

Soft Processor SP1: Introduction

June 15, 2011

Abstract

A computer system consisting of a soft processor, a small data and instruction memory, input from switches, output to LEDs, a serial interface for communication with the PC and a debug interface will be configured out of IP cores and programmed in C.

1 Hardware Design

An embedded computer system consists of a processor, memory and peripheral interfaces for communication with its environment. Figure 1 shows the block diagram of the computer system to be designed. The core is a 32-bit RISC processor, type MicroBlaze. That is a processor especially optimized for the implementation in programmable logical circuits (FPGAs). The processor is connected to the data bus (DLMB), the instruction bus (ILMB) and the peripheral bus (PLB). The data and the instruction bus are each connected via a memory controller to one of the ports of a dual port memory. To the peripheral bus (PLB)

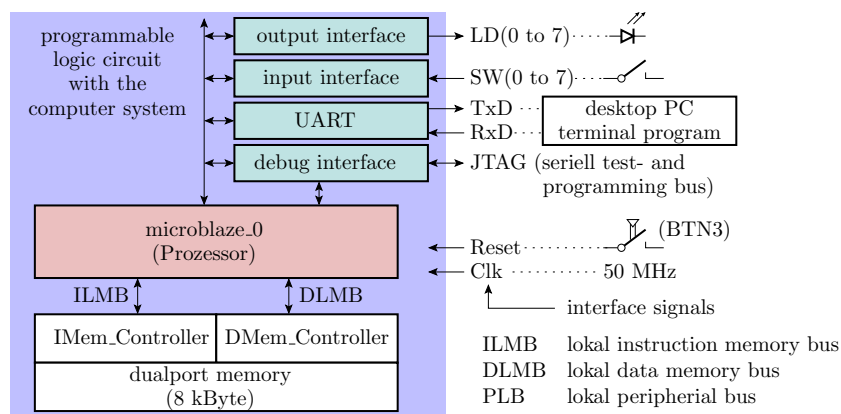


Figure 1: The computer system to be designed

- a parallel interface to drive 8 LEDs,
- a parallel interface to input data from 8 switches,
- a serial interface for communication with the desktop PC and
- and a special debug interface for running programmes in step mode, setting break points etc..

are connected. An embedded computer system does not have a keyboard, a hard disk or a monitor. The program will be downloaded into the programmable circuit in the same way as the hardware configuration via the JTAG bus. The user input and output is carried out via the serial interface and the communication with the debugger via the JTAG. The reset button is »BTN3«. Via the 8 switches a byte can be entered and via the 8 LEDs the value of a single byte can be displayed. The system clock is 50 MHz and will be provided by IC4 at the bottom side of the test board.

1.1 Create a new Project

- Start the Program via the menu: »Anwendungen« ▷ »Umgebung« ▷ »Xilinx Platformstudio (EDK)«
- Select »Blank XPS project« (figure 2 left). Continue with »OK«.
- Define the working directory for the project via »Project file« ▷ »Browse« and select the programmable logic circuit of the test board: »spartan3«, »xc3s1000«, »ft256« and »-4« (figure 2 right¹).
- Continue with »OK«.

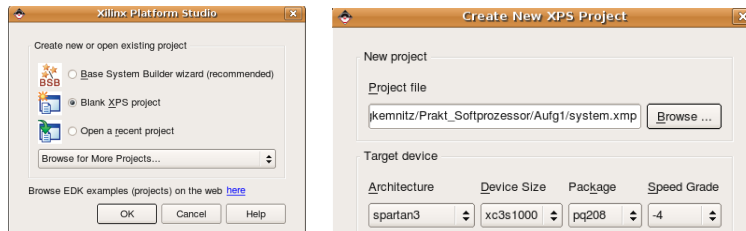


Figure 2: Create a new project

¹Attention input bug: you have to click at all menu items, even if the displayed values don't have to be changed.

