

Übungen zu Systemnahe Programmierung in C (SPiC)

Sebastian Maier
(Lehrstuhl Informatik 4)

Übung 1



Wintersemester 2016/2017



Inhalt

Organisatorisches

Entwicklungsumgebung

Anleitung

Aufgabe 1 und Hands-on



Organisatorisches
Tafelübungen
Aufgaben
Rechnerübungen
Bei Problemen

Entwicklungsumgebung

Anleitung

Aufgabe 1 und Hands-on



Tafelübungen

- Tafelübung Do 10:15 – 11:45 (im Raum 01.153-113)
- 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
 5. Hands-on: gemeinsames Programmieren
- Folien nicht unbedingt zum Selbststudium geeignet
→ Anwesenheit, Mitschrift
- Übersicht aller SPiC-Termine:
https://www4.cs.fau.de/Lehre/WS16/V_SPIC/#woch
- Semesterplan:
https://www4.cs.fau.de/Lehre/WS16/V_SPIC/#sem



■ 8 Aufgaben:

- 4 x Mikrocontroller (SPiCboard)
- 4 x Linux

■ Lösungen:

- Abgabe unter Linux
- Lösung wird automatisch auf Ähnlichkeit mit allen anderen, auch älteren Lösungen verglichen
- “abgeschriebene” Lösungen bekommen 0 Punkte
 - ⇒ Im Zweifelsfall bei einem Übungsleiter melden
- Programm nicht übersetzbar: -50% der möglichen Punkte
- Bei Warnungen des Compilers: Je Warnung -2 Punkte
- Kommentare im Code helfen euch und dem Korrektor
- Nur die Aufgabenstellung lösen → Code auskommentieren
- Lieber Teilaufgaben richtig, als alles, aber falsch lösen



Bonuspunkte

- Abgegebene Aufgaben werden mit Übungspunkten bewertet
- Umrechnung in Bonuspunkte für die Klausur (bis zu 10% der Punkte)
- Bestehen der Klausur durch Bonuspunkte *nicht möglich*
- Bonuspunkte für die Klausur ab 50% der erreichbaren Übungspunkte
- Bonuspunkte können nicht in nächste Semester übernommen werden



- Rechnerübung Fr 14:15 – 15:45 (im Raum 01.153-113)
- Unterstützung durch Übungsleiter bei der Aufgabenbearbeitung
- Falls 30 Minuten nach Beginn der Rechnerübung (also um 14:45) niemand anwesend ist, kann der Übungsleiter gehen



Bei Problemen

- Diese Folien konsultieren
- Häufig gestellte Fragen (FAQ) und Antworten:
https://www4.cs.fau.de/Lehre/WS16/V_SPiC/Uebung/faq/
- 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



