabenbearbeitung

- Betreute Bearbeitung der Aufgaben während der Rechnerübung
  - $\Rightarrow$ Hardware wird während der Übung zur Verfügung gestellt
- Selbständige Bearbeitung teilweise nötig
  - eigenes SPiCboard: Anfertigung am Lötabend (nur im Sommersemester)
  - SPiCboard Simulator: `SPiCsim`

Funktionsbibliothek

- `libspicboard`: Funktionsbibliothek zur Ansteuerung der Hardware

  Beispiel: `sb_led_on(GREEN0);` schaltet 1. grüne LED an
- direkte Konfiguration der Hardware durch Anwendungsprogrammierer nicht nötig
- Verwendung vor allem bei den ersten Aufgaben, später muss `libspicboard` teils selbst implementiert werden
- Dokumentation online:

  `https://www4.cs.fau.de/Lehre/WS18/V_GSPIC/SPiCboard/libapi.shtml`

Wichtige Verzeichnisse

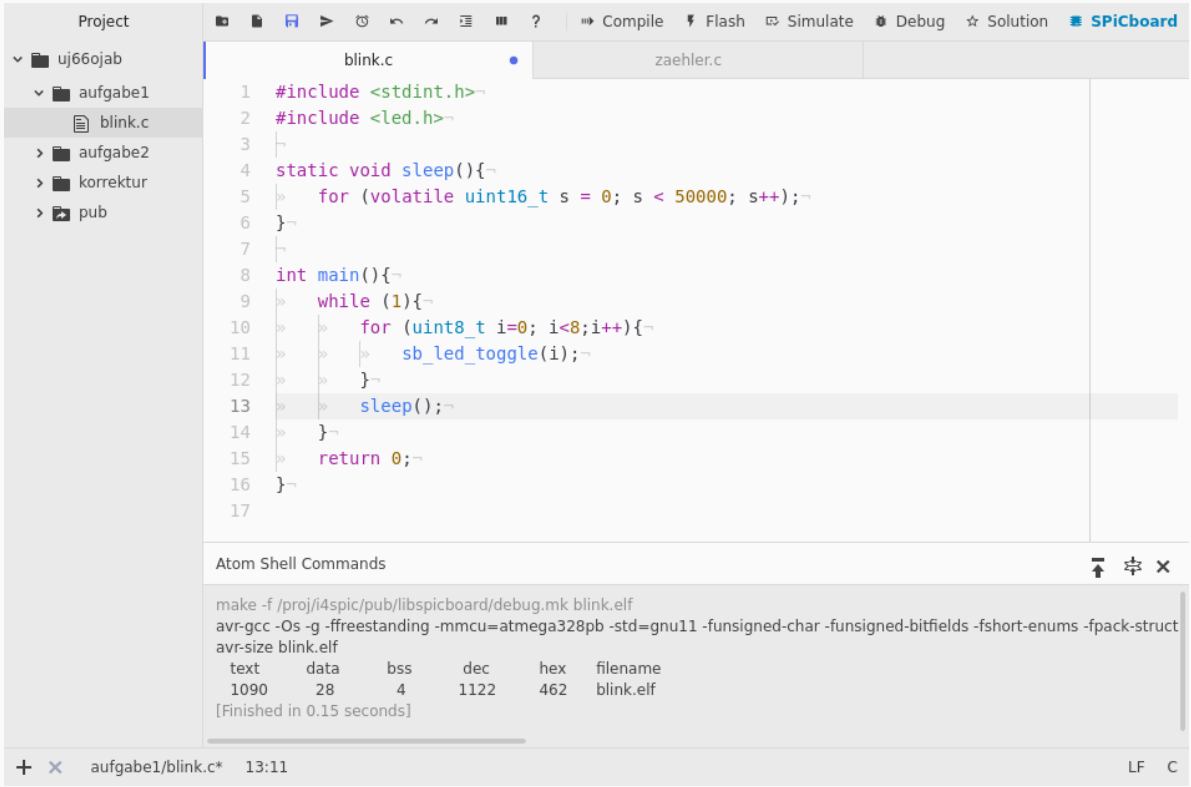
- Vorgabeverzeichnis `/proj/i4gspic/pub/`
  - Hilfsmaterial zu jeder Übungsaufgabe unter `aufgabeX/`
  - die Vorlesungsfolien in `folien/`
  - `libspicboard` mit Dokumentation sowie minimalen Beispiel
  - Hilfestellung zur Programmiersprache *C*

