Übungen zu Systemnahe Programmierung in C (SPiC) – Wintersemester 2022

Übung 3

Phillip Raffeck Maximilian Ott

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



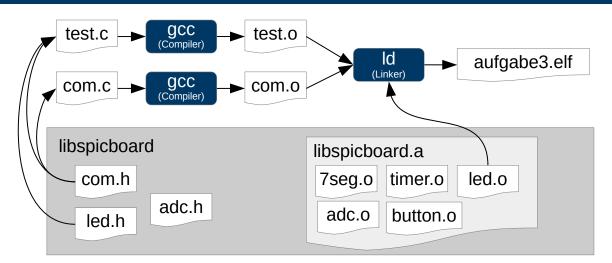


Vorstellung Aufgabe 1

Module

Ablauf vom Quellcode zum laufenden Programm





- 1. Präprozessor
- 2. Compiler
- 3. Linker
- 4. Programmer/Flasher



- Header Dateien enthalten die Schnittstelle eines Moduls
 - Funktionsdeklarationen
 - Präprozessormakros
 - Typdefinitionen
- Header Dateien können mehrmals eingebunden werden
 - led.h bindet avr/io.h ein
 - button.h bindet avr/io.h ein
 - → Funktionen aus avr/io.h mehrmals deklariert
- Mehrfachinkludierung/Zyklen vermeiden → Include-Guards
 - Definition und Abfrage eines Präprozessormakros
 - Konvention: Makro hat den Namen der .h-Datei, "ersetzt durch '_'
 - z.B. für button.h → BUTTON_H
 - Inhalt nur einbinden, wenn das Makro noch nicht definiert ist
- Vorsicht: Flacher Namensraum → möglichst eindeutige Namen

Schnittstellenbeschreibung (2)



■ Erstellen einer .h-Datei (Konvention: gleicher Name wie .c-Datei)

```
#ifndef LED_H
02 #define LED H
   /* Fixed-width Datentypen einbinden (im Header verwendet) */
   #include <stdint.h>
04
05
   /* Datentypen */
06
07
   typedef enum {
80
       RED0
                = 0,
       YELLOW0 = 1,
09
       [\ldots]
10
       GREEN1 = 6,
11
       BLUE1
                = 7
12
   } LED;
13
14
15 /* Funktionen */
16 void sb_led_on(LED led);
17 [...]
18 #endif //LED_H
```



- Interne Variablen und Hilfsfunktionen nicht Teil der Schnittstelle
- C besitzt einen flachen Namensraum
- Unvorhergesehen Zugriffe können Fehlverhalten auslösen
- ⇒ Kapselung: Sichtbarkeit & Lebensdauer einschränken

Implementierung: Sichtbarkeit & Lebensdauer (1)



Sichtbarkeit und Lebensdauer	nicht static	static
lokale Variable	Sichtbarkeit Block	Sichtbarkeit Block
	Lebensdauer Block	Lebensdauer Programm
globale Variable	Sichtbarkeit Programm	Sichtbarkeit Modul
	Lebensdauer Programm	Lebensdauer Programm
Funktion	Sichtbarkeit Programm	Sichtbarkeit Modul

- Lokale Variablen, die **nicht** static deklariert werden:
- → auto Variable (automatisch allokiert & freigegeben)
- Globale Variablen und Funktionen als static, wenn kein Export notwendig



```
static uint8_t state; // global static
02 uint8_t event_counter; // global
03
04 static void f(uint8_t a) {
     static uint8_t call_counter = 0; // local static
05
     uint8_t num_leds; // local (auto)
     /* ... */
07
80
09
  void main(void) {
10
     /* ... */
11
12
```

- Sichtbarkeit & Lebensdauer möglichst weit einschränken
- → Wo möglich: static für globale Variablen und Funktionen

Implementierung: Initialisierung eines Moduls (1)



- Module müssen Initialisierung durchführen
 - Zum Beispiel Portkonfiguration
 - Java: Mit Klassenkonstruktoren möglich
 - C: Kennt kein solches Konzept
- Workaround: Modul muss bei erstem Aufruf einer seiner Funktionen ggf. die Initialisierung durchführen
 - Muss sich merken, ob die Initialisierung schon erfolgt ist
 - Mehrfachinitialisierung vermeiden
- Anlegen einer Init-Variable
 - Aufruf der Init-Funktion bei jedem Funktionsaufruf
 - Init-Variable anfangs o
 - Nach der Initialisierung auf 1 setzen

