Übungen zu Systemnahe Programmierung in C (SPiC)

Sebastian Maier, Heiko Janker
(Lehrstuhl Informatik 4)

Übung 1

Wintersemester 2015/2016

Inhalt

Organisatorisches

Entwicklungsumgebung

Anleitung

Aufgabe 1 und Hands-on

Inhalt

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/WS15/V_SPIC/#woch`
- Semesterplan:
  `https://www4.cs.fau.de/Lehre/WS15/V_SPIC/#sem`

Aufgaben

- 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

    $\Rightarrow$ 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 $\rightsquigarrow$ 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

Rechnerübungen

- 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/WS15/V_SPIC/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:

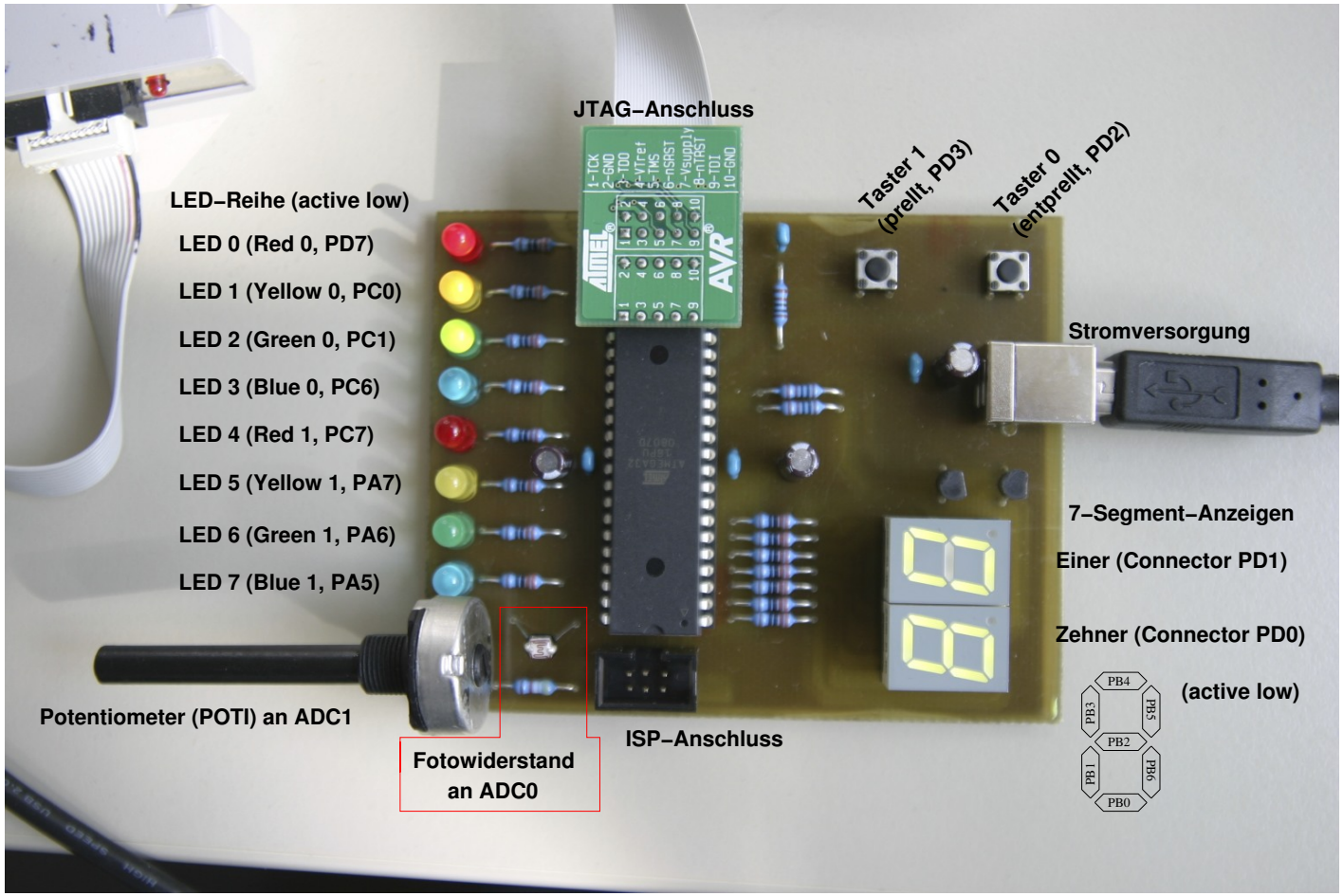
