Aufgabe 1: (16 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) Was versteht man unter Polling? 2 Punkte

- ☐ Wenn ein Programm regelmäßig eine Peripherie-Schnittstelle abfragt, ob Daten oder Zustandsänderungen vorliegen.
- ☐ Das regelmäßige Anheben eines Pegels, um einem Gerät einen bestimmten Zustand zu signalisieren.
- ☐ Wenn ein Programm zum Zugriff auf kritische Daten Interrupts sperrt.
- ☐ Ein Konzept zur Abarbeitung von Interrupts.

b) Welche Aussage zu globalen Variablen ist richtig? 2 Punkte

- ☐ Globale Variablen sind bei sehr großen Programmen vorteilhaft, weil man alle Variablendefinitionen an einer Stelle hinschreiben kann und man sie dadurch sehr schnell finden kann.
- ☐ Durch den Einsatz von globalen Variablen werden vor allem große Programme unübersichtlich und auf Dauer schwer wartbar, da der direkte Bezug zwischen Daten und Funktionen verloren geht.
- ☐ Man sollte globale Variablen sparsam einsetzen, da sie mehr Speicherplatz benötigen als lokale Variablen.
- ☐ Durch die Verwendung von globalen Variablen kann man den Einsatz von Funktionsparametern vermeiden. Dadurch werden Programme übersichtlicher und leichter wartbar.

c) Was bewirken folgende Anweisungen in der Programmiersprache C? 2 Punkte

```
uint8_t x = 42;
x ^= x;
```

Welche Aussage ist richtig:

- ☐ Alle Bits der Variable ändern ihren Wert.
- ☐ Alle Bitwerte werden um eine Stelle nach links verschoben.
- ☐ Die Variable hat nach der Operation den Wert `0`.
- ☐ Die Variable hat nach der Operation den Wert `1`.

d) Welche Aussage zu Zeigern ist richtig? 2 Punkte

- ☐ Zeiger vom Typ `(void *)` sind am besten für Zeigerarithmetik geeignet, da sie kompatibel zu jedem Zeigertyp sind.
- ☐ Ein Zeiger kann zur Manipulation von schreibgeschützten Datenbereichen verwendet werden.
- ☐ Ein Zeiger darf nie auf seine eigene Speicheradresse verweisen.
- ☐ Der Speicherbedarf eines Zeigers ist unabhängig von der Größe des Objekts, auf das er zeigt.

e) Gegeben ist folgendes Makro: 2 Punkte

```
#define MUL(a,b) (a * b)
```

Wie ist das Ergebnis des folgenden Ausdrucks

```
3 * MUL(2 + 1, 4)
```

- ☐ 10
- ☐ 18
- ☐ 28
- ☐ 36

f) Wie viele Bytes belegt die folgende Struktur im Speicher eines AVR-Mikrocontrollers (gehen Sie davon aus, dass der Compiler nur soviel Speicher wie unbedingt nötig verwendet). 2 Punkte

```
union {
    struct {
        uint16_t lo, hi;
    };
    uint16_t r16;
} reg;
```

- ☐ 2 Bytes
- ☐ 4 Bytes
- ☐ 16 Bytes
- ☐ 32 Bytes

g) Welche Aussage zum Thema virtueller Adressraum ist richtig?

2 Punkte

- ☐ Die Abbildung von virtuellen auf physikalische Adressen erfolgt während der Programmlaufzeit durch eine spezielle Softwarekomponente.
- ☐ Dieselbe virtuelle Adresse kann in verschiedenen Prozessen auf unterschiedliche physikalische Adressen abgebildet werden.
- ☐ Virtuelle Adressen entsprechen Variablennamen in einem C-Programm. In Zeigern werden dagegen physikalische Adressen gespeichert, mit denen man die Abbildung umgehen kann.
- ☐ Die Umrechnung von virtuellen zu physikalischen Adressen erfolgt beim Übersetzen durch den Compiler.

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

2 Punkte

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

h) Welche Aussage zum ternären `?`-Operator der Programmiersprache C ist richtig.

