

Übungen zu Systemnahe Programmierung in C (SPiC) – Wintersemester 2019/2020

Übung 1

Benedict Herzog
Bernhard Heinloth
Tim Rheinfels

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



Lehrstuhl für Verteilte Systeme
und Betriebssysteme

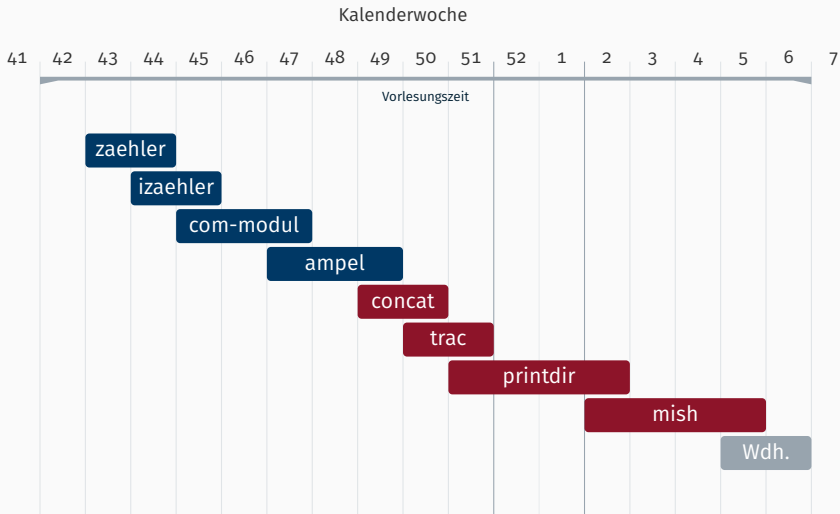


FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Organisatorisches

- 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
 5. Hands-on: gemeinsames Programmieren
- Folien nicht unbedingt zum Selbststudium geeignet
→ Anwesenheit, Mitschrift
- Übersicht aller SPiC-Termine:
https://www4.cs.fau.de/Lehre/WS19/V_SPIC/#woch
- Semesterplan:
https://www4.cs.fau.de/Lehre/WS19/V_SPIC/#sem



- Studierende die GSPiC belegen müssen nur die Mikrocontroller-Aufgaben abgeben
→ zaehler, izaehler, com, ampel
- Freiwillige Teilnahme an den Linux-Aufgaben ist selbstverständlich möglich
- Empfehlung: Letzte bzw. letzten Übungen zur Klausurvorbereitung

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

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

- 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 im Raum 01.153-113
- Unterstützung durch Übungsleiter bei der Aufgabenbearbeitung
- Falls 30 Minuten nach Beginn der Rechnerübung niemand anwesend ist, kann der Übungsleiter gehen
- Termine auf der Webseite:

https://www4.cs.fau.de/Lehre/WS19/V_SPIC/#woch

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

Ode

Odd

Odi

Odh

Odc

Odb

Odg

Odf

Oda

1. Besuche die Seite cipmap.cs.fau.de
2. Wähle den Raum der Rechnerübung aus (z.B. 01.153-113)
3. Klicke auf *Lecture Mode*.
 - **farbiger Rechner:** Hat einen Request gestellt
 - **grauer Rechner:** Kein Request gestellt
4. Durch einen Klick auf *Request Tutor* wird deine Anfrage in die Warteschlange eingereicht
5. Nachdem deine Frage beantwortet wurde: Schaltfläche erneut klicken, um die Anfrage zurückzuziehen

Bitte beachte:

- Anfragen können nur während einer Übung gestellt werden
- Loggst du dich aus, werden deine Requests gelöscht

- Folien konsultieren
- Häufig gestellte Fragen (FAQ) und Antworten:

https://www4.cs.fau.de/Lehre/WS19/V_SPIC/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>

- Darüber hinaus gehende Fragen:

Inhaltliche Fragen (Tutoren):