  i4spic@cs.fau.de

  ⇒ Zum Beispiel auch, wenn kein Übungsleiter auftauchen sollte

Inhalt

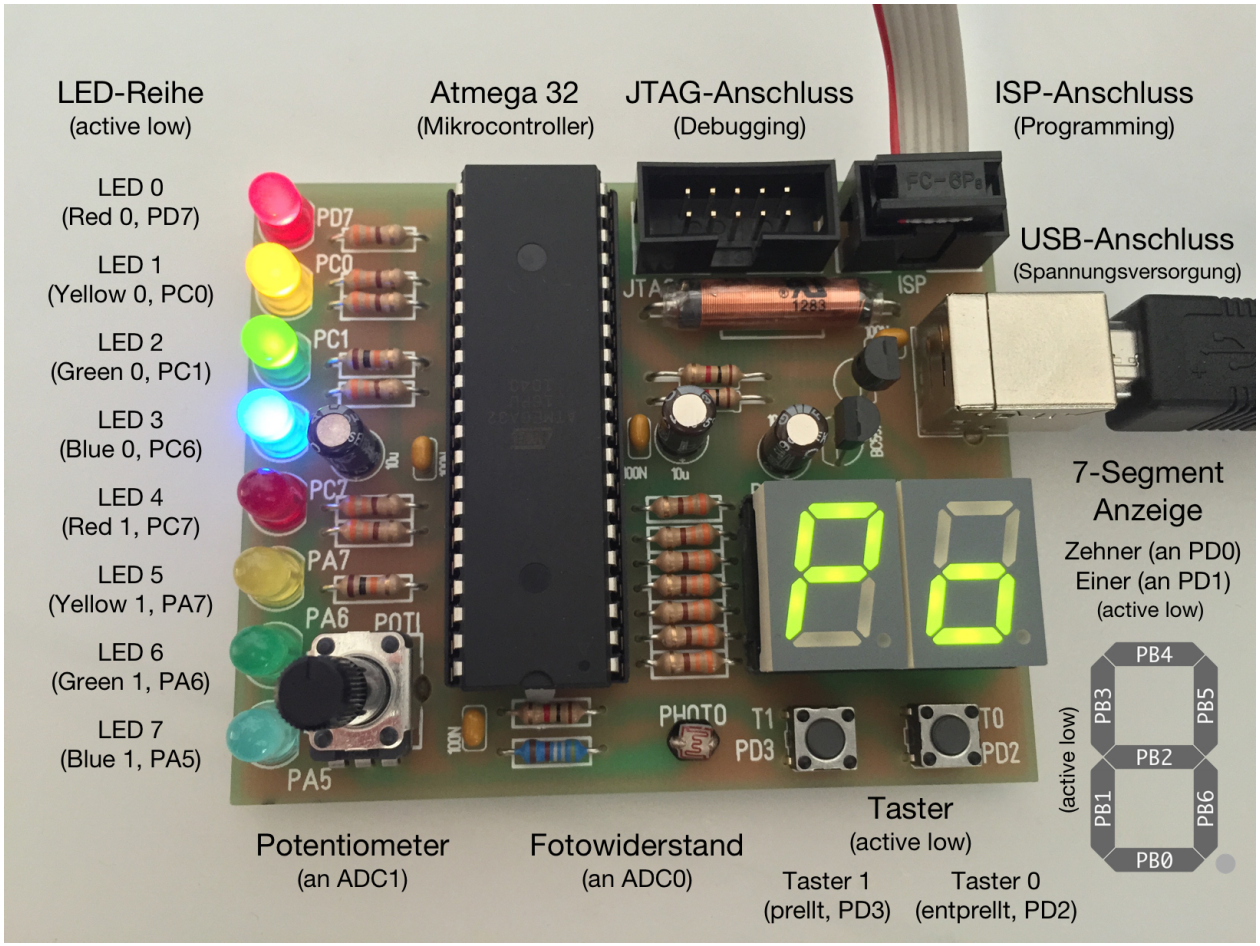
Hardware: SPiCboard – v1

- Speziell für (G)SPiC angefertigte SPiCboards mit AVR-ATmega32-Mikrocontroller
- SPiCboard Version 1:



Hardware: SPiCboard – v2

- SPiCboard Version 2:



Hardware: Werkzeuge

- 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/`
- In 01.155N befinden sich weitere Windows-Rechner

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/WS15/V_SPIC/Uebung/doc`

Wichtige Verzeichnisse (1)

- Heimverzeichnis:
  - Linux: `~`
  - Windows: `Z:\`
- Projektverzeichnis:
  - Linux: `/proj/i4spic/LOGINNAME/`
  - Windows: `P:\`
  - Die Lösungen müssen im Unterordner `aufgabeX` gespeichert werden

