

**Aufgabe 1: (12 Punkte)**

Bei den Einfachauswahlfragen in dieser Aufgabe ist jeweils nur **eine** richtige Antwort eindeutig anzukreuzen. Auf die richtige Antwort gibt es die angegebene Punktzahl.

Wollen Sie eine Antwort korrigieren, streichen Sie bitte die falsche Antwort mit drei waagrechten Strichen durch (~~☒~~) und kreuzen die richtige an.

Lesen Sie die Frage genau, bevor Sie antworten.

a) Gegeben ist folgendes Makro:

```
#define SUM(a,b) a+b
```

2 Punkte

Wie ist das Ergebnis des folgenden Ausdrucks?

```
SUM(2, 3) * SUM(2 - 1, 4)
```

- 9  
 11  
 15  
 25

b) Welche Aussagen zum Schlüsselwort **volatile** sind richtig?

2 Punkte

- Das Schlüsselwort **volatile** erlaubt dem Compiler bessere Optimierungen durchzuführen.  
 Das Schlüsselwort **volatile** unterbindet alle Nebenläufigkeitsprobleme in der entsprechenden Datei.  
 Das Schlüsselwort **volatile** sorgt dafür, dass die damit definierte Variable nur so kurz wie möglich in einem Register gehalten wird.  
 Das Schlüsselwort **volatile** beschleunigt den Zugriff auf die damit definierte Variable.

c) Welche der folgenden Aussagen bzgl. der Interruptsteuerung ist richtig?

2 Punkte

- Während der Bearbeitung eines Interrupts nimmt der Prozessor keine weiteren Interrupts an.  
 Wurde gerade ein Pegel-gesteuerter Interrupt ausgelöst, so muss erst ein Pegelwechsel der Interruptleitung stattfinden, bevor erneut ein Interrupt ausgelöst wird.  
 Flankengesteuerten Interrupts können nicht blockiert werden, da sie völlig unvorhersehbar auftreten.  
 Pegel-gesteuerte Interrupts werden beim Wechsel des Pegels ausgelöst.

d) Gegeben ist folgender Ausdruck:

```
if ( ( a = 5 ) || ( b != 3 ) ) { c = 24; } else { c = 42; }
```

2 Punkte

Welche Aussage ist richtig?

- Falls a den Wert 7 und b den Wert 5 enthält, wird c auf 42 gesetzt.  
 Der Compiler meldet einen Fehler, weil dieser Ausdruck nicht zulässig ist.  
 Falls a den Wert 5 und b den Wert 7 enthält, wird c auf 42 gesetzt.  
 Der Wert von a hat keinen Einfluss auf das Ergebnis in c.

e) Was versteht man unter Polling?

2 Punkte

- Wenn ein Gerät so lange Interrupts auslöst, bis die Daten durch den Mikrocontroller abgeholt wurden.  
 Wenn ein Programm regelmäßig eine Peripherie-Schnittstelle abfragt, ob Daten oder Zustandsänderungen vorliegen.  
 Wenn ein Gerät durch Auslösen eines Interrupts Daten von einem Mikrocontroller anfordert.  
 Wenn ein Programm zum Zugriff auf kritische Daten Interrupts sperrt.

f) Welche Aussage zur Speicherallokation ist richtig?

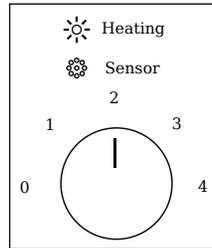
2 Punkte

- Die dynamische Allokation von Speicher ist auf einem Mikrocontroller zu bevorzugen, da erst zur Laufzeit geprüft wird, ob der Speicher wirklich zur Verfügung steht.  
 Die Speicheradresse von statisch allokierten Variablen kann sich zur Laufzeit ändern.  
 **automatic**-Variablen werden im Heap allokiert.  
 Die Verwendung von statisch allokierten Variablen erlaubt den Speicherbedarf bereits nach dem Binden abzuschätzen.

**Aufgabe 2: Heizung (30 Punkte)**

Sie dürfen diese Seite zur besseren Übersicht bei der Programmierung heraustrennen!

Implementieren Sie die Steuerung einer klimafreundlichen Heizungsanlage. Zur Schonung des eigenen Geldbeutels und der Umwelt wird hierzu die Raumtemperatur mittels eines Sensors kontinuierlich gemessen, um so sicherzustellen, dass eine zuvor definierte Maximaltemperatur nie dauerhaft überschritten werden kann. Sofern die gewünschte Maximaltemperatur nicht erreicht wurde, kann die Heizleistung mit Hilfe eines Drehreglers feingesteuert werden. Eine leuchtende Kontrollleuchte soll dabei eine laufende Heizphase signalisieren. Falls die gemessene Maximaltemperatur überschritten wird oder der Regler den linken Anschlag erreicht, soll die Heizleistung auf null reduziert und die Kontrollleuchte ausgeschaltet werden.



