SPiC-Aufgabe #4: spiel

(12 Punkte, keine Gruppen)

Programmieren Sie ein Geschicklichkeitsspiel (Datei `spiel.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). Achten Sie darauf, dass am Anfang und am Ende **nicht** doppelt gewartet wird.
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:
   (a) LED7 – LED0 werden hintereinander ausgeschaltet (anfangend bei LED7).
   (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. Dabei soll die Geschwindigkeit in den ersten Level deutlich schneller steigen als in den späteren Level.

Ihr Programm soll in zwei Hauptteile unterteilt werden, die geeignet aus `main()` aufzurufen sind: `play()` (die Spiellogik) und `show_win()` (Siegessequenz). Überlegen Sie sich geeignete Rückgabewerte und Parameter für diese Funktionen. Sie können zur Kapselung von Funktionen weitere Hilfsfunktionen benutzen.

`play()` Die Funktion `play()` soll die Implementierung eines Levels beinhalten. Die Geschwindigkeit für das Level soll vom Aufrufer der Funktion festgelegt werden können. Wenn der Spieler das Level geschafft hat, soll aus der Funktion zurückgekehrt werden.

`show_win()` Zeigt die Siegessequenz an. Der Ablauf der Siegessequenz soll durch kurzes Warten zwischen den Schritten erkennbar sein.

Es soll an allen Stellen, wo gewartet werden muss, ausschließlich **passives** Warten genutzt werden. Nutzen Sie hierfür die `libspicboard` (siehe Hinweise).

Hinweise

- Nutzen Sie die `libspicboard` sowohl zur Ansteuerung der Siebensegmentanzeige (`sb_7seg_showNumber()`) als auch zum Warten (`sb_timer_delay()`).
- Verwenden Sie Schleifen und Bitoperationen, um die Bitmuster für die Ansteuerung der LEDs zu erstellen und verwenden Sie *ausschließlich* die Funktion `sb_led_setMask()` zur Ansteuerung der LEDs.
- Verwenden Sie wo möglich lokale Variablen und nur wo benötigt globale Variablen mit geeigneter Sichtbarkeit.
- Die Verwendung des `Button`-Moduls der `libspicboard` ist **nicht** zulässig!
  - Konfigurieren Sie direkt die Interruptbehandlung für `BUTTON0`. Dieser ist an Pin `PD2` und damit an der externen Interruptquellen `INT0` des ATmega-Mikrocontrollers angeschlossen.
  - Ein für das Spiel relevanter Tastendruck wird durch eine fallende Flanke signalisiert.

  - Mehrfaches Drücken während der selben Cursorposition muss nicht berücksichtigt werden.
  - Die Unterbrechungsbehandlungsroutine soll möglichst kurz sein.

- Begründen Sie die Verwendung von allen `volatile` Variablen. Wenn für mehrere Variablen die selbe Begründung gilt, dürfen Sie diese gemeinsam begründen.
- Im Verzeichnis `/proj/i4spic/pub/aufgabe4/` befindet sich die Datei `spiel.elf`, welche eine Beispielimplementierung enthält.

Abgabezeitpunkt

alle Gruppen 14.06.2020 18:00:00 Uhr