Übung zum C/C++-Praktikum Fachgebiet Echtzeitsysteme

TECHNISCHE UNIVERSITÄT DARMSTADT

Übungen für den 5. Tag

Aufgabe 1 Umgang mit den Mikrocontroller-Bords

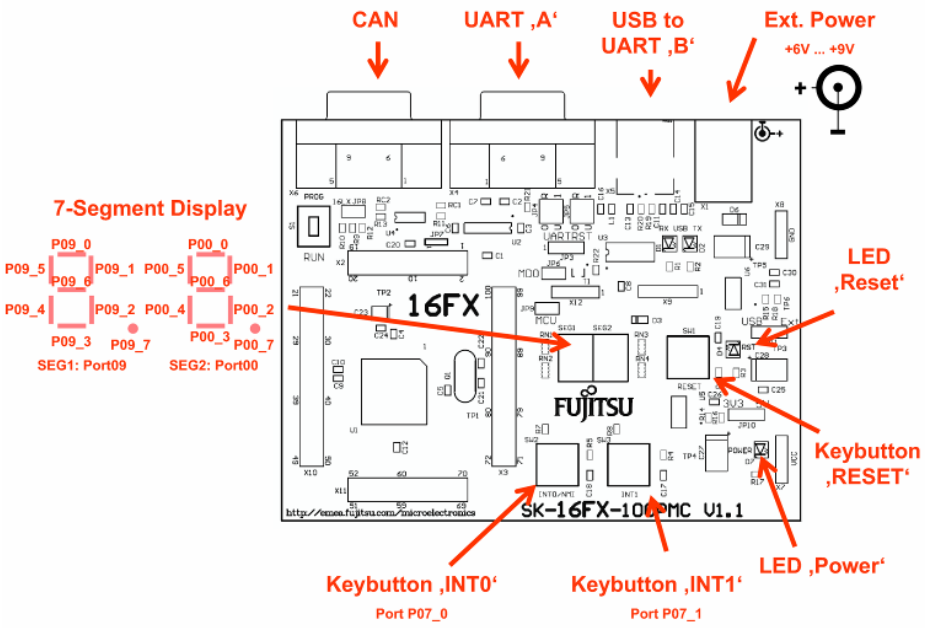
Die Entwicklung für den Mikrocontroller unterscheidet sich kaum von den bisherigen Übungen, da die Toolchain mit in Eclipse integriert ist. Der Programmcode wird in Eclipse in den Projekten, die für die einzelnen Aufgaben vorgesehen sind, geschrieben. Beim Starten des Build-Vorgangs wird zusätzlich zu den eigentlichen Kompilierungsschritten automatisch das Programm auf den Mikrocontroller übertragen und der Controller neu gestartet. Soll manuell das Programm neu gestartet werden, ohne ein neues Programm zu übertragen, hilft ein kurzer Druck auf den blauen Reset-Knopf.

Aufgabe 2 Systemtest

Testen Sie das Zusammenspiel von Eclipse / Compiler / Flash auf Mikrocontroller, in dem Sie das vorgegebene Projekt p00_flash bauen. Dies sollte am Schluss automatisch das Programm auf den angeschlossenen Mikrocontroller übertragen. Das Programm sollte auf der Sieben-Segment-Anzeige des Boards eine *42* anzeigen.

Aufgabe 3 Embedded-Entwicklung

Im Folgenden werden einzelne Hardwarekomponenten angesteuert. Dazu ist hier eine Übersicht über die auf dem Starterkit vorhandenen Komponenten gegeben.



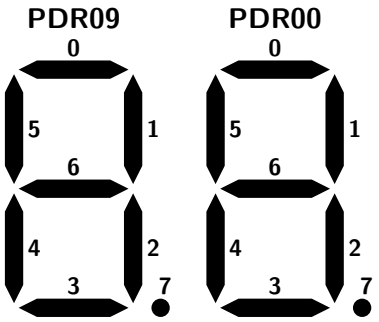
Weitere Informationen, **insbesondere für das Display**, sind im Ordner *Datasheets* im Dokumente-Ordner der Installation zu finden.