Im Detail soll Ihr Programm wie folgt funktionieren:

- Initialisieren Sie die Hardware in der Funktion `void init(void)`. Treffen Sie hierbei keine Annahmen über den initialen Zustand der Hardware-Register.
- Der Eingang PD2 (Interrupt 0) ist mit dem Raumtemperatursensor verbunden. Die externe Beschaltung stellt sicher, dass genau dann eine steigende Flanke auftritt, wenn die Maximaltemperatur erreicht wird und eine fallende Flanke dann, wenn diese den Wohlfühlbereich (Bereich unterhalb der Maximaltemperatur) wieder betritt. Stellen Sie sicher, dass dieses Event unbedingt vor dem Timer-Event abgearbeitet wird.
- Für die Zeittaktung soll ein 8-Bit Timer verwendet werden. Konfigurieren Sie diesen so, dass er alle  $T = 1ms$  einen Interrupt auslöst.
- Der Regler soll alle  $T$  Zeiteinheiten ausgelesen werden. Messen Sie dafür zunächst durch Aufruf von `uint16_t sb_adc_read(ADCDEV)` für POT1 den Wert  $r$  des Drehreglers als vorzeichenlose 10 Bit Ganzzahl. Während des Aufrufs müssen die Interrupts gesperrt sein.
- Sofern die Maximaltemperatur nicht überschritten wurde, soll der gelesene Wert  $r$  mittels der Funktion `uint16_t convert(uint16_t r)` in eine vorzeichenlose 16 Bit Ganzzahl umgewandelt werden. Benutzen Sie für die Umwandlung folgende Gleichung:

$$\text{convert}(r) = \begin{cases} 0 + \text{HEAT\_SCALE} \times (r \bmod 256) & \text{für } r \in \{0, \dots, 255\} \\ 2^{12} - 1 + \text{HEAT\_SCALE} \times (r \bmod 256) & \text{für } r \in \{256, \dots, 511\} \\ 2^{13} - 1 + \text{HEAT\_SCALE} \times (r \bmod 256) & \text{für } r \in \{512, \dots, 767\} \\ 2^{14} - 1 + \text{HEAT\_SCALE} \times (r \bmod 256) & \text{für } r \in \{768, \dots, 1022\} \\ 2^{16} - 1 & \text{für } r \in \{1023\} \end{cases} \quad (1)$$

- Um eine unübersichtliche `if-else`-Kaskade zu vermeiden, sollen die Grundwerte  $(0, 2^{12} - 1, \dots)$  innerhalb des Arrays `uint16_t heating_interval[5]` gespeichert werden. Die Heizstufe  $(0, 1, 2, 3, 4)$ , welche sich aus dem Intervall von  $r$  ergibt, soll als Index dienen.
- Der konvertierte Wert soll schließlich an Funktion `void heat(uint16_t value)` übergeben werden. Diese kapselt die eigentliche Steuerung der Heizleistung. Beachten sie zusätzlich, dass die Kontrollleuchte immer nur genau dann aktiv sein soll, wenn die Maximaltemperatur noch nicht erreicht wurde und der konvertierte Wert größer null ist.
- Stellen Sie sicher, dass sich der Mikrocontroller möglichst oft im Schlafmodus befindet.

**Information über die Hardware**

Sie dürfen diese Seite zur besseren Übersicht bei der Programmierung heraustrennen!

Temperatursensor: Interruptleitung an **PORTD**, Pin 2

- Steigende Flanke: Zulässige Maximaltemperatur wurde überschritten, Heizung wird ausgeschaltet.
- Fallende Flanke: Raumtemperatur hat Wohlfühlbereich erreicht, Feinsteuerung mittels Temperaturregler beginnt.
- Pin als Eingang konfigurieren: Entsprechendes Bit im **DDR**-Register auf 0
- Internen Pull-Up-Widerstand deaktivieren: Entsprechendes Bit im **PORTD**-Register auf 0
- Externe Interruptquelle **INT0**, ISR-Vektor-Makro: **INT0\_vect**
- Aktivieren/Deaktivieren der Interruptquelle erfolgt durch Setzen/Löschen des **INT0**-Bits im Register **EIMSK**

Konfiguration der externen Interruptquelle **INT0** (Bits im Register **EICRA**)

Interrupt 0		Beschreibung
ISC01	ISC00	
0	0	Interrupt bei low Pegel
0	1	Interrupt bei beliebiger Flanke
1	0	Interrupt bei fallender Flanke
1	1	Interrupt bei steigender Flanke

Kontrollleuchte für Aufheizen: Ausgang an **PORTB**, Pin 0

- Gibt an, ob gerade aktiv geheizt wird. Für ausgeschaltene Heizung auf LOW setzen, sonst auf HIGH
- Pin als Ausgang konfigurieren: Entsprechendes Bit im **DDRB**-Register auf 1
- Initialer Zustand zunächst 0, entsprechendes Bit im **PORTB**-Register auf 0

Zeitgeber (8-bit): **TIMER0**

- Es soll die Überlaufunterbrechung verwendet werden (ISR-Vektor-Makro: **TIMER0\_OVF\_vect**)
- Der ressourcenschonendste Vorteiler (*prescaler*) ist 64, wodurch es bei dem 16 MHz CPU-Takt (hinreichend genau) alle  $1ms$  zum Überlauf des 8-bit-Zählers **TCNT0** kommt.
- Aktivieren/Deaktivieren der Interruptquelle erfolgt durch Setzen/Löschen des **TOIE0**-Bits im Register **TIMSK0**