    $\Rightarrow$ Das Abgabeprogramm sucht dort
  - Ist durch andere nicht lesbar
  - Wird automatisch erstellt

Wichtige Verzeichnisse (2)

- Vorgabeverzeichnis:
  - Linux: `/proj/i4spic/pub/`
  - Windows: `Q:\`
  - 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 `i4/`
  - Kleine Hilfsprogramme unter `tools/`
- Falls eines der Verzeichnisse `Z:\`, `P:\`, `Q:\` nicht angezeigt wird:
  - Windows Explorer – Computer – Netzlaufwerk verbinden
  - `Z:\` unter `\\faui03\LOGINNAME`
  - `P:\` unter `\\faui03\i4spichome`
  - `Q:\` unter `\\faui03\i4spicpub`

Entwicklungsumgebung: Atmel Studio

- Programmentwicklung mit Atmel Studio 6 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

Inhalt

Windows-Login

- Zur Bearbeitung der Übungen ist ein Windows-Login nötig
  - Auf einem CIP-Rechner mit Linux-Passwort einloggen
  - Ein Terminalprogramm öffnen und dort folgendes Kommando ausführen:
    `/local/ciptools/bin/setsambapw`
    (hängt auch auf einem Zettel an der Wand zum Raum 01.155-N)

- 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
```

Atmel Studio Einrichtung

- Achtung: Die Anleitung muss genau beachtet werden
- Start von Atmel Studio über:
  Start $\rightsquigarrow$ Alle Programme $\rightsquigarrow$ Atmel $\rightsquigarrow$ Atmel Studio 6.2
- Falls Windows-Firewall einige Funktionen blockiert, auf "Abbrechen" klicken
- Importieren der Projektvorlage (einmalig):

1. File $\rightsquigarrow$ Import $\rightsquigarrow$ Project Template...
2. `Q:\tools\SPiC_Template6.zip`
3. Add to folder: `<Root>`
4. OK

$\Rightarrow$ Meldung: „Project Template has been successfully imported."

Atmel-Studio-Projekt

- Pro Übungsaufgabe ein neues Projekt anlegen:
  1. File ⇝ New ⇝ Project...
  2. Projekttyp: `(G)SPiC-Projekt`
  3. Name: `aufgabeX`, zum Beispiel `aufgabe0` (Achtung: Kleinschreibung!)
  4. Location: `P:\`
  5. Wichtig: Kein Häkchen bei "Create directory for solution"
  6. OK

- Initiale C-Datei zu Projekt hinzufügen:
  1. Rechts Solution Explorer auswählen
  2. Orangefarbenes Projekt auswählen
  3. Project ⇝ Add New Item...
  4. Dateityp: C File
  5. Name: siehe Aufgabenstellung, jetzt `test.c` (Achtung: Kleinschreibung!)
  6. Add

Programmieren (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 $\rightsquigarrow$ Build Solution

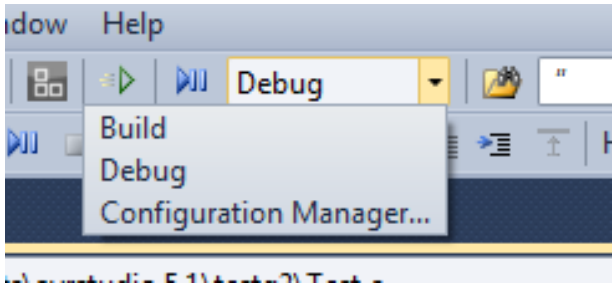
$\Rightarrow$ Programm wurde nur erfolgreich übersetzt, wenn unten steht:
`Build succeeded.`

$\Rightarrow$ Fehlermeldungen erscheinen ggf. unten

Programmieren (2)

- *Achtung:* Zwei verschiedene Compiler-Profile:
  - `Debug`: Ohne Optimierung
  - `Build`: 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

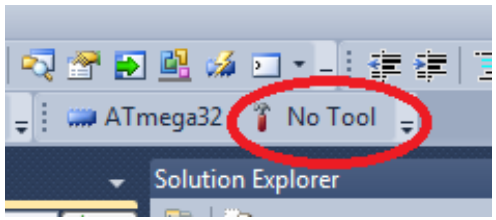


- Letztendlich soll jede Aufgabe mit `Build` kompiliert und getestet werden

  ⇒ *Die `Build`-Konfiguration wird von uns bewertet!*

Flashen mit Programmierer

- 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 das 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

Debuggen (2)

- 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 bestimmen 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 Benutzername und Linux-Passwort
- Im erscheinenden Terminal-Fenster folgendes Kommando ausführen, dabei `aufgabe0` entsprechend ersetzen:

