

Übung zum C/C++-Praktikum Fachgebiet Echtzeitsysteme



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Übungen für den 5. Tag

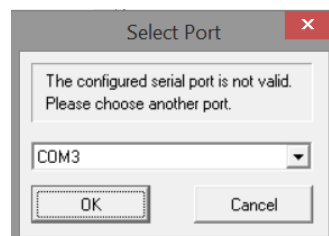
Aufgabe 1 Umgang mit den Mikrocontroller-Boards

Von der Toolseite aus unterscheidet sich die Entwicklung für den Mikrocontroller kaum von den bisherigen Übungen, da die Toolchain für die Embedded-Entwicklung in Eclipse integriert ist. Der Programmcode wird in Eclipse geschrieben, kompiliert und gelinkt. Danach wird das Programm automatisch auf den Mikrocontroller übertragen (via *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, indem du das vorgegebene Projekt *p00_flash* baust.

Du wirst nach dem (emulierten) COM-Port gefragt, an dem der Microcontroller angeschlossen ist:



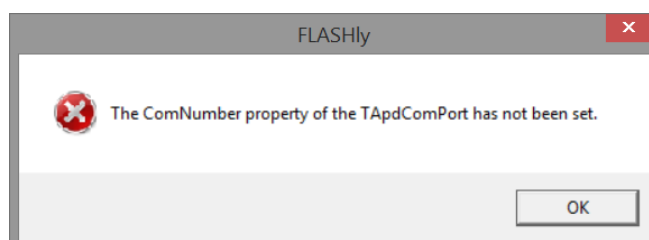
Den entsprechenden Port ermittelst du entweder durch Ausprobieren (hohe Portnummern sind wahrscheinlicher) oder du öffnest den **Geräte-Manager** und suchst unter **Anschlüsse (COM & LPT)** nach dem **USB Serial Port**.

Auf der Sieben-Segment-Anzeige des Boards sollte nun eine 42 erscheinen.

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

Aufgabe 2.1 Standard-COM-Port

Selbst wenn der Build- und Übertragungsprozess erfolgreich war, wirst du zunächst eine Fehlermeldung erhalten:



Übung zum C/C++-Praktikum - Tag 5

Dieser Fehler verschwindet, wenn du einen Standard-COM-Port wie folgt einstellst: Öffne die **Projekteigenschaften** (Rechtsklick auf das Projekt) zu *C/C++-Builder* → *Settings* und wähle den Baumeintrag *FLASHy/General*. Hier kannst du den COM-Port in das Feld *COM port* eintragen.

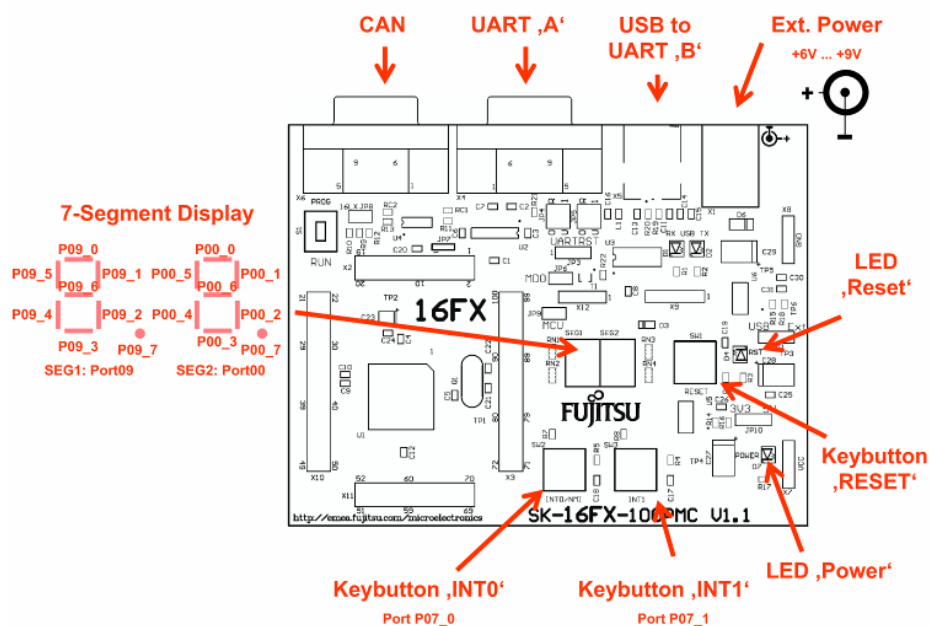
Sehr wichtig: Wenn du das USB-Kabel aus- und wieder einsteckst, kann sich die Nummer des COM-Ports ändern. Du musst diese Änderung in den Einstellungen entsprechend nachziehen.

