Übungen zu Systemnahe Programmierung in C (SPiC) – Sommersemester 2024

Übung 1

Maxim Ritter von Oncuil
Arne Vogel

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

Lehrstuhl für Informatik 4
Systemsoftware

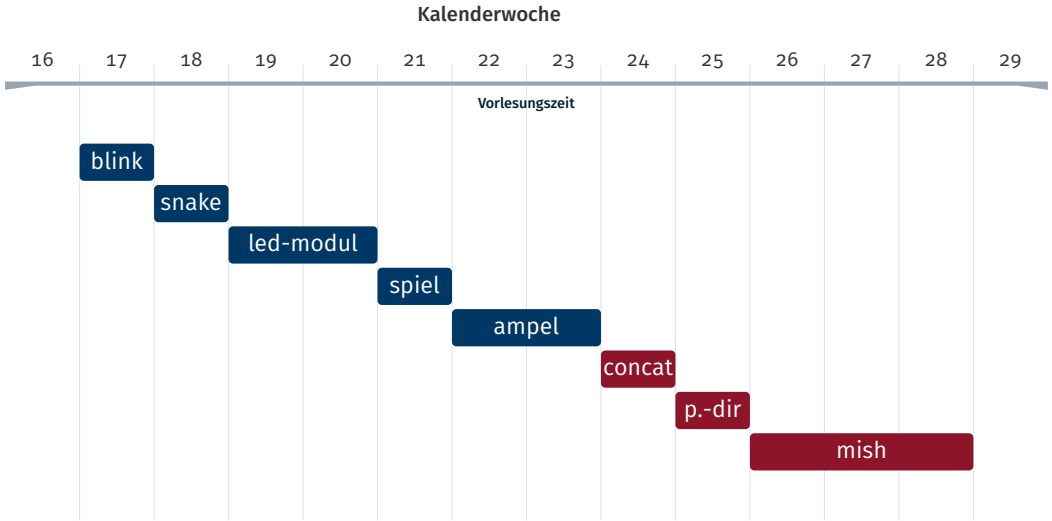
Friedrich-Alexander-Univ
Technische Fakultät

Organisatorisches

Tafelübungen

- 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
- Semesterplan und Übersicht aller SPiC-Termine:
  `https://sys.cs.fau.de/lehre/SS24/spic/`

Aufgaben



GSPiC

- Studierende, die GSPiC belegen, müssen nur die Mikrocontroller-Aufgaben abgeben
  - ⟶ blink, snake, led-modul, spiel, ampel
- Freiwillige Teilnahme an den Linux-Aufgaben ist selbstverständlich möglich
- Empfehlung: Letzte bzw. letzten Übungen zur Klausurvorbereitung

Lösungen

- 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

Bonuspunkte

- Abgegebene Aufgaben werden mit Übungspunkten bewertet
- Ab 20% der erreichbaren Übungspunkte gibt es Bonuspunkte für die Klausur
- Ab 80% der erreichbaren Übungspunkte gibt es die vollen Bonuspunkte
- Umrechnung der Übungspunkte in Bonuspunkte für die Klausur (bis zu 10% der Punkte)
  - ⟶ Beispiel: 80% der Übungspunkte führen bei 90 möglichen Klausurpunkten zu 9 Bonuspunkten
- Bestehen der Klausur durch Bonuspunkte *nicht möglich*

Rechnerübungen

- Raum der Rechnerübungen: 01.153-113 (WinCIP)
- Unterstützung durch Übungsleiter bei der Aufgabenbearbeitung

  Freie Plätze nach dem „First come, first served“-Prinzip
- Falls 30 Minuten nach Beginn der Rechnerübung niemand anwesend ist, kann der Übungsleiter gehen
- Termine auf der Webseite:

  `https://sys.cs.fau.de/lehre/SS24/spic/`

Bei Problemen

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

  `https://sys.cs.fau.de/lehre/SS24/spic/uebung/spicboard/faq`
- Fragen zum Stoff gerne im StudOn Forum:

  `https://www.studon.fau.de/frm5700999.html`
- Darüber hinaus gehende Fragen:

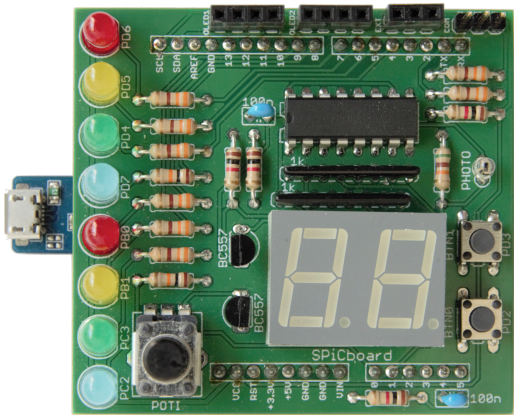
  Inhaltliche Fragen (Tutoren):
  i4spic@lists.cs.fau.de