Übung zum C/C++-Praktikum - Tag 5

Aufgabe 3.1 p01_7seg

Implementieren Sie ein Programm, das die Zahlen 0 bis 99 auf den Sieben-Segment-Anzeigen ausgibt. Nach jeder Ausgabe einer Zahl soll eine Pause eingelegt werden.
Hinweise:

- Für die Ausgabe der Zahlen 0 bis 9 steht Ihnen ein Array DEC7SEG zur Verfügung, welches die benötigten Were für die Ports für die jeweiligen Ziffern enthält.
- Die Pause kann durch eine Schleife produziert werden, die in jedem Zyklus den Befehl `__wait_nop()` aufruft. Eine Konstante DELAY steht Ihnen zur Verfügung für die Anzahl der Schleifendurchläufe. Achten Sie insbesondere darauf, dass Sie hier den Datentyp long verwenden müssen!
- Die beiden Anzeigen sind an den Ports 09 und 00 angeschlossen. Die Ansteuerung erfolgt logisch invertiert, das bedeutet wenn ein Ausgang für ein Segment logisch 0 ist leuchtet dieses, bei 1 ist es aus.
- Beispiel: ein „E“ kann durch das Setzen der Pins 0, 3, 4, 5 und 6 auf low gezeichnet werden. Binär: 10000110, Hexadezimal: 0x86



Aufgabe 3.2 p02_buttons

Erstellen Sie einen Counter. Das Programm soll zu Beginn den Wert 0 anzeigen. Bei Druck auf die rechte Taste soll der Wert erhöht, bei Druck auf die linke Taste erniedrigt werden. Wird 99 angezeigt und die rechte Taste gedrückt, soll der Counter auf 0 gesetzt werden. Umgekehrt ebenso (counter==0, linke Taste gedrückt ⇒ counter=99).
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 aktuell Wert abgefragt werden kann, müssen Sie den aktuellen Wert mit einem gespeicherten Wert aus dem vorigen Durchlauf vergleichen.
- Der linke Taster ist an Port 07 Pin 0, der rechte an Pin 1 angeschlossen. Bei gedrücktem Taster liegt ein Low-Pegel am Eingang, sonst ein High-Pegel.

Aufgabe 3.3 p03_util

Wie in Aufgabe 3.1 sollen hier die Zahlen 0 bis 99 ausgegeben werden, aber unter der Verwendung von eigens dafür geschriebenen Funktionen:

```
void wait(long w)          // für w Zyklen pausieren
void setLeft7Seg(int i)    // Zahl i auf linker Anzeige ausgeben falls i im gültigen Bereich (0 bis 9)
void setRight7Seg(int i)   // ebenso wie eben, nur für die rechte Anzeige
void set7Seg(int i)        // Zahl (0 bis 99) auf 7-Segment-Anzeigen darstellen
```

Aufgabe 3.4 p04_adc

Schreiben Sie ein Programm, das den Spannungswert von AN1 (linker Schieberegler) auf der linken Sieben-Segment-Anzeige und den Spannungswert von AN2 (rechter Schieberegler) auf der rechten Sieben-Segment-Anzeige ausgibt. Skalieren Sie dazu den resultierenden Wertebereich von 0 bis 255 auf 0 bis 9.
Hinweise:

- Einige Funktionen aus Aufgabe 3.3 können hier eingesetzt werden.
- Achten Sie auf die richtige Initialisierung von *ADER0*.
- 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 (AN1) oder 00010 (AN2) an.
- Lesen Sie die beiden Werte nacheinander aus, verwenden Sie also immer die gleiche Zahl für Start- und Endkanal während einer Konvertierung.