Konfiguration der Frequenz des Zeitgebers **TIMER0** (Bits im Register **TCCR0B**)

CS02	CS01	CS00	Beschreibung
0	0	0	Timer aus
0	0	1	CPU-Takt
0	1	0	CPU-Takt / 8
0	1	1	CPU-Takt / 64
1	0	0	CPU-Takt / 256
1	0	1	CPU-Takt / 1024
1	1	0	Ext. Takt (fallende Flanke)
1	1	1	Ext. Takt (steigende Flanke)

Ergänzen Sie das folgende Codegerüst so, dass ein vollständig übersetzbares Programm entsteht.

```
#include <avr/interrupt.h>
#include <avr/io.h>
#include <avr/sleep.h>
#include <stdint.h>
#include <adc.h>
```

```
extern int16_t sb_adc_read(ADCDEV dev);
extern void heat(uint16_t value);
#define HEAT_SCALE 8
```

```
// Funktionsdeklarationen, globale Variablen, etc.
```

```
// Unterbrechungsbehandlungsfunktionen
```

```
// Ende Unterbrechungsbehandlungsfunktionen
```

```
// Funktion main
```

```
// Initialisierung und lokale Variablen
```

```
// Hauptschleife
```

```
// Ereignisse verarbeiten
```







d) Beschreiben Sie welche LEDs nach Aufruf von `func()` leuchten. Sie müssen keine konkreten LEDs nennen. Beispielantworten: *die obersten vier LEDs, alle LEDs für die eine 1 in `leds` gesetzt ist, die unterste LED.*

2 Punkte

```
static void func(uint8_t leds) {
    sb_led_setMask(leds ^ 0xff);
}
```

e) Beschreiben Sie welche LEDs beim Aufruf `func()` je Iteration leuchten. Sie müssen keine konkreten LEDs nennen. Beispielantworten: *die obersten vier LEDs, alle LEDs für die eine 1 in `leds` gesetzt ist, die unterste LED.*

2 Punkte

```
static void func(void) {
    for(uint8_t i = 0; i < 4; i++) {
        sb_led_setMask(0xfe + i);
    }
}
```

Iteration 1 (i=0):

Iteration 2 (i=1):

Iteration 3 (i=2):

Iteration 4 (i=3):

**Aufgabe 4: Nebenläufigkeit (8 Punkte)**

*Sie dürfen diese Seite zur besseren Übersicht heraustrennen!*

Das nachfolgende Codebeispiel für einen 8-Bit-AVR-Mikrocontroller überprüft, ob die modul-globale Variable `counter` ein anliegendes Event signalisiert. Wenn dies der Fall ist, wird darauf reagiert, ansonsten wird mit der Hauptschleife fortgefahren. Der Wert von `counter` wird asynchron durch die Unterbrechungsbehandlung von `INT0` erhöht.

Die Implementierung dieser Funktionalität beinhaltet ein Nebenläufigkeitsproblem.

```
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <stdint.h>
```

```
static volatile uint8_t counter = 0;
```

```
ISR(INT0_vect) {
    counter++;
}
```

```
void main(void) {
    while(1) {
        if(counter > 0) {
            counter--;
            // Verarbeite Event...
        }
    }
}
```

Die beiden folgenden Assemblerausschnitte zeigen Abschnitte der `main()`-Funktion und der Unterbrechungsbehandlungsfunktion für `INT0`. In den Assemblerausschnitten können Sie den Kommentaren entnehmen, welche C-Anweisung die folgenden Assemblerinstruktionen repräsentieren.

**Hauptprogramm**

```
; counter--;
H1: lds  r24, counter
H2: subi r24, 0x01
H3: sts  counter, r24
```

**Interruptbehandlung INT0**

```
; counter++;
I1: lds  r25, counter
I2: addi r25, 0x01
I3: sts  counter, r25
```

a) Benennen Sie das Nebenläufigkeitsproblem. (1 Punkt)

-----

-----

-----

-----

-----

b) Demonstrieren Sie einen konkreten Programmablauf, bei dem das Nebenläufigkeitsproblem auftritt. (4 Punkte)

Tragen Sie dafür die relevanten Speicher- und Registerinhalte **nach** der Ausführung jeder Assemblerinstruktion in die nachfolgende Tabelle ein. Gehen Sie davon aus, dass die Variable `counter` initial den Wert `0x0f` hat und treffen Sie keine Annahmen über andere Variablen- oder Speicherinhalte.

Kennzeichnen Sie auch in der Tabelle, wann gegebenenfalls ein Interrupt auftritt.

Zeile	counter	r24	r25
–	0x0f	–	–

c) Mit welchem Mechanismus ließe sich das Nebenläufigkeitsproblem lösen? (1 Punkt)

-----

-----

d) Ist die Verwendung des Schlüsselworts `volatile` für die Deklaration der Variablen `value` nötig? Begründen Sie kurz. (2 Punkte)

-----

-----

-----

-----

-----

-----

-----

-----