Aufgabe 2.1.1 Troubleshooting

Manchmal scheitert das Beschreiben des Microcontrollers aus ansonsten unerfindlichen Gründen. In diesem Fall kann es helfen, den Schiebeschalter (bei der DSUB9-Buchse) während des Programmierens auf **PROG** und danach wieder auf **RUN** zu stellen.

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* verwendest:

```
int main(void) {  
    long d;  
    // ...  
}
```

Übung zum C/C++-Praktikum - Tag 5

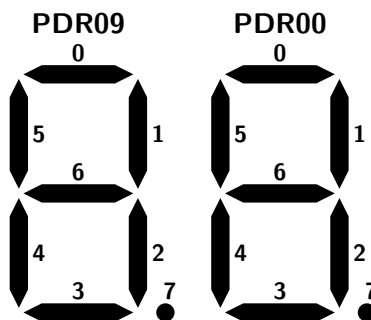
```
for (d = 0; d < DELAY; ++d) {  
    __wait_nop();  
}  
}
```

- 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.



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;  
char stateOfRightButton;  
// ...  
stateOfLeftButton = PDR07_P0;  
stateOfRightButton = PDR07_P1;
```

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

Übung zum C/C++-Praktikum - Tag 5

- Die Modulooperation von negativen Zahlen kann problematisch sein. 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;
unsigned char channelNumber;

// 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:

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

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 einsetzen.
- Achte auf die richtige Aktivierung der analogen Kanäle (ADER0).
- Der Wert für ADSR besteht aus 16 Bits (0110 11xx xxyy yy_b), wobei xxxxx für den Startkanal der Konvertierung und yy_b 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 1
ADSR = 0x6C00 + (1 << 5) + 1;
```

- [illegible]

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, was einem sehr feinen Streifenmuster entspricht.

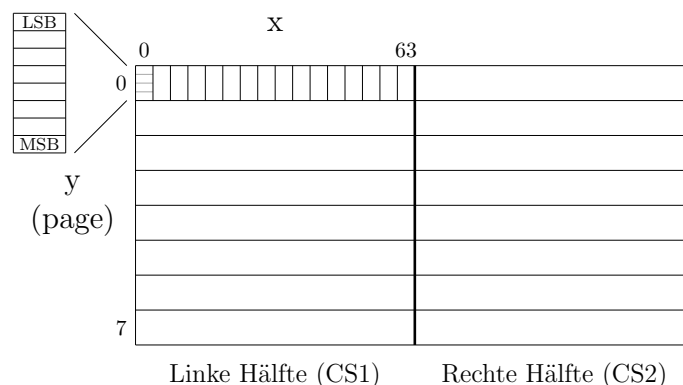
Für diese Aufgabe benötigst du die Dokumentation des Displays ([SVN](https://svn.wsl.ethz.ch/svn/Doku/Display_AV128641YFBS-WSV.pdf)/Doku/Display_AV128641YFBS-WSV.pdf), dort insbesondere den Abschnitt mit den Befehlen: Setzen der x- und y-Adresse, Einschalten des Displays, Senden von Daten.

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 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

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, 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;
```

Übung zum C/C++-Praktikum - Tag 5

Befehle senden

Ein Befehl an das Display besteht grundsätzlich aus zwei Phasen:

- 1.) Setzen der entsprechenden Pins (meist `LCD_PIN_DI`, `LCD_PORT_DB`).

Eventuell selektieren der Displayhälfte über `LCD_PIN_CS1`, `LCD_PIN_CS2`.

- 2.) 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 (`page`)
- b) Setzen der y-Adresse (Spalte)
- c) Senden von 8 Bit Daten (eine „Zelle“ von 8 Pixeln Höhe)

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

Aufgabe 3.6 p06_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 nun folgende Funktionen:

```
/**  
 * Clears the frame buffer, setting all pixels to 0.  
 */  
void lcd_clear();
```

Übung zum C/C++-Praktikum - Tag 5

```
/**
 * Sets (if black == 1) or clears (if black == 0) a pixel in the
 * frame buffer.
 * Does nothing, if any input parameter is invalid.
 */
void lcd_drawPixel(int x, int y, int black);

/**
 * Displays the frame buffer on the LCD
 */
void lcd_flush();
```

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 Bitoperationen AND und OR sowie der Shift-Operator `<<` benötigt. Eine Skizze kann hier sinnvoll sein, um sich vorzustellen, wie die Bitoperationen 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*).