Aufgabe 3.5 p05_lcdbasics

Ziel dieser Aufgabe ist es, die linke Seite des Displays anzusteuern und alle Zellen mit dem Wert aa_h (10101010_b) zu belegen.
Hinweise:

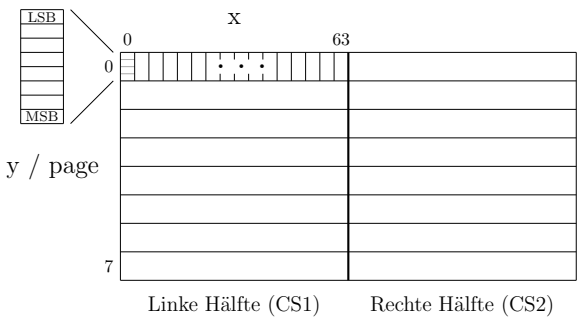
- Für diese Aufgabe benötigen Sie die Dokumentation des Displays (*Display_AV128641YFBY-WSV#.pdf*), dort insbesondere den Abschnitt mit den Befehlen (Setzen der x- und y-Adresse, **Einschalten des Displays**, Senden der Daten).
- Achten Sie darauf, dass Sie nach jedem Befehl (Daten oder Instruktion) das Enable-Signal an das Display senden. Es empfiehlt sich, eine Funktion *void lcd_sendEnable(void)* zu implementieren, die den Enable-Pin auf 1 setzt, kurz wartet und dann wieder auf 0 setzt und erneut kurz wartet. Verwenden Sie als Warteintervall die Konstante LCD_T. Denken Sie daran, das Enable-Pin zu Beginn des Programms mit 0 zu initialisieren.
- Das Display ist logisch aufgeteilt in zwei Hälften zu je 64 x 64 Pixel (siehe untenstehende Graphik). Welche Hälfte einen Befehl verarbeiten soll wird ausgewählt, in dem deren Chip-Select-Signal CS1 bzw. CS2 auf 1 gesetzt wird, das andere auf 0.
- Die Ansteuerung erfolgt über Pins mit festgelegten Funktionen. Zur Vereinfachung wurden bereits im Projekt für diese Aufgabe Definitionen vorgegeben, so dass die Pins über einfache Namen angesteuert werden können. Die Namen entsprechen den Pins, wie sie im Datenblatt ab Seite 10 zu finden sind.

Name	Funktion	Controllerpin/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

- Grundsätzliches Prinzip zum Senden von Befehlen:
  a) Daten am Datenbus bereitstellen: PORT_DB, PIN_DI, PIN_RW wie gewünscht setzen, ein Chip-Select (PIN_CS1 oder PIN_CS2) aktivieren
  b) Enable-Signal schicken (=> high => low)

  Beachten Sie: Das Reset-Signal muss immer 1 (deaktiviert) sein.
- Vorgehen zum Schreiben von Bilddaten:
  a) Setzen der y-Adresse (Zeilenblock)
  b) Setzen der x-Adresse (Spalte)
  c) Senden von 8 Bit Daten (eine „Zelle“ von 8 Pixeln Höhe)

  Vereinfachung: Der *x*-Zähler wird nach dem Senden von Daten im Display automatisch hochgezählt. Somit kann *x* zu Beginn jeder Zeile auf 0 gesetzt werden und muss danach für den Rest der Zeile nicht mehr manuell gesetzt werden.



Aufgabe 3.6 p06_lcd

Nun soll das Display wie ein Bildschirm angesteuert werden können. Es gibt einen Buffer *lcd_buffer*, auf dem Pixeloperationen durchgeführt werden sollen. Schreiben Sie folgende Funktionen:

```
void lcd_clear()                          // Löscht den Buffer (setzt alle Werte im Buffer auf 0).
void lcd_drawPixel(int x, int y, int black) // Setzt einen Pixel an Position (x,y) wenn black == 1 ist,
                                          // bei black == 0 wird der Pixel an Position (x,y) gelöscht.
                                          // Keine Operation bei ungültigen Werten für x, y oder black.
void lcd_flush()                          // Gibt den Buffer an das Display aus.
```

Hinweise:

- Ein Testprogramm steht bereits zur Verfügung, welches ein Schachbrettmuster auf dem Display ausgibt – wenn Ihre Funktionen komplett und korrekt implementiert sind.
- Achten Sie auch hier darauf, dass das Display zunächst per Befehl eingeschaltet werden muss.
- Für die Funktion *lcd_drawPixel* werden Bitoperationen benötigt. Eine Skizze kann hier sinnvoll sein, um sich vorzustellen, wie die Bitoperationen arbeiten.