Aufgabe 1: (14 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) Welche der folgenden Aussagen zur Speicherorganisation auf einem Mikrocontroller ist richtig?

2 Punkte

- ☐ Eine dynamische Allokation ist auf einem Mikrocontroller nicht möglich.
- ☐ Für jede lokale auto-Variable wird bereits beim Übersetzen exklusiv ein Speicherbereich reserviert.
- ☐ Im `.text`-Segment werden ausschließlich Zeichenketten gespeichert.
- ☐ Das `.bss`-Segment enthält ausschließlich mit 0 initialisierte Variablen.

b) Welche der folgenden Aussagen zum Begriff der Rücksprungadresse ist richtig?

2 Punkte

- ☐ Die Rücksprungadresse ermöglicht die Rückkehr ins Betriebssystem. Auf einer Mikrocontroller-Plattform ist sie allerdings nicht vorhanden.
- ☐ Bei rekursiven Funktionsaufrufen erstellt der Compiler eine Rücksprungadresse um sicher zu stellen, dass die Rekursion terminiert.
- ☐ Bei Aufruf einer Funktion sichert der Prozessor selbsttätig die Adresse der folgenden Instruktion. Dies ist die Rücksprungadresse.
- ☐ Bei Aufruf einer Funktion über einen Funktionszeiger muss der Programmierer eine Rücksprungadresse angeben, an der das Programm später fortgesetzt werden soll.

c) Welche Eigenschaften muss der Vorteiler (*Prescaler*) eines Zeitgebers (*Timer*) für periodische Interrupts haben, um möglichst energieeffizient zu sein.

2 Punkte

- ☐ Es sollte der kleinstmögliche Vorteiler gewählt werden, der ausreichend für die geforderte Genauigkeit ist.
- ☐ Es sollte der größtmögliche Vorteiler gewählt werden, der ausreichend für die geforderte Genauigkeit ist.
- ☐ Der energieeffizienteste Vorteiler hängt von der Zählrichtung des Zeitgebers ab: möglichst klein für herunterzählende und möglichst groß für hochzählende Zeitgeber.
- ☐ Der Vorteiler hat keinen Einfluss auf die Energieeffizienz des Mikrocontrollers.

d) Gegeben ist folgender Programmcode:

2 Punkte

```
#define SUB(a,b) a-b
#define ADD(a,b) a+b
```

Wie ist das Ergebnis des folgenden Ausdrucks

```
2 * SUB(3, ADD(3, 2))
```

- ☐ -4
- ☐ 1
- ☐ 4
- ☐ 5

e) Gegeben sei folgender Programmcode. Was gibt das Programm aus?

2 Punkte

```
char s1[] = "SPiC";
char s2[] = "SPIC";
s2[2] = 'i';

if(s1 == s2) {
    printf("match");
} else {
    printf("no match");
}
```

- ☐ Der C-Compiler meldet beim Übersetzen einen Fehler.
- ☐ Das Programm gibt `match` aus.
- ☐ Das Programm gibt `no match` aus.
- ☐ Das Programm stürzt zur Laufzeit ab.

f) 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 zum Zugriff auf kritische Daten Interrupts sperrt.
- ☐ 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.

g) Welche Aussage zu Prozessen ist richtig?

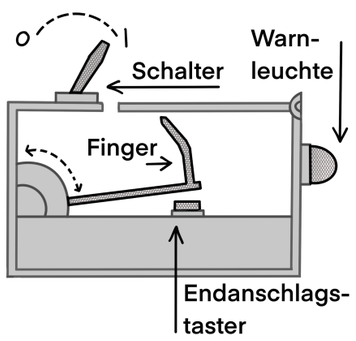
2 Punkte

- ☐ Auf Multi-Core-Systemen können mehrere Prozesse echt parallel ausgeführt werden. Jedoch nur so viele, wie Prozessorkerne zur Verfügung stehen.
- ☐ In einem Prozess können mehrere Programme gleichzeitig ausgeführt werden.
- ☐ Laufen mehrere Prozesse auf einer CPU, können diese direkt Daten austauschen.
- ☐ Auf Single-Core-Systemen gibt es keine Nebenläufigkeit.

