Informatikwerkstatt, Foliensatz 3 Modularisierung

G. Kemnitz

30. Oktober 2023

Inhalt:

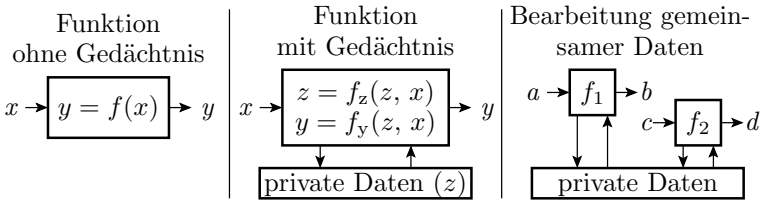
Contents

Interaktive Übungen:

- Simulation eines Modultests (mtest_quad)
- Kapselung von Funktionen und Objekten (bit_io3_mod)

1 Modularisierung

Modularisierung



Zur Beherrschung von Entwurf und Test werden größere Programme aufgeteilt in Module (Programmbausteine):

- Funktionen ohne Gedächtnis,
- Funktionen mit Gedächtnis,
- Funktionen zur Bearbeitung gemeinsamer privater Daten.

und mehrere Datei:

- c-Dateien: separat zu übersetzende Programmdatei.
- h-Dateien (Header): Schnittstellenbeschreibungen.

Funktionen ohne Gedächtnis

Eine c-Funktion berechnet aus Eingabewerten ein Ergebnis:

```
uint8_t Summe(uint8_t a, uint8_t b){
 return a+b;}       //Ergebnisrückgabe
```

und berechnet bei Aufruf aus den Operanden ein Ergebnis:

```
uint8_t x, y, z;
z = Summe(x, y);
```

Für veränderliche Übergabe- oder mehrere Ausgabeparameter werden Adressen (Zeiger) übergeben. Im Bsp. wird der Wert der Variablen »x« auf der übergebenen Adresse »&x« hochgezählt:

```
void inc(uint8_t *a){(*a)++;} // void -- kein
...                           // Rückgabewert
uint8_t x=5;   // Variable aufrufendes Programm
inc(&x);       // Aufruf mit der Adresse von x
```

Feldübergabe: Anfangszeiger und Länge

```
uint8_t SumFeld(const uint8_t *F, uint8_t len){
 uint8_t sum=0, idx;
 for (idx=0;idx<len;idx++){
   sum += F[idx];
 }
 return sum; //Ergebnisrückgabe
} ...

// Summe aller Elemente von x
uint8_t s, x[]={2,4,8,3};   // Vereinbarung
s = SumFeld(x, sizeof(x));  // Funktionsaufruf
```

- Adresse/Wert der Übergabeparameter »F« und »len« sind innerhalb des Unterprogramms beschreibbare Variablen.
- »const« verbietet Schreibzugriffe auf den Inhalt »*F«.
- sizeof(x) – Byteanzahl von x

Funktionen mit Gedächtnis

Zustandsvariablen zur Werteaufbewahrung nach Beendigung einer Funktion sind global / statisch (mit fester Adresse) zu vereinbaren:

```
uint16_t err_ct;   //globaler Fehlerzähler
...
void test_lim(int16_t val, int16_t max,
                           int16_t min){
 if ((val>max) || (val<min))
    err_ct++;       //Fehlerzähler erhöhen
}
```

Vereinbarung einer statischen Variablen innerhalb einer Funktion[1]:

```
void test_lim( ... ){
 static uint16_t err_ct = 0; //globaler Fehlerz.
 if ((val>max) || (val<min)) err_ct++;
}
```

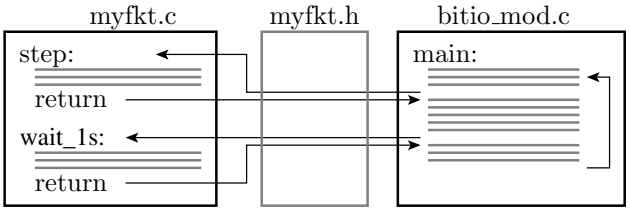
Aufteilung in Programm- und Header-Dateien

In Programmdateien (.c) gehören[2]:

- Funktionsbeschreibungen zur Übersetzung in Anweisungsfolgen.
- globale Variablen, für die Speicherplatz reserviert wird, ...