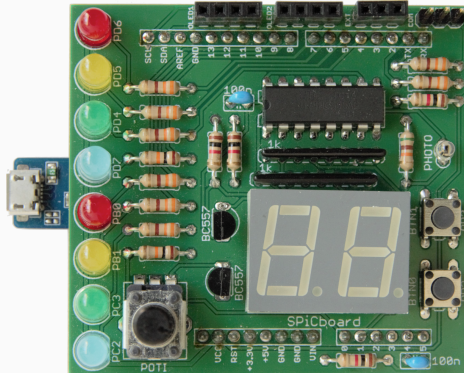
i4spic@lists.cs.fau.de

Organisatorische Fragen (Mitarbeiter):

i4spic-orga@lists.cs.fau.de

Entwicklungsumgebung

- **ATmega328PB Xplained Mini:**
Mikrocontroller-Board mit integriertem Programmer/Debugger
- Speziell für SPiC angefertigte **SPiCboards** als
Erweiterungsplatine



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

- `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/WS19/V_SPIC/SPiCboard/libapi.shtml

- Vorgabeverzeichnis `/proj/i4spic/pub/`
 - Hilfsmaterial zu jeder Übungsaufgabe unter `aufgabeX/`
 - die Vorlesungsfolien in `vorlesung/`
 - die Übungsfolien in `uebung/`
 - `libspicboard` mit Dokumentation sowie minimalen Beispiel
 - Hilfestellung zur Programmiersprache C

- Vorgabeverzeichnis `/proj/i4spic/pub/`
 - Hilfsmaterial zu jeder Übungsaufgabe unter `aufgabeX/`
 - die Vorlesungsfolien in `vorlesung/`
 - die Übungsfolien in `uebung/`
 - `libspicboard` mit Dokumentation sowie minimalen Beispiel
 - Hilfestellung zur Programmiersprache C
- Projektverzeichnis
 - `/proj/i4spic/<login>/`
 - Lösungen hier in Unterordnern `aufgabeX` speichern
 - ⇒ das Abgabeprogramm sucht (nur) dort
 - für andere nicht lesbar
 - wird automatisch erstellt
 - enthält symbolische Verknüpfung zum Vorgabeverzeichnis



Project

- uj66ojob
 - aufgabe1
 - blink.c
 - aufgabe2
 - korrektur
 - pub

blink.c

```
1 #include <stdint.h>
2 #include <led.h>
3
4 static void sleep(){
5
6 }
7
8 int main(){
9
10
11
12
13
14
15
16 }
17
```

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



- im Startmenü unter *FAU Courses* Eintrag *SPiC-IDE*
- speziell für 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/WS19/V_SPIC/SPiCboard/cip.shtml

Anleitung

- 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.



- Spätestens nach erfolgreichem Testen des Programms müssen Übungslösungen zur Bewertung abgegeben werden
- **Bei Zweiergruppen darf nur ein Partner abgeben!**
 - Der Partner muss aus der selben Gruppe sein
 - Bei der Abgabe wird der Partner-Login hinterlegt
- Abgabe entweder per SPiC IDE Button oder
- Terminal-Fenster öffnen und folgendes Kommando ausführen (aufgabeX entsprechend ersetzen):
`/proj/i4spic/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/i4spic/bin/show-submission aufgabeX`
- Unterschiede zwischen abgegebener Version und Version im Projektverzeichnis `/proj/i4spic/<login>` anzeigen:
`/proj/i4spic/bin/show-submission aufgabeX -d`
- Eigenen Abgabetermin anzeigen:
`/proj/i4spic/bin/get-deadline aufgabeX`

Variablen



- 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
(SPICBOARD/ATMEGA328PB hat nur 2048 Byte SRAM)
- ↪ Nur so viel Speicher verwenden, wie tatsächlich benötigt wird!



```
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
15     ↪ sb_button_getState(0)
16     /* ... */
17 }
```

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

Compileroptimierung



- AVR-Mikrocontroller, sowie die allermeisten CPUs, können ihre Rechenoperationen nicht direkt auf Variablen ausführen, die im Speicher liegen
- Ablauf von Operationen:
 1. **Laden** der Operanden aus dem Speicher in Prozessorregister
 2. **Ausführen** der Operationen in den Registern
 3. **Zurückschreiben** des Ergebnisses in den Speicher

