Praktikum Softprocessor SP2: Grundkurs C-Programmierung

7. Juli 2014

Zusammenfassung

Mit dem im ersten Praktikumsversuch entwickelten und getesteten eingebetteten Rechnersystem werden am Beispiel kleiner Programmieraufgaben Fallunterscheidungen und Schleifen in C eingeführt bzw. aufgefrischt.

1 Einleitung

1.1 Hardware-Konfiguration und Header-Dateien

Das im ersten Praktikumsversuch entwickelte Rechnersystem besteht aus einem MicroBlaze-Prozessor, einem Speicher, zwei parallelen, einer seriellen und einer Debug-Schnittstelle (Abb. 1) und muss nach Zuschalten der Versorgungsspannung als Bitdatenstrom in den programmierbaren Schaltkreis auf der Versuchsbaugruppe geladen werden. Das im ersten Praktikum entwickelte C-Programm bildet in diesem Praktikum den Rahmen für eigene Progragrammentwicklungen. Es beginnt mit den Include-Anweisungen der Header-Dateien:

```
#include "xparameters.h"
#include "xgpio_l.h"
#include "xuartlite_l.h"
```

Die Datei »xparameters.h« enthält die Parameter der Hardware-Konfiguration und die beiden anderen die Schnittstellenbeschreibungen der elementaren Lese- und Schreibfunktionen für die Ein- und Ausgabeschnittstellen.

1.2 Das Rahmenprogramm

Das Rahmenprogramm für die nachfolgenden Programmieraufgaben sei das vorgegebene Programm aus der ersten Übung mit kleinen Verbesserungen. Die Lese- und Schreibfunktion für die serielle Schnittstelle soll durch die in C üblichen Funktionsaufrufe ersetzt werden:

```
char z;
...
putchar(z);     // Zeichen versenden
z = getchar();  // Zeichen empfangen
```

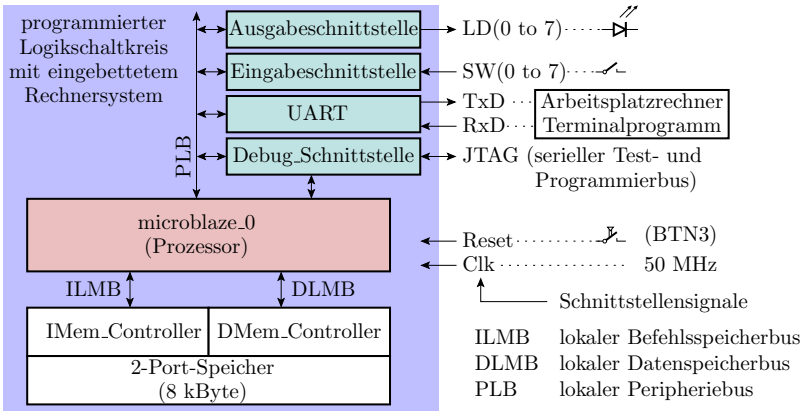

Abbildung 1: Das entworfene Rechnersystem

Dazu sind nach den Include-Anweisungen folgende Makros zu definieren[1]:

```
#define putchar(c) \
     XUartLite_SendByte(XPAR_UART_BASEADDR, c)
#define getchar() \
     XUartLite_RecvByte(XPAR_UART_BASEADDR)
```

(\ – Zeichen zur Fortsetzung der Makro-Definition in der nachfolgenden Zeile). Das Hauptprogramm »main«, das nach »run« oder Betätigung der Reset-Taste »BTN3« aufgerufen wird, vereinbart zu Beginn die Zeichenkettenkonstante »Hallo Welt\r\n«, eine Zeigervariable »str«, die auf den Anfang der Zeichenkette zeigt, und eine 8-Bit-Variable zur Speicherung eines Zeichens:

```
int main()
 {
 char *str = "Hallo Welt\r\n";
 char data;
```

In der nachfolgenden Schleife zum Versenden der Zeichenkette soll die Sendefunktion für Einzelzeichen durch das zuvor definierte Makro »putchar(...)« ersetzt werden:

```
 while (*str != 0)
  {
  putchar(*str);
  str++;
 }
```

Der Ausdruck »*str« beschreibt den Wert, der im Speicher auf der Adresse »str« steht. Der Ausdruck »str++« schaltet den Zeiger ein Zeichen weiter. In der nachfolgenden Endlosschleife wird jeweils auf ein Byte von der seriellen Scnittstelle gewartet, der Bytewert auf die Leuchtdioden ausgegeben und zurückgesendet, der Byte-Wert von den Schaltern eingelesen und gleichfalls an den PC gesendet.

```
  while (1)
   { // ------------ Schleifenkörper-------------------
   data = getchar();
   XGpio_mWriteReg(XPAR_AUSGABESCHNITTSTELLE_BASEADDR, 0, data);
   putchar(data);
   data = XGpio_mReadReg(XPAR_AUSGABESCHNITTSTELLE_BASEADDR);
   putchar(data);
    // ------------------------------------------------------
   }
}
```

