Übung zum C/C++-Praktikum Fachgebiet Echtzeitsysteme

TECHNISCHE UNIVERSITÄT DARMSTADT

Übungen für den 5. Tag

Aufgabe 1 Umgang mit den Mikrocontroller-Boards

Von der Toolseite aus unterscheidet sich die Entwicklung für den Mikrocontroller kaum von den bisherigen Übungen, da die Toolchain mit in Eclipse integriert ist. Der Programmcode wird in Eclipse geschrieben, kompiliert und gelinkt. Danach wird das Programm automatisch auf den Mikrocontroller übertragen (*FLASHly*) und der Controller neu gestartet. Soll das Programm manuell neu gestartet werden, ohne ein neues Programm zu übertragen, hilft ein kurzer Druck auf den blauen Reset-Knopf.

Aufgabe 2 Systemtest

Teste das Zusammenspiel von Eclipse / Compiler / Flash auf Mikrocontroller, in dem du das vorgegebene Projekt *p00_flash* baust. 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.

Wichtig: Das USB-Kabel muss am Hub in der Buchse stecken, die mit *Data* beschriftet ist.

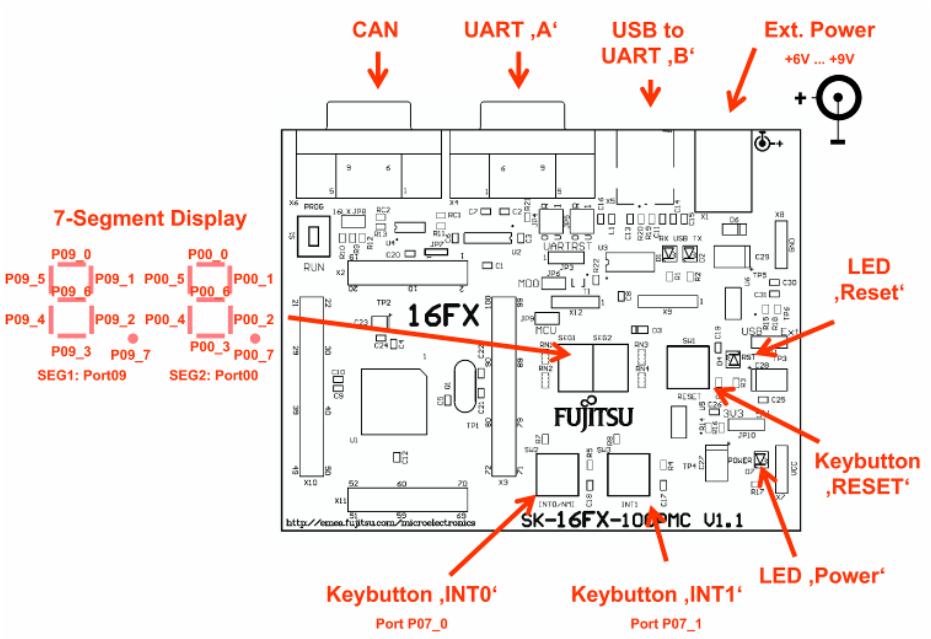
Tipp: Selbst wenn der Build- und Übertragungsprozess erfolgreich war, wirst du zunächst eine Fehlermeldung erhalten. Dieser Fehler verschwindet, wenn du einen Standard-COM-Port einstellst.

Du kannst den Standard-COM-Port in den Projekteigenschaften festlegen. Gehe dazu in den Projekteigenschaften (Rechtsklick auf das Projekt) zu *C/C++-Builder* → *Settings* und wähle den Baumeintrag *FLASHly/General*. Hier kannst du den Standardport in das Feld *COM port* eintragen.

Wichtig: Wenn du das USB-Kabel aus- und wieder einsteckst, wird sich die Nummer des COM-Ports ändern. Du musst diese Änderung in den Settings entsprechend nachziehen.

Aufgabe 3 Embedded-Entwicklung

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



Aufgabe 3.1 p01_7seg

Implementiere ein Programm, das die Zahlen 0 bis 99 auf der 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 Werte 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 dir zur Verfügung für die Anzahl der Schleifendurchläufe. Achte insbesondere darauf, dass du für die Schleifenzählervariable den Datentyp *long* verwendst:

```
int main(void) {
    long d;
    // ...
    for (d = 0; d < DELAY; ++d) {
        __wait_nop();
    }
}
```

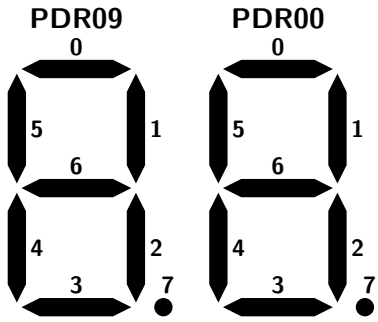
- Die beiden Anzeigen sind an den **Ports 09** und **00** angeschlossen. Die Ansteuerung erfolgt logisch invertiert, das bedeutet, wenn ein Ausgang für ein Segment logisch 0 ist, leuchtet dieses, bei 1 ist es aus.
- Beispiel: ein „E" kann durch das Setzen der Pins 0, 3, 4, 5 und 6 auf low gezeichnet werden.[1]

  Binär: 10000110

  Hexadezimal: 0x86

- Es kann sinnvoll sein, deine Anzeige für hexadezimale Zeichen zu erweitern. Ergänze dazu das Array *DEC7SEG*, sodass die Zahlen 10 bis 15 als `A`,...,`F` dargestellt werden können.