  Organisatorische Fragen (Mitarbeiter):
  i4spic-orga@lists.cs.fau.de

Entwicklungsumgebung

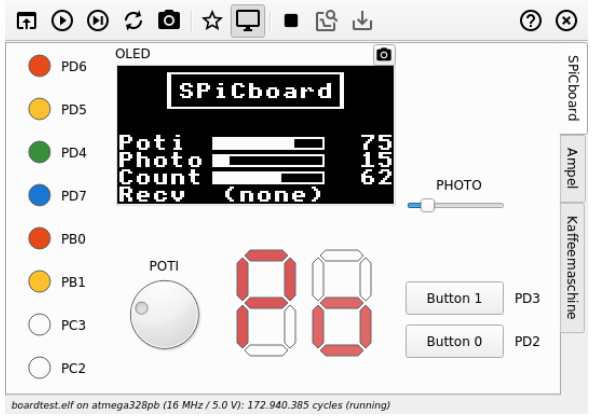
Hardware: SPiCboard

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



Simulator: SPiCsim

- **SPiCsim:**
  Simuliert ATmega328PB und SPiCBoard
- Erlaubt Aufzeichnung und Darstellung der Signale



Aufgabenbearbeitung

- Betreute Bearbeitung der Aufgaben während der Rechnerübungen
  - ⇒ 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:

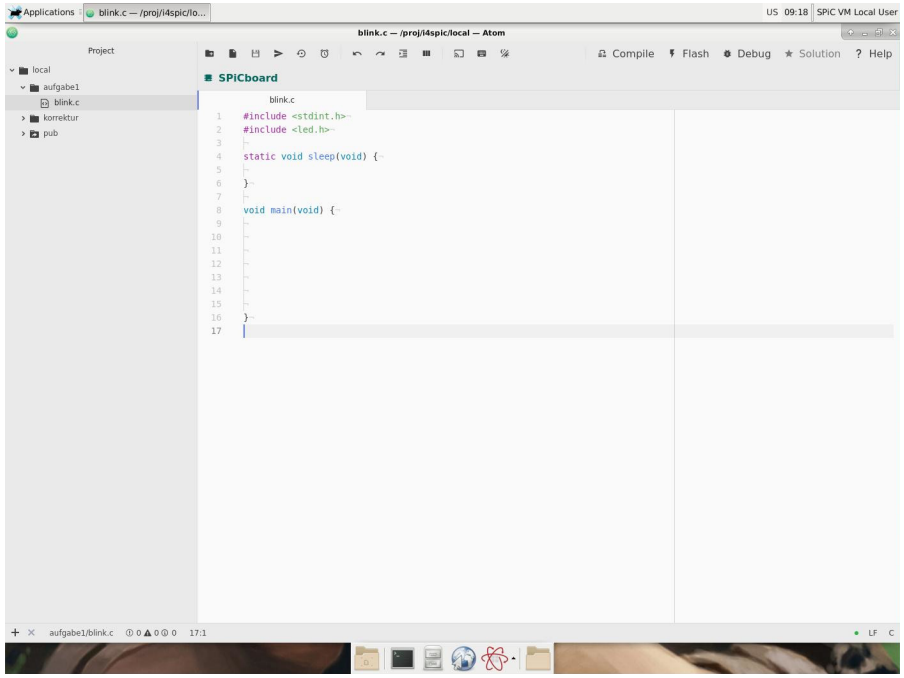
  `https://sys.cs.fau.de/lehre/SS24/spic/uebung/spicboard/libapi`

Wichtige Verzeichnisse

- Vorgabeverzeichnis `/proj/i4spic/<login>/pub/`
  - Hilfsmaterial zu jeder Übungsaufgabe unter `aufgabeX/`
  - `libspicboard` mit Dokumentation sowie minimalem Beispiel
  - Die Vorlesungsfolien in `vorlesung/` (VM: Nur in der Remote-IDE)
  - Die Übungsfolien in `uebung/` (VM: Nur in der Remote-IDE)
  - Hilfestellung zur Programmiersprache *C* (VM: Nur in der Remote-IDE)

Wichtige Verzeichnisse

- Vorgabeverzeichnis `/proj/i4spic/<login>/pub/`
  - Hilfsmaterial zu jeder Übungsaufgabe unter `aufgabeX/`
  - `libspicboard` mit Dokumentation sowie minimalem Beispiel
  - Die Vorlesungsfolien in `vorlesung/` (VM: Nur in der Remote-IDE)
  - Die Übungsfolien in `uebung/` (VM: Nur in der Remote-IDE)
  - Hilfestellung zur Programmiersprache *C* (VM: Nur in der Remote-IDE)
- 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



Der Editor

- 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 unter
  `https://sys.cs.fau.de/lehre/SS24/spic/uebung/spicboard/cip`

13

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.