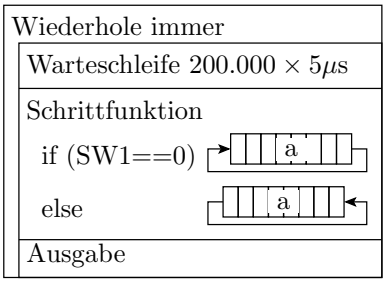
In Header- (Schnittstellen-) Dateien gehören:

- übereinstimmende Vereinbarungen mehrerer Programmdateien: Konstaten, Typen, Funktionsschnittstellen, ...



Lauflichtprogramm zur Modularisierung

```
#include <avr/io.h>      // Beispielprogramm
uint32_t Ct;             // bit_io3.c
uint8_t a=1;             // ================
int main(void){          //
 DDRA = 0;               //
 DDRJ = 0xFF;            //
 while(1){for (Ct=0;     //
    Ct<200000; Ct++);    //
  if (PINA & 0b1)        //
   a = (a<<1) | (a>>7); //
  else                   //
   a = (a>>1) | (a<<7); //
  PORTJ = a;             //
 }
}
```



Auslagerung der Schrittfunktion und der Warteschleife in je eine Funktion in eine neue c-Datei »myfkt.c«.

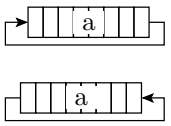
[1] Der Name einer Variablen ist nur in dem Block bekannt, in dem sie vereinbart ist. Eine mit »static« vereinbarte Variable behält ihren Wert nach Beendigung einer Funktion bis zum Neuaufruf, ist aber in anderen Funktionen nicht sichtbar.

[2] Der Compiler übersetzt Programmdateien in Speichernutzungstabellen, Code-Bausteine, ... und der Linker verbindet diese über Namenstabellen.

Schritt- und Wartefunktion in »myfkt.c«

```
#include "myfkt.h"        //Header
uint8_t a=1;     //Zustand (global)
uint8_t step(uint8_t x){ //Funktionsvereinbarung
  if (x & 0b1)    //Rotation links
    a = (a<<1) | (a>>7);
  else            //Rotation rechts
    a = (a>>1) | (a<<7);
  return a;

void wait_1s(){      //Wartefunktion
 uint32_t Ct;        //Zähler (lokal)
 for (Ct=0; Ct<200000; Ct++);
}                    // 200.000 ×5 µs
```



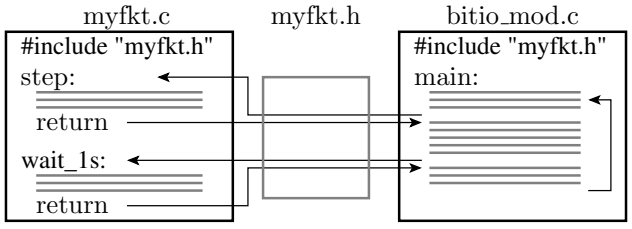
- Automatenzustand: globale (statische) Variable.
- Nur die Funktionsschnittstellen nach außen sichtbar.

Hauptprogramm (bit_io3_mod.c)

```
#include <avr/io.h>
#include "myfkt.h" //Header eigene Funktionen
int main(void){
  DDRA = 0;         //Port A Schaltereingänge
  DDRJ = 0xFF;      //Port J LED-Ausgänge
  while(1){
    PORTJ = step(PINA);
    wait_1s();
 }
}
```

- Ersatz der Warteschleife und der Anweisungen für die Warteschleife und die Übergangs- (Schritt-) Funktion durch Funktionsaufrufe.
- Interaktive Übung 3.1, Folie 8.

Header-Datei




```
#ifndef MYFKT_H_
#define MYFKT_H_
#include <avr/io.h>         // Definition:
  uint8_t step(uint8_t x); // * Schrittfunktion
  void wait_1s();          // * Wartefunktion
#endif /* MYFKT_H_ */
```

In mehrfach eingefügte Header gehört nichts, was in Code übersetzt wird oder Speicher reserviert, sonst Duplikate.

