

Aufgabe 1: (14 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 beim Zugriff auf I/O-Register unter dem Begriff *Memory-mapped*? 2 Punkte

- Der Zugriff auf die Register erfolgt mit speziellen mmap-Instruktionen des Prozessors.
- Die Register sind nicht real, sondern nur virtuell im Hauptspeicher vorhanden (sog. "virtual devices").
- Beim Zugriff auf spezielle Speicherbereiche des Hauptspeichers werden die Inhalte der Hauptspeicherzellen automatisch in Geräteregister umkopiert.
- Die Register sind in den Adressraum des Prozessors eingeblendet und der Zugriff erfolgt mit den normalen Speicherzugriffsinstruktionen.

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

- 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.
- Das Schlüsselwort *volatile* hat nur noch eine historische Bedeutung und ist in heutigen Programmen nicht mehr nötig.

c) Was versteht man unter Nebenläufigkeit? 2 Punkte

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

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

- Pegelgesteuerte Interrupts werden bei jedem Wechsel des Pegels ausgelöst.
- Interrupts sind eine Besonderheit von AVR-Mikroprozessoren. Auf anderen Architekturen müssen externe Ereignisse durch Pollen abgefragt werden.
- Wurde gerade ein Flanken-gesteuerter Interrupt ausgelöst, so muss erst ein Pegelwechsel der Interruptleitung stattfinden, damit erneut ein Interrupt ausgelöst werden kann.
- Pegelgesteuerte Interrupts müssen durch Pollen des Pegels abgefragt werden.

e) Was bewirken folgende Programmanweisungen? 2 Punkte

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

Welche Aussage ist richtig:

- Alle Bits der Variable ändern ihren Wert.
- Die Variable hat nach der Operation den Wert 0.
- Alle Bitwerte werden um eine Stelle nach links verschoben.
- Die Variable hat nach der Operation den Wert 255.

f) Gegeben ist folgendes Makro: 2 Punkte

```
#define SQ(x) (x * x)
```

Wie ist das Ergebnis des folgenden Ausdrucks

```
2 * SQ(1 - 3)
```

- 8
- 10
- 2
- 4

g) Gegeben ist folgender Ausdruck: 2 Punkte

```
if ((foo > 42) && (foo = 23)) {
    func();
}
```

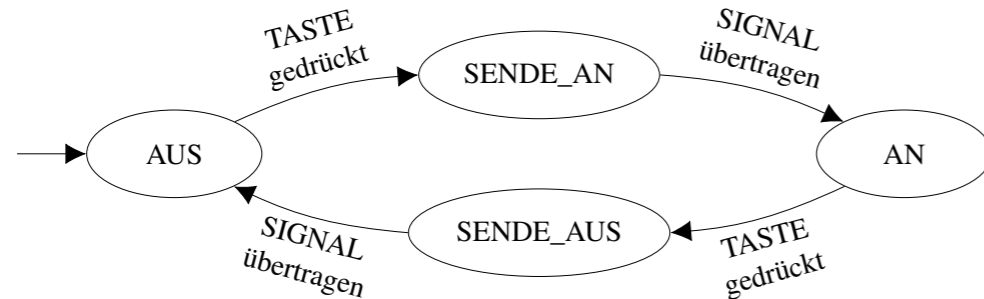
Welche Aussage ist richtig?

- Der Compiler meldet einen Fehler, weil dieser Ausdruck nicht zulässig ist.
- Falls foo den Wert 23 enthält, wird *func()* aufgerufen.
- Falls foo den Wert 65 enthält, wird *func()* aufgerufen.
- Die Bedingung wird niemals wahr, somit wird *func()* im gegebenen Ausdruck nicht aufgerufen.

Aufgabe 2: Funksteckdose (30 Punkte)

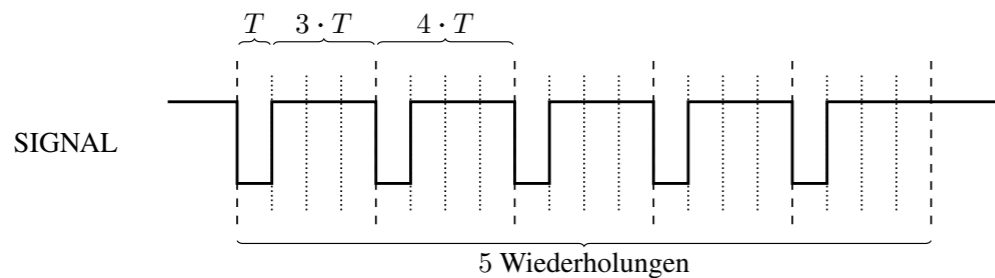
Sie dürfen diese Seite zur besseren Übersicht bei der Programmierung heraustrennen!

Übersicht: Schreiben Sie ein Programm für den AVR-Mikrocontroller, welches die batteriebetriebene Fernbedienung einer Funksteckdose implementiert: Auf Knopfdruck soll der Zustand der Steckdose durch Ausgabe eines Signals umgeschaltet werden. Das Signal soll unter Zuhilfenahme eines Zeitgebers erzeugt werden. Bis zum Ende der Signalübertragung sollen weitere Knopfdrücke ignoriert werden. Die folgende Zustandsmaschine soll den Kern Ihrer Implementierung bilden:



Signal: Das Signal zum Umschalten ist abschnittsweise konstant. Der Abstand zwischen zwei Pegeln beträgt ganzzahlige Vielfache von $T = 16\mu s$.

