Prof. Dr.-Ing. Ingo Elsen,
B.Sc. Hanna Babilon, M.Sc. Tim Höner, M.Eng. Marcel Stüttgen

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

ARBKVS, Modulnr.: 53107

Praktikum

Liebe Studierende,

dieses Dokument enthält alle Praktikumsaufgaben. Es gibt während des Semesters Meilensteine, zu denen bestimmte Aufgaben erledigt sein müssen. Die Aufgabe ist dann gelöst, wenn die vollständige und richtige Lösung fristgerecht in Ilias im Abgabeordner für die entsprechende Aufgabe eingereicht wurde **und** sie einen Testattermin für die Aufgabe innerhalb der Frist erfolgreich absolviert haben. Ist die Aufgabe zu dem Meilensteintermin nicht fertig, gilt das Praktikum als nicht erfolgreich absolviert und ich kann leider kein Gesamttestat erteilen!
Es werden nur Lösungen akzeptiert, die folgendem Namesschema genügen:

<Name>_<MatrNr>_ARBKVS_<Aufgabe>.<endung>

Bitte reichen Sie den Code als einzelne Datei ein (Endung „asm“, „c“, „cpp“), oder, wenn Sie mehrere Dateien einreichen, als ZIP-Datei (Endung „zip“).

Viel Erfolg und freundliche Grüße
Ingo Elsen

Teil 1
ARBKVS Testate

Name:	Vorname:	Matr.-Nr.:

Tabelle 1: Meilensteine im Praktikum

Aufgabe	Meilenstein[1]		Datum	Teiltestat
Aufgabe 1	A:15.11.2021 C:02.11.2021 E:03.11.2021 G:04.11.2021	B:25.10.2021 D:26.10.2021 F:27.10.2021 H:28.10.2021		
Aufgabe 2	A:29.11.2021 C:16.11.2021 E:17.11.2021 G:18.11.2021	B:08.11.2021 D:09.11.2021 F:10.11.2021 H:11.11.2021		
Aufgabe 3	A:13.12.2021 C:30.11.2021 E:01.12.2021 G:02.12.2021	B:22.11.2021 D:23.11.2021 F:24.11.2021 H:25.11.2021		
Aufgabe 4	A:10.01.2022 C:14.12.2021 E:15.12.2021 G:16.12.2021	B:06.12.2021 D:07.12.2021 F:08.12.2021 H:09.12.2021		
Aufgabe 5	A:17.01.2022 C:11.01.2022 E:12.01.2022 G:13.01.2021	B:03.01.2022 D:04.01.2021 F:05.01.2022 H:06.01.2021		
Gesamttestat				

Hinweise zu den Testaten:

1. Der Testatbogen ist der einzige Nachweis Ihrer Teilnahme am Praktikum und ist sorgfältig aufzubewahren. Es ist empfehlenswert diesen Testatbogen auch nach erfolgreichem Abschluss des Praktikums aufzubewahren. Er kann, z.B. bei Verlust des Praktikumspasses, zur Neuausstellung als Nachweis herangezogen werden.

[1] je nach Gruppe. Meilenstein bedeutet, dass zu diesem Datum der Versuch testiert sein muss.

Sofern das Praktikum als Online-Praktikum durchgeführt wird, wird der Testatbogen bei der nächsten Anwesenheitsveranstaltung unterschrieben. Beim Online-Praktikum führen die Betreuer zusätzliche Testatlisten.

2. Der Versuch ist von jedem aufzubauen und einzeln zu programmieren. Gruppenlösungen sind nicht zulässig.

3. Zur erfolgreichen Teilnahme an einem Praktikumsversuch gehören die Vorbereitung zum Versuch, praktische Durchführung während des Praktikumstermins, sowie das nachzuweisende Verständnis für die durchgeführten Versuche. Anderenfalls kann eine aktive Mitarbeit bei der Versuchsdurchführung nicht bescheinigt und damit kein Testat erteilt werden. Die erfolgreiche Teilnahme wird am Ende des Versuchstermins durch den Betreuer testiert. Beim Online-Praktikum gilt ein Versuch als erfolgreich absolviert, wenn eine richtige Lösung vollständig, fristgerecht und richtig eingereicht wird und die Lösung einem der das Praktikum Betreuenden vorgestellt und von der Betreuung akzeptiert wurde. Von anderen kopierte Lösungen führen sofort zum Nichtbestehen des gesamten Praktikums!

4. Zur Anerkennung des Praktikums und zur Erteilung des Gesamttestates im Praktikumspass (bitte zum letzten Termin mitbringen) müssen alle Versuche testiert sein.