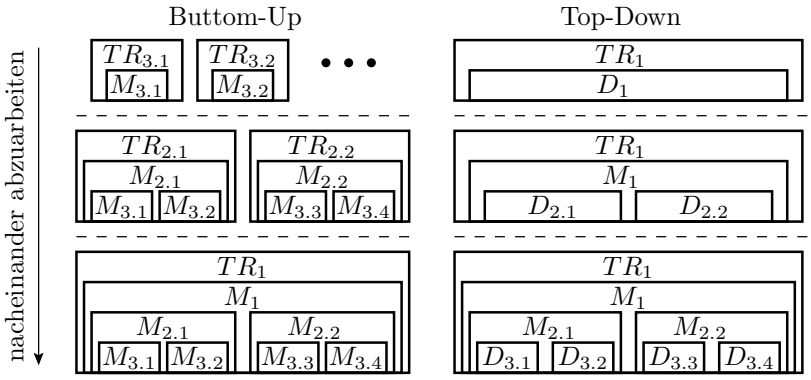
```
#ifndef MYFKT_H_
#define MYFKT_H_
#include <avr/io.h>
#include <avr/io.h>          // Definition:
  uint8_t step(uint8_t x); // * Schrittfunktion
  void wait_1s();          // * Wartefunktion
#endif /* MYFKT_H_ */
```

- #ifndef ... #define ... #endif: Präprozessoranweisungen, verhindern Mehrfacheinfügen desselben Headers in dieselbe C-Datei, wenn Header selbst Header enthalten.
- Regel für einen guten Programmierstil.
- Selbst ausprobieren, ob der Compiler mehrfach eingefügte gleiche Header verträgt.

2 Modultest

Testrahmen und Dummies

- Entwurf und Test komplexer Systeme erfolgt modulweise.
- Ein Modultest verlangt einen Testrahmen, der Eingaben bereitstellt, die Abarbeitung startet (steuert) und die Ergebnisse kontrolliert/ausgibt.
- Nicht vorhandene Teilmodule werden durch Dummies ersetzt[3].



Ein Testobjekt und sein Testrahmen

Testobjekt sei das folgende Unterprogramm zur Quadrierung[4]:

```
uint32_t quad(int16_t a){ // Quadratberechnung
  return (uint32_t)a * a; // Warum mit Typcast?
};
```

Testbeispiele sind Tupel aus Eingaben und Sollausgaben. Man kann dafür einen neuen Datentyp definieren:

[3] Testrahmen und Dummies oft umfangreicher als das Programm

[4] Im nachfolgenden Experiment sollen Sie selbst herausfinden, welche der Testbeispiele ohne den Typecast falsch ausgeführt werden.

```
typedef struct{
  int16_t x;              //Eingabe
  uint32_t y;             //Sollausgabe
} test_t;
```

Ein »struct« ist eine Zusammenfassung aus bereits definierten Datentypen[5].

Testsatz

Eine Testsatz als Menge von Tests ist im einfachsten Fall ein initalisiertes Feld von Testbeispielen

```
test_t testsatz[] ={{<Tupel1>},{...},...};
```

Testbeispiele für die Quadratberechnung:

```
test_t testsatz[] ={      //Eingabe  Sollwert
  {0, 0},                 // 0x0000  0x00000000
  {1, 1},                 // 0x0001  0x00000001
  {9, 81},                // 0x0009  0x00000051
  {-5, 25},               // 0xFFFB  0x00000019
  {463, 214369},          // 0x01CF  0x00034561
  {0x7FFF, 1073676289}    // 0x7FFF  0x3FFF0001
};
```

Welche Testbeispiele führt das Programm falsch aus? Ausprobieren!

Testrahmen

Programm, das in einer Schleife alle Testbeispiele abarbeitet und die Ergebnisse kontrolliert oder zur Kontrolle ausgibt:

```
int main(){
 uint8_t idx, err_ct=0;
 uint32_t erg;
 for (idx=0; idx<6;idx++){
 erg = quad(testsatz[idx].x);//Istwert
 if (erg != testsatz[idx].y) //Soll/Ist-Vergl.
  err_ct++;                  //Fehlfkt. zählen
 }
}
```

Testdurchführung mit dem im Simulator im Debug-Modus:

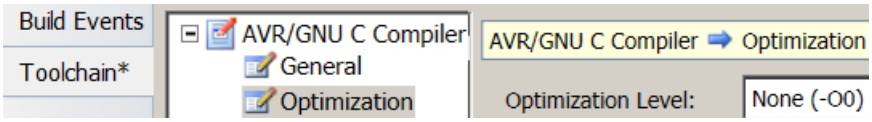
- Unterbrechungspunkt vor dem Fehlerzähler.
- Bei jeder Fehlfunktion Halt am Unterbrechungspunkt.

