

Übungen zu Grundlagen der systemnahen Programmierung in C (GSPIC) im Sommersemester 2018

2018-04-13

Bernhard Heinloth

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



Lehrstuhl für Verteilte Systeme
und Betriebssysteme

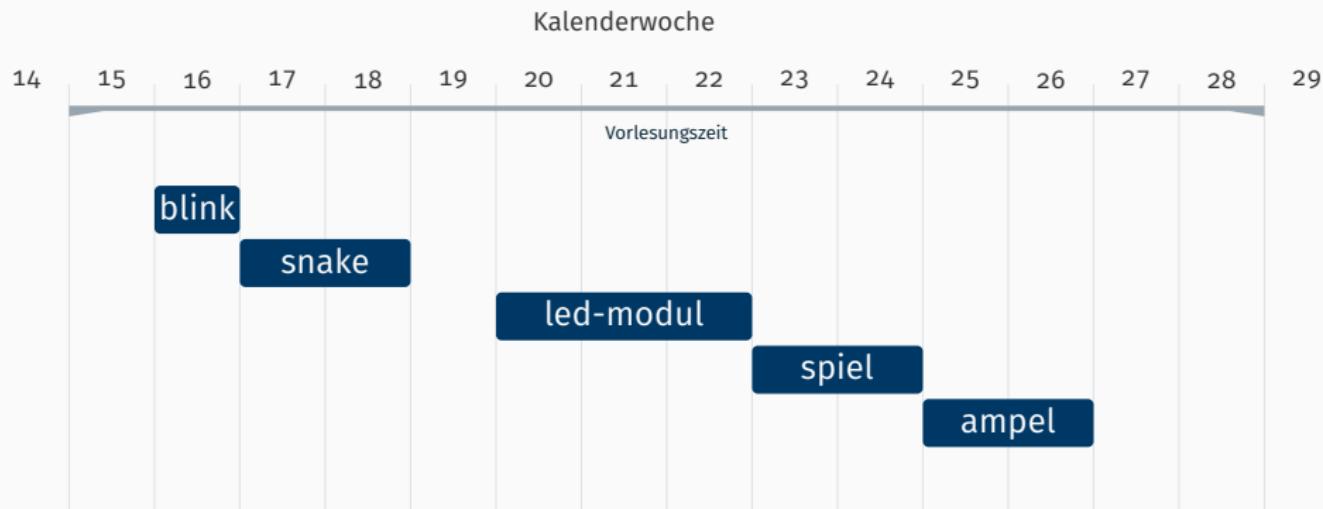


FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Organisatorisches

- 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/SS18/V_GSPIC/#woch
- Semesterplan:
https://www4.cs.fau.de/Lehre/SS18/V_GSPIC/#sem

Aufgaben



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 Punkte je Compilerwarnung
 - -50% der möglichen Punkte falls nicht übersetzbare
- (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

- Räume der Rechnerübungen: 01.153-113 (und 01.155N-113)
- 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://www4.cs.fau.de/Lehre/SS18/V_GSPIC/#woch

CipMap

02.151 02.135 01.155 01.155N 01.153 00.153 00.156 0.01-142 Tutorlogin

Lecture Mode

Opt-in

FAQ

Settings

Legal Notice

Privacy Policy

< Collapse sidebar

09a

09e 09d 09c 09b

09i 09h 09g
User: N/A 09f

01d
User: N/A 01c 01b 01a

09j

Anfragen via CipMap stellen

1. besuche die Seite cipmap.cs.fau.de
2. wähle an der obigen Bildschirmleiste den Raum der Rechnerübung aus (01.153-113 bzw Win-CIP)
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

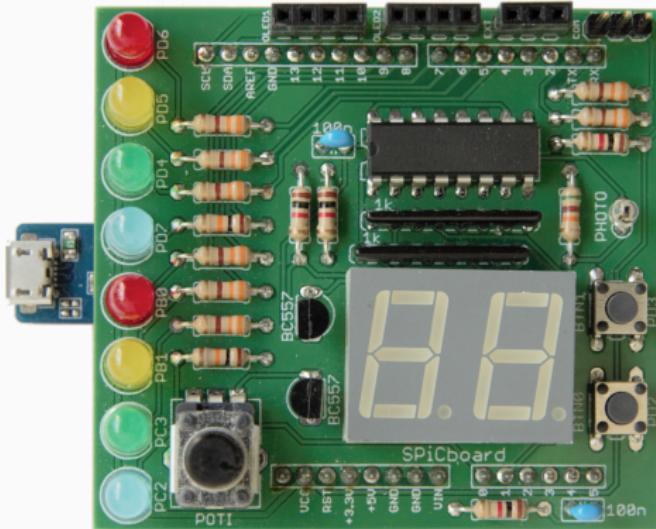
Bei Problemen

- diese Folien konsultieren
- häufig gestellte Fragen (FAQ) und Antworten:
https://www4.cs.fau.de/Lehre/SS18/V_GSPIC/Uebung/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
⇒ 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 **SPiCboards** als Erweiterungsplatine



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
 - 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/SS18/V_GSPIC/Uebung/doc