Wichtige Verzeichnisse

- Vorgabeverzeichnis `/proj/i4gspic/pub/`
  - Hilfsmaterial zu jeder Übungsaufgabe unter `aufgabeX/`
  - die Vorlesungsfolien in `folien/`
  - `libspicboard` mit Dokumentation sowie minimalen Beispiel
  - Hilfestellung zur Programmiersprache *C*
- Projektverzeichnis
  - `/proj/i4gspic/LOGINNAME/`
  - Lösungen hier in Unterordnern `aufgabeX` speichern

    $\Rightarrow$ das Abgabeprogramm sucht (nur) dort
  - für andere nicht lesbar
  - wird automatisch erstellt
  - enthält symbolische Verknüpfung zum Vorgabeverzeichnis

Der Editor


Project
uj66ojab
aufgabe1
blink.c
aufgabe2
korrektur
pub
Compile
Flash
Simulate
Debug
Solution
SPiCboard
blink.c
zaehler.c
#include <stdint.h>
#include <led.h>
static void sleep(){
for (volatile uint16_t s = 0; s < 50000; s++);
}
int main(){
while (1){
for (uint8_t i=0; i<8;i++){
sb_led_toggle(i);
}
sleep();
}
return 0;
}
Atom Shell Commands
make -f /proj/i4spic/pub/libspicboard/debug.mk blink.elf
avr-gcc -Os -g -ffreestanding -mmcu=atmega328pb -std=gnu11 -funsigned-char -funsigned-bitfields -fshort-enums -fpack-struct
avr-size blink.elf
text data bss dec hex filename
1090 28 4 1122 462 blink.elf
[Finished in 0.15 seconds]
aufgabe1/blink.c* 13:11
LF
C

Der Editor

- im Startmenü unter *FAU Courses* Eintrag *SPiC-IDE*
- speziell für (G)SPiC entwickelt, basierend auf Atom
- vereint Editor, Compiler und Debugger in einer Umgebung
- Cross-Compiler zur Erzeugung von Programmen für unterschiedliche Architekturen
  - Wirtssystem (engl. host): Intel-PC
  - Zielsystem (engl. target): AVR-Mikrocontroller
- detaillierte Anleitung auf `https://www4.cs.fau.de/Lehre/WS18/V_GSPIC/SPiCboard/cip.shtml`

Die SPiC-IDE wird im Wintersemester 2018/19 erstmalig eingesetzt

Wir wollen die IDE gerne noch besser machen und sind über Feedback und Hinweise auf Probleme bzw. Fehler sehr dankbar.

Hinweise an Bernhard Heinloth, per Mail an heinloth@cs.fau.de oder einfach im Büro 0.055 vorbei kommen!

Anleitung

CIP-Login

- Für die Benutzung der CIP Infrastruktur (und damit des Abgabesystems) ist ein CIP Login nötig
  - Bei Problemen bitte an die CIP Admins wenden
- Kriterien für sicheres Passwort:
  - Mindestens 8 Zeichen, besser 10
  - Mindestens 3 Zeichensorten, besser 4 (Groß-, Kleinbuchstaben, Zahlen, Sonderzeichen)
  - Keine Wörterbuchwörter, Namen, Login, etc.
- Passwort-Generierung zum Aussuchen mit folgendem Kommando:

```
pwgen -s 12
```

Abgabe

- Spätestens nach erfolgreichem Testen des Programms müssen Übungslösungen zur Bewertung abgegeben werden
- Bei Zweiergruppen darf nur ein Partner abgeben!
- Abgabe entweder per SPiC IDE Button oder
- Terminal-Fenster öffnen und folgendes Kommando ausführen (`aufgabeX` entsprechend ersetzen):

  ```
  /proj/i4gspic/bin/submit aufgabeX
  ```

  - Wichtig: **Grüner Text** signalisiert erfolgreiche Abgabe, **roter Text** einen Fehler!