Wie die nachstehende Grafik illustriert, besteht der zeitliche Verlauf des Signals zum Anschalten (SENDE_AN) aus fünf Wiederholungen der Sequenz low, high, high, high, wobei der Ruhepegel high ist:



Das Signal zum Ausschalten (SENDE_AUS) besteht entsprechend aus fünf Wiederholungen der Sequenz low, low, low, high.

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.
- Implementieren Sie in der Funktion `main()` die oben stehende Zustandsmaschine.
- Verwenden Sie eine externe Interruptquelle zur Erkennung der Tastendrucke.
- Erzeugen Sie die Signale unter Zuhilfenahme eines Zeitgebers. Konfigurieren Sie diesen so, dass er alle $T = 16\mu s$ einen Interrupt auslöst.
- Stellen Sie sicher, dass sich der Mikrocontroller möglichst oft im Schlafmodus befindet um die Batterie der Fernbedienung zu schonen. Deaktivieren Sie hierfür insbesondere in den Zuständen AUS und AN die Überlaufinterrupts des Zeitgebers und in den Zuständen SENDE_AN und SENDE_AUS die Interrupts der Taste.

Hinweis: Wir empfehlen die Implementierung der Zustandsmaschine mit `if-else` statt mit `switch-case` Anweisungen zu realisieren.

Information über die Hardware

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

Taste: Interruptleitung an **PORTD**, Pin 2

- active-low: Wird die Taste 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**

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

Sendeleitung: Ausgang an **PORTD**, Pin 6

- Pin als Ausgang konfigurieren: Entsprechendes Bit im **DDRD**-Register auf 1
- Ruhepegel high: Entsprechendes Bit im **PORTD**-Register auf 1

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

- Es soll die Überlaufunterbrechung verwendet werden (ISR-Vektor-Makro: **TIMER0_OVF_vect**)
- Der ressourcenschonendste Vorteiler (*prescaler*) ist 1, wodurch es bei dem 16 MHz CPU-Takt alle $16\mu s$ zum Überlauf des 8-bit-Zählers **TCNT0** kommt. Somit entspricht jeder Überlauf genau einer Zeiteinheit T
- 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/interrupt.h>
#include <avr/io.h>
#include <avr/sleep.h>
#include <stdint.h>
```

```
enum Zustand {
  AUS      = 0,
  SENDE_AUS = 1,
  AN       = 2,
  SENDE_AN  = 3
};
```

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

```
// Unterbrechungsbehandlungsfunktionen
```

```
// Ende Unterbrechungsbehandlungsfunktionen
```

 A:

```
// Funktion main
```

```
// Initialisierung und lokale Variablen
```

```
// Hauptschleife
```

```
// Warten auf Ereignisse
```

```
// Tastendruck verarbeiten
```

// Überlauf verarbeiten

// Ende main

M:

// Initialisierungsfunktion

// Ende Initialisierungsfunktion

I:

Aufgabe 3: Speicherorganisation (7 Punkte)

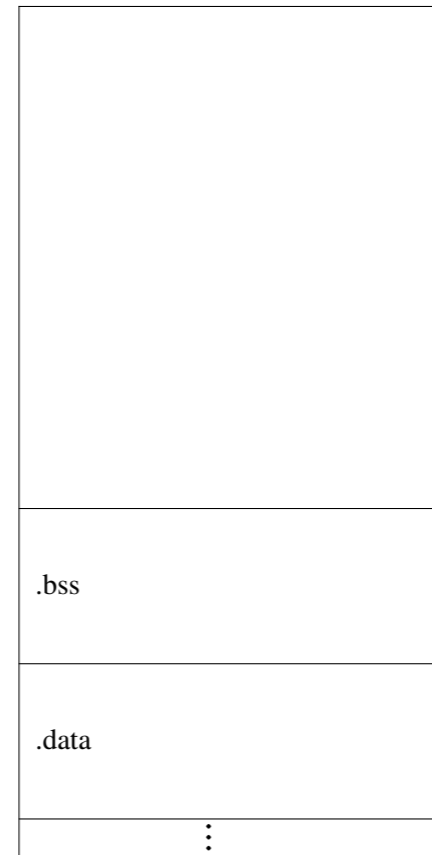
Das folgende Programm wird ohne Optimierungen übersetzt und auf einem 8-Bit AVR Mikrocontroller ausgeführt.

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

```

1 #include <avr/io.h>
2
3 #define pins 4
4
5 uint8_t pin[pins] = { PD4, PD5, PD6, PD7 };
6 static uint16_t count = 1;
7
8 uint16_t toggle(uint8_t x) {
9     static uint8_t init[pins] = { 0 };
10    if (init[x] == 0) {
11        DDRD |= (1 << pin[x]);
12        init[x] = 1;
13    }
14    PORTD ^= (1 << pin[x]);
15    return count++;
16 }
17
18 uint16_t toggle_all(void) {
19     uint16_t max;
20     for(uint8_t i = 0 ; i < pins ; i++) {
21         max = toggle(i);
22     }
23     return max;
24 }
    
```

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) Wie muss der RAM-Speicher beim Start des Mikrocontrollers vorbereitet werden, ehe die eigentliche Programmausführung in `main()` beginnen kann? (2 Punkte)

Aufgabe 4: C-Module (9 Punkte)

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

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

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

c) Welche Bedeutung kommt der Header-Datei (.h) bei C-Modulen zu und was beinhaltet sie? (2 Punkte)

d) Welche besondere Bedeutung hat das Schlüsselwort `static` in C-Modulen? (1 Punkt)

