SPiC-Aufgabe #3: Geschicklichkeitsspiel

(12 Punkte, keine Gruppen)

Programmieren Sie ein Geschicklichkeitsspiel (Datei `gesch.c`) zum Training der Hand-Augen-Koordination. Ein Spielcursor wandert dabei über die LED-Reihe des SPiCboards. Wird der Taster 0 gedrückt, wird die LED auf die der Cursor im Moment des Tastendrucks zeigt, abhängig von ihrem vorherigen Zustand, an- bzw. ausgeschaltet. Eine bereits leuchtende LED wird wieder ausgeschaltet, eine nicht leuchtende LED wird angeschaltet. Ziel des Spiels ist es, dass alle 8 LEDs leuchten.

1. Zu Beginn des Spiels sind LED0 – LED7 ausgeschaltet.
2. Der aktuell erreichte Level wird, beginnend mit 1, auf der Sieben-Segmentanzeige dargestellt.
3. Der Spielcursor wandert fortlaufend von LED0 zu LED7 und wieder zurück zu LED0. Dazu wird an der aktuellen Cursorposition der Zustand der LED kurzzeitig invertiert (eine *aus*geschaltete LED wird *ein*geschaltet; eine *ein*geschaltete LED wird *aus*geschaltet).
4. Drücken von Taster 0 hält diese Invertierung fest, dass heißt die kurzzeitige Invertierung bleibt dauerhaft bestehen, auch wenn der Cursor weiter wandert.
5. Wenn alle 8 LEDs leuchten, ist das Spiel gewonnen. Es folgt die Siegessequenz, deren Ablauf durch kurzes Warten zwischen den Schritten erkennbar sein muss:
   (a) LED7 – LED0 werden hintereinander ausgeschaltet.
   (b) Der Cursor wandert einmal von LED7 zu LED0 und wieder zurück.
   (c) Die LEDs füllen sich von außen nach innen und werden dort beginnend wieder ausgeschaltet (*leeren* sich wieder).
6. Das Spiel geht in den nächsten Level (die Cursorgeschwindigkeit nimmt dabei zu und nähert sich einer Maximalgeschwindigkeit) und beginnt erneut.

Teilen Sie ihr Programm in mehrere Teilaufgaben auf, um diese Aufgabe zu lösen. Überlegen Sie sich auch einen geeigneten Rückgabewert und geeignete Parameter für Ihre Hilfsfunktionen:

`wait_key()` Schreiben Sie eine Funktion `wait_key()`, welche, abhängig vom Level, eine bestimmte Zeit wartet und einen in dieser Zeit erfolgten Tastendruck (logisch fallende Flanke, d. h. ein Wechsel des `BUTTONSTATE` von `RELEASED` zu `PRESSED`) von `BUTTON0` geeignet zurückgibt. Beachten Sie hierbei, dass die Wartezeit nicht von der Dauer des Tastendrucks abhängen darf.

`show_cursor()` Schreiben Sie eine Funktion `show_cursor()`, welche abhängig vom aktuellen Zustand der LEDs (an bzw. aus) und der Position des Cursors die Ansteuerung der LEDs übernimmt.

`play()` Nutzen Sie die Funktionen `show_cursor()` und `wait_key()` in einer Funktion `play()`, um die eigentliche Spiellogik zu implementieren und rufen Sie diese geeignet aus der `main()` auf.

`show_win()` Implementieren Sie die Siegessequenz in einer Funktion `show_win()` und rufen Sie diese geeignet aus der `main()` auf.

Hinweise

- Wenn es im Programmablauf nötig ist zu warten, implementieren Sie dies durch aktives Warten.
- Verwenden Sie Schleifen und Bitoperationen, um die Bitmuster zur Ansteuerung der LEDs zu erstellen und verwenden Sie *ausschließlich* die Funktion `sb_led_setMask()` zur Ansteuerung der LEDs.
- Verwenden Sie keine globalen Variablen, sondern *ausschließlich* lokale Variablen.
- Die libspicboard verwendet für die Sieben-Segmentanzeige Interrupts, welche jedoch erst später besprochen werden. Binden Sie die Headerdatei `avr/interrupt.h` ein und rufen Sie am Anfang der `main()`-Funktion den Befehl `sei()` auf, um Interrupts zu aktivieren.
- Im Verzeichnis `/proj/i4spic/pub/aufgabe3/` unter Linux bzw. in `P:\aufgabe3\` unter Windows befindet sich die Datei `gesch.elf`, welche eine Beispielimplementierung enthält.
- Ihr Programm muss mit der `Release`-Compiler-Konfiguration kompilieren und funktionieren; Diese Konfiguration wird zur Bewertung herangezogen.

Abgabezeitpunkt

T01	06.05.2018	18:00:00
T02	06.05.2018	18:00:00
T03	06.05.2018	18:00:00
T04	07.05.2018	18:00:00
T05	08.05.2018	18:00:00
T06	08.05.2018	18:00:00
T07	09.05.2018	18:00:00
T08	09.05.2018	18:00:00
T09	09.05.2018	18:00:00