5. Das Praktikum muss in dem laufenden Semester abgeschlossen werden. Eine Verteilung einzelner Versuche über mehrere Semester ist nicht möglich.

6. Zur sinnvollen Durchführung des Praktikums ist eine Vorbereitung mit Hilfe der Versuchsanleitung und eventuell weiterer Literatur erforderlich. In ILIAS findet sich bereits Material zum Praktikum.

Teil 2
Sicherheitshinweise und Rückgabe der Boards

1. In den Praktikumsräumen herrschen Laborbedingungen. Es wird dazu besonders vor Berührung spannungsführender Teile gewarnt.

2. Im Gefahrenfall sofort den NOT-AUS-Schalter betätigen und den Betreuer umgehend verständigen. Erst nach Freigabe durch den Betreuer darf die Spannung wieder eingeschaltet werden.

3. Die Geräte und Versuchsanordnungen bitte sorgfältig behandeln. Bitte beachten Sie, dass das Praktikumsboard empfindlich gegenüber statischer Elektrizität ist. Bitte berühren Sie einen geerdeten Gegenstand (z.B. einen Heizkörper) an einer nicht-isolierten Stelle, bevor Sie mit dem Board arbeiten.

4. Beschädigte Boards bitte bei einer das Praktikum betreuenden Person melden.

5. Bei der Rückgabe des Boards in der Bibliothek bitte vorher das Blinkprogramm aus der Anleitung aufspielen (steht in Ilias unter „Praktikum→Datenblätter etc.→test.zip“ (LED Blinkprogramm in Assembler). Das Bibliothekspersonal testet so, ob das Board funktioniert.

Teil 3
Aufgaben

Aufgabe 1 Lauflicht in Assembler

Steuern Sie die zehn LEDs (D0-D9) auf dem Microcontrollerboard so an, dass sich ein Lauflicht ergibt. Implementieren Sie das Lauflicht so, dass es am Ende seine Bewegungsrichtung ändert. Als Takt soll zunächst ca. 1/5 Sekunde Leuchtdauer für jede LED genommen werden. Der Arduino ist mit 16MHz getaktet.

a) Verkabeln Sie das Board so, dass DO an PD0 liegt, D1 an PD1 usw. So ergeben sich die kürzestmöglichen, kreuzungsfreien Verbindungen. Achten Sie auch darauf, dass zwischen dem Anschluss PD1 und PD2 noch zwei weitere Anschlüsse ($\overline{\texttt{Reset}}$ und `GND`) liegen, die Sie nicht verkabeln dürfen! Für die LEDs D8 und D9 nutzen Sie am besten die Pins PB0 und PB1.

b) Verkabeln Sie die LEDs über Steckbrücken, so genannte Jumper-Wires!

c) Legen Sie im Atmel-Studio ein entsprechendes Assembler-Projekt an!

d) Programmieren Sie die Lichtsteuerung und legen die Verzögerungsschleife als Unterprogramm (Aufruf mit RCALL) an. Kommentieren Sie Ihren Assembler-Code so, dass ein Fremder ihn verstehen kann, d.h. nicht jede Zeile einzeln, sonder sinnvolle funktionale Blöcke, wie z.B. für die Laufrichtung nach rechts und dann für die Laufrichtung nach links!

e) Starten Sie das Programm im Simulator, setzen Sie einen Breakpoint (Haltepunkt für der Debugger) und zeigen Sie den Betreuern wie das Lauflicht funktioniert!

f) Bringen Sie das Programm auf dem Controller zum Laufen. Übertragen Sie dazu das Programm auf den Controller.

Ziele:

- Die Entwicklungsumgebung bedienen können.
- Ein Assemblerprogramm schreiben und debuggen können.
- Den Simulator bedienen können.
- Die ATMega Microcontroller-Architektur kennen lernen.
- Ein Programm auf einen Microcontroller übertragen können.