2 Punkte

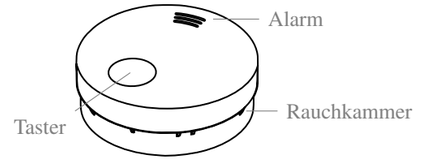
```
return c == 0 ? a : b;
```

- ☐ Wenn `c` gleich `42` ist, wird `a` zurückgegeben.
- ☐ Wenn `c` gleich `42` ist, wird `b` zurückgegeben.
- ☐ Es wird immer `a` zurückgegeben.
- ☐ Der Code kompiliert nicht.

Aufgabe 2: Rauchmelder (30 Punkte)

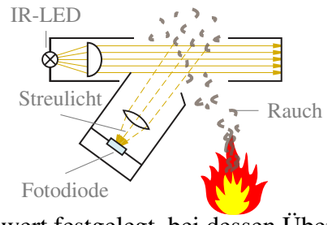
Sie dürfen diese Seite zur besseren Übersicht bei der Programmierung heraustrennen!

Der Brandschutz ist ein leidiges Thema an der FAU, an welchem bereits einige Experten verzweifelt sind. Deshalb soll die derzeitige Situation durch die Entwicklung eines eigenen, batteriebetriebenen, foto-optischen Rauchmelders verbessert werden.



Schreiben Sie ein Programm für den AVR-Mikrocontroller, welches diesen Rauchmelder steuert.

Prinzip:



Bei einem foto-optischen Rauchmelder sind die Infrarotleuchtdiode, sowie die dazugehörige Fotodiode, derart positioniert, dass nur ein geringer Anteil des ausgesendeten Lichts auf den Empfänger trifft. Sobald jedoch Rauch in den Melder gelangt, wird das Licht an den Rauchteilchen gestreut – und die Fotodiode detektiert einen höheren Lichteinfall. Da die Menge an Rauchteilchen diesen Lichteinfall beeinflusst, wird ein Grenzwert festgelegt, bei dessen Überschreitung der akustische Alarm auslöst. Ein aktiver Alarm kann durch Betätigen einer Zurücksetztaste („Reset“) wieder beendet werden. Zudem muss zur Gewährleistung der Sicherheit auch der Batterieladezustand regelmäßig geprüft und gegebenfalls ein Batteriewechsel durch einen Warnton signalisiert 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 16-bit-Messwert der Fotodiode wird periodisch bestimmt. Steht ein neuer Messwert zur Verfügung, wird dies durch einen Interrupt signalisiert. Die 8-bit Register `PHOTOL` und `PHOTOH` entsprechen den nieder- und höherwertigen Bytes des vorzeichenlosen Messwerts. Mit dem Verlassen des Interrupthandlers beginnt das Messen des nächsten Wertes, daher muss das Auslesen der Register im Interruptkontext geschehen.
- Nach dem Lesen eines Messwertes prüft das Hauptprogramm ob der Grenzwert `THRESHOLD` überschritten wurde. Ist dies der Fall, so wird der Brandmeldealarm ausgelöst.
- Ebenfalls soll nach jedem neuen Messwert auch der Ladezustand der Batterie überprüft werden. Unterschreitet dieser ein kritisches Level, so ist ein Bit gesetzt und es soll ein Warnton ausgegeben werden.
- Mit dem Lautsprecher kann ein lautes, monotones, akustisches Signal ausgegeben werden. Die Dauer gibt Aufschluss über die Bedeutung:
  - Ein niedriger Batterieladezustand wird mit einem einzelnen kurzen (20ms langen) Warnton nach jeder Ladezustandsprüfung signalisiert.
  - Beim Brandmeldealarm ertönt ein regelmäßig wiederholendes Warnsignal (Rechteckschwingung) mit einer Periodendauer von 500ms: 500ms Signal, 500ms Pause, 500ms Signal, 500ms Pause, ...
- Eine Zeitperiode wird durch aktives Warten in der Funktion `void wait(uint16_t ms)` implementiert: Für jede Millisekunde werden `LOOPS_PER_MS` Schleifendurchlaufe gewartet. Eine Verarbeitung von Ereignissen muss während dieser Funktion nicht durchgeführt werden.
- Ein Brandmeldealarm kann **nach** dem ersten Signalton durch Drücken der Resettaste beendet werden – ein erneutes Überschreiten des Grenzwertes löst jedoch erneut einen Alarm aus.
- Um die Laufzeit des Brandmelders zu maximieren, soll – solange kein akustisches Signal ausgegeben wird – der Mikrocontroller so viel Zeit wie möglich im Schlafmodus verbringen.