In den Aufgaben, die in diesem Praktikum gelöst werden sollen, ist die Zeichenkettenkonstante, die in der ersten Schleife an den PC gesendet wird, durch »Aufgabe *i.j*« (*i.j* – Aufgabennummer) und der Schleifenkörper der Endlosschleife durch das zu entwickelnde Programmstück zu ersetzen.

1.3 Neues Programmierprojekt vorbereiten

Starten Sie SDK::

- »Anwendungen« ▷ »Umgebung« ▷ »Xilinx Platformstudio (SDK)«

wählen Sie als Workspace das Unterverzeichnis »Aufg1-4/SDK/SDK_Workspace«. Legen Sie für jede Aufgabe ein Software-Projekt anzulegen

- »File« ▷ »New« ▷ »Manages Make C ...« ▷ Project Name: »Aufg_....«

In dem neuen Projektordner ist eine Programmdatei »main_a?_?.c« zu erstellen:

- »File« ▷ »New« ▷ »Source File« ▷ Folder: »Aufg_...«, File: »main_2_1.c«

[1] Eine Makro-Definition beschreibt eine intelligente Textersetzung, bei der der Präprozessor – ein Programm, das den Programmtext vor der Übersetzung vorverarbeitet – den Funktionsaufruf hinter »#define« durch den nachfolgenden Funktionsaufruf ersetzt.