Aufgabe 2: Useless Box (30 Punkte)

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

Implementieren Sie die Steuerung einer sogenannten *nutzlosen Kiste (engl. Useless Box)*. Auf der Oberseite der Kiste befindet sich ein Schalter. Wird dieser vom Nutzer umgelegt, leuchtet eine Warnleuchte für eine zufällige Zeitspanne. Danach startet eine Fingeranimation: Ein motorgesteuerter Finger fährt aus der Kiste und schiebt den Schalter zurück in seine ursprüngliche Position. Im Anschluss verschwindet der Finger wieder im Kisteninneren und aktiviert einen Endanschlagstaster. Damit ist der Zyklus beendet und die Kiste wartet, bis der Nutzer den Schalter erneut umlegt.



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 sowohl mit dem Schalter, als auch mit dem Endanschlag verbunden. Die externe Beschaltung stellt sicher, dass genau dann eine steigende Flanke auftritt, wenn der Schalter vom Nutzer aktiviert wird. Eine fallende Flanke an `PD2` signalisiert hingegen das Erreichen des Endanschlags.
- Nach Aktivierung des Schalters soll die Warnleuchte, welche am Pin `PB1` angeschlossen ist, für eine zufällige Zeitspanne leuchten. Diese Zeitspanne soll (annähernd) gleichverteilt aus den Ganzzahlen zwischen 200ms und 2000ms (einschließlich) gezogen werden.
- Implementieren Sie dafür die Funktion **`uint16_t`** `get_random(`**`void`**`)`, welche einen 16-Bit Zufallswert generiert, indem sie sechs mal mittels **`uint16_t`** `sb_adc_read(ADCDEV)` den Wert des Zufallsgebers `RANDOM` misst. Jede Messung liegt im Wertebereich 0 bis 1023. Während des Messvorgangs müssen die Interrupts deaktiviert sein! Die Messungen sollen mittels XOR-Operation verknüpft werden, wobei zunächst der Wert der i-ten Messung m_i mit 2^{i-1} multipliziert werden soll, also:

$$(2^0 \cdot m_1) \text{ XOR } (2^1 \cdot m_2) \text{ XOR } \ldots \text{ XOR } (2^5 \cdot m_6)$$

- Für die Zeittaktung soll ein 8-Bit Timer so konfiguriert werden, dass er in der Lage ist in Intervallen von 1ms zu zählen: Konfigurieren Sie diesen so, dass er alle $T = 1ms$ einen Interrupt auslöst.
- Nach Ablauf der Zeitspanne soll die Warnleuchte ausgeschaltet und die Fingeranimation gestartet werden. Sie müssen die Animation lediglich am Pin `PB0` aktivieren und sie wieder deaktivieren, nachdem an `PD2` eine fallende Flanke aufgrund des Endanschlags aufgetreten ist.
- Wird die Schalterstellung nach dem initialen Aktivieren verändert (zum Beispiel durch wiederholtes Umlegen des Schalters), erzeugt die externe Beschaltung so lange **keinen** weiteren Pegelwechsel an `PD2`, bis die Fingeranimation den Endanschlag erreicht hat.
- 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!

Eingangspin: Interruptleitung an **`PORTD`**, Pin 2

- Steigende Flanke: Schalter aktiviert, fallende Flanke: Endanschlag erreicht
- Pin als Eingang konfigurieren: Entsprechendes Bit im **`DDRD`**-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 Interruptquellen **`INT0`** (Bits im Register **`EICRA`**)

Interrupt 0		Beschreibung	Interrupt 1	
ISC01	ISC00		ISC11	ISC10
0	0	Interrupt bei low Pegel	0	0
0	1	Interrupt bei beliebiger Flanke	0	1
1	0	Interrupt bei fallender Flanke	1	0
1	1	Interrupt bei steigender Flanke	1	1

Fingeranimation: Ausgang an **`PORTB`**, Pin 0

- Bei anliegendem LOW-Pegel läuft die Fingeranimation
- Pin als Ausgang konfigurieren: Entsprechendes Bit im **`DDRB`**-Register auf 1
- Animation zunächst aus, entsprechendes Bit im **`PORTB`**-Register auf 1