Organisatorisches

Entwicklungsumgebung

Hardware

Funktionsbibliothek

Wichtige Verzeichnisse

Atmel Studio

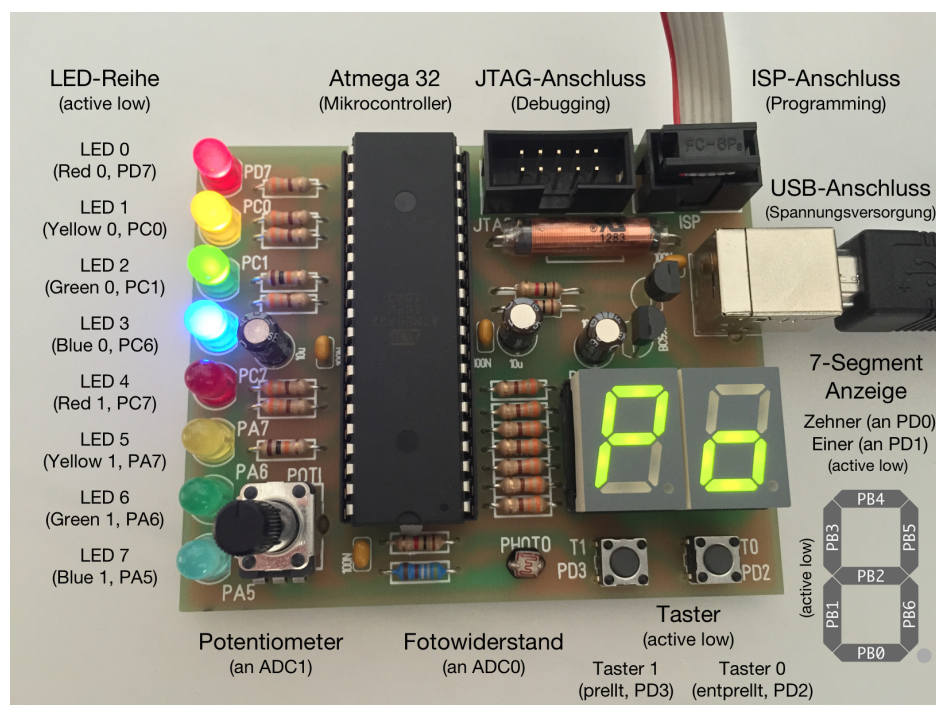
Anleitung

Aufgabe 1 und Hands-on

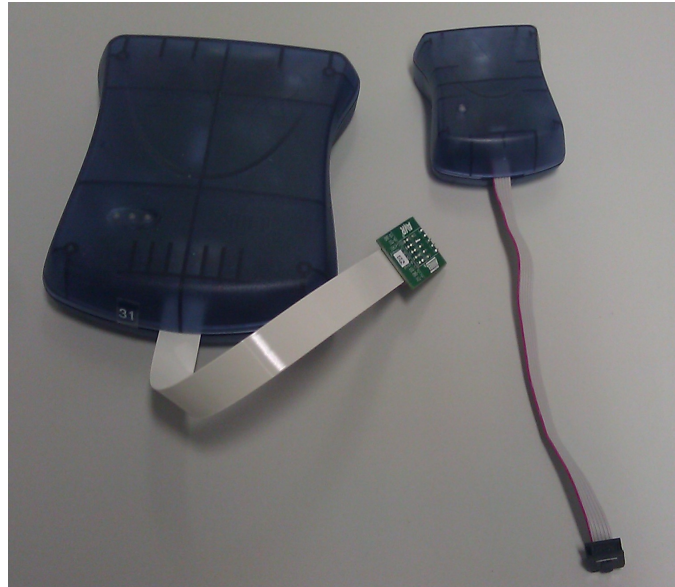


Hardware: SPiCboard

- Speziell für (G)SPiC angefertigte SPiCboards mit AVR-ATmega32-Mikrocontroller



- JTAG-Debugger (links) zur Überwachung der Programmausführung direkt auf dem Board (z. B. Schritt-für-Schritt-Ausführung, Untersuchung von Variablenwerten, etc.)
- ISP-Programmierer (rechts) zur Übertragung des eigenen Programms auf den Mikrocontroller



Aufgabenbearbeitung

- Betreute Bearbeitung der Aufgaben während der Rechnerübungen
⇒ Hardware wird zur Verfügung gestellt
- Selbständige Bearbeitung teilweise nötig
- Ausleihe von SPiCboard, Kabeln und Programmierer/Debugger tagsüber möglich:
 - Bei Harald Junggunst, Büro 0.046 (Erdgeschoss RRZE-Gebäude)
 - Übliche Bürozeiten: von 8:00 bis 15:00
 - <https://www4.cs.fau.de/~jungguns/>



- 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/WS16/V_SPIC/Uebung/doc



Wichtige Verzeichnisse (1)

- Projektverzeichnis:
 - Windows: `P:\`
 - Linux: `/proj/i4spic/LOGINNAME/`
 - Lösungen müssen hier in Unterordnern `aufgabeX` gespeichert werden
⇒ Das Abgabeprogramm sucht dort
 - Ist durch andere nicht lesbar
 - Wird automatisch erstellt
- Heimverzeichnis:
 - Windows: `Z:\`
 - Linux: `~`