Information über die Hardware

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

Rauchsensor: Interruptleitung an **PORTD**, Pin 2
- Wurde ein neuer Messwert bestimmt, so ändert sich der anliegende Pegel.
- Pin als Eingang konfigurieren: entsprechendes Bit in **DDRD**-Register auf 0
- Internen Pull-Up-Widerstand aktivieren; entsprechendes Bit in **PORTD**-Register auf 1
- Externe Interruptquelle **INT0**, ISR-Vektor-Makro: **INT0_vect**.
- Aktivierung der Interruptquelle erfolgt durch Setzen des **INT0**-Bits im Register **EIMSK**.

Resettaste: Interruptleitung an **PORTD**, Pin 3
- active-low: Wird die Taste gedrückt, so liegt ein LOW-Pegel an.
- Pin als Eingang konfigurieren: entsprechendes Bit in **DDRD**-Register auf 0
- Internen Pull-Up-Widerstand aktivieren; entsprechendes Bit in **PORTD**-Register auf 1
- Externe Interruptquelle **INT1**, ISR-Vektor-Makro: **INT1_vect**.
- Aktivierung der Interruptquelle erfolgt durch Setzen des **INT1**-Bits im Register **EIMSK**.

Batteriestatus: **PORTB**, Pin 1
- Solange der Ladezustand der Batterie ausreichend ist, liegt ein HIGH-Pegel an.
- Bei Erreichen eines kritischen Ladezustands, liegt ein LOW-Pegel an.
- Pin als Eingang konfigurieren: entsprechendes Bit in **DDRB**-Register auf 0
- Internen Pull-Up-Widerstand deaktivieren; entsprechendes Bit in **PORTB**-Register auf 0
- Auslesen des Zustands über entsprechendes Bit im **PINB**-Register

Akustischer Alarm: **PORTC**, Pin 3
- Die integrierter Elektronik des Summers (Piezoschallwandler) ermöglicht eine einfache Verwendung mit Gleichstrom.
- Bei anliegendem HIGH-Pegel erzeugt der Summer einen ständigen, lauten Ton (in der Eigenresonanzfrequenz).
- Pin als Ausgang konfigurieren: entsprechendes Bit in **DDRC**-Register auf 1
- Alarm zunächst abgeschalten; entsprechendes Bit in **PORTC**-Register auf 0

Konfiguration der externen Interruptquellen **INT0** und **INT1** (Bits in 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

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


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

#define PHOTOL (*((volatile uint8_t *) 0x170))
#define PHOTOH (*((volatile uint8_t *) 0x180))
#define LOOPS_PER_MS 250

const uint16_t THRESHOLD = 23000;

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

```
// Unterbrechungsbehandlungsfunktionen
```

A:

```
// Funktion main

  // Initialisierung und lokale Variablen

  // Hauptschleife

    // Warten auf Ereignisse
```

M:


```
    // Behandlung von Grenzwert und Batteriestatus

// Ende der Main

// Wartefunktion

// Ende Wartefunktion
```

E:

```
// Initialisierungsfunktion
```

```
// Ende Initialisierungsfunktion
```

I:

Aufgabe 3: pfind (15 Punkte)

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

Schreiben Sie ein Programm `pfind`, das parallel mehrere Verzeichnisse (nicht rekursiv) nach Einträgen, die auf ein Suchmuster passen, durchsucht. Der Aufruf des Programms erfolgt mit dem Suchmuster als erstem Parameter und einer Liste von einem oder mehreren zu durchsuchenden Verzeichnissen als weitere Parameter, zum Beispiel:

```
pfind search_pattern dir1/ dir2/
```

