Aufgabe 1: (16 Punkte)

Bei den Multiple-Choice-Fragen ist jeweils nur **eine** richtige Antwort eindeutig anzukreuzen. Auf die richtige Antwort gibt es die angegebene Punktzahl.

Wollen Sie eine Multiple-Choice-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) Welchen Wert hat die Variable a nach Ausführung der folgenden Zeilen Code: 2 Punkte

```
uint8_t a = 3 | 16;
a &= (1 << 4);
```

- ❏ 1
- ❏ 3
- ❏ 16
- ❏ 0

b) Worin liegt der Unterschied zwischen den wie folgt in der Datei **prog.c** deklarierten Funktionen? 2 Punkte

```
uint8_t testA(void);
static uint8_t testB(uint8_t param);
```

- ❏ Die Function **testA** ist nur für Funktionen in der Datei **prog.c** aufrufbar.
- ❏ Die Funktion **testB** wird vom Linker statisch in das Programm gebunden, **testA** wird hingegen dynamisch (zur Laufzeit) eingebunden.
- ❏ Die Function **testB** ist nur für Funktionen in der Datei **prog.c** aufrufbar.
- ❏ Die Funktion **testB** ist nur über einen Funktionszeiger aufrufbar.

c) Gegeben ist folgender Programmcode: 2 Punkte

```
int32_t x[] = {3, 8, -13, 5, 4};
int32_t *y = &x[4];
y -= 3;
```

Welchen Wert liefert die Dereferenzierung von y (also ***y**)?

- ❏ Zur Laufzeit tritt ein Fehler auf.
- ❏ 8
- ❏ 4
- ❏ 1

d) Welche Aussage zu Zeigern ist richtig? 2 Punkte

- ❏ Ein Zeiger kann zur Manipulation von schreibgeschützten Datenbereichen verwendet werden.
- ❏ Die Speicherstelle, auf die ein Zeiger verweist, kann niemals selbst einen Zeiger enthalten.
- ❏ Beim Rechnen mit Zeigern muss immer der Typ des Zeigers beachtet werden.
- ❏ Zeiger vom Typ **void*** benötigen weniger Speicher als andere Zeiger, da bei anderen Zeigertypen zusätzlich die Größe gespeichert werden muss.

e) Was bewirkt folgende Funktion? 2 Punkte

```
void func(int a, int b) {
    int c = a; a = b; b = c;
}
```

- ❏ Bei einem Aufruf vertauscht die Funktion die Inhalte der an die formalen Parameter a und b übergebenen Variablen.
- ❏ Da in C Funktionen mit "call by value" aufgerufen werden, erhält die Funktion Kopien der Aufrufparameter, die sie vertauscht. Beim Aufrufer hat dies allerdings keine Auswirkungen.
- ❏ Der Compiler meldet bei der Übersetzung einen Fehler, weil die Funktionsparameter nicht verändert werden dürfen.
- ❏ Die von b adressierte Speicherzelle enthält nach dem Aufruf die in der Variablen a abgelegte Speicheradresse.

f) Wann kann es zu Nebenläufigkeitsproblemen kommen? 2 Punkte

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

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

- ❏ Globale Variablen sind bei sehr großen Programmen vorteilhaft, weil man alle Variablendefinitionen in einer Datei an einer Stelle hinschreiben kann und man sie dadurch sehr schnell findet.
- ❏ Durch die Verwendung von globalen Variablen kann man den Einsatz von Funktionsparametern vermeiden. Dadurch werden Programme übersichtlicher und leichter wartbar.
- ❏ Durch den Einsatz von globalen Variablen werden vor allem große Programme unübersichtlich und auf Dauer schwer wartbar, da der direkte Bezug zwischen Daten und den Funktionen verloren geht.
- ❏ Globale Variablen sind die einzige Möglichkeit, Daten über Modulgrenzen hinweg auszutauschen.

h) Welche Aussage zu *volatile* ist richtig? 2 Punkte

- ❏ Das Schlüsselwort *volatile* hat nur noch eine historische Bedeutung und ist in heutigen Programmen nicht mehr nötig.
- ❏ Das Schlüsselwort *volatile* unterbindet Optimierungen an einer damit definierten Variable.
- ❏ Das Schlüsselwort *volatile* unterbindet alle Nebenläufigkeitsprobleme.
- ❏ Das Schlüsselwort *volatile* erlaubt dem Compiler bessere Optimierungen durchzuführen.

Aufgabe 2: Heißer Draht (30 Punkte)

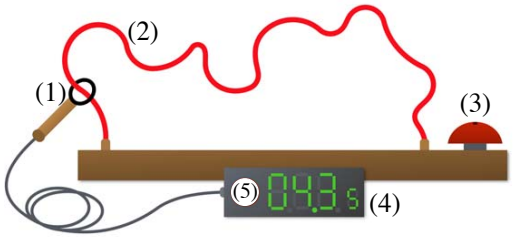
Sie dürfen diese Seite zur besseren Übersicht bei der Programmierung heraustrennen!

