Softprocessor SP2: Basic Course C-Programming

July 7, 2011

Abstract

In the first softprocessor lecture an embedded computer system was developed and tested. In this lecture small programming exercises using distinctions of cases and loops in C will be introduced respectively refreshed.

1 Introduction

1.1 Hardware-Configuration and Header-Files

The developed computer system from the first lecture consists of a MicroBlace processor, a dualport memory, two parallel interfaces, one serial interface (UART) and one debug interface (fig. 1). After switching on the supply power the bitstream of the computer system has to be transferred into the FPGA on the development board. The C-program from the first lecture is the frame for own programs in this lecture. It starts with the »inlcude« statements of the header files:

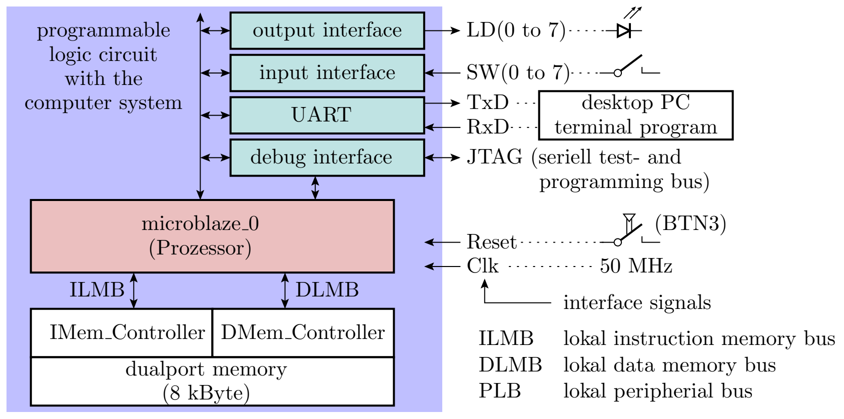

programmable logic circuit with the computer system
output interface
input interface
UART
debug interface
LD(0 to 7)
SW(0 to 7)
TxD
RxD
desktop PC terminal program
JTAG (seriell test- and programming bus)
microblaze_0 (Prozessor)
Reset
(BTN3)
Clk
50 MHz
interface signals
ILMB
DLMB
IMem_Controller
DMem_Controller
dualport memory (8 kByte)
ILMB lokal instruction memory bus
DLMB lokal data memory bus
PLB lokal peripherial bus

Figure 1: Developed computer system from the first lecture

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

The file »xparameters.h« contains the hardware-configuration parameters and the other two files the parameters for basic read- and write-functions.

1.2 Main program

The main program for the next exercises is the given program of the first exercise with minor improvements.

```
char z;
...
putchar(z);     // send character
z = getchar();  // receive character
```

After the include-statements the following macros are defined[1]:

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

(\ – character for continuation of a macro definition in the next line). The main program »main«, which starts after »run« or pressing the reset-button »BTN3«, defines at begin the string »Hallo Welt\r\n« as constant, a pointer variable »str«, pointing on the beginning of the string and an 8 bit variable for saving one character:

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

In the following loop, to send the string, the function to send single characters should be replaced with the earlier defined macro »putchar(...)«:

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

The statement »*str« describes the value at the memory address of »str«. The statement »str++« is shifting the pointer one character ahead. In the following infinite loop every time when a byte from the UART is received it will be shown on the LEDs and sent back to the PC. Also the byte-value of the switches will be read and sent to the PC.

[1]A macro definition describes an intelligent textsubstitution, the preprocessor – a program for preconditioning the program before compiling – replaces the function call after »#define« with the following function call.

```
    while (1)
     { // ------------ body of the loop-------------------
     data = getchar();
     XGpio_mWriteReg(XPAR_AUSGABESCHNITTSTELLE_BASEADDR, 0, data);
     putchar(data);
     data = XGpio_mReadReg(XPAR_AUSGABESCHNITTSTELLE_BASEADDR);
     putchar(data);
      // -----------------------------------------------------------
     }
  }
```

In the next excercises of this course the string which was sent to the PC in the first loop will be replaced with »excercise *i.j*« (*i.j* – excercise number) and the loop body of the infinite loop will be replaced by the programm, which should be developed.