  ```
  /proj/i4spic/bin/submit aufgabe0
  ```

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

Abgabe (2)

- Fehlerursachen
  - `aufgabeX` muss klein geschrieben sein
  - Häkchen bei “Create directory for solution” nicht entfernt:

    ⇒ Dateien sind im Ordner `aufgabeX/aufgabeX`
  - `.c`-Datei falsch benannt

- Anzeigen der abgegebenen Aufgabe

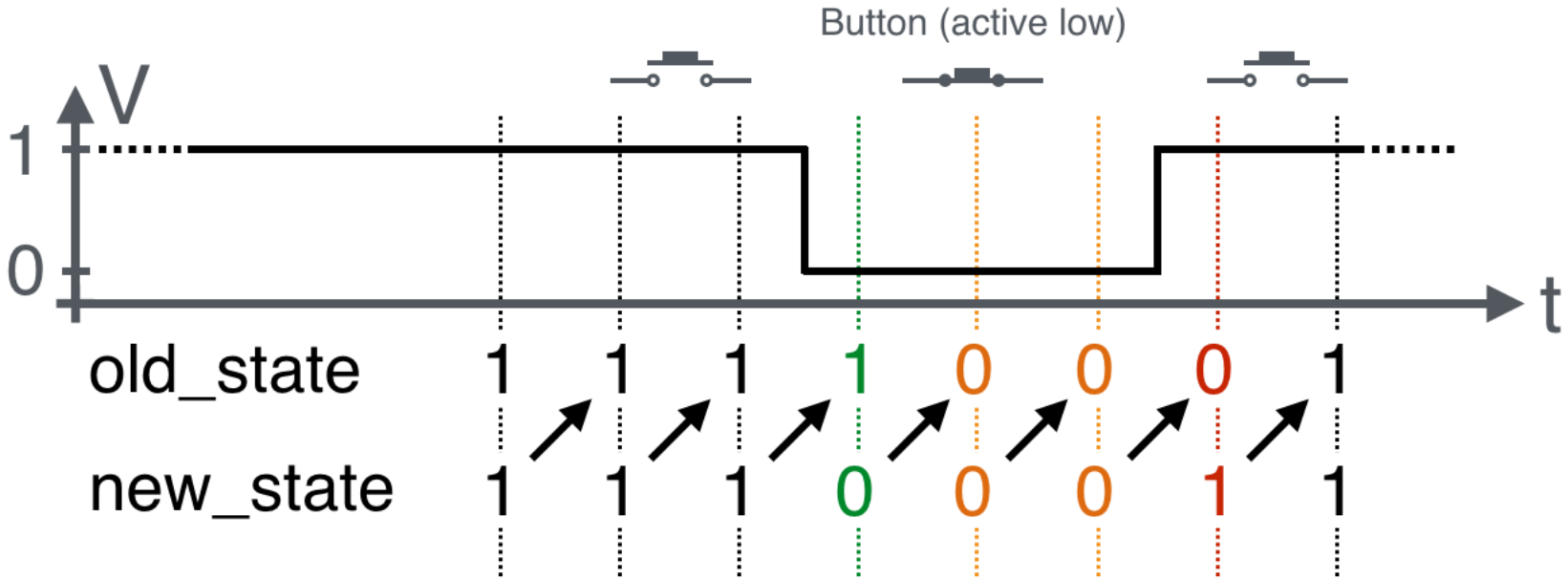
  `/proj/i4spic/bin/show-submission aufgabe0`
  - Zeigt abgegebene Version an
  - Zeigt ggf. Unterschied zwischen abgegebener Version und Version im Projektverzeichnis `P:\aufgabeX` an

Inhalt

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 Bibliothesfunktionen für Button, Display und LED
- Dokumentation der Bibliothek:
  `https://www4.cs.fau.de/Lehre/WS15/V_SPIC/Uebung/doc`
- Implementierung durch Polling $\rightsquigarrow$ keine Interrupts erforderlich

Flankendetektion ohne Interrupts



- 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!

Übersicht: Sichtbarkeit & Lebensdauer

SB Sichtbarkeit LD Lebensdauer		nicht `static`	`static`
Variablen	lokal	Block SB — LD: Block (`auto` Variable)	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 ≠ lokale Variable in `f()`
- Globale `static` Variablen: Sichtbarkeit auf Modul beschränken

⇝ `static` bei Funktionen und globalen Variablen verwenden, wo möglich

Typdefs & Enums

```
1  #define PD3 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

Hands-on: Signallampe

- Morsesignale über LED 0 ausgeben
- Steuerung über Taster 1
- Nutzung der Bibliothesfunktionen für Button und LED
- Dokumentation der Bibliothek:
  `https://www4.cs.fau.de/Lehre/WS15/V_SPIC/Uebung/doc`