Aufgabe 2 Interrupts — Blinker mit Steuerung über Tasten

In diesem Versuch steht es Ihnen frei, ob Sie mit C oder Assembler programmieren. Hinweis: Sollten Sie in C programmieren müssen sie spezielle ISR (Interrupt Service Routinen Schreiben). Wie das geht finden Sie in der Dokumentation zum AVR GCC.
Bisher haben Sie alle Versuche ohne Nutzung von Interrupts realisieren können. Entwickeln Sie ein Programm, das einen Blinker realisiert. Hierzu sollen D0 und D9 genutzt werden. Über die beiden Taster können die LEDs gesteuert werden. Taster SW1 schaltet LED D0 zwischen Dauerlicht und blinken um, Taster SW2 macht dasselbe für LED D9. Dabei soll es nicht möglich sein, dass beide LEDs gleichzeitig blinken und/oder eingeschaltet sind. Blinkt also beispielsweise die D0 und Taster SW2 wird gedrückt, wird D0 ausgeschaltet und D9 fängt an zu blinken.
Hinweis: Hierzu müssen Sie die entsprechende Include–Datei `<avr/interrupt.h>` in C einbinden.

a) Zeichnen Sie einen UML Zustandsautomaten der Funktion und zeigen Sie diesen den Betreuern vor!

b) Ermitteln Sie, an welche Anschlüsse Sie die Taster anschließen können, so dass Sie diese über Interrupts abfragen können!

c) Schließen Sie zwei LEDs und zwei Taster an!

d) Legen Sie ein Projekt entsprechend der von Ihnen gewählten Programmiersprache an und realisieren Sie die Aufgabe über Polling, also die dauernde Abfrage der Taster!

e) Legen Sie ein Projekt entsprechend der von Ihnen gewählten Programmiersprache an und realisieren Sie die Aufgabe unter Nutzung von Interrupts für die Eingabe!

f) (Falls Sie in C programmieren): Wie bildet der Compiler die ISR auf die Assembler Interrupt Vektoren ab? Hinweis: Sie können dafür den Disassembler von AtmelStudio nutzen, oder sich die `.lst`-Datei anschauen.

g) Welche Vor- und Nachteile hat die Implementierung über Interrupts?

Hinweis: Sie können für die Zeitverzögerung beim Blinken die Funktion `_delay_ms()` nutzen. Hier ein Beispiel für ein einfaches Blinkprogramm:

```
1  #include <avr/io.h>
2  #define F_CPU 16000000UL // hier muss die passende Taktfrequenz des Microcontrolle
3  #include <util/delay.h>
4
5  int main( void )
6  {
7      DDRB |= (1 << PB0);
8
9      while(1) {
10         PORTB ^= (1 << PB0);
11         _delay_ms(1000);
12     }
13     return 0;
14 }
```

Ziele:

- Funktionsweise Interrupts verstehen.
- Interrupts als Mittel für nebenläufige Programmierung nutzen können.
- Verstehen, wie ein Betriebssystem mit Eigenheiten der Hardware umgehen muss.
- Verstehen, wie ein BS Hardware-Funktionalitäten abbildet.

Aufgabe 3 Multitasking — Zeitbasis

Implementierungssprache für diese Aufgabe ist C.

Schreiben Sie ein Programm, dass eine Zeitmessung auf dem AVR realisiert. Gehen Sie dazu folgendermaßen vor:

a) Nutzen Sie einen Timer des AVR um eine Taktrate von einer Millisekunde (oder einer guten Annäherung) zur erzeugen!

b) Mit diesem Takt soll ein monotoner Zähler realisiert werden, der bei Programmstart losläuft und die Millisekunden seit Programmstart zählt.

c) Erstellen Sie zwei Funktionen

- `waitFor(uint32_t ms)` Wartet unter Nutzung des Millisekundenzählers für `ms` millisekunden.
- `waitUntil(uint32_t ms)` Wartet bis der Millisekundenzähler den Wert `ms` erreicht hat.

d) Schreiben Sie ein Testprogramm, in dem Sie eine oder mehrere LEDs unter Nutzung dieser Funktionen blinken lassen!

Hinweis: Damit der Zähler richtig aktualisiert werden kann müssen Sie das Schlüsselwort `volatile` der Variablendefinition voran stellen:
`volatile uint32_t systemClock = 0;`

Beantworten Sie folgende Fragen:

a) Wozu dient `volatile` und was kann passieren, wenn Sie das nicht verwenden?

b) Wie lange dauert es, bis der Zähler überläuft?

c) Wie vermeiden Sie Fehler bei der Zeitberechnung in `waitFor(uint32_t ms)`, wenn der Zähler überläuft? Hinweis: Überlegen Sie sich die verschiedenen Ergebnisse von mathematischen Operationen mit `unsigned` Datentypen.

d) Was müssten Sie im Programm ändern, um größere Zeiten zu verarbeiten?

Ziele:

- Einfache Scheduling Algorithmen verstehen und anwenden können.
- Eine komplexe Lösung Schritt für Schritt entwickeln
- Komplexen Code testen können.
- Verständnis für die verschiedenen Datentypen einer Rechnerarchitektur entwickeln.