Warnleuchte: Ausgang an **`PORTB`**, Pin 1

- Bei anliegendem LOW-Pegel leuchtet die Warnleuchte
- Pin als Ausgang konfigurieren: Entsprechendes Bit im **`DDRB`**-Register auf 1
- Warnleuchte zunächst aus, entsprechendes Bit im **`PORTB`**-Register auf 1

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

- Es soll die Überlaufunterbrechung verwendet werden (ISR-Vektor-Makro: **`TIMER0_OVF_vect`**)
- Der ressourcenschonendste Vorteiler (*prescaler*) ist 8, wodurch es bei dem 2,048 MHz CPU-Takt 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 <timer.h>

#define DELAY_MIN 200
#define DELAY_MAX 2000

#define RANDOM 17
extern uint16_t sb_adc_read(ADCDEV dev);

// Funktionsdeklarationen, globale Variablen, etc.

// Unterbrechungsbehandlungsfunktionen

// Ende Unterbrechungsbehandlungsfunktionen
```

U:


```
// Funktion main

  // Initialisierung und lokale Variablen

  // Hauptschleife

    // Warten auf Ereignisse

    // Ereignisse verarbeiten
```

```
// Ende main
```

M:


```
// Zufallsfunktion
```

```
// Ende Zufallsfunktion
```

Z:

```
// Initialisierungsfunktion
```

```
// Ende Initialisierungsfunktion
```

I:

Aufgabe 3: pish (19 Punkte)

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

Schreiben Sie ein Programm `pish` (pico shell), welches eine sehr vereinfachte Shell implementiert.

```
$> ./pish
pish> ls
file_a.txt file_b.txt
Return code: 0
pish>
```

Implementieren Sie zunächst die Hilfsfunktion `int sys(const char *cmd)`, welche eine vereinfachte Form der Bibliotheksfunktion `system(3)` nachbilden soll:

- `sys()` prüft zunächst den übergebenen Parameter `cmd`. Im Fall `cmd` gleich `NULL`, gibt die Funktion `-1` zurück und **keine** Fehlermeldung aus.
- Anschließend erzeugt `sys()` einen neuen Kindprozess.
- Der **Kindprozess** setzt die Signalbehandlung für das Signal `SIGINT` auf die Standardbehandlung zurück (siehe unten) und ersetzt anschließend das aktuell laufende Programm durch das Programm:

  ```
  /bin/sh -c <cmd>
  ```

  wobei `<cmd>` der übergebene Parameter `cmd` ist.
- Der **Elternprozess** wartet unterdessen auf die Beendigung des zuvor gestarteten Kindes. Wenn sich das Kind *normal* durch `exit()` beendigt, gibt `sys()` den Exitcode des Kindes zurück, ansonsten `-1`.
- Tritt in der Funktion `sys()` während der Bearbeitung im Elternprozess ein Fehler auf, wird auf `stderr` eine Fehlermeldung ausgegeben und `-1` zurückgegeben.

Die Funktion `main()` soll die folgende Funktionalität implementieren:

- Die Signalbehandlung für `SIGINT` soll zunächst auf *Ignorieren* gesetzt werden.
- Anschließend wird in einer Schleife der Prompt "`pish> `" ausgegeben und anschließend ein Kommando von `stdin` eingelesen. Ein Kommando wird durch einen Zeilenumbruch (`\n`) beendet. Die Schleife soll solange ausgeführt werden, bis `EOF` oder ein Fehler auftritt.
- Das eingelesene Kommado soll an `sys()` übergeben werden und anschließend der Rückgabewert von `sys()` auf `stderr` ausgegeben werden.

Hinweise:

- Sie dürfen davon ausgehen, dass eingegebenen Kommandos kleiner `1024` Zeichen sind.
- Zur effizienten Initialisierung von Strukturen des Typs `struct sigaction` können Sie die folgende Syntax verwenden (`sa_flags` und `sa_mask` werden so automatisch richtig initialisiert):