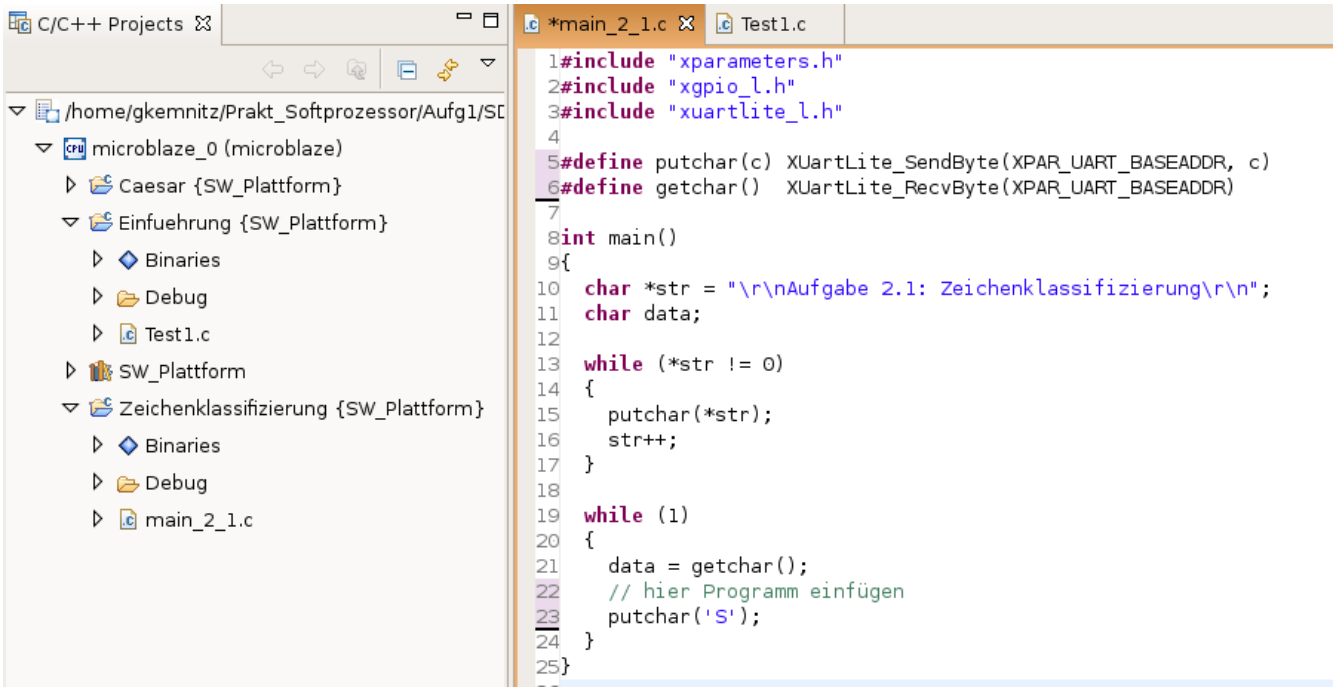

Abbildung 2: Projektvorbereitung für die erste Programmieraufgabe

(..._2_1. – Praktikums- und Aufgabennummer). In die Programmdatei ist der Inhalt der Datei »hallo.c« aus dem Projekt »HalloWelt« zu kopieren und anzupassen. Abbildung 2 zeigt das Ergebnis der vorbereitenden Schritte für die erste Programmieraufgabe. Bevor die selbst entwickelten Programme mit »Run« oder »Debug« getestet werden können, ist die Versorgungsspannung an die Versuchsbaugruppe anzuschließen und die Bitdatei mit der Hardware-Beschreibung in den programmierbaren Logikschaltkreis zu laden:

- »Tools« ▷ »Programm FPGA« ▷ »Save and Program«

2 Fallunterscheidungen

Fallunterscheidungen sind Kontrollflussanweisungen zur Beschreibung einer bedingten Ausführung von Anweisungsfolgen. Wir beschränken uns hier auf die if-Anweisung.

Die nachfolgenden Beschreibungen der Programmkonstrukte lehnen sich an die BMF (Bakus-Naur-Form) an. Schlüsselworte werden **fett** dargestellt und Platzhalter, die nach weiteren Regeln durch Programmtexte zu ersetzen sind, werden *kursiv* dargestellt. Texte in **[**...**]** können null oder einmal und Texte in **{**...**}** beliebig oft vorkommen. Eine if-Anweisung beginnt mit dem Schlüsselwort **if** gefolgt von einer Bedingung, die in Klammern stehen muss, gefolgt von der Anweisungsfolge, die, wenn die Bedingung erfüllt ist, ausgeführt werden soll. Jede Anweisung wird mit einem Semikolon abgeschlossen. Ein Folge von mehreren Anweisungen ist in {...} einzurahmen:

if (*Bedingung*) {
Anweisung;
}
[else if (*Bedingung*) {
Anweisung;
}
]
[else{
Anweisung;
}**]**

Nach dem if-Zweig können beliebig viele Else-If-Zweige folgen, die abgearbeitet werden, wenn die eigene Bedingung

erfüllt und alle Bedingungen der vorherigen if- und else-if-Zweige nicht erfüllt sind. Den Abschluss kann ein else-Zweig bilden, dessen Anweisungen abgearbeitet werden, wenn keine der vorherigen Bedingungen erfüllt ist.

Eine Bedingung – ein Ausdruck mit einem Zahlenwert – ist erfüllt, wenn der Wert ungleich null, und nicht erfüllt, wenn der Wert gleich null ist. Bedingungen werden häufig durch den Vergleich zweier Operanden gebildet:

gleich	ungleich	kleiner	kleiner gleich	größer	größer gleich
==	!=	<	<=	>	>=

Die Vergleichsergebnisse haben den Wertebereich 0 für »falsch« und 1 für »wahr«. Die Verknüpfung von Vergleichsergebnissen erfolgt über logische Operatoren:

Negation	UND	ODER
!	&&	||

Der logische Ausdruck

$$((a < b) \vee (b \geq 5)) \wedge (d = 3)$$

wird z.B. in C wie folgt beschrieben:

```
((a<b || b>=5) && d==3)
```

Im nachfolgenden Beispiel sind zwei Zähler »CtA« und »CtB« als vorzeichenbehaftete ganze Zahlen und »s« als Zeichen vereinbart. Wenn »s« das Zeichen »a« ist, soll der erste, und wenn es das Zeichen 'b' ist, soll der zweite Zähler um eins erhöht werden:

```
int CtA, CtB;
char s;
...
if      (s=='a') CtA++;
else if (s=='b') CtB++;
```

Der Ausdruck 'a' beschreibt den Zeichenwert des Zeichens »a«, der 0x61 ist, und der Operator »CtA++« erhöht den Zählerwert um eins. Die {...} um die Anweisungsfolgen des If- und des Else-If-Zweigs sind hier nicht erforderlich, weil beide Anweisungsfolgen nur je aus einer Anweisung bestehen. Abbildung 3 zeigt noch einmal die ASCII-Tabelle aus der vorherherigen Praktikumsanleitung, die für die Lösung der Aufgaben benötigt wird.

Testbeispiel (Monitorausgaben):

```
1a3f56y
G:y K:1
Hallo␣Welt
G:t K:␣
```

In der ersten Zeile hat »y« den größten Zeichenwert von 0x74 und »1« den kleinsten Zeichenwert von 0x31. In der zweiten Eingabezeile hat »t« den größten und das nicht dargestellte Leerzeichen den kleinsten Zeichenwert.