- Vorgabeverzeichnis:
 - Windows: Q:\
 - Linux: /proj/i4spic/pub/
 - Projektvorlage vorlage.cproj für Atmel Studio
 - Aufgabenstellungen unter aufgaben/
 - Hilfsmaterial und Binärmusterlösungen zu einzelnen Übungsaufgaben unter aufgabeX/
 - Programm zum Testen der Einheiten auf den Boards unter boardtest/
 - libspicboard-Bibliothek und -Dokumentation unter libspicboard/
 - Kleine Hilfsprogramme unter tools/
- Falls eines der Verzeichnisse Z:\, P:\, Q:\ nicht angezeigt wird:
 - Windows Explorer – Computer – Map network drive
 - Z:\ unter \\fau03\LOGINNAME
 - P:\ unter \\fau03\i4spichome
 - Q:\ unter \\fau03\i4spicpub



Entwicklungsumgebung: Atmel Studio

- Programmentwicklung 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



Organisatorisches

Entwicklungsumgebung

Anleitung

CIP Login

Projekt und Dateien anlegen

Programmieren und Übersetzen

Flashen

Debuggen

Abgabe

Aufgabe 1 und Hands-on



Windows-Login

- 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`



- Lehrstuhl Informatik 4 Übungen zu SPiC (WS 2016/17)

16-33

Lehrstuhl Informatik 4 Übungen zu SPiC (WS 2016/17)

17-33

Programmieren und Übersetzen (1)

- Beispielprogramm, um erste grüne LED einzuschalten:

```
1 #include <led.h>
2
3 void main(void) {
4     sb_led_on(GREEN0);
5     while(1) {
6         /* Endlosschleife:
7            Mikrocontrollerprogramm darf nie terminieren */
8     }
9 }
```

- Programm kompilieren mit Build → Build Solution

- ⇒ Programm wurde nur erfolgreich übersetzt, wenn unten steht:
Build succeeded.
- ⇒ Fehlermeldungen erscheinen ggf. unten



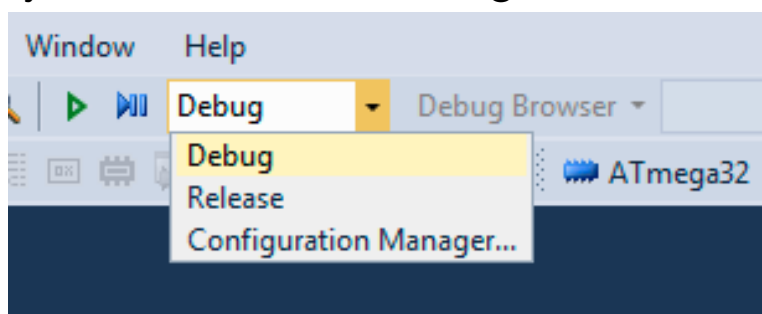
Programmieren und Übersetzen (2)

- **Achtung:** Zwei verschiedene Compiler-Profile:

- Debug: Ohne Optimierung
- Release: Mit Optimierung

- ⇒ Optimierung macht den Code *sehr* viel schneller, kann aber den Debugger “verwirren”

- Umstellung des Profils in Drop-Down-Box rechts neben dem Play-Button in der Werkzeugleiste

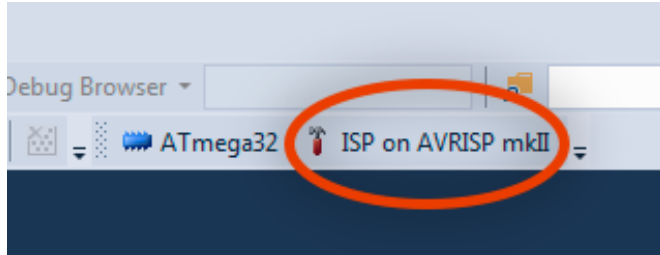


- Letztendlich soll jede Aufgabe mit Release kompiliert und getestet werden

- ⇒ Die Release-Konfiguration wird von uns bewertet!