Schreiben Sie ein Programm für einen AVR-Mikrocontroller, welches die Steuerung des Spiels "Heißer Draht" implementiert.
Ziel des Spiels ist es, einen Metallring (1) möglichst schnell vom Anfang bis zum Ende eines Drahtes (2) zu bewegen ohne dabei den Draht zu berühren.

Nach einem erfolgreichem Spieldurchgang meldet eine Anzeige die benötigte Gesamtdauer in Sekunden (4). Bei Berühren des Drahtes wird das Spiel abgebrochen. Dies wird durch ein rotes Licht (5) angezeigt.

Im Detail soll Ihr Programm wie folgt funktionieren:

- Initialisieren Sie die Hardware in der Funktion **`void init(void)`**. Treffen Sie keine Annahmen über den initialen Zustand der Hardware-Register. Zu Beginn sollen Licht und Anzeige ausgeschaltet sein.
- Ein Spieldurchgang beginnt durch Drücken des Tasters (3) und startet die Zeiterfassung. Während ein Spiel aktiv ist, sind das rote Licht und die Anzeige ausgeschaltet.
- Erreicht der Spieler das Ende des Drahtes, drückt er so schnell wie möglich erneut den Taster (3). Die Spieldauer wird dann auf der eingebauten Anzeige (4) angezeigt.
- Sollte der Spieler vor Erreichen des Endes den Draht mit dem Metallring berühren, ist das Spiel vorzeitig beendet und ein rotes Licht (5) leuchtet auf.
- Um die aktuelle Zeit zu ermitteln, existiert die Funktion **`uint32_t getTime(void)`**. Sie liefert die Millisekunden seit Start des Mikrocontrollers. Mögliche Überläufe können hierbei vernachlässigt werden.
- Für die Anzeige der Spieldauer steht die Funktion **`void displayTime(uint32_t msec)`** zur Verfügung. Die Funktion **`void displayOff(void)`** schaltet die Anzeige aus.
- Sowohl Berührungen des Drahtes als auch Tastendrücke sollen mit Hilfe von Interrupts erkannt werden. (kein Polling!)
- Während des Wartens auf Tastendrücke oder Drahtberührungen soll der Mikrocontroller zum Stromsparen in den Schlafmodus gehen.



Information über die Hardware

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

Licht: **PORTB**, Pin 2
- active-low: Das Licht leuchtet sobald ein LOW-Pegel anliegt.
- Pin als Ausgang konfigurieren: entsprechendes Bit in **DDRB**-Reg. auf 1
- Licht zu Beginn ausschalten; entsprechendes Bit in **PORTB**-Register auf 1

Taster: Interruptleitung an **PORTD**, Pin 2
- active-low: Wird der Taster 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 **INT0**, ISR-Vektor-Makro: **INT0_vect**.
- Aktivierung der Interruptquelle erfolgt durch Setzen des **INT0**-Bits im Register **GICR**.

Draht: Interruptleitung an **PORTD**, Pin 3
- active-low: Wird der Draht berührt, 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 **GICR**.

- Konfiguration der externen Interruptquellen 0 und 1 (Bits in Register **MCUCR**)

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>

void displayOff(void);
void displayTime(uint32_t msec);
uint32_t getTime(void);


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

```
/* Unterbrechungsbehandlungsfunktionen */
```

A:

```
/* Funktion main */

   /* Variablen, Initialisierung */

  /* Hauptschleife */

      /*  Warten auf Interrupt */
```

M:


```
      /*  Taster wurde gedrueckt */

      /*  Draht wurde beruehrt */

/* Ende main */
```

B:

```
/* Initialisierungsfunktion */
```

I:

Aufgabe 3: (5 Punkte)

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

a) Was versteht man unter einem Interrupt? (3 Punkte)

b) Beschreiben Sie den Unterschied zwischen Pegel- und Flanken-gesteuerten Interrupts. (2 Punkte)

Aufgabe 4: (9 Punkte)

a) Was versteht man unter der Sichtbarkeit und Lebensdauer einer Variable und wie werden diese Konzepte in C umgesetzt? (6 Punkte)

b) Welche Bedeutung haben die unterschiedlichen Gültigkeitsbereiche von Variablen in einem C-Programm in Bezug auf Nebenläufigkeit bei Interrupts (d. h. wie sind Variablen der verschiedenen Kategorien von Nebenläufigkeitsproblemen betroffen und warum ist dies so)? (3 Punkte)