3 Schleifen

In C gibt es zwei Schleifentypen. Die For-Schleife ist für Zählschleifen vorgesehen. Sie beginnt mit dem Schlüsslewort **for** gefolgt von einem geklammerten Tupel mit drei durch Semikolon getrennten Einträgen. »*Anfangswert*« ist ein Platzhalter für eine oder mehrere Anweisungen vor der Schleife und dient üblicherweise zur Initialisierung des Schleifenzählers. »*Interationsbedingung*« ist ein logischer Ausdruck, bei dem, wenn er nicht erfüllt ist, die Schleife verlassen wird. »*Inkrement*« ist ein Platzhalter für ein oder mehrere Anweisungen im Schleifenkörper, in der Regel Anweisungen zum Weiterzählen des Schleifenzählers. Nach dem Tupel folgen die (übrigen) Anweisungen des Schleifenkörpers:

```
for ([Anfangswert]; [Iterationsbedingung]; [Inkrement])
 {
  {Anweisung;}
 }
```

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Abbildung 3: ASCII-Tabelle zur Umrechnung der Zeichen in Zeichenwerte

Das universellere Schleifenkonstrukt ist die While-Schleife. Sie beginnt mit dem Schlüsselwort **while** gefolgt von der in Klammern gesetzten Iterationsbedingung und der Anweisungsfolge des Schleifenkörpers:

```
[Anfangswert;]
while (Iterationsbedingung)
 {
  {Anweisung;}
  [Increment;]
 }
```

Solange die Iterationsbedingung erfüllt ist, wird die Anweisungsfolge im Schleifenkörper wiederholt. Mit einer Anfangswertzuweisung an den Schleifenzähler vor der Schleife und einer Inkrement-Anweisung für den Schleifenzähler im Schleifenkörper bildet eine While-Schleife eine Zählschleife nach. Die bisher verwendete Endlosschleife hat die Iterationsbedingung »1« (immer wahr), d.h. sie wird nie beendet. Für Schleifenkörper gibt es im Zusammenhang mit Fallunterscheidungen noch zwei weitere Kontrollflussanweisungen:

```
if (Bedingung) break;
```

beendet bei erfüllter Bedingung die aktuelle Schleife und

```
if (Bedingung) continue;
```

beendet bei erfüllter Bedingung die Abarbeitung des Schleifenkörpers und bewirkt einen Sprung zur Prüfung der Interationsbedingung.

Aufgabe 2.1: Zeichenklassifizierung

In jedem Durchlauf der Endlosschleife soll ein Zeichen vom PC über die serielle Schnittstelle gelesen, entsprechend der nachfolgenden Tabelle klassifiziert und das Ergebnis – ein Zeichen – über die serielle Schnittstelle zurück zum PC gesenden werden.

Eingabezeichen	Ausgabezeichen
Ziffer ('0', ..., '9')	'Z'
Buchstabe ('a', ..., 'z', 'A', ..., 'Z'	'B'
Steuerzeichen mit einem Zeichenwert kleiner 32 (hexadezimal 0x20 oder größer 126 (hexadezimal 0x7E)	'C'
alle anderen Zeichen	'S'

Testbeispiel:

Eingabe:	`aBc2011␣!#:↩`
Ausgabe:	`BBBZZZZSSSSC`

(␣ – Leerzeichen; ↩ – Enter)

Aufgabe 2.2: Cesar-Chiffre

Der römische Feldherr Gaius Julius Cesar verwendete für geheime Korrespondenzen folgende Verschlüsselung. Jeder Buchstabe im Text wurde um den um n Stellen zirkular verschobenen Buchstaben im Alphabet ersetzt. Zur Rückgewinnung des ursprünglichen Textes wird das Verfahren zeichenweise umgekehrt.

Entwickeln Sie ein Programm, das nach diesem Schema für alle von der seriellen Schnittstelle empfangenen Zeichen die Großbuchstaben, die Kleinbuchstaben und die Ziffern jeweils den um n Zeichen zirkular verschobenen Großbuchstaben, den um n Zeichen zirkular verschobenen Kleinbuchstan bzw. die um n Zeichen zirkular verschobene Ziffer ersetzt. Alle anderen Zeichen sollen nicht verändert werden. Der Wert von n sei der mit den Schaltern auf der Baugruppe eingestellte Wert, der in der Schleife vor jeder Zeichenkonvertierung mit

```
n = XGpio_ReadReg(XPAR_AUSGABESCHNITTSTELLE_BASEADDR, 0);
```

eingelesen wird.

