Aufgabe 1: (18 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) Was versteht man unter Nebenläufigkeit? 2 Punkte

- ❏ Wenn ein Programm abwechselnd auf zwei verschiedene Speicherbereiche zugreift.
- ❏ Wenn ein Programmabschnitt in einer Schleife mehrfach durchlaufen wird.
- ❏ Die Programmabschnitte im if- und else-Teil einer bedingten Anweisung.
- ❏ Wenn für zwei Befehle aus zwei Programmabläufen nicht feststeht, welcher von beiden tatsächlich zuerst ausgeführt werden wird.

b) Wie löst man das Nebenläufigkeitsproblem "lost-update" zwischen Hauptprogramm und Interrupthandler auf einem Mikrocontroller?

- ❏ 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

c) Wie viele Bytes belegt die folgende Struktur im Speicher eines AVR-Mikrocontrollers: 2 Punkte

```
union {
    struct {
        uint8_t lo, hi;
    };
    uint16_t r16;
} reg;
```

- ❏ 2 Bytes
- ❏ 4 Bytes
- ❏ 16 Bytes
- ❏ 32 Bytes

d) 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.

e) Was ist ein Stack-Frame? 2 Punkte

- ❏ Der Speicherbereich, in dem der Programmcode einer Funktion abgelegt ist.
- ❏ Ein spezieller Registersatz des Prozessors zur Bearbeitung von Funktionen.
- ❏ Ein Fehler, der bei unberechtigten Zugriffen auf den Stack-Speicher entsteht.
- ❏ Ein Bereich des Speichers, in dem u.a. lokale automatic-Variablen einer Funktion abgelegt sind.

f) Welche der folgenden Aussagen zum Begriff der Rücksprungadresse ist richtig? 2 Punkte

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

g) Gegeben ist folgender Programmcode: 2 Punkte

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

Was ist das Ergebnis des folgenden Ausdrucks?

```
2 * SUB(2, ADD(3, 4))
```

- ❑ -10
- ❑ -3
- ❑ 5
- ❑ 6

Aufgabe 2: Airbagsteuerung (30 Punkte)

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

Schreiben Sie ein Programm für den AVR-Mikrocontroller, das Messwerte von einem Verzögerungssensor entgegen nimmt und bei Überschreitung eines vorgegebenen Grenzwertes den Airbag auslöst. Die Airbagsteuerung bekommt hierfür Interrupts des Verzögerungssensors, sobald neue Messwerte zum Auslesen zur Verfügung stehen. Wird über einen vordefinierten Zeitraum kein Messwert vom Sensor zur Verfügung gestellt, dann wird dies mit Hilfe eines Watchdog-Timers erkannt und in einen Fehlerzustand gewechselt, der durch eine LED visualisiert wird.

Im Detail soll Ihr Programm wie folgt funktionieren:

- Initialisieren Sie die Hardware in der Funktion **`void init(void),`** so dass auch die LED für die Fehleranzeige ausgeschaltet ist. Treffen Sie keine Annahmen über den initialen Zustand der Hardware-Register.
- Zu Beginn prüft das Programm einmalig, ob der Airbag einsatzbereit ist. Wurde der Airbag bereits ausgelöst, wechselt das Programm durch Aufruf von **`error()`** in den Fehlermodus.
  Implementieren Sie hierfür die Funktion
  ```
  void error(void);
  ```
  welche die LED zur Fehleranzeige aktiviert und anschließend in einer Endlosschleife wartet.
- Ist der Airbag einsatzbereit, wird der Watchdog-Timer aktiviert, der später zur Erkennung von Fehlfunktionen des Sensors dient.
  Hierfür muss eine Behandlungsfunktion mit der Signatur
  ```
  void handler(void);
  ```
  durch Aufruf der vorgegebenen Bibliotheksfunktion
  ```
  void registerWatchdog(void (*func_ptr) (void));
  ```
  registriert werden.
  Anschließend geht das Programm in den Steuerungsbetrieb über, in dem der Mikrocontroller im stromsparenden Schlafmodus auf Ereignisse von Sensor oder Watchdog wartet.
- Steht ein neuer Messwert des Verzögerungssensors zur Verfügung, so wird das durch einen Interrupt signalisiert. Der vorzeichenlose 16-Bit Messwert kann dann aus den beiden Registern **`SENSOR_LOW`** und **`SENSOR_HIGH`** ausgelesen werden. Wurde der Messwert abgeholt, muss dies dem Sensor durch Schreiben des Registers **`SENSOR_RESET`** mit dem Wert 1 mitgeteilt werden. Aus Latenzgründen muss das Auslesen und Zurücksetzen im Interrupt-Handler stattfinden.
- Nachdem ein neuer Messwert gelesen wurde, prüft das Hauptprogramm ob der Grenzwert der maximalen Verzögerung überschritten wurde. Ist dies der Fall, so wird der Airbag ausgelöst und anschließend in den Fehlermodus gewechselt.
- In regelmäßigen Abständen wird der Watchdog-Timer ausgelöst, dessen Handler im Interruptkontext ausgeführt wird. Wurde seit dem vorhergehenden Watchdog-Ereignis kein Sensorwert empfangen, ist von einer Fehlfunktion des Sensors auszugehen und das Hauptprogramm wechselt in den Fehlermodus.