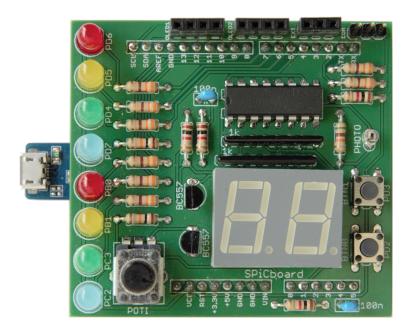


- initDone ist initial 0
- Wird nach der Initialisierung auf 1 gesetzt
- → Initialisierung wird nur einmal durchgeführt

```
static void init(void) {
01
        static uint8_t initDone = 0;
02
        if (initDone == 0) {
03
          initDone = 1;
04
05
        }
06
07
08
   void mod_func(void) {
09
        init();
10
11
12
```

Aufgabe: BUTTON-Modul





■ Die libspicboard besteht aus verschiedenen Modulen

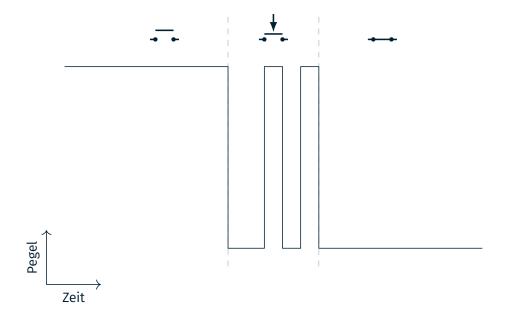
9

BUTTON-Modul – Aufgabe



- BUTTON-Modul der libspicboard selbst implementieren
 - Zustandsabfrage: Gleiches Verhalten wie das Original
 - Callbacks: Platzhalter
 - Beschreibung: https://sys.cs.fau.de/lehre/WS22/spic/uebung/spicboard/ libapi/extern/group__Button.html
- Testen des Moduls
 - Eigenes Modul gegen ein Testprogramm (test-button.c) linken
 - Andere Teile der Bibliothek können für den Test benutzt werden
- Taster des SPiCboards
 - Beide Taster sind active-low beschalten
 - Anschluss: BUTTON0 an PD2, BUTTON1 an PD3
 - Nomenklatur: PD2 = Port D, Pin 2

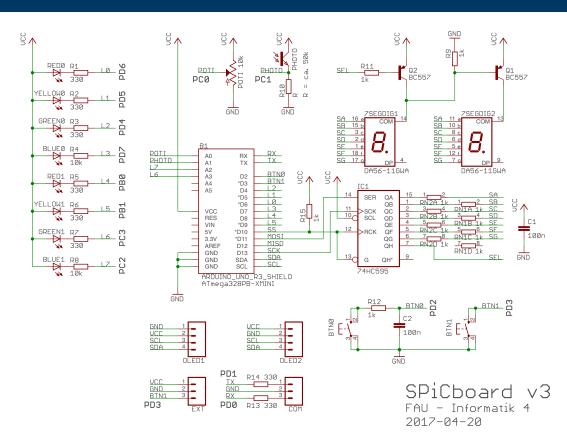




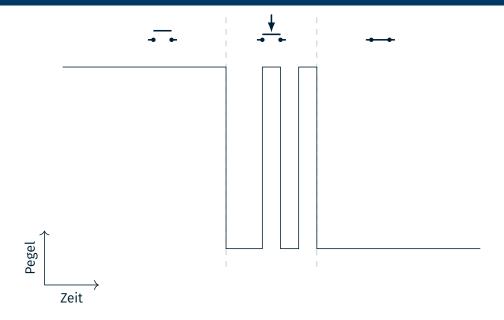
- Mechanischer Effekt beim Umschalten
- Erzeugt überzählige Flanken
- Entprellen in Hardware/Software möglich

SPiCboard Schaltplan





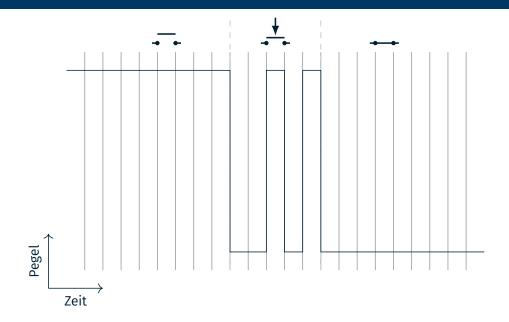




- Summe des Signalwertes über Zeitfenster
- Entscheidung für häufiger auftretenden Pegel
 - ⇒ Glättet die Kontaktprellen
- Aktives Warten mittels _delay_us() aus <util/delay.h>

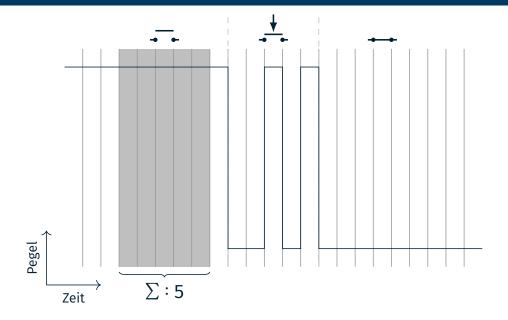
Mittelwertfilter





- Summe des Signalwertes über Zeitfenster
- Entscheidung für häufiger auftretenden Pegel
 - ⇒ Glättet die Kontaktprellen
- Aktives Warten mittels _delay_us() aus <util/delay.h>

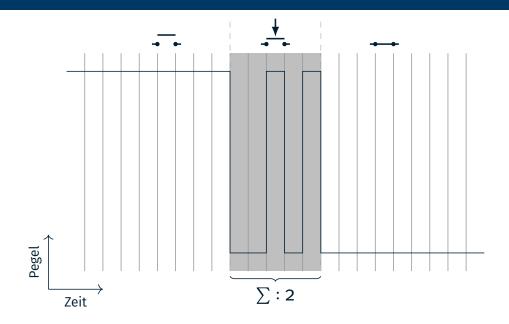




- Summe des Signalwertes über Zeitfenster
- Entscheidung für häufiger auftretenden Pegel
 - ⇒ Glättet die Kontaktprellen
- Aktives Warten mittels _delay_us() aus <util/delay.h>

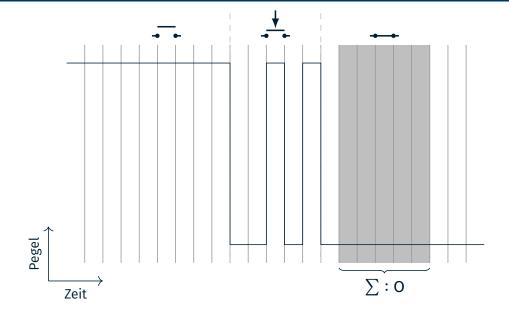
Mittelwertfilter





- Summe des Signalwertes über Zeitfenster
- Entscheidung für häufiger auftretenden Pegel
 - ⇒ Glättet die Kontaktprellen
- Aktives Warten mittels _delay_us() aus <util/delay.h>





- Summe des Signalwertes über Zeitfenster
- Entscheidung für häufiger auftretenden Pegel
 - ⇒ Glättet die Kontaktprellen
- Aktives Warten mittels _delay_us() aus <util/delay.h>

Übersetzereinstellungen



- Projekt wie gehabt anlegen
 - Initiale Quelldatei: test-button.c
 - Dann weitere Quelldatei button.c hinzufügen
- Wenn nun übersetzt wird, werden die Funktionen aus dem eigenen BUTTON-Modul verwendet
- Andere Teile der Bibliothek werden nach Bedarf hinzugebunden
- Temporäres Deaktivieren zum Test der Originalfunktionen:

```
01 #if 0
02 ....
03 #endif
```

- ⇒ Sieht der Compiler diese "Kommentare"?
- ⇒ Wie kann der Code wieder einkommentiert werden?



```
void main(void){
01
02
     // 1.) Callback Funktionen
03
     int8_t result = sb_button_registerCallback(BUTTONO, ONPRESS,
04
        → NULL);
     if(result != -1) { // Nicht implementiert -> -1
05
       // Test fehlgeschlagen
06
       // Ausgabe z.B. auf 7-Segment-Anzeige
07
08
09
     // 2.) Testen bei ungueltiger Button ID
10
11
12
```

- Schnittstellenbeschreibung genau beachten
- Testen aller möglichen Rückgabewerte
- Fehler wenn Rückgabewert nicht der Spezifikation entspricht
- Für sb_button_getState(): Endlosschleife mit LEDs

Hands-on: Statistikmodul

Screencast: https://www.video.uni-erlangen.de/clip/id/16328



- Statistikmodul und Testprogramm
- Funktionalität des Moduls (Schnittstelle):

```
// Schnittstelle
uint8_t avgArray(uint16_t *a, size_t s, uint16_t *avg);
uint8_t minArray(uint16_t *a, size_t s, uint16_t *min);
uint8_t maxArray(uint16_t *a, size_t s, uint16_t *max);

// interne Hilfsfunktionen
uint16_t getMin(uint16_t a, uint16_t b);
uint16_t getMax(uint16_t a, uint16_t b);
```

- Rückgabewert: 0: OK; 1: Fehler
 - 0: OK
 - 1: Fehler
- Vorgehen:
 - Header-Datei mit Modulschnittstelle (und Include-Guards)
 - Implementierung des Moduls (Sichtbarkeit beachten)
 - Testen des Moduls im Hauptprogramm (inkl. Fehlerfälle)