Testbeispiel für $n = 2$:

Eingabe:	`98 ist ziemlich gross`
Ausgabe:	`10 kuv bkgonkej itquu`

Zusatzfrage:

- Für welche Werte von n werden alle Ziffern und Buchstaben verschlüsselt?

Aufgabe 2.3: Knobelaufgabe

Aus einer Folge von Zeichen soll das Zeichen mit dem kleinsten und das Zeichen mit dem größten Zeichenwert zurückgegeben werden. Schleifen sind nicht erlaubt, bis auf die vorgegebene Endlosschleife. Die Zeichenauswertung soll mit jeder Zeile neu beginnen. Jedes empfangene Zeichen soll zur Darstellung auf dem Terminal zurückgesendet werden. Nach einem Zeilenumbruch ist zusätzlich eine Zeile mit der Angabe des größten und des kleinsten Zeichens in der Form:

`G:`G `K:`K`\n'`

(G – durch größte Zeichen zu ersetzen; K – durch kleinste Zeichen zu ersetzen) zurückzusenden. Zeichen mit einem Zeichenwert kleiner 0x20 oder größer 0x7E sind zu ignorieren.

Aufgabe 2.4: Zeichenstromformatierung

Schreiben Sie mit Hilfe einer in die Endlosschleife eingebetteten For-Schleife und einer Fallunterscheidung ein Zeichenstromformatierungsprogramm, das alle empfangenen Zeichen mit Zeichenwerten von 0x20 bis 0x7E zurücksendet und nach jedem zehnten zurückgesendeten Zeichen zusätzliche einen Zeilenumbruch aus den Zeichen »\n\r« einfügt.

Testbeispiel:

Eingabe:	`aBc2011␣!#:absftz76ghf` `345`

Ausgabe:	`aBc2011␣!#`
	`:absftz76g`
	`hf345`

(␣ – Leerzeichen). Der Zeilenumbruch der Eingabe geht, da er durch Zeichenwerte kleiner 0x20 dargestellt wird, verloren.

Aufgabe 2.5: Unäre Wertedarstellung

Das Unärsystem ist ein Additionssystem, das lediglich ein Symbol mit der Wertigkeit »1« besitzt. Jeder Einer wird durch dasselbe Symbol repräsentiert, üblicherweise durch einen senkrechten Strich. Nach Eingabe einer Ziffer soll zuerst die Ziffer, dann ein Doppelpunkt, gefolgt von der unären Darstellung des Ziffernwertes und einem Zeilenumbruch zurück geschickt werden. Alle anderen Eingabezeichen sind zu ignorieren.

Testbeispiel:

Eingabe:	`197a3`

Ausgabe:	`1:|`
	`9:|||||||||`
	`7:|||||||`
	`3:|||`

Aufgabe 2.6: Schachbrettausgabe

Schreiben Sie ein Programm, das in der vorgegebenen Endlosschleife auf ein beliebiges Zeichen von der seriellen Schnittstelle wartet und nach dessen Empfang mit zwei ineinander verschachtelten Schleifen das nachfolgende Schachbrettmuster zurückschickt:

```
A1 A2 A3 A4 A5 A6 A7 A8
B1 B2 B3     ...     B8
C1 C2 C3     ...     C8
D1 D2 C3              .
E1           ...      .
F1                    .
G1                    .
H1 H2 H3     ...     H8
```

Aufgabe 2.7: Text-Wert-Text-Konvertierung

Schreiben Sie ein Programm, das in der Endlosschleife immer auf drei Ziffern von der seriellen Schnittstelle wartet, die Ziffern gefolgt von einem Doppelpunkt zurückschickt, nach der Formel

$$w = \sum_{i=0}^{2} z_i \cdot 10^i$$

den Zahlenwert in der mit

```
unsigned short w; // vorzeichenfrei, 16 Bit
```

vereinbarten Variablen berechnet, den an den Schaltern eingestellten Wert einliest, den eingelesenen Schalterwert zum Wert der Variablen w addiert, die Summe in eine Ziffernfolge der Länge 4 zurückwandelt und an den PC gefolgt von einem Zeilenumbruch schickt.

Testbeispiel für Schalterstellung 0010 0011 (0x23=35):

Eingabe:	197480012030
Ausgabe:	197:0232 480:0515 012:0047 030:0065

Zusatzfrage:

- Wie viele Binärziffern werden maximal für die Darstellung der Summe benötigt? Reicht »short unsigned« (vorzeichenfrei, 16-Bit) als Datentyp dafür in jedem Fall aus?