

---

# SPiC-Aufgabe #5: Ampel

(15 Punkte, keine Gruppen)

Entwerfen Sie die Steuerung für eine Ampelanlage an einem Fußgängerüberweg in einer Datei `ampel.c`. Die Steuerung soll im Normalzustand grün für den Autoverkehr und rot für den Fußgängerüberweg anzeigen. Möchte ein Fußgänger die Straße überqueren, kann er eine Umschaltung per Tastendruck anfordern. Nach der Umschaltungsanforderung wird die Autoampel schrittweise auf rot geschaltet und die Fußgängerampel auf grün. Der Fußgänger hat eine gewisse Zeit, um die Straße zu überqueren, bevor die Schaltung wieder schrittweise in den Normalzustand zurückkehrt und der Autoverkehr wieder freigegeben wird, bis der nächsten Fußgänger die Straße überqueren möchte.

Die Aufgabe ist in zwei Teilaufgaben aufgeteilt: Teilaufgabe a) implementiert die Ampelsteuerung und Teilaufgabe b) führt einen Fehlerzustand der Ampelanlage ein. Ist die Ampelanlage im Fehlerzustand und der Fehlergrund fällt wieder weg, wird, ohne Gefährdung von Verkehrsteilnehmern, wieder in den Normalbetrieb umgeschaltet. Im Folgenden werden die Teilaufgaben im Detail beschrieben.

## Teilaufgabe a: Ampelsteuerung (11 Punkte)

Die den Autos zugewandte Ampel wird durch die LEDs `RED0`, `YELLOW0` und `GREEN0` dargestellt. Die Fußgängerampel wird durch die LEDs `RED1` und `GREEN1` (kein gelbes Licht) dargestellt. Durch das Drücken von `BUTTON0` können Fußgänger eine Umschaltung anfordern. Die LED `BLUE1` signalisiert den Fußgängern, dass eine Umschaltanforderung entgegengenommen wurde.

Die Steuerung soll im Detail wie folgt arbeiten:

- Eine Umschaltanforderung wird durch Druck auf `BUTTON0` ausgelöst. Der Druck der Taste wird durch Aktivierung der LED `BLUE1` bestätigt (entspricht „Signal kommt“). Diese LED wird wieder deaktiviert, sobald die Fußgängerampel grünes Licht zeigt. Weitere Tastendrucke an `BUTTON0` werden ignoriert, bis die Fußgängerampel wieder rot zeigt.
- Nach erfolgter Umschaltanforderung im Normalzustand (Autoampel grün, Fußgängerampel rot) zählt die Ampel über die Siebensegmentanzeige 8 Sekunden herunter, welche die Fußgänger noch warten müssen, bis ihre Ampel grün wird; in den übrigen Phasen bleibt die Siebensegmentanzeige aus. Von den insgesamt 8 Sekunden bleibt die Autoampel noch 5 Sekunden grün, dann wechselt sie für 1 Sekunde in den Zustand gelb, bevor sie schließlich rot wird. Erst nach weiteren 2 Sekunden, in denen beide Ampeln rot sind, schaltet die Fußgängerampel auf grün.
- Die Grünphase der Fußgängerampel soll 5 Sekunden andauern, bevor sie wieder auf rot wechselt. Nach einer weiteren Sekunde wechselt die Autofahrerampel für eine Sekunde auf gelb-rot und nach einer weiteren Sekunde wieder auf grün in den Normalzustand.
- Initial sollen beide Ampeln rot anzeigen und die Autoampel soll in der üblichen Geschwindigkeit über rot-gelb auf grün wechseln, während die Fußgängerampel auf rot verweilt (aber bereits Umschaltanforderungen annimmt).
- Eine Umschaltanforderung kann angenommen werden, sobald die Fußgängerampel auf rot steht. Das bedeutet insbesondere, dass bereits Umschaltanforderungen angenommen werden können, bevor die Autoampel auf rot-gelb schaltet (d.h. beide Ampeln rot sind) – während die LED `BLUE1` sofort die erfolgreiche Anforderung signalisiert, beginnt der Zähler (inklusive Anzeige der verbleibenden Wartezeit) jedoch erst nachdem die Autoampel auf grün steht.