⇒ Detaillierte Behandlung in der Vorlesung
- Der Compiler darf den Code nach Belieben ändern, solange der “globale” Zustand beim Verlassen der Funktion gleich bleibt
- Optimierungen können zu drastisch schnellerem Code führen



■ Typische Optimierungen:

- Beim Betreten der Funktion wird die Variable in ein Register geladen und beim Verlassen in den Speicher zurückgeschrieben
- Redundanter und "toter" Code wird weggelassen
- Die Reihenfolge des Codes wird umgestellt
- Für automatic Variablen wird kein Speicher reserviert; es werden stattdessen Prozessorregister verwendet
- Wenn möglich, übernimmt der Compiler die Berechnung (Konstantenfaltung):
 $a = 3 + 5$; wird zu $a = 8$;
- Der Wertebereich von automatic Variablen wird geändert:
Statt von 0 bis 10 wird von 246 bis 256 ($= 0$ für `uint8_t`) gezählt und dann geprüft, ob ein Überlauf stattgefunden hat



```
01 void wait(void) {  
02     uint8_t u8 = 0;  
03     while(u8 < 200) {  
04         u8++;  
05     }  
06 }
```

- Inkrementieren der Variable u8 bis 200
- Verwendung z.B. für aktive Warteschleifen



■ Assembler ohne Optimierung

```
01 ; void wait(void){
02 ; uint8_t u8;
03 ; [Prolog (Register sichern, Y initialisieren, etc)]
04 rjmp while      ; Springe zu while
05 ; u8++;
06 addone:
07 ldd r24, Y+1     ; Lade Daten aus Y+1 in Register 24
08 subi r24, 0xFF   ; Ziehe 255 ab (addiere 1)
09 std Y+1, r24     ; Schreibe Daten aus Register 24 in Y+1
10 ; while(u8 < 200)
11 while:
12 ldd r24, Y+1     ; Lade Daten aus Y+1 in Register 24
13 cpi r24, 0xC8    ; Vergleiche Register 24 mit 200
14 brcs addone     ; Wenn kleiner, dann springe zu addone
15 ;[Epilog (Register wiederherstellen)]
16 ret             ; Kehre aus der Funktion zurück
17 ;}
```



■ Assembler mit Optimierung

```
01 ; void wait(void){  
02 ret          ; Kehre aus der Funktion zurück  
03 ; }
```

- Die Schleife hat keine Auswirkung auf den Zustand

→ Die Schleife wird komplett wegoptimiert



- Variable können als `volatile` (engl. unbeständig, flüchtig) deklariert werden
- ↪ Der Compiler darf die Variable nicht optimieren:
 - Für die Variable muss **Speicher reserviert** werden
 - Die **Lebensdauer** darf nicht verkürzt werden
 - Die Variable muss vor jeder Operation aus dem **Speicher geladen** und danach ggf. wieder in diesen zurückgeschrieben werden
 - Der **Wertebereich** der Variable darf nicht geändert werden
- Einsatzmöglichkeiten von `volatile`:
 - Warteschleifen: Verhinderung der Optimierung der Schleife
 - nebenläufigen Ausführungen (später in der Vorlesung)
 - Variable wird im Interrupthandler und der Hauptschleife verwendet
 - Änderungen an der Variable müssen “bekannt gegeben werden”
 - Zugriff auf Hardware (z.B. Pins) ↪ wichtig für das LED Modul
 - (Debuggen: der Wert wird nicht wegoptimiert)

Aufgabe: 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:
 $\text{LED0} \hat{=} 100, \text{LED1} \hat{=} 200, \text{etc.}$
 - 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/WS19/V_SPIC/SPiCboard/libapi.shtml

- 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

Hands-on: Signallampe



- Morsesignale über LED o ausgeben
- Steuerung über Taster 1
- Nutzung der Bibliotheksfunktionen für Button und LED
- Dokumentation der Bibliothek:

https://www4.cs.fau.de/Lehre/WS19/V_SPIC/SPiCboard/libapi.shtml