[5] Grundregel der Programmierung: Alles was man benutzt (jeder Bezeichner, jede Variable, jeder Typ, ...), muss vorher im Programmtext definiert sein.

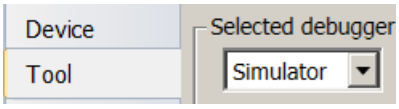
Projekt mtest_quad« ausprobieren

- Projekt »mtest_quad« öffnen.
- Compiler-Optimierung ausschalten (in -O0 ändern):

```
Project > mtest_quad Properties... (Alt+F7)
```



- Auswahl des Simulators als »Debugger«:



- Übersetzen.
- Debugger starten:

Test und Fehlerlokalisierung im Schrittbetrieb:

```
struct{
  int16_t x;
  uint32_t y;
} testsatz[]={
  {0, 0}, {1, 1}, {9, 81},
  {-5, 25}, {463, 214369},
  {0x7FFF, 1073676289}
};

int main(){
  uint8_t idx, err_ct=0;
  uint32_t erg;
  for (idx=0; idx<6;idx++){
    erg = quad(testsatz[idx].y);
    if (erg != testsatz[idx].y)
      err_ct++;
  }
}
```

Watch 1

Name	Value
testsatz	{struct [6]{data}@0x0200}
[0]	{struct {data}@0x0200}
[1]	{struct {data}@0x0206}
[2]	{struct {data}@0x020c}
x	0x0009
y	0x00000051
[3]	{struct {data}@0x0212}
[4]	{struct {data}@0x0218}
[5]	{struct {data}@0x021e}
idx	0x02
erg	0x000019a1

Fehler- und Fehlerbehebung

- Test {9, 81} versagt, weil im Testrahmen statt »if(y_soll != f(x))« »if(y_soll!=f(y_soll)«, steht:

$$\text{Vergleich } 81 \text{ mit } 81^2 \text{ statt mit } 9^2$$

- Nach Fehlerbehebung versagt der Test {-5, 25}, weil die erste »-5« vor der Quadrierung auf eine positive Zahl $\neq 5$ gecastet wird:

$$\text{Berechnung von } \left(2^{16}-5\right)\cdot(-5) \text{ statt } (-5)\cdot(-5)$$

- Bei Cast erst nach Produktbildung

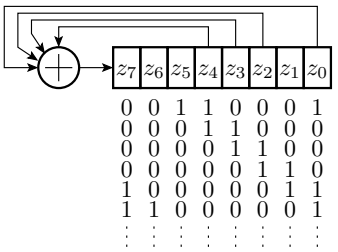
```
return (uint32_t)(a * a);
```

  versagt der Test {0x01CF, 0x00034561}, weil »a*a« als »uint16_t« ohne die führende 3 berechnet und erst dann mit führenden Nullen auf 32 Bit gecastet wird:

$$\text{Berechnung von } (\text{0x1CF})^2 \text{ \& 0xFFFF statt 0x1CF}^2$$

Viele Programmierfehler fallen erst bei Beispielrechnungen auf.

Hausaufgabe



Die nachfolgende Funktion »rand8()«zur Erzeugung von 8-Bit Pseudo-Zufallszahlen verwendet eine globale uint8_t-Zustandsvariable »rand_state«, bildet die Folgezustands- und Ergebisbits 0 bis 6 durch Rechtsverschiebung und Bit 7 als EXOR-Summe der Bits 0, 2, 3, und 4 des alten Wertes der Zustandsvariablen.

```
uint8_t lfsr(){
              // Vereinb. Zustand, Anfangswert 0x31
              // Berechnung Folgezustand (eine Anweisung)
              // Rückgabe Folgezustand (eine Anweisung)
}
```