- Flashen: Kompiliertes Programm in den Speicher des Mikrocontrollers kopieren
- Programmierer auswählen:
 - Project → aufgabeX Properties
 - Tool → Selected Debugger → AVRISP mkII
 - ISP Clock: 150,00 kHz
 - File → Save Selected Items (CTRL-S)
- Schnellauswahl des Werkzeugs:



- Übersetzen, in den Speicher kopieren und laufen lassen:
Debug → Continue (F5)
Beim ersten Mal ggf. Firmware-Upgrade durchführen



Debuggen (1)

- JTAG-Debugger zum Untersuchen des Programmablaufs “live” auf dem Board
- Debugger auswählen:
 - Project → aufgabeX Properties
 - Tool → Selected Debugger → JTAGICE mkII
 - JTAG Clock: 200,00 kHz
 - File → Save Selected Items
- Direkt in den Speicher kopieren und laufen lassen:
Debug → Continue (F5)
- Beim ersten Mal ggf. Firmware-Upgrade durchführen
- Sollte sich der Debugger eigenartig verhalten ist wahrscheinlich die Clock verstellt



- Programm laden und beim Betreten von `main()` anhalten:
Debug → Start Debugging and Break
- Schrittweise abarbeiten mit
 - F10 (Step Over): Funktionsaufrufe werden in einem Schritt bearbeitet
 - F11 (Step Into): Bei Funktionsaufrufen wird die Funktion betreten
- Debug → Windows → I/O View: I/O-Ansicht gibt Einblick in die Zustände der I/O-Register; die Werte können dort auch direkt geändert werden
- Breakpoints unterbrechen das Programm an einer bestimmten Stelle
 - Setzen durch Codezeile anklicken, dann F9 oder Debug → Toggle Breakpoint
 - Programm laufen lassen (F5 oder Debug → Continue): stoppt, wenn ein Breakpoint erreicht wird



Binärabbild flashen

- Nötig, um vorgefertigte Binärabbilder (.hex-Images) zu testen, z. B. Binärmusterlösungen unter `Q:\aufgabeX`
- Möglich mit Debugger (ICE) oder Programmierer (ISP)
 - Tools → Device Programming
 - Tool: JTAGICE mkII bzw. AVRISP mkII
 - Device: ATmega32
 - Interface: JTAG bzw. ISP
 - Apply
 - Verbindung überprüfen mit Device Signature – Read
 - Ergebnis: 0x1E9502
 - ⇒ Eignet sich gut um die Verbindung zwischen PC und µC zu testen
 - Memories → Flash: .hex-Datei auswählen
 - Program
- Nach erfolgreichem Flashen führt das Board das Programm direkt aus
- Ein Neustart des Programms ist durch Trennung und Wiederherstellung der USB-Spannungsversorgung möglich



Abgabe (1)

- Nach erfolgreichem Testen des Programms müssen Übungslösungen zur Bewertung abgegeben werden
- Wichtig: Bei Zweiergruppen darf nur ein Partner abgeben!
- Die Abgabe erfolgt 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 Benutzernamen und **Linux**-Passwort
- Im erscheinenden Terminal-Fenster folgendes Kommando ausführen, dabei 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
- Weitere nützliche Tools:
 - Quelltext der abgegebenen Aufgabe anzeigen:
`/proj/i4spic/bin/show-submission aufgabeX`
 - Unterschiede zwischen abgegebener Version und Version im Projektverzeichnis P:\aufgabeX anzeigen:
`/proj/i4spic/bin/show-submission aufgabeX -d`
 - Eigenen Abgabetermin anzeigen:
`/proj/i4spic/bin/get-deadline aufgabeX`