Abgabe (1)

- 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!

Abgabe (2)

- 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`

Eure Todos

1. Anmeldung in StudOn: `https://www.studon.fau.de/crs5610197.html`
   - Forum zum Fragen stellen
2. Anmeldung zu den Übungen über Waffel: `https://waffel.cs.fau.de`
   - Für Abgabe und Korrektur der Aufgaben

   ⇒ ab **Donnerstag, 18.04.2024, 18:00 Uhr**
3. Anmeldung im Informatik CIP: `https://account.cip.cs.fau.de`
   - Für Bearbeiten, Abgabe und Korrektur der Aufgaben

Eure Todos

1. Anmeldung in StudOn: `https://www.studon.fau.de/crs5610197.html`
   - Forum zum Fragen stellen
2. Anmeldung zu den Übungen über Waffel: `https://waffel.cs.fau.de`
   - Für Abgabe und Korrektur der Aufgaben

   ⇒ ab **Donnerstag, 18.04.2024, 18:00 Uhr**
3. Anmeldung im Informatik CIP: `https://account.cip.cs.fau.de`
   - Für Bearbeiten, Abgabe und Korrektur der Aufgaben

Da es bis zu 24h dauern kann, bis nach der Anmeldung die erforderlichen Änderungen aktiv sind, solltet ihr euch **umgehend darum kümmern**. Vorher ist eine Bearbeitung und Abgabe der Übungsaufgaben nicht möglich!

Compileroptimierung

Compileroptimierung: Hintergrund

- 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

Compileroptimierung: Beispiele

- 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

Compileroptimierung: Beispiel (1)

```
01 void wait(void) {
02   uint8_t u8 = 0;
03   while(u8 < 16) {
04     u8++;
05   }
06 }
```

- Inkrementieren der Variable u8 bis 16
- Verwendung z.B. für aktive Warteschleifen

Compileroptimierung: Beispiel (2)

- 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 < 16)
11 while:
12 ldd r24, Y+1    ; Lade Daten aus Y+1 in Register 24
13 cpi r24, 0x10   ; Vergleiche Register 24 mit 16
14 brcs addone     ; Wenn kleiner, dann springe zu addone
15 ;[Epilog (Register wiederherstellen)]
16 ret             ; Kehre aus der Funktion zurück
17 ;}
```


Compileroptimierung: Beispiel (3)

- Assembler mit Optimierung

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


Compileroptimierung: Beispiel (3)

- Assembler mit Optimierung

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

- C kennt die Wartesemantik der Schleife nicht
- Die Schleife hat keine Auswirkung auf den (globalen) Zustand

⇝ Der Compiler optimiert sie komplett weg

Schlüsselwort volatile

- 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: blink

Aufgabenbeschreibung: blink

- Lernziel:
  - Umgang mit Programmierwerkzeugen und dem Abgabesystem
  - Aktives Warten
- Blinkende LEDs `YELLOW0` und `YELLOW1`
  - Abwechselnd an- bzw. ausschalten (Warnlicht)
  - Frequenz ca. 1 mal pro halbe Sekunde
  - Nutzung der Bibliotheksfunktionen für LEDs
  - Implementierung durch aktives Warten (Schleife mit Zähler)
- Dokumentation der Bibliothek:

  `https://sys.cs.fau.de/lehre/SS24/spic/uebung/spicboard/libapi`
- Abzugebende Datei: `blink.c`

Hands-on: Licht

Screencast: `https://www.video.uni-erlangen.de/clip/id/13444`

Hands-on: Licht

- In der SPiC-IDE:
  - Neuen Ordner erstellen (z.B. hands-on/licht)
  - Neue Quellcodedatei erstellen (z.B. `licht.c`)
- Programm erstellen:
  - Schalte eine LED ein (z.B. `GREEN0`)
  - Warte in einer Endlosschleife
- In der SPiC-IDE:
  - Programm übersetzen
  - Programm im Simulator oder auf dem SPiCboard testen.