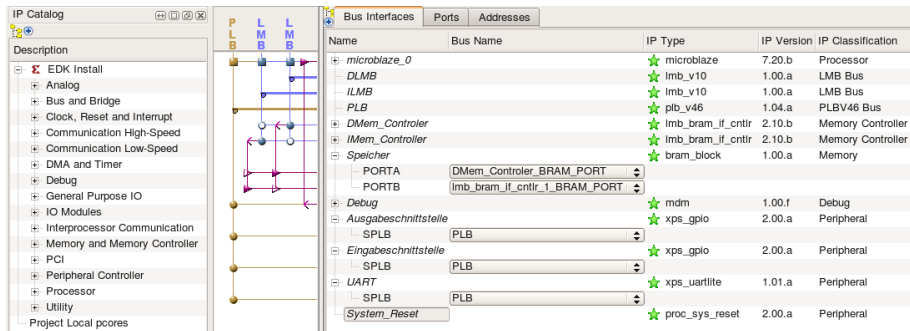


Figure 3: Selecting the functional blocks from the IP catalog

1.2 Selection, Configuration and Connection of the IP-Cores²

At the user interface beneath the left window the tab »IP Catalog« has to be selected. The required IP-cores have to be dragged with the mouse to the right side. In figure 3 it has already been done. The following table summarizes for all 10 functional blocks the category, the original name and how they have to be renamed in correspondence to figure 1.

category	circuit type	to rename in
Prozessor	MicroBlaze	microblaze_0 ^{*1}
Bus and Bridge	Local Memory Bus (LMB)	DLMB
Bus and Bridge	Local Memory Bus (LMB)	ILMB
Bus and Bridge	Processor Local Bus (plb_v46)	PLB
Memory and Memory Contr.	LMB BRAM Controller	IMem_Controller
Memory and Memory Contr.	LMB BRAM Controller	DMem_Controller
Memory and Memory Contr.	Block RAM (BRAM)	Speicher
Debug	MicroBlaze Debug Module (MDM)	Debug
General Purpose IO	XPS General Purpose IO	Ausgabeschchnittstelle ^{*2}
General Purpose IO	XPS General Purpose IO	Eingabeschchnittstelle ^{*3}
Communication Low Speed	XPS UART (Lite)	UART
Clock, Reset and Interrupt	Processor System Reset Module	System_Reset

^{*1} The name of the processor should not be changed. Otherwise there may arise difficult to understand problems within the software development system SDK, e.g. using capitalized characters causes confusion with the search paths of design objects; ^{*2} output interface; ^{*3} output interface;

For renaming the original name has to be selected with the left mouse button and overwritten. Next the bus connections have to be arranged. Of the processor the data bus (DLMB), the instruction bus (ILMB) and the peripheral bus (DPLB) have to be connected with system busses as shown in figure 4 by clicking

²IP-Cores are pre-designed, configurable circuit descriptions. The acronym IP stands for »intellectual property«.

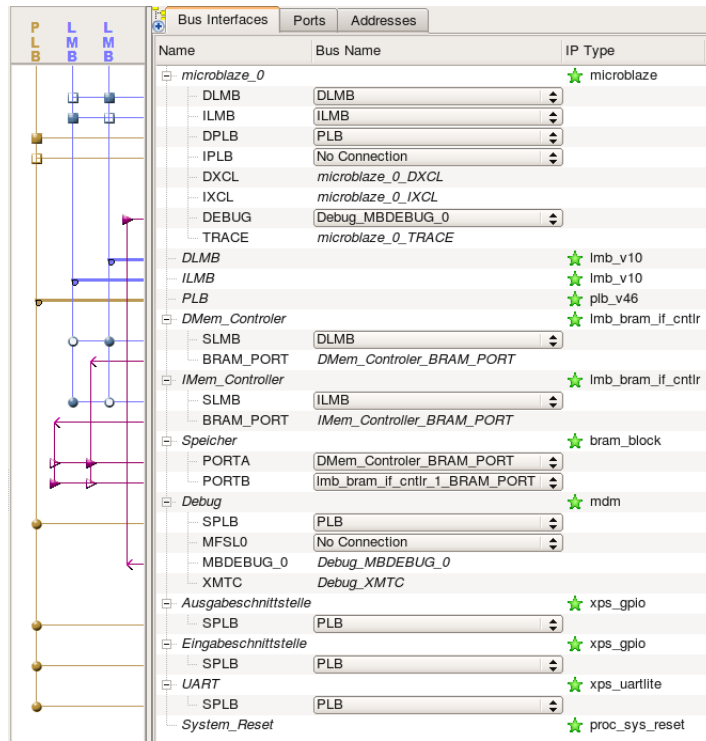


Figure 4: Renaming and bus connections

at the cross points at the graphic. Each memory controller has to be linked to the memory bus, each port of the dual port memory with the red bus connector of the memory controllers and the three interface circuits with the peripheral bus (PLB). The debug interface has to be connected with peripheral bus and a special debug port of the processor.