Information über die Hardware

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

LED für Fehleranzeige: **PORTB**, Pin 5
- active-low: Die LED leuchtet sobald ein LOW-Pegel anliegt.
- Pin als Ausgang konfigurieren: entsprechendes Bit in **DDRB**-Register auf 1
- LED zu Beginn ausschalten; entsprechendes Bit in **PORTB**-Register auf 1

Verzögerungssensor: Interruptleitung **INT0** an **PORTD**, Pin 2
- active-low: steht ein neuer Messwert zur Verfügung, 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**
- Abfrage des aktuellen 16-Bit Messwertes durch Auslesen von Register **SENSOR_HIGH** (oberes Byte) und **SENSOR_LOW** (unteres Byte)
- Sensor für erneute Messung zurücksetzen durch Schreiben in Register **SENSOR_RESET** mit Wert 1
- Das Auslesen und das Zurücksetzen des Sensors muss im Interruptkontext stattfinden

Airbag Statusabfrage: **PORTB**, Pin 6
- active-low: Ist der Airbag einsatzbereit, so liegt ein LOW-Pegel an
- Pin als Eingang konfigurieren: entsprechendes Bit in **DDRB**-Register auf 0
- Internen Pull-Up-Widerstand aktivieren; entsprechendes Bit in **PORTB**-Register auf 1

Airbag Auslösung: **PORTB**, Pin 7
- active-high: Der Airbag wird ausgelöst, wenn ein HIGH-Pegel anliegt
- Pin als Ausgang konfigurieren: entsprechendes Bit in **DDRB**-Register auf 1
- Verfrühte Aktivierung des Airbags verhindern; entsprechendes Bit in **PORTB**-Register auf 0

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>

// Grenzwert der Verzoegerung
#define DECELERATION_THRESHOLD (0x1AD3)

// Verzoegerungssensor
#define SENSOR_LOW       (*((volatile uint8_t *)0xa0))
#define SENSOR_HIGH      (*((volatile uint8_t *)0xb0))
#define SENSOR_RESET     (*((volatile uint8_t *)0xc0))

/* Bibliotheksfunktionen */
void registerWatchdog(void (*func_ptr) (void));

/* Funktionsdeklarationen, globale Variablen, etc. */
static void init(void);
static void error(void);

/* Unterbrechungsbehandlungsfunktion / Watchdog-Handler */
```

A:

```
/* Funktion main */

   /*  Initialisierung und lokale Variablen */

  /* Hauptschleife */

      /*  Warten auf Ereignisse */
```

B:


```
   /*  Ereignis "neuer Sensorwert" bearbeiten */

   /*  Watchdog-Ereignis bearbeiten */

/* Ende main */
```

E:

```
/* Funktion error */
```

```
/* Ende error */
```

```
/* Initialisierungsfunktion */
```

FI:

Aufgabe 3: Module (6 Punkte)

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

a) Nennen Sie drei Gründe warum man Software in Module gliedert? (3 Punkte)

b) Nennen und beschreiben Sie zwei Schritte, die eine in der Programmiersprache C geschriebene Anwendung (bestehend aus mehreren Modulen) in ein ausführbares Programm überführen? (3 Punkte)

Aufgabe 4: Speicher (10 Punkte)

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

a) Das folgende Programm wird auf einem 8-Bit AVR/Atmega32 Mikrocontroller ausgeführt.
Ergänzen Sie in der untenstehenden Tabelle die Eigenschaften der genannten Variablen und Ausdrücke. (8 Punkte)

```
volatile uint16_t a;
char *b = (char *)0x1234;
const char c = 'm';
static uint16_t d = 0;

static uint8_t next() {
    static uint8_t s = 3;
    return s++;
}

void main(void){
    char *p = malloc(132);
    uint8_t x = next();

    b = &c;

    while(1){
        // wait forever
    }
}
```

Variable	Sichtbarkeit	Lebensdauer	Speichersegment	Speicherbedarf in Bytes
a	Programm		.bss	
b	Programm	Programm		
c		Programm	.rodata	1
d		Programm		2
s			.data	1
p	Block	Block		
***p** (deref.)	---	malloc --> free		
x				1

b) Warum sollte man die Sichtbarkeit und Lebensdauer von Variablen so weit wie möglich einschränken? (2 Punkte)