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) Wann kann es zu Nebenläufigkeitsproblemen kommen? 2 Punkte

- ☐ Wenn ein Programmabschnitt in einer Schleife mehrfach durchlaufen wird.
- ☐ Wenn aus dem Hauptprogramm und einer Unterbrechungsbehandlungsfunktion dieselbe globale Variable geschrieben wird.
- ☐ Wenn ein Programm in einer Funktion mehrere lokale Variablen verwendet.
- ☐ Wenn die Programmabschnitte im `if`- und `else`-Teil einer bedingten Anweisung auf dieselbe Variable zugreifen.

b) Welche der folgenden Aussagen zum Linken eines Programms ist **richtig**? 2 Punkte

- ☐ Beim Linken werden Zugriffe aus einem C-Modul auf globale **`static`**-Variablen eines anderen C-Moduls optimiert.
- ☐ Der Linker fügt Objektdateien und Bibliotheken zu einer ausführbaren Datei zusammen.
- ☐ Der Linker löst **`#include`**-Anweisungen in einem C-Programm auf, indem er sie durch den Inhalt der einzubindenden Datei ersetzt.
- ☐ Auf einer Mikrocontroller-Plattform wird normalerweise dynamisch gelinkt, weil dies Speicherplatz spart.

c) Welche der folgenden Aussagen bzgl. der Interruptsteuerung ist richtig? 2 Punkte

- ☐ Beim Auftreten eines Interrupts sichert der Prozessor automatisch einige Register-Werte.
- ☐ Wurde gerade ein Pegel-gesteuerter Interrupt ausgelöst, so muss erst ein Pegelwechsel der Interruptleitung stattfinden, bevor erneut ein Interrupt ausgelöst wird.
- ☐ Flanken-gesteuerten Interrupts können nicht blockiert werden, da sie völlig unvorhersehbar auftreten.
- ☐ Pegel-gesteuerte Interrupts werden beim Wechsel des Pegels ausgelöst.

d) Wie löst man das Nebenläufigkeitsproblem „Lost-Update“ zwischen Hauptprogramm und Interrupthandler auf einem Mikrocontroller? 2 Punkte

- ☐ Durch die Verwendung von pegelgesteuerten anstelle von flankengesteuerten Interrupts.
- ☐ Durch den Aufruf einer Callback-Funktion im Interrupthandler.
- ☐ Durch Synchronisation mittels kurzzeitigem Sperren der Interrupts.
- ☐ Die Verwendung des Schlüsselwortes `volatile` löst alle Nebenläufigkeitsprobleme.

e) Gegeben ist folgender Programmcode: 2 Punkte

```
int32_t x[] = {1, 2, 4, 8};
int32_t *y = &x[1];
y += 2;
```

Welchen Wert liefert die Dereferenzierung von `y` nach der Ausführung des Programmcodes?

- ☐ 3
- ☐ 8
- ☐ Zur Laufzeit tritt ein Fehler auf.
- ☐ 2

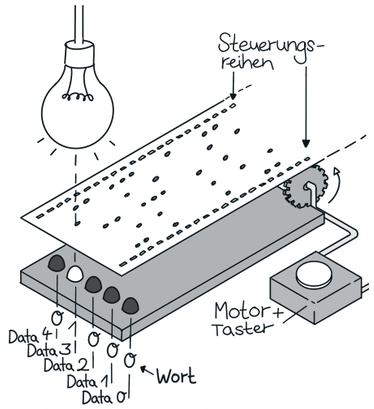
f) In welcher der folgenden Situationen wird ein laufender Prozess in den Zustand blockiert überführt? 2 Punkte

- ☐ Der Scheduler teilt die CPU einem anderen Prozess zu.
- ☐ Der Prozess greift lesend auf eine Datei zu und der entsprechende Datenblock ist noch nicht im Hauptspeicher vorhanden.
- ☐ Ein Kindprozess des Prozesses terminiert.
- ☐ Das Betriebssystem kann einen laufende Prozess nicht in den Zustand blockiert überführen, weil der Prozess sonst einen Trap auslösen würde.

