Übung zum C/C++-Praktikum Fachgebiet Echtzeitsysteme

TECHNISCHE UNIVERSITÄT DARMSTADT

Übungen für den 5. Tag

Musterlösungs-/Microcontroller-Projekte in CodeLite importieren

Die angebotenen Musterlösungs-/Microcontroller-Projekte basieren auf Makefiles. Daher ist es wichtig, dass du sie entsprechend importierst: **Workspace -> Add an existing project** und dann zur entsprechenden `.project`-Datei navigieren.

Einführung

Von der Toolseite aus unterscheidet sich die Entwicklung für den Mikrocontroller kaum von den bisherigen Übungen. Der Aufruf der Toolchain (Compiler, Linker und Flashprogramm) für die Embedded-Entwicklung ist in einem Makefile integriert, das in CodeLite durch den Buildprozess automatisch aufgerufen wird.
Verwende für jede Aufgabe das zugehörige Vorlageprojekt aus dem Übungsrepository (`https://github.com/Echtzeitsysteme/tud-cpp-exercises`), da diese sowohl das Makefile, als auch notwendige Bibliotheken im Ordner `uc_includes` zur Verwendung des Boards enthalten.

Hinweise

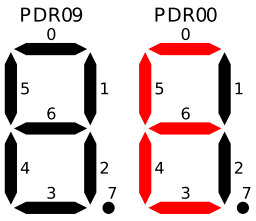
- Sollte der Flashvorgang fehlschlagen, kann es helfen den Schiebeschalter auf dem Board auf **PROG** zu schalten. Nach dem Flashvorgang setzt man dann den Schiebeschalter auf **RUN** und startet das Programm mit Hilfe der blauen Resettaste.
- Pool-PCs: Das USB-Kabel muss am Hub in der Buchse stecken, die mit *Data* beschriftet ist.
- Variablen dürfen in dieser Übung im Gegensatz zu C++ nur am Anfang eines Anweisungsblocks deklariert werden. Erst ab dem C99 Standard (welche hier nicht verwendet wird) dürfen Deklarationen an beliebigen Stellen erfolgen.

Aufgabe 0 Test

Teste das Zusammenspiel von CodeLite, Compiler und Flashvorgang auf dem Mikrocontroller, indem du das vorgegebene Projekt zu dieser Aufgabe baust. Auf der Siebensegmentanzeige des Boards sollte im Erfolgsfall eine 42 erscheinen.

Aufgabe 1 Siebensegmentanzeige

Implementiere ein Programm, das die Zahlen 0 bis 99 auf der Siebensegmentanzeige ausgibt. Nach jeder Ausgabe einer Zahl soll eine Pause eingelegt werden.



Für die Ausgabe der Zahlen 0 bis 9 steht dir das Array `DEC7SEG` zur Verfügung, welches die benötigten Werte für die jeweiligen Ziffern enthält. Die beiden Anzeigen sind an den Ports 09 und 00 angeschlossen. Die Ansteuerung erfolgt logisch invertiert, das bedeutet, dass ein Segment bei einer logischen 0 leuchtet und umgekehrt. Beispiel: Ein „E“ kann durch das Setzen der Pins 0, 3, 4, 5 und 6 (Binär: 10000110, bzw. Hexadezimal: 0x86) auf low gezeichnet werden.

Die Pause kann durch eine Schleife produziert werden, die in jedem Zyklus den Befehl `__wait_nop()` aufruft. Eine Konstante `DELAY` steht dir für die Anzahl der Schleifendurchläufe zur Verfügung. Achte insbesondere darauf, dass du für den Zähler den Datentyp `long` verwendest.

```
long d;              // always define variables at the beginning of a block!
for (d = 0; d < DELAY; ++d) {
   __wait_nop();     // wait one cycle
}
```

Aufgabe 2 Buttons

Implementiere einen Counter der bei 00 startet und bis 99 zählen kann. Bei Druck auf die rechte Taste soll der Wert erhöht, bei Druck auf die linke Taste verringert werden. Nutze die Siebensegmentanzeige aus der vorherigen Aufgabe zur Anzeige des aktuellen Wertes.
Wird 99 angezeigt und die rechte Taste gedrückt, soll der Counter auf 00 gesetzt werden. Umgekehrt gilt: Falls der Counter 00 anzeigt und die linke Taste gedrückt wird, soll er auf 99 umspringen.
Der linke Taster ist an Port 07 Pin 0, der rechte an Port 07 Pin 1 angeschlossen. Bei gedrücktem Taster liegt ein Low-Pegel am Eingang, sonst ein High-Pegel. Du kannst den Zustand eines Pins wie folgt abfragen:

```
char leftButton, rightButton;
leftButton = PDR07_P0;
rightButton = PDR07_P1;
```