- Fehlerursachen
  - Notwendige Dateien liegen nicht im richtigen Ordner
  - `aufgabeX` muss klein geschrieben sein
  - `.c`-Datei falsch benannt
  - Abgabetermin verpasst
- Nützliche Tools
  - Quelltext der abgegebenen Aufgabe anzeigen:
    `/proj/i4gspic/bin/show-submission aufgabeX`
  - Unterschiede zwischen abgegebener Version und Version im Projektverzeichnis `/proj/i4gspic/<login>` anzeigen:
    `/proj/i4gspic/bin/show-submission aufgabeX -d`
  - Eigenen Abgabetermin anzeigen:
    `/proj/i4gspic/bin/get-deadline aufgabeX`

Variablen

Verwendung von `int`

- Die Größe von `int` ist nicht genau definiert
- zum Beispiel beim ATmega328PB: 16 bit
  - ⇒ Gerade auf µC führt dies zu Fehlern und/oder langsameren Code
- Für die Übung gilt
  - Verwendung von `int` ist ein Fehler
  - Stattdessen: Verwendung der in der `stdint.h` definierten Typen: `int8_t`, `uint8_t`, `int16_t`, `uint16_t`, etc.
- Wertebereich
  - `limits.h`: `INT8_MAX`, `INT8_MIN`, ...
- Speicherplatz ist sehr teuer auf µC
  (SPIC BOARD/ATmega328PB hat nur 2048 Byte SRAM)

⇝ Nur so viel Speicher verwenden, wie tatsächlich benötigt wird!

Sichtbarkeit & Lebensdauer

Sichtbarkeit und Lebensdauer	nicht `static`	`static`
lokale Variable	Sichtbarkeit **Block** Lebensdauer **Block**	Sichtbarkeit **Block** Lebensdauer **Programm**
globale Variable	Sichtbarkeit **Programm** Lebensdauer **Programm**	Sichtbarkeit **Modul** Lebensdauer **Programm**
Funktion	Sichtbarkeit **Programm**	Sichtbarkeit **Modul**

- Lokale Variablen, die **nicht** `static` deklariert werden:

  ⇝ `auto` Variable (automatisch allokiert & freigegeben)
- Funktionen als `static`, wenn kein Export notwendig

Globale Variablen

```
01 static uint8_t state; // global static
02 uint8_t event_counter; // global
03
04 void main(void) {
05   /* ... */
06 }
07
08 static void f(uint8_t a) {
09   static uint8_t call_counter = 0; // local static
10   uint8_t num_leds; // local (auto)
11   /* ... */
12 }
```

- Sichtbarkeit/Gültigkeit möglichst weit **einschränken**
- Globale Variable $\neq$ lokale Variable in `f()`
- Globale `static` Variablen: Sichtbarkeit auf Modul beschränken

$\rightsquigarrow$ wo möglich, `static` für Funktionen und Variablen verwenden

Typedefs & Enums

```
01 #define PB3 3
02 typedef enum {
03     BUTTON0 = 0,
04     BUTTON1 = 1
05 } BUTTON;
06
07 void main(void) {
08     /* ... */
09     PORTB |= (1 << PB3); // nicht (1 << 3)
10
11     BUTTONSTATE old, new; // nicht uint8_t old, new;
12
13     // Deklaration: BUTTONSTATE sb_button_getState(BUTTON btn);
14     old = sb_button_getState(BUTTON0); // nicht
       ↪ sb_button_getState(0)
15     /* ... */
16 }
```

- Vordefinierte Typen verwenden
- Explizite Zahlenwerte nur verwenden, wenn notwendig

Aufgabe: Zähler

Aufgabenbeschreibung: Zähler

- Automatisches Hochzählen mit einer über das Potentiometer einstellbare Geschwindigkeit
- Anzeige erfolgt über 7-Segmentanzeige und LEDs
  - LEDs zu Beginn aus
  - Hunderterstelle durch LED visualisieren:
    `LED0` $\hat{=}$ 100, `LED1` $\hat{=}$ 200, etc.
  - bei Verlassen des anzeigbaren Wertebereichs Zähler zurücksetzen
- Bibliotheksfunktionen für Potentiometer, 7-Segmentanzeige und LED – Dokumentation auf der Webseite unter `https://www4.cs.fau.de/Lehre/WS18/V_GSPIC/SPiCboard/libapi.shtml`

Verwendung von Modulo

- Modulo ist der Divisionsrest einer Ganzzahldivision
- **Achtung**: In C ist das Ergebnis im negativen Bereich auch negativ
- Beispiel:

```
b = a % 4;
```

a =	-5	-4	-3	-2	-1	0	1	2	3	4	5	6
b =	-1	0	-3	-2	-1	0	1	2	3	0	1	2