Praktikum Softprocessor SP3: Unterprogramme und Header

7. Juli 2014

Zusammenfassung

Für das im ersten Praktikumsversuch entwickelte eingebetteten Rechnersystem soll eine kleine Bibliothek nützlicher Funktionen entwickelt und getestet werden.

1 Datentypen, Funktionen und Header

1.1 Datentypen

Die Variablen in einem C-Programm haben alle einen Typ, der den Wertebereich und die Wertedarstellung beschreibt und gemeinsam mit dem Namen vor der Nutzung einer Variablen vereinbart werden muss. Der im ersten Praktikumsversuch konfigurierte Prozessor kann nur ganze Zahlen mit und ohne Vorzeichen verarbeiten[1]. Die Typen dafür heißen:

	8 Bit	16 Bit	32 Bit
vorzeichenfrei	**unsigned char**	**unsigned short**	**unsigned int**
vorzeichenbehaftet	**char**	**short**	**int**

Im nachfolgenden Beispiel werden zwei Variablen CtA und CtB als vorzeichenbehaftete 32-Bit- und eine Variable »z« als vorzeichenfreie 8-Bit-Variable vereinbart:

```
int CtA, CtB;
unsigned char z;
```

1.2 Funktionen

Ein größeres Programm wird zur Wahrung der Übersichtlichkeit in Teile gegliedert, die in unterschiedlichen Dateien stehen. Eine erste Untergliederung ist in ein Hauptprogramm, dass mit »Run« oder nach Reset« gestartet wird, und mehrere Unterprogramme, die das Hauptprogramm bei Bedarf aufruft und die Teilaufgaben abarbeiten. Ein Unterprogramm besteht aus einer Schnittstellenbeschreibung, die den Aufrufnahmen und die Übergabeparameter festlegt, und der auszuführenden Anweisungsfolge. In C werden Unterprogramme als Funktionen mit einer beliebigen Anzahl von Aufrufparametern und einem Rückgabewert vereinbart:

```
Typ function funktionsname(Parameter)
{
 {Anweisung;}
 [return Rückgabewert;]
}
```

Die Parameter sind die Eingabevariablen, für die jeweils ein Typ und der lokale Name vereinbart werden. Ihnen werden beim Funktionsaufruf Werte übergeben, aus den die Funktion ein Ergebnis berechnet. Das Ergebnis muss den Datentyp der Funktion haben und wird mit der abschließenden Return-Anweisung zurückgegeben. Die nachfolgende Funktion hat zwei vorzeichenbehaftete 32-Bit-Zahlen als Eingabeparameter, eine vorzeichenbehaftete 32-Bit-Zahl als Rückgabewert und gibt das Vergleichsgebnis der beiden Eingabeparameter zurück:

```
int function Test(int a, b) return a==b;
```

[1] Für die Nutzung von Gleitkommazahlen ist eine Gleitkommarecheneinheit in den Prozessor mit einzubauen.

Das Hauptprogramm ist auch eine Funktion. Der Rückgabetyp ist in der Regel vom Typ »int« und der Rückgabewert der Ausführungsstatus (korrekter Abschluss oder Nummer des aufgetretenen Fehlertyps):

```
int main(Parameter){
 {Anweisung;}
 [return Ausführungsstatus;]
}
```

In einem eingebetteten System enthält das Hauptprogramm eine Endlosschleife und beendet sich nicht. Damit wird auch kein Wert zurückgegeben. Funktionen ohne Rückgabewert werden mit dem Typ »void[2]« vereinbart.

1.3 Header

Größere Programme werden in Übersetzungseinheiten unterteilt. Zusammengehörige Funktionen werden in Quellcodedateien zusammengefasst. Die Definitionen der Funktionsaufrufe aus den Quellcodedateien und auch die Definitionen von projektübergreifend genutzten Konstanten und Makros gehören in die zugehörige Header-Datei, auf die durch Einbindung in den Quelltext von anderen Dateien Bezug genommen werden kann. Die Header-Dateien bilden dadurch die Schnittstelle zwischen den einzelnen Einheiten. In Programmbibliotheken bilden Header-Dateien den einsehbaren Teil der Bibliothek, wohingegen der Rest in Übersetzungseinheiten oft vorübersetzt, also nicht in Form von Quelltext, vorliegt.