Organisatorisches

Entwicklungsumgebung

Anleitung

Aufgabe 1 und Hands-on

Aufgabenbeschreibung: Zähler

Flankendetektion ohne Interrupts

Verwendung von int

Sichtbarkeit & Lebensdauer

Typdefs & Enums

Verwendung von Modulo

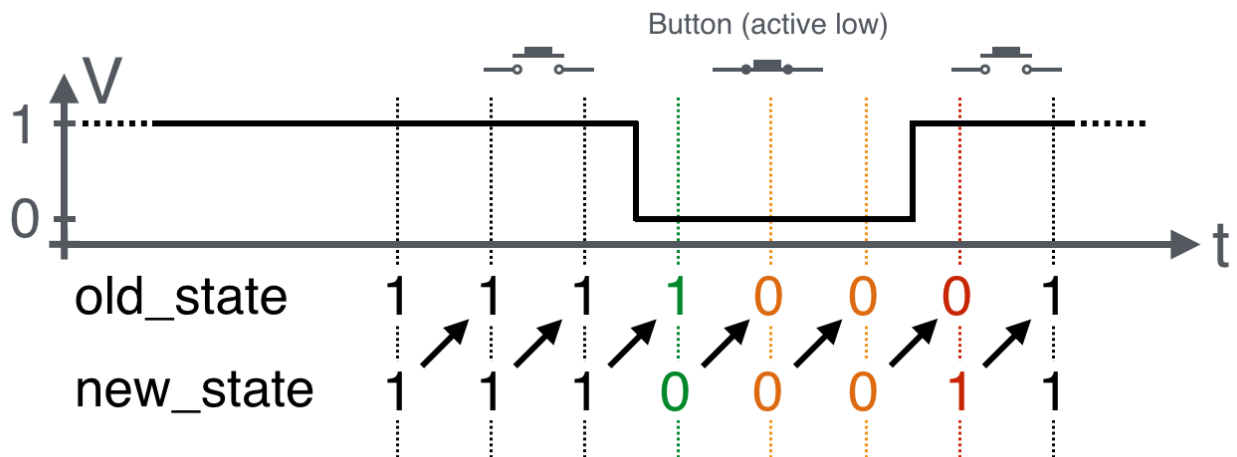
Hands-on: Signallampe



Aufgabenbeschreibung: Zähler

- Zählen der Tastendrücke an Taster 1 (fallende Flanke)
- Aktualisieren der Anzeige durch Taster 0 möglich (fallende Flanke)
- Anzeige erfolgt über 7-Segmentanzeige und LEDs (zu Beginn aus)
- Hunderterstelle durch LED visualisieren:
LED0 = 100, LED1 = 200, etc.
- Bei Verlassen des anzeigbaren Wertebereichs Zähler zurücksetzen
- Nutzung der Bibliothefunktionen für Button, Display und LED
- Dokumentation der Bibliothek:
https://www4.cs.fau.de/Lehre/WS16/V_SPIC/Uebung/doc
- Implementierung durch Polling → keine Interrupts erforderlich





- Detektion der Flanke durch aktives, **zyklisches Abfragen** (engl. Polling) eines Pegels
- Unterscheidung zwischen **active-high** & **active-low** notwendig
- Später: Realisierung durch Interrupts



Verwendung von `int`

- Die Größe von `int` ist nicht genau definiert (ATmega32: 16 bit)
⇒ Gerade auf μC führt dies zu Fehlern und/oder langsameren Code
- Für die Übung:
 - 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
⇒ Nur so viel Speicher verwenden, wie tatsächlich benötigt wird!



SB Sichtbarkeit LD Lebensdauer		nicht static	static
Variablen	lokal	Block SB <i>auto Variable</i> LD Block	Block SB LD Programm
	global	Programm SB LD Programm	Modul SB LD Programm
Funktionen		SB Programm	SB Modul

- Lokale Variable, nicht static = auto Variable
 - automatisch allokiert & freigegeben
- Funktionen als static, wenn kein Export notwendig



Globale Variablen

```
1 static uint8_t state; // global static
2 uint8_t event_counter; // global
3
4 void main(void) {
5     ...
6 }
7
8 static void f(uint8_t a) {
9     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
 - static bei Funktionen und globalen Variablen verwenden, wo möglich



```
1 #define PB3 3
2 typedef enum { BUTTON0 = 4, BUTTON1 = 8
3 } BUTTON;
4 #define MAX_COUNTER 900
5 ...
6 void main(void) {
7     ...
8     PORTB |= (1 << PB3); // nicht (1 << 3)
9     ...
10    BUTTONEVENT old, new; // nicht uint8_t old, new;
11    ...
12    // Deklaration: BUTTONEVENT sb_button_getState(BUTTON btn);
13    old = sb_button_getState(BUTTON0); // nicht sb_button_getState(4)
14    ...
15 }
```

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



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



- Morsesignale über LED 0 ausgeben
- Steuerung über Taster 1
- Nutzung der Bibliotheksfunktionen für Button und LED
- Dokumentation der Bibliothek:
https://www4.cs.fau.de/Lehre/WS16/V_SPiC/Uebung/doc