Hinweise

- Ein Button ist üblicherweise für mehrere tausend CPU-Zyklen gedrückt.
- Ein Tastendruck ist durch den Übergang von *high* auf *low* definiert. Da in jedem Zyklus nur der aktuelle Wert abgefragt werden kann, musst du den aktuellen Wert mit einem gespeicherten Wert aus dem vorigen Durchlauf vergleichen.
- Die Moduloberechnung von negativen Zahlen kann problematisch sein. Hier hilft es, wenn du vor der Berechnung den Divisor hinzuaddierst.

Aufgabe 3 Hilfsfunktionen

Implementiere wie in Aufgabe 1 einen Zähler von 0 bis 99, jedoch unter Verwendung von eigens dafür geschriebenen Hilfsfunktionen:

```
void wait(long w)          // wait for w cycles
void setLeft7Seg(int i)    // set left display to the given number i (if 0 <= i <= 9)
void setRight7Seg(int i)   // set right display to the given number i (if 0 <= i <= 9)
void set7Seg(int i)        // set the seven-segment display to the given number
```

Aufgabe 4 A/D-Wandler

Schreibe ein Programm, das den Spannungswert von *AN1* (linker Schieberegler) auf der linken Siebensegmentanzeige und den Spannungswert von *AN2* (rechter Schieberegler) auf der rechten Siebensegmentanzeige ausgibt. Skaliere dazu den resultierenden Wertebereich (0 bis 255) auf 0 bis 9.
Mache dich mit folgenden Funktionen vertraut und verwende sie zur Initialisierung und anschließender Verwendung des A/D-Wandlers:

```
void initADC(void)            // initialize the ADCs
int getADCValue(int channel)  // read the value from channel 1 or 2
```

Übungen für den 5. Tag

Hinweise

- Einige Funktionen aus Aufgabe 3 kannst du hier wiederverwenden.
- Der Wert für `ADSR` besteht aus 16 Bits (0110 11xx xxxy $yyyy_b$), wobei xxxxx für den Startkanal der Konvertierung und yyyyy für den Endkanal steht. Für unsere Zwecke nehmen diese beiden 5 Bit Blöcke immer entweder $00001_b = 1$ (AN1) oder $00010_b = 2$ (AN2) an. Daher lässt sich der `ADSR` auch wie folgt über einen binären Shift und zwei Addition bestimmen:

$$\texttt{ADSR} = 0110110000000000_b + (xxxxx_b << 5) + yyyyy_b = 0x6c00 + (x << 5) + y$$

Einführung LCD

Die Ansteuerung des LCD erfolgt über Pins mit festgelegten Funktionen. Zur Vereinfachung wurden bereits im Projekt für diese Aufgabe Definitionen vorgegeben, sodass die Pins über einfache Namen angesteuert werden können. Die Namen entsprechen den Pins, wie sie im Datenblatt (`https://github.com/Echtzeitsysteme/tud-cpp-exercises/blob/master/doc/Display_AV128641YFBY-WSV.pdf`) ab Seite 10 zu finden sind.

Name	Funktion	Pin/Port
LCD_PORT_DB	Datenbus (DB0 - DB7)	P01
LCD_PIN_DI	Data (1) / Instruction (0)	P02_0
LCD_PIN_RW	Read (1) / Write (0)	P02_1
LCD_PIN_E	Enable	P02_2
LCD_PIN_CS1	Linker Chip (1 = aktiv)	P02_3
LCD_PIN_CS2	Rechter Chip (1 = aktiv)	P02_4
LCD_PIN_RESET	Reset-Signal (0 = aktiv)	P02_5

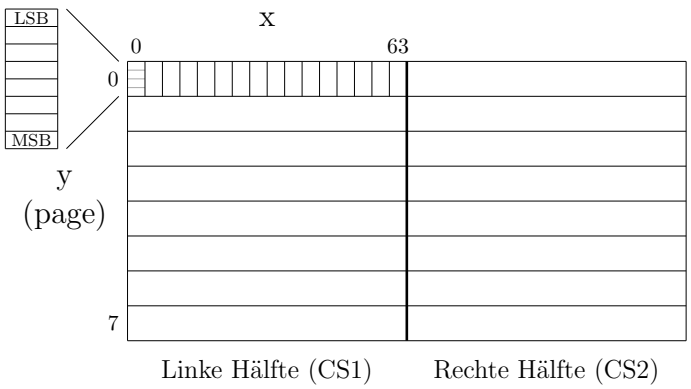
Display einschalten

Das Display muss vor der Benutzung eingeschaltet werden:

```
// Display on
LCD_PORT_DB = 0x3F;
// Display off
LCD_PORT_DB = 0x3E;
```

Displayhälften (Chip-Select Signal)

Das Display ist logisch aufgeteilt in zwei Hälften zu je 64 x 64 Pixel (siehe untenstehende Grafik, MSB/LSB = Most/Least Significant Bit).