In Aufgabe 3.1 sollen Sie eine kleine Bibliothek »utils.c« mit nützlichen Unterprogrammen und Makros für die Zeichenbearbeitung schreiben. Die Header-Datei »utils.h« dafür soll lauten:

```
#ifndef UTILS_H
#define UTILS_H

#include <xparameters.h>  // Hardware-Adressen
#include <xuartlite_l.h>  // Treiber für die serielle Schnittstelle
#include <xgpio_l.h>      // Treiber für die parallele Schnittstelle

// Makro zum Empfang eines Zeichens von der seriellen Schnittstelle
#define getchar() \
              XUartLite_RecvByte(XPAR_UART_BASEADDR)

// Makro zum Versenden eines Zeichens über die serielle Schnittstelle
#define putchar(c) \
          XUartLite_SendByte(XPAR_UART_BASEADDR, c)

// Definitionen der zu exportierenden Funktionen
int isupper(char c);
int islower(char c);
int isalpha(char c);
int isdigit(char c);
int iscntrl(char c);
int isspace(char c);
int isalnum(char c);
int ispunct(char c);
int isprint(char c);
int toupper(char c);
int tolower(char c);

#endif
```

Anweisungen, die mit »#« beginnen, sind für den Präprozessor und werden vor der Programmübersetzung abgearbeitet. Im Beispiel prüft der Präprozessor zu Beginn, ob das Symbol »UTILS_H« definiert ist. Wenn ja, überspringer den Rest des Inhalts der Datei. Wenn nicht, definiert er dieses Symbol und arbeitet alle Zeilen bis »#endif« ab. Die Include-Anweisungen fügen den Inhalt der drei Header-Dateien »xparameters.h« etc. in das zu übersetzende Programm ein. Auch die einzufügenden Header sind in derselben Weise in »#ifndef ... #endif« eingerahmt. So wird verhindert, dass derselbe Header mehrfach eingebunden wird. Die beiden mit »#define« definierten Makros zur Vereinfachung der Funktionsaufrufe für den Empfang und das Versenden eines Zeichen über die serielle Schnittstelle

[2] Schlüsselwort für »kein Typ«

wurden bereits im Praktikumsversuch zuvor eingeführt und verwendet. Die Definitionen der zu exportierenden Funktionen entsprechen den Funktionsdefinitionen nur ohne die in der Funktion auszuführende Anweisungsfolge. In der zugehörigen Quellcodedatei muss jeweils genau dieselbe Aufrufdefinition mit den denselben Typen und Bezeichnern stehen, nur ergänzt um die auszuführende Anweisungsfolge.

Aufgabe 3.1: Zeichenbearbeitungsbibliothek

Entwickeln sie die Quellcodedatei »utils.c« für den Header »utils.h«. Die einzelnen Funktionen sollen folgendes bewirken:

1. isupper: Rückgabe einer »1«, wenn »c« ein Großbuchstabe ist, sonst einer »0«.
2. islower: Rückgabe einer »1«, wenn »c« ein Kleinbuchstabe ist, sonst einer »0«.
3. isalpha: Rückgabe einer »1«, wenn »c« ein Buchstabe ist, sonst einer »0«.
4. isdigit: Rückgabe einer »1«, wenn »c« eine Ziffer ist, sonst einer »0«.
5. iscntrl: Rückgabe einer »1«, wenn »c« ein Steuerzeichen, d.h. ein Zeichen mit einem Wert kleiner 0x20 oder größer 0x7E, ist, sonst Rückgabe einer »0«.
6. isspace: Rückgabe einer »1«, wenn »c« ein Trennzeichen ist, sonst einer »0«. Trennzeichen sind das Leerzeichen, der Tabulator »\t«, Zeilenwechsel »\n«, Wagenrücklauf »\r«, »\f« und »\v«.
7. isalnum: Rückgabe einer »1« für wahr, wenn »c« eine Ziffer oder ein Buchstabe ist, sonst Rückgabe einer »0«.
8. ispunct: Rückgabe einer »1«, wenn »c« ein druckbares Zeichen, aber keine Ziffer und kein Buchstabe ist, sonst Rückgabe einer »0«.
9. isprint: Rückgabe einer »1«, wenn »c« ein druckbares Zeichen, d.h. wenn es kein Steuerzeichen ist, sonst Rückgabe einer »0«.
10. toupper: Wenn das Zeichen ein Kleinbuchstabe ist, soll der zugehörige Großbuchstabe zurückgegeben werden, sonst das Eingabezeichen.
11. tolower: Wenn das Zeichen ein Großbuchstabe ist, soll der zugehörige Kleinbuchstabe zurückgegeben werden, sonst das Eingabezeichen.