  ```
  struct sigaction act = {
     .sa_handler = <handler>,
  };
  ```

Achten Sie auf eine korrekte Fehlerbehandlung der verwendeten Funktionen. Fehlermeldungen sollen generell auf `stderr` erfolgen. Zur kompakten Fehlerbehandlung können die vorgegebenen Funktionen `die()` (`errno` gesetzt) und `err()` (`errno` nicht gesetzt) genutzt werden.

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

```
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>

static void die(const char message[]) {
  perror(message);
  exit(EXIT_FAILURE);
}

static void err(const char message[]) {
  fputs(message, stderr);
  exit(EXIT_FAILURE);
}

// Funktion sys()

  // Fehlerbehandlung

  // Kindprozess erzeugen
```



```
  // Kindprozess

  // Elternprozess

// Ende sys()
```

S:

```
// Funktion main
```

```
// Ende main()
```

L:

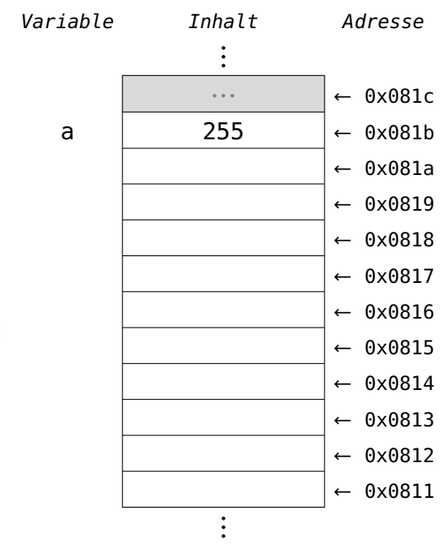
Aufgabe 4: Speicher (10 Punkte)

Das folgende Programm wird ohne Optimierungen übersetzt und auf einem 8-Bit AVR Mikrocontroller ausgeführt.

Hinweis: Lesen Sie zuerst die Aufgabenstellung – ein vollständiges Verständnis des Programms ist zur Bearbeitung der Aufgabe nicht notwendig.

```
1  #include <stdint.h>
2
3  static int16_t *find(const uint8_t array[], uint8_t size, uint8_t value) {
4    int16_t index = -1;
5    for(uint8_t i = 0; i < size; i++) {
6      if(array[i] == value) {
7        index = i;
8        break;
9      }
10   }
11   return &index;
12 }
13
14 void main(void) {
15   uint8_t a = 255;
16
17   // Aufgabe a)
18   uint8_t *b = &a;
19   uint16_t c = *b + a;
20   uint8_t d[3] = {23, 42, 0};
21   uint8_t e = d[((uint16_t) a + 1) >> 8];
22   // Ende Aufgabe a)
23
24   // Aufgabe b)
25   int16_t *m = find(d, 3, 42);
26   int32_t n = 42 + *m;
27
28   /* ... */
29 }
```

Dazugehöriger Stack (Auszug):



a) Obige Grafik zeigt einen Speicherauszug des Stacks nach der Ausführung der Zuweisung in Zeile 15 an. Erweitern Sie die Grafik um einen möglichen Aufbau des Stacks (Variable und Inhalt) nach der Ausführung der Zeilen 17 bis 22. (4 Punkte)

b) Die Funktion `find()` aus obiger Grafik soll das erste Element eines Feldes (Arrays) finden, das den Wert `value` hat. Welches Problem kann bei der Verarbeitung des Rückgabewertes von `find()` (Zeile 25ff) auftreten? Wieso? (2 Punkte)

---

c) Die folgende Tabelle enthält vier Zeiger-Datentypen. Geben Sie für alle Datentypen an, ob der dereferenzierte Wert und der Zeiger veränderlich sind (jeweils **ja** oder **nein**). Sollte ein Datentyp so in C nicht möglich sein, kreuzen Sie bitte die Spalte **Syntaxfehler** an. (4 Punkte)

	Wert veränderlich	**Zeiger veränderlich**	**Syntaxfehler**
`uint8_t *`			
`const uint8_t *`			
`uint8_t * const`			
`const uint8_t * const`			

Aufgabe 5: Nebenläufigkeit (8 Punkte)

Sie implementieren einen digitalen Regler, welcher präzise alle $10ms$ ausgeführt werden muss, um die gewünschten physikalischen Eigenschaften zu gewährleisten. Zur Umsetzung verwenden Sie einen Zeitgeber.

Durch eine unabhängig Zeitmessung des Signals am Pin `PD5` wissen Sie, dass Ihre Implementierung ein Nebenläufigkeitsproblem enthalten muss.