Achten Sie darauf, dass der Mikrocontroller in Ruhephasen, in denen keine Berechnungen durchgeführt werden, in den Schlafmodus wechselt. Dies geschieht durch die entsprechenden Funktionen in `avr/sleep.h`.

Achten Sie weiterhin auf die korrekte Verwendung des `volatile`-Schlüsselworts. Beschreiben Sie in einem Kommentar zu jeder verwendeten `volatile`-Variable, weshalb Sie dieses Schlüsselwort dort benötigen.

---

## Teilaufgabe b: Fehlerzustand (4 Punkte)

Die Ampel soll um einen Fehlerzustand (die Autoampel blinkt gelb, während die Fußgängerampel und Siebensegmentanzeige deaktiviert sind) erweitert werden. Als Fehlerindikator soll die externe Schnittstelle EXT verwendet werden. Da dieser jedoch mit dem selben Pin wie BUTTON1 verbunden ist, kann die Fehlerschaltung vollständig mit diesem Taster getestet werden.

- Während eines niedrigen Pegel (PD3 verbunden mit GND, was dem gedrückten BUTTON1 entspricht) soll der Fehlerzustand aktiviert werden.
- Die Blinkwechselgeschwindigkeit ist eine Sekunde. Zwischen den Wechseln soll die CPU wie üblich schlafen.
- Der Fehlerzustand wird beendet, sobald wieder ein hoher Pegel anliegt. Um Unfälle zu vermeiden, soll die Ampelschaltung (wie beim Start der Anwendung) bei rot-rot beginnen und die Autoampel auf grün schalten.
- Der Fall, dass beim Starten der Ampel bereits der Fehlermodus aktiviert ist, darf ignoriert werden.

### Hinweise:

- Verwenden Sie die Module LED und 7SEG der `libspicboard` für die Ausgabe.
- Verwenden Sie *weder* das Modul `Button` *noch* `Timer` der `libspicboard`!
  - Konfigurieren Sie stattdessen direkt die Interruptbehandlung und die Interrupthandler für `BUTTON0` und `BUTTON1`; diese sind angeschlossen an den Pins `PD2` (`BUTTON0`) bzw. `PD3` (`BUTTON1`) und damit an den externen Interruptquellen `INT0` bzw. `INT1` des ATmega-Mikrocontrollers. Bedenken Sie jedoch die ggf. unterschiedliche Interrupterkennungskonfiguration.
  - Für die Zeittaktung soll der `TIMERO` verwendet werden. Es darf die Überlaufunterbrechung `OVF` verwendet werden (Fehler  $\leq 50$  ms sind tolerierbar). Es soll ein möglichst ressourcenschonender Vorteiler (`prescaler`) gewählt werden.
- Bauen Sie ihr Programm so auf, dass in der `main()` Funktion nur an einer zentralen Stelle die Programmlogik zum Schlafen der CPU implementiert ist. Insbesondere soll die Programmlogik zum Schlafen nicht für jeden Zustandsübergang extra implementiert oder aufgerufen werden.
- Im Verzeichnis `/proj/i4spic/pub/aufgabe5/` befindet sich die Datei `ampel.elf`, welche eine Beispielimplementierung enthält.

### Abgabezeitpunkt

T01	16.06.2019	18:00:00
T02	16.06.2019	18:00:00
T03	16.06.2019	18:00:00
T04	16.06.2019	18:00:00
T05	17.06.2019	18:00:00
T06	17.06.2019	18:00:00
T07	17.06.2019	18:00:00
T08	18.06.2019	18:00:00
T09	18.06.2019	18:00:00
T10	18.06.2019	18:00:00
T11	18.06.2019	18:00:00
T12	19.06.2019	18:00:00
T13	19.06.2019	18:00:00
T14	19.06.2019	18:00:00
T15	19.06.2019	18:00:00
T16	19.06.2019	18:00:00
T17	20.06.2019	18:00:00
T18	20.06.2019	18:00:00