Das Programm soll im Detail wie folgt funktionieren:

- Das Programm prüft zu Beginn, ob mindestens zwei Parameter übergeben wurden (das Suchmuster und mindestens ein Verzeichnisname). Sollte dies nicht der Fall sein, gibt es eine entsprechende Fehlermeldung aus und beendet sich.
- Wurde das Programm korrekt aufgerufen, wird für jedes Verzeichnis mittels `fork()` ein Suchprozess erstellt, der die Suche in dem Verzeichnis durchführt.
- Um die Suche in einem Verzeichnis durchzuführen, ruft jeder Suchprozess die zu implementierende Funktion

  ```
  void find(const char *pattern, const char *dir);
  ```

  mit dem Suchmuster und seinem Verzeichnis auf.
- Hat ein Suchprozess alle Einträge seines Verzeichnisses durchsucht, beendet er sich.
- Der Hauptprozess wartet nach dem Starten aller Suchprozesse mittels `wait()` auf den Abschluss aller Suchprozesse.
- Die Funktion `find()` bekommt neben einem Zeiger auf das Suchmuster einen Zeiger auf das zu durchsuchende Verzeichnis. Für jeden Verzeichniseintrag soll mittels der vorgegebenen Funktion

  ```
  int match(const char *pattern, const char *str);
  ```

  überprüft werden, ob der Name des Verzeichniseintrags zu dem Suchmuster passt. Passt der Name des Verzeichniseintrags auf das Suchmuster gibt `match()` als Rückgabewert 1 zurück, ansonsten 0.
- Passt ein Verzeichniseintrag auf das Suchmuster, soll der Eintrag auf der Konsole ausgegeben werden, wofür die vorgegebene Funktion

  ```
  void print_entry(const char *dir, const char *entry_name);
  ```

  zur Verfügung steht, die jeweils einen Zeiger auf den Verzeichnisnamen und den Namen des Eintrags entgegen nimmt. Passt der Name nicht auf das Suchmuster soll der Eintrag ignoriert werden.

Hinweise:

- Achten Sie auf eine korrekte Fehlerbehandlung der verwendeten Funktionen.
- Für Fehler, bei denen die `errno` Variable passend gesetzt wurde, kann die vorgegebene Funktion `die()` zur kompakten Fehlerbehandlung genutzt werden.
- Die Ausgabe von Fehlern soll (ggf. mit Hilfe der `errno`-Variable) auf `stderr` erfolgen.

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

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <dirent.h>
#include <errno.h>

extern int match(const char *pattern, const char *str);
extern void print_entry(const char *dir, const char *entry_name);

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

// Funktion find
```

X:


```c
// Funktion main

  // Parameter Fehlerbehandlung

  // Pro Verzeichnis neuen Suchprozess erzeugen
```

Y:

```
  // Warten auf die Beendigung aller Suchprozesse
```

```
// Ende main
```

Z:

Aufgabe 4: Module und Speicherorganisation (10 Punkte)

Sie dürfen diese Seite zur besseren Übersicht heraustrennen!

Das folgende Programm wird auf einem 8-Bit AVR/ATmega32 Mikrocontroller ausgeführt.

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

Datei main.c:

```
#include <stdint.h>
#include <util/delay.h>
#include "led.h"

void main(void){
  for (uint8_t x = 0; x < 16; x++){
    schalte_led(x);
    _delay_ms(1000); // Warte 1 Sekunde
  }
}
```

Datei led.c:

```
#include <avr/io.h>
#include <stdint.h>
#include "led.h"

