Aufgabe 1: (10 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) Gegeben ist folgender Programmcode: 2 Punkte

```
int32_t x[] = {-1, 7, -3, 5};
int32_t *y = &x[3];
y -= 2;
```

Welchen Wert liefert die Dereferenzierung von `y` (also `*y`) nach der Ausführung des Programmcodes?

- ☐ Zur Laufzeit tritt ein Fehler auf.
- ☐ 3
- ☐ -5
- ☐ 7

b) Folgende Makrodefinition findet sich in der AVR-Bibliothek: 2 Punkte

```
#define PINA (*(volatile uint8_t *)0x39)
```

Welche der folgenden Aussagen bezüglich der Verwendung des `volatile`-Schlüsselworts ist in diesem Fall richtig?

- ☐ Das `volatile`-Schlüsselwort stellt hier sicher, dass der Zugriff auf `PINA` mit Interrupts synchronisiert wird.
- ☐ Das `volatile`-Schlüsselwort ermöglicht den sicheren Zugriff auf einzelne Bits des Registers.
- ☐ Auf der AVR-Plattform werden Hardware-Register (wie `PINA`) im RAM eingeblendet. Das `volatile`-Schlüsselwort zeigt an, dass es sich dabei um flüchtigen Speicher handelt.
- ☐ Wird der Port A als Eingang konfiguriert, könnte sich der Wert von `PINA` jederzeit ändern. Durch volatile wird der Compiler angewiesen, stets den aktuellen Wert aus `PINA` zu lesen.

c) 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(2, ADD(3, 4))`

- ☐ -10
- ☐ 5
- ☐ -3
- ☐ 6

d) Gegeben sei folgende Enumeration:

2 Punkte

```
enum SPRACHE {Deutsch, Englisch, Russisch};
```

Welche Aussage ist richtig:

- ☐ Der Wert von `Russisch` ist unbekannt; der Compiler weist zur Übersetzungszeit jedem `enum`-Element einen zufälligen, aber eindeutigen Wert zu.
- ☐ Der Compiler meldet einen Fehler, weil den `enum`-Elementen kein Wert zugewiesen wurde.
- ☐ Der Wert von `Russisch` ist `3`.
- ☐ Der Wert von `Russisch` ist `2`.

e) Wozu dient das **`#ifdef`**-Konstrukt in C?

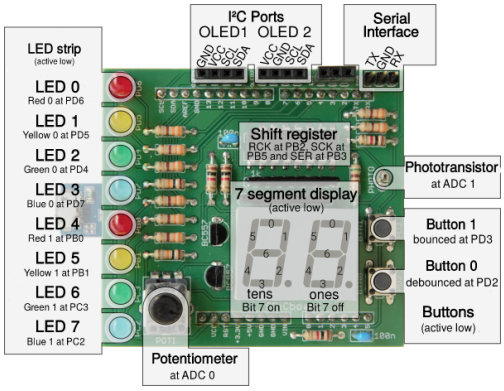
2 Punkte

- ☐ Es kann eingesetzt werden, um sicherzustellen, dass ein Programmteil ein definiertes Ergebnis liefert.
- ☐ Man kann damit Programmteile bei der Übersetzung ausblenden.
- ☐ Es überprüft, ob die danach angegebenen Variablen definiert wurden.
- ☐ Es kann alternativ zu `if`-Abfragen eingesetzt werden.

Aufgabe 2: Boardtest (30 Punkte)

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

Implementieren Sie das Testprogram für das SPiCBoard. Zum Testen der einzelnen Funktionalitäten soll das Programm mittels `BUTTON0` zwischen drei verschiedenen Testmodi (siehe `enum Mode`) wechseln können: Im Modus `TEST_POTI` soll durch Drehen des Potentiometers die Anzahl der leuchtenden LEDs bestimmt werden. Im Modus `TEST_PHOTO` soll eine Anzahl von LEDs proportional zur Helligkeit leuchten. Im Modus `TEST_SEG` soll auf der Siebensegmentanzeige wiederholt von 99 bis 0 runtergezählt werden.



Das Programm soll im Detail 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 mit dem Taster verbunden. Eine fallende Flanke tritt dann auf, wenn der Taster gedrückt wird und eine steigende Flanke dann, wenn dieser losgelassen wird. Sie dürfen davon ausgehen, dass der Taster initial nicht gedrückt gehalten wird.
- Wenn der Taster gedrückt wurde, soll die dafür registrierte `ISR(INT0_vect)` das entsprechende Ereignis per modullokaler Variable an die `main`-Funktion weitergeben. Dabei soll der nächste Testmodus entsprechend der Reihenfolge im `enum` ausgewählt werden. Nach dem letzten Testmodus wird wieder mit dem ersten Modus begonnen. Bei jedem Modusübergang soll sämtlicher testbezogener Zustand zurückgesetzt werden: Siebensegmentanzeige deaktivieren mittels `sb_7seg_disable(void)`, LEDs ausschalten mit Hilfe von `sb_led_setMask(uint8_t)`, ...
- Für die Zeittaktung soll ein 8-Bit-Timer verwendet werden. Konfigurieren Sie den sparsamsten Prescaler und lösen Sie einmal pro Millisekunde ein Event aus. Auf der nächsten Seite finden Sie Details hierzu.
- Wenn der konfigurierte Zählwert des Timers erreicht wird, soll in `ISR(TIMER0_OVF_vect)` ebenfalls nur das Ereignis per modullokaler Variable signalisiert werden. In der `main`-Funktion soll dann der interne Zähler der vergangenen Millisekunden inkrementiert werden.
- Für die Testmodi `TEST_POTI` und `TEST_PHOTO` soll das jeweilige ADC-Gerät (`POTI` bzw. `PHOTO`) mittels `sb_adc_read` ausgelesen werden. Vom gelesenen ganzzahligen 10-bit-Wert sollen anschließend die 8 höchstwertigen Bits als vorzeichenloser 8-bit-Wert interpretiert werden. Mittels der bereitgestellten Funktion `sb_led_showLevel` soll die diesem Wert entsprechende Anzahl an LEDs zum Leuchten gebracht werden.
- Für den Testmodus `TEST_SEG` soll wiederholt von 99 bis 0 auf der Siebensegmentanzeige runtergezählt werden. Zwischen den einzelnen Aufrufen von `sb_7seg_showNumber` soll jeweils $500ms$ gewartet werden. Verwenden Sie hierfür einen internen Zählerwert, der jede Millisekunde inkrementiert wird.
- Benutzen Sie keine Button/Timer Funktionalität aus der libspicboard (`button.h`/`timer.h`).
- Benutzen Sie keine Gleitkommazahlen oder sonstige Mathematikbibliotheksfunktionen.
- 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