1. Vervollständigen Sie das Programm (3 Zeilen).
2. Berechnung der nächsten beide Folgezustände nach 0xC1.

Abgabe bis kommenden Mo. an »ha-iw@in.tu-clausthal.de«.[6]

Aufgabe 3.1: Modularisierung

- Stecken Sie ein Schaltermodul PmodSWT an Stecker JA oben.
- Kopieren Sie aus dem Projektverzeichnis »bit_io3« die Dateien »bit_io3.c« und »bit_io3.cproj« in ein neu anzulegendes Projektverzeichnis »bit_io3_mod«.
- Öffnen des neuen Projekts.
- Erzeugung eine neue c-Datei »myfkt.c« und eine neue Header-Datei »myfkt.h«[7].
- Kopieren Sie die Programmzeilen für die Warteschleife und die Schrittfunktion aus »bit_io3.c« in die neue c-Datei und passen Sie beide c-Dateien und die Header-Datei entsprechend der beiden nachfolgenden Folien an.

[6] Wenn Test- und HA-Ergebnisse ok, weitere Wiederholung nicht erforderlich.
[7] Solution Explorer > Rechtsklick auf bit_io3 > Add > New Item > ...

c-Datei »myfkt.c« (zu erstellen)

```
#include <avr/io.h>
uint8_t a=1; // Zustand (global, warum?)

uint8_t step(uint8_t x){
  if (x & 0b1)        // Rotation links
    a = (a<<1) | (a>>7);
  else                // Rotation rechts
    a = (a>>1) | (a<<7);
  return a;
}

void wait_1s(){
  for (uint32_t Ct=0; Ct<200000; Ct++);
}
```

Header und Hauptprogramm

Zu erstellender Header-Datei »myfkt.h«:

```
#ifndef MYFKT_H_
#define MYFKT_H_
#include <avr/io.h>
  uint8_t step(uint8_t x);
  void wait_1s();
#endif /* MYFKT_H_ */
```

Angepasstes Hauptprogramm »bit_io3_mod.c«:

```
#include "myfkt.h" //Header eigene Funktionen
int main(void){
  DDRA = 0;         //Port A Schaltereingänge
  DDRJ = 0xFF;      //Port J LED-Ausgänge
  while(1){
    PORTJ = step(PINA);
    wait_1s();
  }
}
```

Durchzuführende Tests

- Übersetzen und im Debugger starten.
- Test freilaufend. Sollfunktion: LED-Lauflicht mit Schalter zur Richtungsumschaltung.
- Test auch im Schrittbetrieb und mit Unterbrechungspunkten.

Ein Vorteil gegenüber dem Test des bisherigen Programms ist, dass sich die Wartefunktion im Schrittbetrieb auch mit "Step-Over" und nicht nur mit »Abarbeitung bis zum nächsten Unterbrechungspunkt« überspringen lässt.

Aufgabe 3.2: Modultest Quadrierung

- Legen Sie ein neues Projekt mtest_quad mit einer Datei mtest_quad.c an und kopieren Sie den Dateiinhalt von der Web-Seite »http://techwww.in.tu-clausthal.de/site/Lehre/IW_P3/« aus »der dort angezeigten gleichnamigen Datei.
- Durchführung der Tests ab Seite 17.

Aufgabe 3.3: Modultest Wurzelberechnung

Die nachfolgende Funktion berechnet die ganzzahlige Quadratwurzel einer 16-Bit-Zahl:

```
uint8_t wurzel(uint16_t x){
  uint8_t w=0; uint16_t sum=0;
  while (sum<x){
    sum += (w<<1)+1; w++;
  }
  return w;
}
```

- Ist das Ergebnis des folgenden Testbeispiels richtig?

```
int main(){
  uint16_t x = 37481;
  uint8_t y = wurzel(x);
}
```

Watch 1

Name	Value	Type
x	37481	uint16_t{data}@0x21f8
y	194	uint8_t{data}@0x21fa

Test mit dem Debugger

- Projekt anlegen. Mit dem Simulator ausprobieren.
- Führen Sie die Tests auch mit folgenden Testbeispielen durch:

x		Sollwert von y		Istwert von y
0	0x0000	0	0x00	
1				
257				
8.351				
65025				
65026				
	0xFFFF	256	0x100	

- Sollwert-Bestimmung mit Taschenrechner und Soll-Ist-Vergleich.
- Für welche der Testeingaben ist das vom Programm berechnete Ergebnis falsch?
- Für welche Testeingaben berechnet das Programm kein Ergebnis, weil »sum« immer kleiner »x« bleibt?