1.3 Preparation for a new project

Copy the SDK subfolder from the previous exercise in the project directory for the second exercise:

- Change to the »soft processor« directory
- mkdir Aufg2
- cp - r Aufg1/SDK Aufg2/SDK

Start the program SDK via the menu:

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

go to the subdirectory »Aufg2/SDK/SDK_workspace«. In the workspace make a new softwareproject for every exercise:

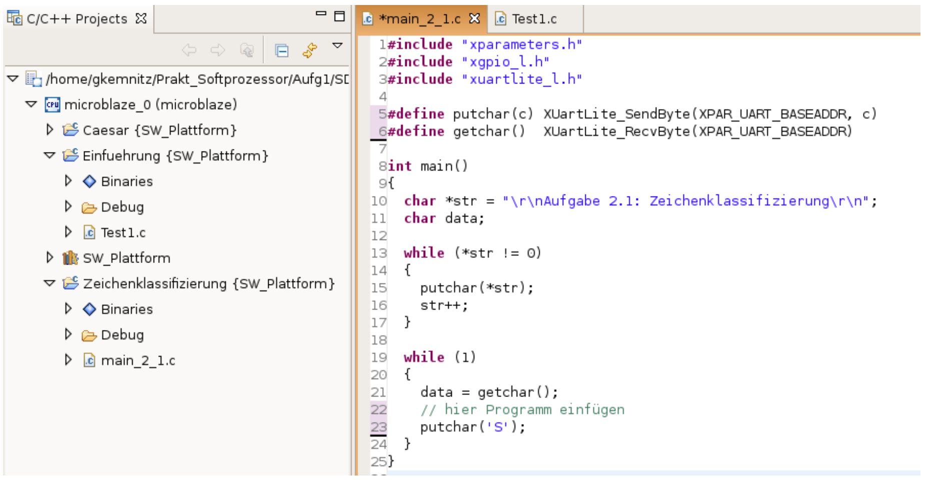
- »File« ▷ »New« ▷ »Manages Make C ...« ▷ Project Name: »Aufg_....«

Make a new program file »main.c« in the new project folder:

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

(..._2_1. – Number of practical course and exercise). Copy the content from the project »Introduction« to the file »Test1.c« and modify it. Figure 2 shows the result of the preparation steps for the first programming exercise. Before it is possible to test the self developed program with »Run« or »Debug« the supply power have to be connected to the developing board and the bitstream with the hardware description has to be loaded into the FPGA:

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


C/C++ Projects
/home/gkemnitz/Prakt_Softprozessor/Aufg1/SD
microblaze_0 (microblaze)
Caesar {SW_Plattform}
Einfuehrung {SW_Plattform}
Binaries
Debug
Test1.c
SW_Plattform
Zeichenklassifizierung {SW_Plattform}
Binaries
Debug
main_2_1.c
*main_2_1.c
Test1.c

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

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

int main()
{
  char *str = "\r\nAufgabe 2.1: Zeichenklassifizierung\r\n";
  char data;

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

  while (1)
  {
    data = getchar();
    // hier Programm einfügen
    putchar('S');
  }
}
```

Figure 2: Project preparation of the first programming exercise

2 Case distinctions

Case distinctions are control flow statements to describe a conditional execution of statements. Only the if-statement will be considered here.

The following describtions of program constructs lean on BNF (Bakus-Naur-Form). Keywords will be **bold** printed and placeholder, which have to be replaced with program lines after some rules, will be printed in *italic*. Strings in [...] can be there one or none time, while strings in {...} can be there manytimes. The if-statement **if** is followed by a condition in brackets, a sequence of statements executed if the condition is true. Each statement is closed ba a semicolon. A sequence of some statements has to be in {...}:

```
if (condition) {
 {statement;}
 }
{else if (condition) {
 {statement;}
 }
}
 [else{
 {condition;}
 }]
```

After the if-branch can as many as needed else-if-branches follow which will be executed if the own condition is true and none of the if-conditions before is true. At the end there could be an else-branch, then the following statements will be executed if none of the if-conditions is true.

The condition – one expression with a number value – is - true - if the value not equals zero, and - false - if the value is zero :

equal	unequal	less	less or equal	greater	greater or equal
==	!=	<	<=	>	>=

The comparison results have the value range false« for 0 and 1 for »true«. The combination of comparison results is handled via logical operations:

negation	AND	OR
!	&&	||

The logic expression

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

is written in C as:

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

In the next example two counters »CtA« und »CtB« as unsigned integer and »s« as character are defined. If the value of »s« is the characer »a« the first counter should increase, if it is »b« the second counter should increase:

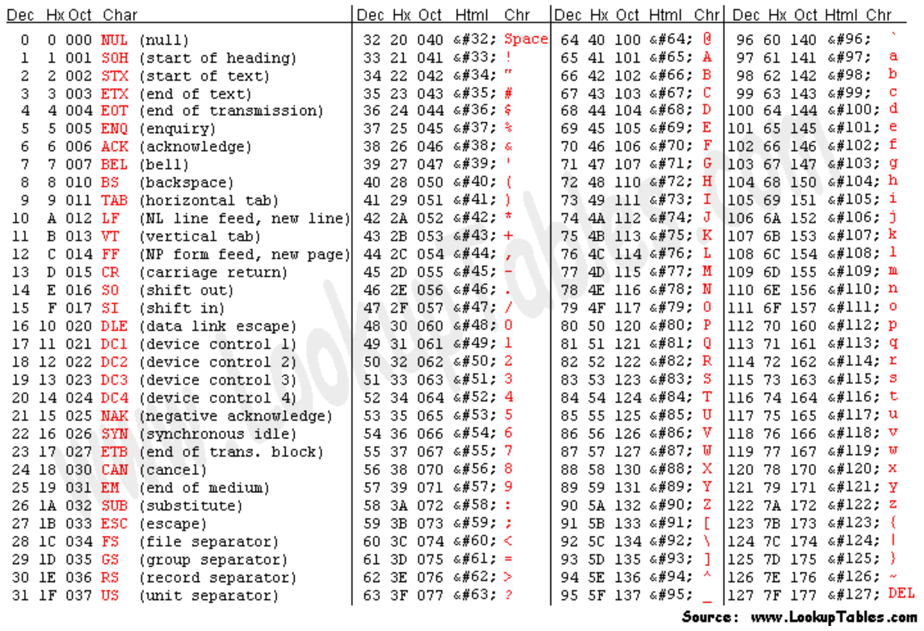
```
int CtA, CtB;
char s;
...
if      (s=='a') CtA++;
else if (s=='b') CtB++;
```

The statement 'a' stands for the character value »a«, it is 0x61, and the statement »CtA++« increases the counter value by one. The {...} around the sequences of the if- and else-if-branch are not necessary here, because both sequences consists of only one statement. Figure 3 shows the ASCII table again, it is needed for the exercises.

Aufgabe 2.1: String classification

In every pass of the infinite loop a character from the PC via UART should be read and classfied after the table below. The result – a character – shall be send back to the PC.

Inputcharacter	Outputcharacter
Digit ('0', ..., '9')	'Z'
Alphabetic character ('a', ..., 'z', 'A', ..., 'Z'	'B'
Control character with a character value less than 32 (hexadecimal 0x20) or greater than 126 (hexadecimal 0x7E)	'C'
all other characters	'S'



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

Source: www.LookupTables.com

Figure 3: ASCII table for conversion of characters in values

Test example:

Input:	aBc2011␣!#:↩
Output:	BBBZZZZSSSSC

(␣ – space; ↩ – enter)

Aufgabe 2.2: Cesar-Code

The roman commander Gaius Julius Cesar used for secret communication the following code. Every character in the text was shifted circular by n places and replaced with the resulting character in the alphabet (see example below). To recreate the original text, the procedure has to be reversed character-wise.

Write a program which receives characters from the UART. The program shall send back every received upper character as the by n shifted upper character and every received lower character as the by n shifted upper character. Every received digit should send back as the by n shifted digit. All other characters shall be returned without shifting. The value of n will be adjusted by the switches on the developing board. Before every loop of conversion the value should be read with