- Fallende Flanke: Taster wird gedrückt
- Steigende Flanke: Taster wird losgelassen
- 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**

Konfiguration der externen Interruptquelle **INT0** (Bits im Register **EICRA**)

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

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

- Es soll die Überlaufunterbrechung verwendet werden (ISR-Vektor-Makro: **TIMER0_OVF_vect**)
- Der ressourcenschonendste Vorteiler (*prescaler*) ist 64, wodurch es bei dem 16 MHz CPU-Takt (hinreichend genau) 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/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <stdint.h>
#include <adc.h>
#include <led.h>
#include <7seg.h>

enum Mode { TEST_POTI = 0, TEST_PHOTO = 1, TEST_SEG = 2 };

// Allows the array of LEDs to be used as a (fill) level, progress, or
// similar indicator. The 8 LEDs are used to display the current level
// as a ratio of the maximum value (<=255) in 9 steps.
//
// level: the current level to be shown
// max: the maximum possible value for level
extern int8_t sb_led_showLevel(uint8_t level, uint8_t max);

// Function Declarations, Global Variables, etc.
```

```
// End Function Declarations, Global Variables, etc.

// Interrupt Service Routines
// ISRs **must** only set an event variable
```

```
// End Interrupt Service Routines
```

D:

```
// Function main

    // Initialization and Local Varibables

    // Event Loop
```

```
// Processing of Button Event
```

```
// Processing of Timer Event
```

```
        // Test required Mode
```

```
// End main
```

M:

```
// Initialization Function
```

```
// End Initialization Function
```

I:

Aufgabe 3: Nebenläufigkeit (8 Punkte)

a) Betrachten Sie folgenden Codeausschnitt. Beschreiben Sie das Lost-Update-Problem anhand des Beispiels! (4 Punkte)

```
1  static volatile uint8_t counter = 0;
2  ISR(INT0_vect) {
3      counter++;
4  }
5
6  void main(void) {
7      while(1) {
8          if(counter > 0) {
9              counter--;
10             // Process key press
11             ...
12         }
13         ...
14     }
15 }
```

Sie finden die nächste Teilaufgabe auf der nachfolgenden Seite!

b) Betrachten Sie folgenden Codeausschnitt. Beschreiben Sie das Lost-Wakeup-Problem und seine Auswirkungen anhand des Beispiels! (3 Punkte)

```
1  ISR(TIMER1_COMPA_vect) {
2      event = 1;
3  }
4
5  void main(void) {
6      sleep_enable();
7      event = 0;
8      while(!event) {
9          sleep_cpu();
10     }
11     sleep_disable();
```

---

c) Beschreiben Sie, wie das Lost-Update-/Lost-Wakeup-Problem in den Beispielen gelöst werden kann! (1 Punkt)

Aufgabe 4: Speicherorganisation (12 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. (6 Punkte)

```
static const char *text = "C i5 c00l";
const uint8_t BUFFER_SIZE = 3;

static volatile uint8_t move_text;

static void move_text_timer_callback(void) {
    move_text = 1;
}

void main(void) {
    sei();
    static uint8_t time = 400;

    sb_timer_setAlarm(
        move_text_timer_callback,
        time, time
    );

    const char *text_start = text;
    move_text = 1;

    while(42) {
        if(move_text){
            move_text = 0;
            if((*text_start) == '\0') {
                text_start = text;
            }

            char buffer[BUFFER_SIZE];
            buffer[0] = text_start[0];
            buffer[1] = text_start[1];
            buffer[2] = '\0';

            text_start++;
        }
        ...
    }
}
```

Variable	Sichtbarkeit	Lebensdauer	Speichersegment	Speicherbedarf in Bytes
text	Modul		.data	
BUFFER_SIZE	Programm	Programm		
move_text		Programm	.bss	1
time				1
text_start			Stack	2
buffer	Block	Block		

b) Wann geschieht die statische und wann die dynamische Allokation? Ordnen Sie die genannten Speichersektionen entsprechend zu. Welcher Zusammenhang besteht dabei bezüglich der Lebensdauer der Variablen? (6 Punkte)