Testautomatisierung

- Vereinbaren Sie einen »struct«-Typ aus Eingabe und Sollausgabe und ein Feld mit allen Testbeispielen.
- Erweitern Sie das vorgegebene Testbeispiel zu einem im Simulator abarbeitbaren Modultest für alle Testbeispiele mit Fehlerzähler.
- Führen Sie den Test wie in der Vorlesung gezeigt im Simulator mit Unterbrechungspunkt so durch, dass der Test nach jedem Testbeispiel, bzw. nach jedem fehlerhaften Testbeispiel anhält.
- Beseitigen Sie alle vom Test nachweisbaren Fehler.

Aufgabe 3.4: Multiplikationsfehler

Das Ergebnis der nachfolgenden Multiplikation ist falsch.

```
#include <avr/io.h>
int main(void){
   uint16_t a = 0x1FA;
   uint16_t b = 0x100;
   uint32_t p = a*b;
}
```

Locals

Name	Value	Type
a	0x01fa	uint16_t{data}@0x21f3
b	0x0100	uint16_t{data}@0x21f5
p	0x0000fa00	uint32_t{data}@0x21f7

- Überprüfen Sie das im Simulator!
- Wie lautet das richtige Ergebnis von 0x1FA*0x0100?
- Ändern Sie die Multiplikation bzw. die Datentypen so, dass das Ergebnis richtig berechnet wird.

- Projekt anlegen, Programm vervollständigen.
- Abarbeitung im Schrittbetrieb mit dem Simulator.

Aufgabe 3.5: Festkommamultiplikation

Festkommazahlen haben eine gedachte Kommaposition. Im nachfolgenden Programm haben alle Variablen 8 gedachte Nachkommastellen:

```
uint16_t a, b, p;
uint32_t tmp;
tmp = (a * b) >> 8;
if (tmp > 0xFFFF) p = 0xFFFF;
else              p = tmp;
```

Das Produkt von zwei ganzen 16-Bit-Zahlen ist 32 Bit groß. Bei gedachten 8 Nachkommastellen sind von den 32 Bit die führenden 16 Bit Vorkomma- und die niederwertigen 16 Bit Nachkommastellen. Vor der Produktzuweisung an das 16-Bit Ergebnis werden die niederwertigsten 8 Nachkommastellen abgeschnitten und der Wert auf 0xFF,FF begrenzt.

Aufgaben:

- Betten Sie den vorgegebenen Code-Schnipsel in einen Programmrahmen für den Test mit dem Simulator mit 20 zufällig ausgewählten Testbeispielen[8] ein.
- Ändern Sie solange die Testeingaben des ersten Testbeispiels, bis es den fehlenden Typecast nachweist.
- Beheben Sie den Fehler in den zu testenden Programmzeilen und kontrollieren sie, dass danach alle Testbeispiele korrekt abgearbeitet werden.

[8] Auswahl zufälliger Werte für a und b und manuelle Berechnung des Sollwerts für p.

Aufgabe 3.6: Modultest Sättigungsbegrenzer

Die Funktion »limit()« soll einen 32-Bit-Festkommawert mit 8 Nachkommastellen betragsmäßig auf den ganzzahligen Wert »maxabs« begrenzen. Das Testbeispiel liefert ein falsches Ergebnis[9].

```
int32_t limit(int32_t val, uint16_t maxabs){
  if ((val>>8)> maxabs) return  maxabs<<8;
  if ((val>>8)<-maxabs) return -(maxabs<<8);
  return val;
}

int main(){
  int32_t v = limit(0, 0xFFFF);
}
```

Locals

Name	Value	Type
v	0x00000100	int32_t{data}@0x21f7 ([R28]+1)

- Ausprobieren im Simulator und Fehlerbeseitigung.
- Erweiterung um einen Testrahmen mit mehreren Testbeispielen.

[9] *Vermutlich ein Compiler-Fehler im gcc.*