Aufgabe 2: Lochstreifen-Lesegerät (30 Punkte)

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

Um den Transistorrechner der Informatik (Zuse Z23) weiter betreiben zu können, sollen Sie ein Programm für den AVR-Mikrocontroller zur Steuerung eines Lochstreifen-Lesegeräts schreiben.



Die Zuse nutzt Lochstreifen mit fünf Lochstellen (Bits) pro Wort und zwei Steuerungsreihen. Ein Loch an einer Lochstelle repräsentiert hierbei eine logische *1*, kein Loch eine *0*. Ein Steuerungsfühler erkennt, wann das nächste Wort zum Auslesen bereit ist.

Mittels unterhalb der Lochstellen angebrachten Fototransistoren erkennt der AVR, ob der von oben beleuchtete Lochstreifen an den entsprechenden Stellen gelocht wurde. Ein Motor kann den Lochstreifen weiterziehen, so dass das nächste Wort über den Fototransistoren liegt. Das Gerät wird durch einen Tastendruck angeschaltet und liest so lange Wörter, bis es durch einen erneuten Tastendruck wieder ausgeschaltet wird.

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.
- Das Lesegerät wird jeweils durch einen Tastendruck an- bzw. ausgeschaltet (Interrupt 0).
- Ist das Lesegerät aktiviert, zieht der Motor so lange am Lochstreifen, bis der Steuerungsfühler erkennt, dass das nächste Wort bereit zum Auslesen ist. Dies wird durch eine Unterbrechung (Interrupt 1) signalisiert. Ist das nächste Wort zum Auslesen bereit, soll der Motor sofort (noch im Interrupthandler) ausgeschaltet werden.
- Um ein Wort zu lesen, werden die unter den Lochstellen liegenden Fototransistoren (`DATA0`-`DATA4`) ausgelesen, wobei `DATA0` der Fototransistor für Bit 0 ist, `DATA1` für Bit 1 usw.
- Das Auslesen eines einzelnen Fototransistors soll in der von Ihnen zu implementierenden Funktion **`uint8_t`** `read_bit(ADCDEV dev)` gekapselt werden. Dazu soll mittels **`int16_t`** `sb_adc_read(ADCDEV dev)` der Wert des Fototransistors (`dev`) fünf mal ausgelesen werden. Wenn er in mehr als der Hälfte der Fälle über `PHOTO_THRESHOLD` liegt, nimmt das Lesegerät ein Loch an und es soll eine `1` zurückgegeben werden, ansonsten nimmt es kein Loch an und es wird eine `0` zurückgegeben. Während des Aufrufs von `sb_adc_read()` müssen die Unterbrechungen **gesperrt** sein.
- Sind alle fünf Bits eines Wortes ausgelesen, soll das gelesene Wort mit der vorgegebenen Funktion `send_word()` an die Zuse geschickt und zusätzlich per `sb_7seg_showHexNumber()` auf der 7-Segment-Anzeige ausgegeben werden. Anschließend soll der Motor wieder aktiviert werden, um das nächste Wort des Lochstreifens über die Fototransistoren zu bewegen.
- Wird das Lesegerät ausgeschaltet während gerade ein Wort gelesen wird, soll dieses noch zu Ende verarbeitet werden. Wird das Gerät ausgeschaltet und es liegt noch keine Lochreihe über den Fototransistoren oder das Auslesen wurde noch nicht gestartet, soll auch kein Wort mehr gelesen und der Motor ausgeschaltet werden. Sie dürfen davon ausgehen, dass der Steuerungsfühler kein neues Wort mehr signalisiert, wenn der Motor ausgeschaltet 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!