Welche Hälfte einen Befehl verarbeiten soll, wird ausgewählt, indem deren Chip-Select-Signal auf 1 gesetzt wird, das andere auf 0:

```
// select left display part (CS1)
LCD_PIN_CS1 = 1;
LCD_PIN_CS2 = 0;
// select right display part (CS2)
LCD_PIN_CS1 = 0;
LCD_PIN_CS2 = 1;
```

Befehle senden

Ein Befehl an das Display besteht grundsätzlich aus zwei Phasen:

a) Setzen der entsprechenden Pins (meist `LCD_PIN_DI`, `LCD_PORT_DB`). Eventuell selektieren der Displayhälfte über `LCD_PIN_CS1`, `LCD_PIN_CS2`.

b) Senden des `enable`-Signals. Setze dazu den Enable-Pin auf 1, warte kurz, setze den Pin wieder auf 0 und warte erneut kurz. Verwende als Warteintervall die Konstante `LCD_T`.

   Es empfiehlt sich, für dieses Signal eine eigene Funktion zu schreiben (`void lcd_sendEnable(void)`).

Auf das Display zeichnen

Das Vorgehen zum Zeichnen sieht in etwa wie folgt aus:

a) Setzen der x-Adresse (Zeile)

b) Setzen der y-Adresse (Spalte)

c) Senden von 8 Bit Daten (eine „Zelle“ von 8 Pixeln Höhe)

Aufgabe 5 LCD Basics

Implementiere ein Programm, dass die linke Seite des Displays ansteuert und alle Zellen mit dem Wert AA_h (10101010_b) belegt, was einem sehr feinen Streifenmuster entspricht.
Der `x`-Zähler wird nach dem Senden von Daten zum Display automatisch hochgezählt. Somit ist es sinnvoll, beim Zeichnen zuerst über die Seiten (`y`) zu iterieren und innerhalb dieser Schleife über die Spalten (`x`):

```
char x, y;
for (y = 0; y < 8; ++y) {
   // LCD: set y address to 'y', set x address to 0
   for (x = 0; x < 64; ++x) {
       // LCD: draw pixel -> column/x address is incremented automatically
   }
}
```

Hinweise

- Denke daran, `LCD_PIN_E` zu Beginn des Programms mit 0 zu initialisieren.
- Das Reset-Signal (`LCD_PIN_RESET`) muss immer 1 (deaktiviert) sein.

Aufgabe 6 LCD

Nun soll das Display wie ein Bildschirm angesteuert werden können. Dazu wirst du einen Framebuffer verwenden, der den Bildschirminhalt repräsentiert. In jeder Iteration wird der Framebuffer zunächst vollständig vorbereitet und dann am Stück übertragen.
Die Vorlage enthält bereits ein Array `lcd_buffer`, das als Framebuffer fungieren soll.
Implementiere folgende Funktionen:

```
void lcd_clear();     // clear the frame buffer, setting all pixels to 0
void lcd_drawPixel(int x, int y, int black);    // set (black == 1) or clear (black == 0) a pixel
    in the frame buffer, do nothing, if any input parameter is invalid
void lcd_flush();    // display the frame buffer on the LCD
```

Übungen für den 5. Tag

Hinweise

- Ein Testprogramm steht bereits zur Verfügung, welches ein Schachbrettmuster auf dem Display ausgibt – wenn deine Funktionen komplett und korrekt implementiert sind.
- Achte auch hier darauf, dass das Display zunächst per Befehl eingeschaltet werden muss (*Display on*).
- Für die Funktion `lcd_drawPixel` werden die bitweisen Operationen AND und OR sowie der Shift-Operator benötigt. Eine Skizze kann hier sinnvoll sein, um sich vorzustellen, wie die bitweisen Operationen arbeiten.
- Wenn du später bewegte Dinge visualisierst, kann es sein, dass das Display flimmert. Hier hilft es, entweder die Anzeige nur zu verändern, wenn sich tatsächlich etwas bewegt oder das Display vor dem Schreiben des Framebuffers aus- (*Display off*) und danach wieder einzuschalten (*Display on*).