Aufgabe 4 Ein-Ausgabe, Multiplexing mit Zeitbasis, Tastensteuerung über Interrupts

Implementierungssprache für diese Aufgabe ist C.

Schreiben Sie ein Programm, mit dem Sie die Sieben-Segment-Anzeigen ansteuern, so dass mit SW2 von 00 bis 99 gezählt wird (pro Tastendruck einmal aufwärts zählen) und mit SW1 von 99 bis 0 gezählt wird (pro Tastendruck einmal abwärts zählen).
Hinweise zur Lösung:

- Teilen Sie Ihr Programm in mehrere Module auf, die in separaten .c und .h Dateien liegen. Vorschlag:
  - `main.c` Hauptprogramm, dass die verschiedenen Unterprogramme aufruft
  - `sevenseg.c`, `sevenseg.h` Ansteuerung der Sieben-Segment-Anzeigen.
  - `keys.c`, `keys.h` Ansteuerung/Auswertung der Tasten
- Für die Verkabelung der Anzeige nutzen Sie am Besten PD0-PD6 für die Segmente a-g (den Dezimalpunkt DP verkabeln Sie nicht), sowie PB0 für das Signal A (unterhalb der Leiste für die Sieben-Segment-Anzeige). Mit dem Signal A schalten Sie zwischen der Zehner- und Einerstelle um. Das müssen Sie so schnell machen, dass für das Auge der Eindruck entsteht, dass beide Ziffern gleichzeitig dargestellt werden. Das bedeutet, Sie müssen mindestens 50 mal pro Sekunde umschalten. Hierzu sollten Sie einen Timer verwenden.
- Die Sieben-Segment-Anzeigen werden active-Low angesteuert, d.h. eine logische 0 im Port-Register schaltet das betreffende Segment ein, eine logische 1 das Segment aus.
- Testen Sie die Komponenten erst einmal einzeln, also nur die Tasteneingabe, oder nur die Anzeige und zum Schluss erst das Zusammenspiel aller Komponenten.

Ziele:

- Eine komplexe Lösung Schritt für Schritt entwickeln
- Komplexen Code testen können.
- Ein-Ausgabe-Verfahren realisieren können.
- Einen Eindruck von der Komplexität von Ein-Ausgabe-Systemen bekommen.

Aufgabe 5 Multitasking — Nutzung von Pre-Emptive-Multitasking und geteilten Resourcen

Implementierungssprache für diese Aufgabe ist C++. Implementierungsumgebung ist der PC, als Entwicklungsumgebung können Sie z.B. VisualStudio C++ benutzen!

Schreiben Sie ein Programm das drei Threads hat. Sie können hierfür die Funktionalitäten aus C++11 nutzen, d.h. insbesondere die Containerklassen und Funktionen der Standardbibliothek:

```
1 #include <iostream>
2 #include <thread>
3 #include <mutex>
4
5 // ...
```

T1 Thread T1 schreibt alle Kleinbuchstaben des Alphabets auf `cout`!

T2 Thread T2 schreibt alle natürlichen Zahlen von $0 \ldots 32$ auf `cout`!

T3 Thread T3 schreibt alle Großbuchstaben des Alphabets auf `cout`!

a) Entwickeln Sie ein Programm, dass die obige Ausgabe so realisiert, dass alle drei Threads unsynchronisert laufen! Wie sieht die Ausgabe aus?

b) Synchronisieren Sie die Ausgabe mittels Mutex!

c) Schreiben Sie eine Klasse `Semaphore` die ein Semaphor realisiert! Wie müssen Sie das Semaphor initialisieren, damit sie es statt des Mutexes nutzen können? Demonstrieren Sie die Lösung im Programm! Hinweis: Beachten Sie dazu die Hinweise aus der Vorlesung, mit welchen Klassen das Semaphor umzusetzen ist!

Ziele:

- Einfache Scheduling Algorithmen verstehen und anwenden können.
- Die Notwendigkeit von Thread-Synchronisierung bei Nutzung geteilter Betriebsmittel verstehen!
- Synchronisationsprimitive verstehen und anwenden können.
- Eine komplexe Lösung Schritt für Schritt entwickeln.
- Komplexen Code testen können.

Name:	Vorname:	Matr.-Nr.:

Name:	Vorname:	Matr.-Nr.:

Name:	Vorname:	Matr.-Nr.:

Name:	Vorname:	Matr.-Nr.:

Name:	Vorname:	Matr.-Nr.:

Name:	Vorname:	Matr.-Nr.: