

Übungen zu Systemnahe Programmierung in C

Abschnitt 5.1: Module

25.05.2020

Tim Rheinfels
Benedict Herzog
Bernhard Heinloth

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

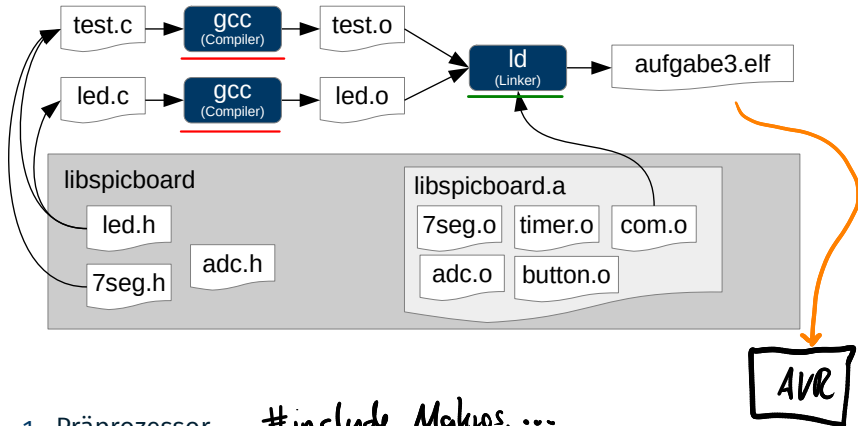


Lehrstuhl für Verteilte Systeme
und Betriebssysteme



FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT



1. Präprozessor `#include, Makros, ...`
2. Compiler
3. Linker
4. Programmer/Flasher

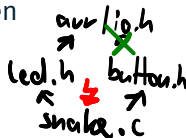
Modul: $\left. \begin{array}{l} \text{-Daten} \\ \text{-funktion} \end{array} \right\} .c$
-Schnittstelle: `.h`
~VL12

- 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
- Inhalt nur einbinden, wenn das Makro noch nicht definiert ist

- **Vorsicht:** Flacher Namensraum → möglichst eindeutige Namen

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

```
01 #ifndef COM_H
02 #define COM_H
03 /* Fixed-width Datentypen einbinden (im Header verwendet) */
04 #include <stdint.h>
05
06 /* Datentypen */
07 typedef enum {
08     ERROR_NO_STOP_BIT, ERROR_PARITY,
09     ERROR_BUFFER_FULL, ERROR_INVALID_POINTER
10 } COM_ERROR_STATUS;
11
12 /* Funktionen */
13 void sb_com_sendByte(uint8_t data);
14 [...]
15 #endif //COM_H
```

Include Guard

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

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

Hilfsvariablen & -funktionen



```
01 static uint8_t state; // global static
02 uint8_t event_counter; // global
03
04 static void f(uint8_t a) {
05     static uint8_t call_counter = 0; // local static
06     uint8_t num_leds; // local (auto)
07     /* ... */
08 }
09
10 void main(void) {
11     /* ... */
12 }
```

- Sichtbarkeit & Lebensdauer möglichst weit **einschränken**

~> Wo möglich: static für globale Variablen und Funktionen



- 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 0
 - Nach der Initialisierung auf 1 setzen



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

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