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

- active-low: Wird der Taster gedrückt, so liegt ein LOW-Pegel an
- Pin als Eingang konfigurieren: Entsprechendes Bit im **`DDRD`**-Register auf 0
- Internen Pull-Up-Widerstand aktivieren: Entsprechendes Bit im **`PORTD`**-Register auf 1
- Externe Interruptquelle **`INT0`**, ISR-Vektor-Makro: **`INT0_vect`**
- Aktivieren/Deaktivieren der Interruptquelle erfolgt durch Setzen/Löschen des **`INT0`**-Bits im Register **`EIMSK`**

Steuerungsfühler: Interruptleitung an **`PORTD`**, Pin 3

- active-high: Erkennt der Fühler ein neues Wort, wechselt der Pegel von LOW zu HIGH
- Pin als Eingang konfigurieren: Entsprechendes Bit im **`DDRD`**-Register auf 0
- Internen Pull-Up-Widerstand aktivieren: Entsprechendes Bit im **`PORTD`**-Register auf 1
- Externe Interruptquelle **`INT1`**, ISR-Vektor-Makro: **`INT1_vect`**
- Aktivieren/Deaktivieren der Interruptquelle erfolgt durch Setzen/Löschen des **`INT1`**-Bits im Register **`EIMSK`**

Konfiguration der externen Interruptquellen **`INT0`** und **`INT1`** (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

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

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

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 <7seg.h>
#include "adc.h"

#define PHOTO_THRESHOLD 512
ADCDEV photo[5] = { DATA0, DATA1, DATA2, DATA3, DATA4 };

extern void send_word(uint8_t word);
extern void sb_7seg_showHexNumber(uint8_t nmbr);
extern int16_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:

// Lesefunktion

// Ende Lesefunktion

R:

```
// Initialisierungsfunktion
```

```
// Ende Initialisierungsfunktion
```

I:

Aufgabe 3: Hexdump (17 Punkte)

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

Schreiben Sie ein Programm `hexdump`, welches die Inhalte von Dateien in hexadezimaler Darstellung ausgibt. Neben den Inhalten wird auch die jeweilige Position in der Datei ausgegeben. Ein Aufruf kann wie folgt aussehen:

```
$> ./hexdump /usr/bin/xxd /usr/bin/
/usr/bin/xxd:
00000000: 7f454c4602010100000000000000000
00000010: 03003e0001000000f027000000000000
00000020: 4000000000000000d841000000000000
[...]
Skip /usr/bin/ (no file)
```

Das Programm soll im Detail wie folgt funktionieren:

- Das Programm prüft zu Beginn, ob mindestens ein Kommandozeilenparameter übergeben wurden. Sollte dies nicht der Fall sein, gibt es eine entsprechende Fehlermeldung aus und beendet sich.
- Für jeden übergebenen Kommandozeilenparameter wird zunächst überprüft, ob es sich um eine reguläre Datei handelt.
- Schlägt die Überprüfung fehl oder handelt es sich nicht um eine reguläre Datei, wird eine Meldung auf `stdout` ausgegeben, der Exitcode auf `EXIT_FAILURE` gesetzt und mit dem nächsten Argument fortgefahren.
- Handelt es sich um eine reguläre Datei, wird der übergebene Pfad ausgegeben und die von Ihnen zu implementierende Funktion `void print(const char *path)` aufgerufen, die die hexadezimal formatierte Ausgabe der Datei implementiert.
- In `print()` wird die Datei byteweise gelesen und jedes Byte als zweistellige Hexadezimalzahl ausgegeben. Der entsprechende Formatstringmodifier für `printf()` um ein Byte als zweistellige Hexadezimalzahl auszugeben lautet `%02x`.

  Beispiel:
  `printf("%02x", 15)` → `0f`
- Pro Zeile sollen 16 Byte ausgegeben werden. Es kann sein, dass in der letzten Zeile weniger als 16 Byte übrig bleiben, dann sollen nur die restlichen Bytes ausgegeben werden. Am Anfang jeder Zeile soll die Position in der Datei (Anzahl der Bytes vom Dateianfang) als achtstellige Hexadezimalzahl ausgegeben werden. Der entsprechende Formatstringmodifier für `printf()` lautet `%08lx`. Nutzen Sie zum Speichern der Position eine Variable des Typs `unsigned long`.