```
1  #include <avr/interrupt.h>
2  #include <avr/io.h>
3  #include <avr/sleep.h>
4
5  extern void run_controller(void);
6
7  #define COUNTER_MAX 3 // Wert für 10ms
8  static volatile uint8_t counter = 0;
9
10 ISR(TIMER0_OVF_vect) {
11     if(counter < COUNTER_MAX) {
12         ++counter;
13     }
14 }
15
16 void main(void) {
17     // [...]
18     sleep_enable();
19     while(1) {
20         cli();
21         while(counter < COUNTER_MAX) {
22             sei();
23             sleep_cpu();
24         }
25         sei();
26
27         run_controller(); // Regler ausführen
28
29         counter = 0; // Alarm zurücksetzen
30
31         PORTD ^= (1 << PD5); // Für unabhängige Messung
32     }
33 }
```

a) Benennen Sie das Nebenläufigkeitsproblem und kennzeichnen Sie die Stelle wo ein Interrupt das Nebenläufigkeitsproblem verursachen kann. (2 Punkte)

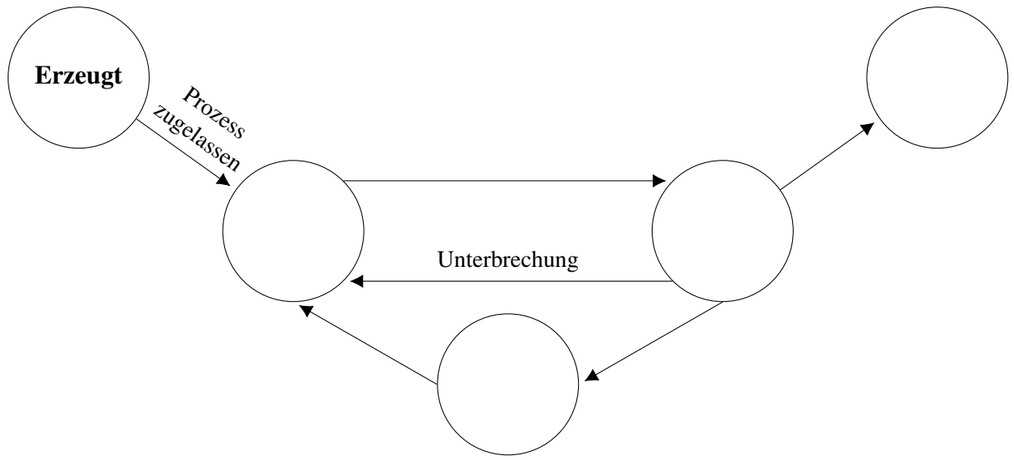
b) Skizzieren Sie einen konkreten Ablauf, der hier zu einem Nebenläufigkeitsfehler führt und seine Auswirkungen (Stichpunkte, kurze Sätze). (4 Punkte)

c) Beschreiben Sie die notwendigen Änderungen am Programmcode, um dieses Problem zu vermeiden. (1 Punkt)

d) Begründen Sie, warum die Variable `counter` in Zeile 8 als `volatile` deklariert ist. (1 Punkt)

Aufgabe 6: Prozesse (9 Punkte)

a) Vervollständigen Sie den folgenden Graphen zu den Prozesszuständen unter Linux mit den entsprechenden Zustandsübergängen. Jeder Knoten entspricht einem Prozesszustand und jede Kante entspricht einem Zustandsübergang. Achten Sie darauf **alle** noch nicht beschrifteten Knoten und Kanten zu beschriften. (6 Punkte)



b) Nennen Sie drei Informationen, die ein Betriebssystem (wie zum Beispiel Linux) in einem Prozesskontrollblock vorhält. (3 Punkte)