#define BV(X) (1 << (X))

const uint8_t pin_bit[] = { BV(PD7), BV(PD4), BV(PD5), BV(PD6) };
static uint8_t zaehler;

void init(void){
  static uint8_t initialisiert = 0;
  if (initialisiert == 0){
    initialisiert = 1;
    for (zaehler = 0; zaehler < 4; zaehler++){
      DDRD  |= pin_bit[zaehler];
      PORTD |= pin_bit[zaehler];
    }
  }
}

void schalte_led(uint8_t maske){
  init();
  for (zaehler = 0; zaehler < 4; zaehler++){
    if (maske & BV(zaehler)){
      PORTD &= ~pin_bit[zaehler];
    } else {
      PORTD |= pin_bit[zaehler];
    }
  }
}
```

a) Vervollständigen Sie in der Tabelle die Eigenschaften der genannten Variablen (6 Punkte)

Variable	Sichtbarkeit	Lebensdauer	Speichersegment
`x`	Block	Block	Stack
`pin_bit`			
`zaehler`			
`initialisiert`			
`maske`			

b) Nennen Sie alle Deklarationen, welche in der Headerdatei `led.h` mindestens genannt sein müssen, damit die vorliegenden C-Quelldateien zu einem lauffähigen Programm übersetzt werden können. Begründen Sie Ihre Auswahl! (2 Punkte)

c) Nennen Sie die einzelnen Schritte (als Stichpunkte), die durchzuführen sind, um aus den C-Quelldateien ein lauffähiges Programm zu erstellen und es auf einem Mikrocontroller auszuführen. (2 Punkte)

Aufgabe 5: Betriebssysteme (9 Punkte)

Die folgenden Beschreibungen sollen kurz und prägnant erfolgen (Stichworte, kurze Sätze).

a) Nennen Sie drei Eigenschaften der Programmiersprache C, die diese besonders geeignet für die systemnahe Softwareentwicklung machen. (3 Punkte)

b) Beschreiben Sie die Begriffe Programm und Prozess und erläutern Sie den Unterschied. (3 Punkte)

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

Aufgabe 6: Nebenläufigkeit (10 Punkte)

Sie dürfen diese Seite zur besseren Übersicht heraustrennen!

Das nachfolgende Codebeispiel für einen 8-Bit-AVR-Mikrocontroller überprüft, ob der Wert der modul-globalen Variable `value` über einem bestimmten Schwellwert liegt. Wenn dies der Fall ist, wird darauf reagiert, ansonsten wird mit der Hauptschleife fortgefahren. Der Wert von `value` wird asynchron durch die Unterbrechungsbehandlung von `INT0` erhöht.

Die Implementierung dieser Funktionalität beinhaltet ein Nebenläufigkeitsproblem.

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

#define THRESHOLD 0x0150
static volatile uint16_t value = 0;

ISR(INT0_vect) {
    value += 3;
}

void main(void) {
    /* ... */
    sei();
    while (1){
        uint16_t local_value = value;
        if(local_value > THRESHOLD) {
            // handle case ...
        }

        // do something else ...
    }
}
```

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

```
    ;local_value=value;
H1: lds r22, value   ;low  8-bit
H2: lds r23, value+1 ;high 8-bit
    ;if(local_value > THRESHOLD)
H3: cpi r22, 0x51
H4: sbci r23, 0x01
H5: brcs <target>
```

Interruptbehandlung `INT0`

```
    ;value += 3;
I1: lds r24, value   ;low  8 bit
I2: lds r25, value+1 ;high 8 bit
I3: adiw r24, 3
I4: sts value+1, r25 ;high 8 bit
I5: sts value, r24   ;low  8 bit
```


a) Benennen Sie das Nebenläufigkeitsproblem. (1 Punkt)

---

b) Demonstrieren Sie einen konkreten Programmablauf, bei dem das Nebenläufigkeitsproblem auftritt. (5 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 `value` initial den Wert `0x00ff` hat und treffen Sie keine Annahmen über andere Variablen- oder Speicherinhalte. Kennzeichnen Sie auch, wann gegebenfalls ein Interrupt auftritt.

Zeile	value	r22	r23	r24	r25
–	0x00ff	–	–	–	–

c) Welche(r) Speicher- bzw. Registerinhalt(e) enthalten nach dem von Ihnen beschriebenen Programmablauf nun falsche Werte. Nennen Sie außerdem die/den jeweils korrekten Wert(e).
(1 Punkt)

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

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