```
    n = XGpio_mReadReg(XPAR_AUSGABESCHNITTSTELLE_BASEADDR);
```

Test example for $n = 2$:

Input:	`98 is quite big`
Output:	`10 ku swkvg dki`

Additional question:

- Which values of n encodes every digits and characters?

Aufgabe 2.3: Riddle

From a character sequence the character with the biggest an the character with the lowest value shall be returned. Loops are not allowed, aside from the given infinite loop. The character analysis shall be restarted with every new line. Every received character should be returned to the terminal for visualisation. After »Enter« an extra line with the value of the biggest and the lowest value shall be inserted and sended to the terminal:

```
B:G L:K\n'
```

(G – replaced by the biggest character; K – replaced by the lowest character). Characters with value less than 0x20 or greater than 0x7E will be ignored.

Test example (monitor output on the PC):

```
1a3f56y
B:y L:1
Hallo␣Welt
B:t L:␣
```

In the first line »y« has the biggest character value of 0x74 and »1« the lowest character value of 0x31. In the second line »t« has the biggest and the not pictured space character has the lowest character value.

3 Loops

C has two loop types. The for-loop is a counting loop. The loop begins with **for** followed by three entries in brackets separated by semicolons. The »*start-value*« is a place holder for one or more statements before the loop, usually they are for initialisation of the loop counter. The »*iterationcondition*« is a logical statement, if it is false the loop will be left. The »*inkrement*« is a place holder for one or more statements in the loop body, usually statements to increment the loop counter. After this three entries followes the (other) statements of the loop body:

```
for ([startvalue]; [iterationcondition]; [increment])
 {
  {statement;}
 }
```

The universal loop is the while loop. It starts with the keyword **while** followed by the iterationcondition in brackets and the statement sequence of the loop body:

```
[startvalue;]
while (iterationcondition)
 {
  {statement;}
  [increment;]
 }
```

If the iterationcondition is true, the statement sequence in the loop body is repeated. With the assignment of an initial value before the loop body and an increment statement for the loop counter in the loop body, a counting loop can be reproduced. The already used infinite loop has the iterationcondition »1« (always true), that means there is no termination of the loop. For loop bodies exists two additional control flow statements in combination with case distinctions :

```
if (condition) break;
```

terminates the loop in case the condition is »true« and

```
if (condition) continue;
```

ends the current loop execution if the condition is »true« and jumps to the next cycle.

Aufgabe 2.4: Character stream formatting

Write a character stream formatting program using the infinite loop including a for loop and a case distinction, which returnes all received charactersfrom 0x20 to 0x7E to the PC and inserts a line break consisting of the chracters »\n\r« after every tenth sended character.

Test example:

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

Output:	`aBc2011␣!#`
	`:absftz76g`
	`rhf345`

(␣ – space). The received line break will be lost, its chracter value is less than 0x20.

Aufgabe 2.5: Unary value representation

The unary system is an adding system with only one symbol with ther value of »1«. Each one will be represented through the same symbol, usually a vertical line. After the input of a digit, a double point followed by the unary representation of the digit value and a line break shall be returned. All other input characters will be ignored.

Test example:

Input:	`197a3`

Output:	`1:|`
	`9:|||||||||`
	`7:|||||||`
	`3:|||`

Aufgabe 2.6: Chess board output

Write a program waiting in an infinite loop on any character via UART. After having received the character the programm shall return the following chess board model, realised with two nested loops, to the PC:

```
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-value-text-conversion

Write a program, which is always waiting in an ifinite loop for three digits received via UART. The three digits shall be returned followed by a double point. After the formula

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

the numbervalue of the three received digits will be saved in the variable

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

The value set by the switches shall be, added to variable$w$. This sum shall be converted into a character sequence with the length of 4 and returned to the PC followed by a line break.

Test example for switches representing 0010 0011 (0x23=35):

Input:	`197480012030`
Output:	`197:0232`
	`480:0515`
	`012:0047`
	`030:0065`

Additional question:

- How may binary digits are necessary at most for representation of the sum? Is »short unsigned« (unsigned, 16-bit) as datatyp enough in any case?