[1] Im Internet gibt es zahlreiche (Online-)Tools zum Umrechnen, z.B. `http://binaer-dezimal-hexadezimal-umrechner.miniwebapps.de/`



Aufgabe 3.2 p02_buttons

Erstelle einen Counter. Das Programm soll zu Beginn den Wert 00 anzeigen. Bei Druck auf die rechte Taste soll der Wert erhöht, bei Druck auf die linke Taste verringert werden. 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.

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.
- Der linke Taster ist an Port 07 Pin 0, der rechte an Port 7 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 stateOfLeftButton = PDR07_P0;
  char stateOfRightButton = PDR07_P1;
  ```

- Die Moduloberechnung von negativen Zahlen kann problematisch werden. Hier hilft es, wenn du vor der Modulooperation den Divisor hinzuaddierst. Beispiel:

  ```
  int value = (value + 42) % 42;
  ```

Aufgabe 3.3 p03_util

Wie in Aufgabe 3.1 sollen hier die Zahlen 0 bis 99 ausgegeben werden, 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 i is between 0 and 9)
void setRight7Seg(int i) // set right display to the given number i (if i is between 0 and 9)
void set7Seg(int i)    // set the seven-segment display to the given number in the range of 0 to 99
```

Aufgabe 3.4 p04_adc

Schreibe ein Programm, das den Spannungswert von *AN1* (linker Schieberegler) auf der linken Sieben-Segment-Anzeige und den Spannungswert von *AN2* (rechter Schieberegler) auf der recht Sieben-Segment-Anzeige ausgibt. Skaliere dazu den resultierenden Wertebereich (0 bis 255) auf 0 bis 9.

Verwende zur Initialisierung des A/D-Wandlers folgenden Code:

```
// Start of function.
unsigned char result;

// Initialisation
ADCS_MD  = 3;   // ADC Stop Mode
ADCS_S10 = 1;   // 8 Bit Precision
ADER0_ADE1 = 1;    // Activate analog inputs AN1 + AN2
ADER0_ADE2 = 1;  //  (ADER0: Inputs AN0 to AN7)
```

Anschließend kannst du mit folgendem Code (hier für den Kanal 1) eine A/D-Wandlung vornehmen:

```
ADSR = 0x6C00 + (3 << 5) + 3;  // Start and end channel is 1

ADCS_STRT = 1;       // Start A/D conversion
while (ADCS_INT == 0) { } // Wait for A/D conversion to finish
result = ADCRL;      // store result (1 Byte)
ADCS_INT = 0;        // Set bit to 0 for next conversion
```

Hinweise:

- Einige Funktionen aus Aufgabe 3.3 kannst du hier eingesetzt werden.
- Achte 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. Daher sieht die Initialisierung von ADSR etwas kryptisch aus:

  ```
  // Start and end channel is 3
  ADSR = 0x6C00 + (3 << 5) + 3;
  ```

- Lies die beiden Werte nacheinander aus, verwende 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 benötigst du die Dokumentation des Displays (*<SVN>/Doku/Display_AV128641YFBY-WSV.pdf*), dort insbesondere den Abschnitt mit den Befehlen: Setzen der x- und y-Adresse, Einschalten des Displays, Senden von Daten.
- Achte darauf, nach jedem Befehl (Daten oder Instruktion) das Enable-Signal an das Display zu 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. Verwende als Warteintervall die Konstante *LCD_T*. Denke 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 Grafik). 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	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

- 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 (z.B. *void lcd_sendEnable(void)*)

  Beachte: Das Reset-Signal muss immer 1 (deaktiviert) sein.

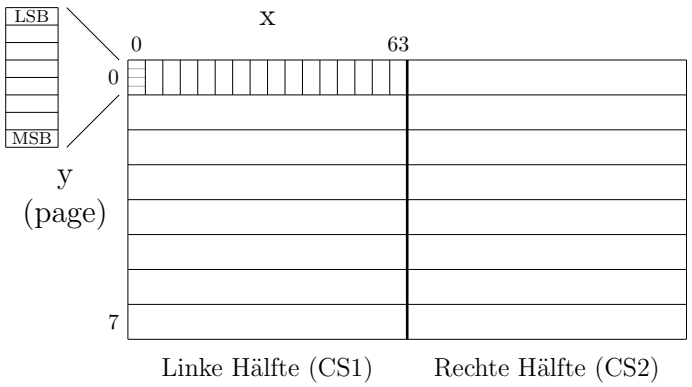
- Vorgehen zum Schreiben von Bilddaten:

  a) Setzen der x-Adresse (Zeilenblock)

  b) Setzen der y-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 *y* 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. Schreibe folgende Funktionen:

```
void lcd_clear()                      // Clear buffer (set all slots to 0)
void lcd_drawPixel(int x, int y, int black) // Sets a pixel at (x,y) if black == 1
                                      // Clears a pixel at (x,y) if black == 0
                                      // Ignore if x,y, or black is invalid
void lcd_flush()                      // Show buffer on display
```

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.
- Für die Funktion *lcd_drawPixel* werden Bitoperationen benötigt. Eine Skizze kann hier sinnvoll sein, um sich vorzustellen, wie die Bitoperationen arbeiten.