  Beispiel:
  `printf("%08lx:", 16)` → `00000010:`
- Die Ausgabe der Inhalte einer Datei soll mit einer neuen Zeile abgeschlossen werden.
- Achten Sie darauf allokierte Ressourcen wieder freizugeben (`free()`, `fclose()`, ...).

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

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

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.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 print
```

```
  // Öffne Datei

  // Gebe Inhalte aus
```



```
// Ende Funktion print()
```

C:

```
// Funktion main

  // Parameter Fehlerbehandlung

  // Dateityp überprüfen und ggf. print() aufrufen

// Ende main()
```

L:

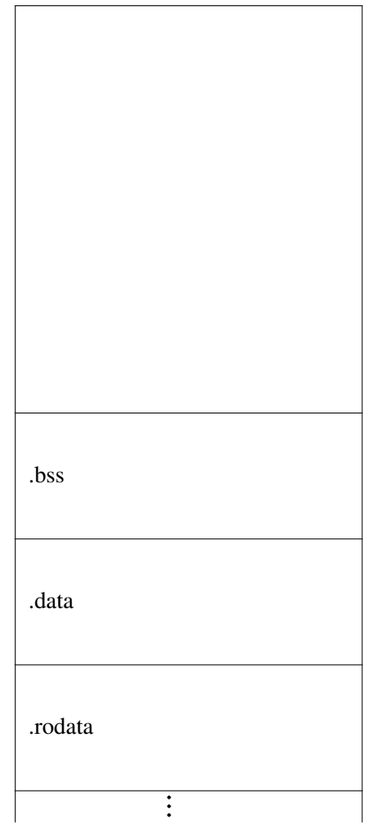
Aufgabe 4: Speicher (8 Punkte)

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  extern uint8_t read_byte(void);
4
5  #define size 8
6  static uint8_t buffer[size];
7  uint8_t event = 0;
8
9  void put(uint8_t in) {
10   static uint8_t write_index = size - 1;
11   write_index = (write_index + 1) % size;
12   buffer[write_index] = in;
13 }
14
15 uint8_t get(void) {
16   static uint8_t read_index = 0;
17   uint8_t out = buffer[read_index];
18   read_index = (read_index + 1) % size;
19   return out;
20 }
21
22 void main(void) {
23   while(1) {
24     while(event == 0) {
25         // [...]
26     }
27
28     event = 0;
29     put(read_byte());
30   }
31 }
```

Speicheraufbau (vereinfacht):



a) Vervollständigen Sie den (vereinfachten) Speicheraufbau:

5 Punkte

1) Ergänzen Sie *Stack* und *Heap* sowie deren Wachstumsrichtung in der Abbildung. (2 Punkte)

2) Ordnen Sie alle im Quelltext vorkommenden Variablen den dargestellten Speichersektionen zu. (3 Punkte)

b) Nennen Sie Lebensdauer und Sichtbarkeit der Variablen `buffer` (Zeile 6), `write_index` (Zeile 10) und `out` (Zeile 17). (3 Punkte)

Aufgabe 5: Zeitgeber & Nebenläufigkeit (12 Punkte)

Aus den Übungen kennen Sie die Zeitgeber des AVR Mikrocontrollers als mögliche Quelle von Unterbrechungen. Nehmen Sie in dieser Aufgabe einen vergleichbaren Zeitgeber an, der wie folgt aufgebaut ist:

Ein konfigurierbarer Eingangsvorteiler (*engl. prescaler*) $p \in \{1, 2, 4, 8\}$ stellt das Verhältnis zwischen CPU-Takt f_{cpu} und Zählfrequenz f_{count} her:

$$\frac{f_{cpu}}{f_{count}} = p. \tag{1}$$

Der nachgeschaltene Zähler besitzt eine Breite von $n = 4\,\text{Bit}$. Für die Überlauffrequenz f_{ovf} gilt also

$$f_{ovf} = \frac{f_{count}}{2^n}. \tag{2}$$

Jeder Überlauf erzeugt eine Unterbrechung am Vektor `OVF_vect`. Zur Erinnerung: Eine Frequenz ist der Kehrwert der dazugehörigen Periode:

$$f = \frac{1}{T}. \tag{3}$$

a) Der CPU-Takt sei $f_{cpu} = 1{,}6\,\text{MHz}$. Wählen Sie den Vorteiler p möglichst energieeffizient, so dass die Überlauffrequenz f_{ovf} mindestens $20\,\text{kHz}$ beträgt. Begründen Sie Ihre Wahl. (4 Punkte)

b) Für eine bestimmte Konfiguration des Zeitgebers (Vorteiler p und m Überlaufe) ist der relative Fehler zwischen der gewünschten Periode T_{soll} und der tatsächliche Periode T_{ist} zu hoch. Beschreiben Sie, wie Sie diesen Fehler bei gleichbleibender Soll-Periode reduzieren können. (2 Punkte)

c) Durch die Verwendung von Unterbrechungen im Allgemeinen (nicht nur durch Unterbrechungen des Zeitgebers) kann es zu Nebenläufigkeitsproblemen kommen. Ein mögliches Nebenläufigkeitsproblem ist die *Lost-Update* Anomalie. Beschreiben Sie exemplarisch wie es zu einer Lost-Update Anomalie kommen kann. (4 Punkte)

d) Ein anderes mögliches Nebenläufigkeitsproblem ist die *Read-Write* Anomalie. Welche Voraussetzung bezüglich des Datentyps der betroffenen Variablen muss gegeben sein, damit eine Read-Write Anomalie auftreten kann? Sind auch andere Architekturen außer der AVR-Architektur potenziell betroffen? (2 Punkte)

Aufgabe 6: Betriebssysteme (11 Punkte)

Im folgenden sehen Sie zwei inhaltsgleiche Programme, einmal für den AVR-Mikrocontroller und einmal für Linux, welche in einer Endlosschleife jeweils einen Integer-Wert einlesen, damit Berechnungen durchführen und das Ergebnis anschließend ausgeben.

AVR:

```
1  #include <7seg.h>
2  #include <adc.h>
3  #include <avr/interrupt.h>
4
5  void main(void) {
6    sei();
7
8    while(1) {
9      uint16_t in = sb_adc_read(POTI);
10     int8_t out = 127 / (1023 - in);
11
12     sb_7seg_showHexNumber(out);
13   }
14 }
```

Linux:

```
1  #include <stdio.h>
2
3  int main(int argc, char *argv[]) {
4    while(1) {
5      unsigned int in;
6      scanf("%u", &in);
7      // [...] error handling scanf()
8      if(in > 1023) {
9        in = 1023;
10     }
11     int out = 127 / (1023 - in);
12     printf("%d\n", out);
13   }
14 }
```

Hinweis: Gehen Sie davon aus, dass es eine Fehlerbehandlung für die Funktion scanf() gibt.

a) Bei beiden Implementierungen kann es für einen oder mehrere Eingabewerte zu fehlerhaften Programmausführungen kommen. Nennen Sie den oder die Eingabewerte und den Grund für die fehlerhafte Ausführung. (3 Punkte)

b) Beschreiben Sie jeweils wie die Plattformen (AVR und Linux) mit einer fehlerhaften Ausführung umgehen. Was ist der Unterschied zwischen den Plattformen? (4 Punkte)

c) Erläutern Sie kurz den Ablauf eines Systemaufrufs (*System Call*) in Linux. Worin besteht der Unterschied zu einem *normalen* Funktionsaufruf? (4 Punkte)