Kopieren Sie sich zum Testen der einzelnen Bibliotheksfunktionen die Datei »main.c« in das neue Projekt und nehmen Sie folgende Anpassungen vor:

- Zusätzlich Include-Anweisung für »utils.h«.
- Löschen der Makro-Definitionen in »main.c«, die jetzt in »utils.h« stehen, zur Vermeidung von Mehrfachdefinitionen.
- In der Endlosschleife zwischen dem Empfang und dem Versenden des Zeichens soll der Schalterwert eingelesen und anhand des Zahlenwertes, den die Schalterstellung beschreibt, eines der zu testenden Unterprogramme aufgerufen werden. Beim Test der ersten neun Funktionen soll für den Ergebniswert »0« die Ziffer null und für den Ergebniswert »1« die Ziffer »1« zurückgesendet werden. Beim Test der beiden letzten Funktionen soll nach jedem Aufruf der Ergebniswert zurückgesendet werden.

Testbeispiel:

Schalterstellung:	1 (isupper)	...	10 (toupper)
Eingabe:	`aBc2011␣Hallo`	...	`aBc2011␣Hallo`
Ausgabe:	`0100000010000`	...	`ABC2011␣HALLO`

Aufgabe 3.2: Ein- und Ausgabe von Zahlenwerten

1. Ergänzen Sie in »utils.h« und »utils.c« eine Funktion

   ```
   unsigned short term_read_unsigned();
   ```

   die auf Ziffern von der seriellen Schnittstelle gefolgt von einem Trennzeichen wartet und nach der Rekursion

   $$\begin{aligned} W_0 &= 0 \\ W_{n+1} &= 10 * W_n + z \end{aligned}$$

   (z – Wert der empfangenen Ziffer; W_n – Zahlenwert nach Empfang von n Ziffern) den Zahlenwert berechnet. Trennzeichen vor der ersten Ziffer sind zu ignorieren. Wenn die Ziffenfolge danach von unzulässigen Zeichen unterbrochen ist oder der Wert der Zahl größer als der größte darstellbare Zahlenwert ist, soll der größte darstellbare Zahlenwert 0xFFFF zurückgegeben werden.

2. Ergänzen Sie weiterhin in »utils.h« und »utils.c« eine Funktion

   ```
   void term_write_unsigned(unsigned z);
   ```

   die den Zahlenwert »z« in eine Zifferenfolge übersetzt und zeichenweise versendet. Der Algorithmus hier besteht aus zwei Schleifen. In der ersten Schleife wird in einer Variablen »x« die größte Zehnerpotenz kleiner dem darzustellenden Zahlenwert »z« gesucht. In der zweiten Schleife wird solange $x > 0$ ist

   - der Ziffernwert »c« als ganzzahliger Quotient aus Zahlenwert und Stellenwert (Zehnerpotenz) gebildet

     $$c = z/x$$

   - vom Zahlenwert das Produkt aus Zahlenwert und Stellenwert subtrahiert

     $$z = z - c \cdot x$$

   - der Stellenwert durch zehn geteilt:

     $$x = x/10$$

   - der Ziffernwert in den Zeichenwert der Ziffer umgerechnet und versendet.

   Nach der Ziffernfolge soll noch ein Leerzeichen versendet werden.

3. Ändern Sie zum Testen in der Endlosschleife des Hauptprogramms die Anweisungsfolge so ab, dass in der Endlosschleife immer auf zwei mit Trennzeichen abgeschlossene Ziffernfolgen gewartet und die Summe gefolgt von einem Leerzeichen zurückgesendet wird.

Testbeispiel:

Eingabe:	137␣344␣1647␣56
Ausgabe:	481␣1703␣