Wichtige Verzeichnisse

■ Projektverzeichnis

- in Windows: Q:\
- in Linux: /proj/i4gspic/LOGINNAME/
- Lösungen hier in Unterordnern aufgabeX speichern
 ⇒ das Abgabeprogramm sucht (nur) dort
- für andere nicht lesbar
- wird automatisch erstellt

■ Heimverzeichnis

- in Windows: Z:\
- in Linux: ~

Wichtige Verzeichnisse

- Vorgabeverzeichnis P:\ (Windows) bzw. /proj/i4gspic/pub/ (Linux) mit
 - Hilfsmaterial und Binärmusterlösungen zu jeder Übungsaufgabe unter aufgabeX/
 - die Vorlesungsfolien in folien/
 - Programm zum Testen der Einheiten auf dem SPiCboard unter boardtest/
 - libspicboard-Bibliothek und -Dokumentation sowie ein minimales Beispiel
 - Hilfestellung zur Programmiersprache C
- Falls eines der Netzlaufwerke nicht angezeigt wird:
 - Windows Explorer – Computer – Map network drive
 - Z:\ unter \\faui03\LOGINNAME
 - P:\ unter \\faui03\i4gspicpub
 - Q:\ unter \\faui03\i4gspichome

Entwicklungsumgebung: Atmel Studio

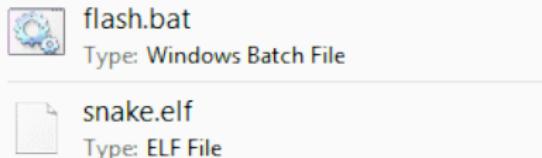
- Programmierung mit Atmel Studio 7 unter Windows
- vereint Editor, Compiler und Debugger in einer Umgebung
- Cross-Compiler zur Erzeugung von Programmen für unterschiedliche Architekturen
 - Wirtssystem (engl. host): Intel-PC
 - Zielsystem (engl. target): AVR-Mikrocontroller

Anleitung

- Für die Benutzung der CIP Infrastruktur (und damit des Abgabesystems) ist ein CIP Login nötig
- Zur Bearbeitung der Übungen ist ein Windows-Login nötig:
 - Im Raum 01.155 mit Linux-Passwort einloggen
 - Ein Terminalprogramm öffnen und dort folgendes Kommando ausführen:
`cip-set-windows-password`
- 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`

Binärabbild flashen

- Nötig, um vorgefertigte Binärabbilder (.elf-Images) zu testen, z. B. Binärmusterlösungen unter P:\aufgabeX
- Möglich mit Skript flash.bat im jeweiligen Verzeichnis, Ausführen mit Doppelklick



- Nach erfolgreichem Flashen führt das Board das Programm direkt aus
- Neustart des Programms ist durch Trennen und Wiederherstellen der USB-Stromversorgung möglich

Abgabe (1)

- Spätestens nach erfolgreichem Testen des Programms müssen Übungslösungen zur Bewertung abgegeben werden
- Bei Zweiergruppen darf nur ein Partner abgeben!
- Abgabe unter einer Linux-Umgebung per Remote Login
 - Start ~ Alle Programme ~ PuTTY ~ PuTTY
 - Host Name: faui0sr0 bzw. von Zuhause faui0sr0.cs.fau.de
 - Open
 - PuTTY Security Alert mit "Ja" bestätigen
 - Login mit Benutzername und **Linux**-Passwort
- Im erscheinenden Terminal-Fenster folgendes Kommando ausführen (aufgabeX entsprechend ersetzen):
`/proj/i4gspic/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/i4gspic/bin/show-submission aufgabeX`
 - Unterschiede zwischen abgegebener Version und Version im Projektverzeichnis `Q:\aufgabeX` anzeigen:
`/proj/i4gspic/bin/show-submission aufgabeX -d`
 - Eigenen Abgabetermin anzeigen:
`/proj/i4gspic/bin/get-deadline aufgabeX`

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 < 200) {  
04         u8++;  
05     }  
06 }
```

- Inkrementieren der Variable u8 bis 200
- 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 < 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 ;}
```

Compileroptimierung: Beispiel (3)

- 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

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 gegebenenfalls 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 in 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 BLUE0 und BLUE1
 - Abwechselnd an- bzw. ausschalten (Blaulicht)
 - Frequenz ca. 1 mal pro Sekunde
 - Nutzung der Bibliotheksfunktionen für LEDs
 - Implementierung durch aktives Warten (Schleife mit Zähler)
 - Übersetzung in Compiler-Konfiguration Release
- Dokumentation der Bibliothek:
https://www4.cs.fau.de/Lehre/SS18/V_SPIC/Uebung/doc
- Abzugebende Datei: `blink.c`

Hands-on: Licht

Demo

Hands-on: Licht

- Projekt aufgabe0 erstellen
 - Ordner erstellen
 - Projektvorlage kopieren und Dateinamen anpassen
 - Projekt in Atmel Studio öffnen
- Minimalprogramm test.c erstellen
 - schaltet LED GREEN0 ein
 - wartet dann endlos
- Programm übersetzen und im Simulator testen
- Lösung abgeben
 - mit Putty zu Linux-Rechner verbinden
 - Abgabeprogramm ausführen