The peripheral interfaces have to be configured. For the output interface for »Channel 1«, tab »User« data width has to be reduced down to 8 (figure 5 a). In addition »Channel 1 Data Default Value« should be changed so that after initialization four LEDs are on and four LEDs are off. As shown in figure

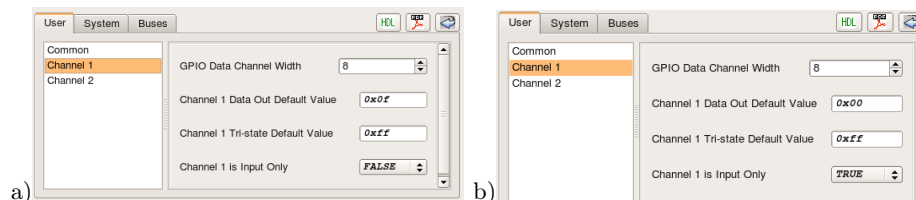


Figure 5: Configuration a) output interface b) input interface

5 b for the input interface the data width should be also reduced down to eight and for »Channel 1« the flag »Input Only« has to be set to true.

For the UART (universal asynchronous receiver transmitter) the setting in the menu »User« in figure 6 a can be kept or changed. It is only important that the settings conform with those in the terminal program later used for communication with the PC. In menu »System, PLB« figure 6 b, clock frequency has to be reduced to »50000000Hz« (50 MHz)³.

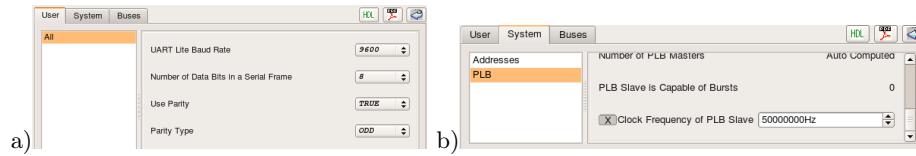


Figure 6: UART settings a) baud rate und transmission parameters b) clock frequency

The configuration of the debug port can be kept unchanged and for the processor via »Config IP« tab »Debug« »Enable Microblaze Debug Module Interface« has to be selected. Otherwise during program start of SDK an error will arise.

1.3 External Connections

Change to the view »Ports (figure 7). The circuit in figure 1 has a clock input (Clk), an initialization input (Reset), a serial input (RxD), a serial output (TxD), an 8-bit output vector »LD« to the LEDs and an 8-bit input vector »SW« from the switches. First by button »Add External Port« top right for each interface signal a declaration line has to be added and then to be adjusted.

The names in column »External Ports« must be the same as in the constraint file. In column »Net« the internal name has to be declared, normally the same as for the port. In column »Direction« »I« means »input« and »O« means »output«. For both bitvectors in column »Range« the index range should be specified ascending as the data. In column »Class« clock and initialization signals has to be labeled as special signals. Next the external ports are connected to the computer components by entering there names in corresponding menus.

³First click at the »X«. Afterward the frequency value can be edited.

Name	Net	Direction	Range	Class	IP Type	Reset Polarity	Frequency
External Ports							
Clk	Clk	I		CLK			50000000
Reset	Reset	I		RST		1	
LD	LD	O	[7:0]	NONE			
SW	SW	I	[7:0]	NONE			
RxD	RxD	I		NONE			
TxD	TxD	O		NONE			

Figure 7: External Ports

To the data, instruction and peripheral bus the clock and the initialisation signal, preprocessed by the system reset unit are connected. The output port has to be connected to the LEDs, the input port to the switches and the UART with serial signals »RxD« and »TxD«. The internal reset signal, provided by the debug interface has to be connected to the corresponding input of the system reset unit. The »Slowest_sync_clk« for sampling and time alignment is the clock »Clk« and the external reset signal is reset »Reset«. The unused additional reset input »Aux_Reset_In« has to be connected to »net_gnd« and the unused input »Dcm_locked«⁴ to »net_vcc«.

1.4 Define the Address Ranges

In tab »Adresses« to the data memory controller, the instruction memory controller, the parallel, the serial and the debug interface address ranges has to be defined (figure 9). With the button »Generate Adresses« default values are

⁴Using a DCM this signal delays the deactivation of the internal reset signal until the delay loop is locked.

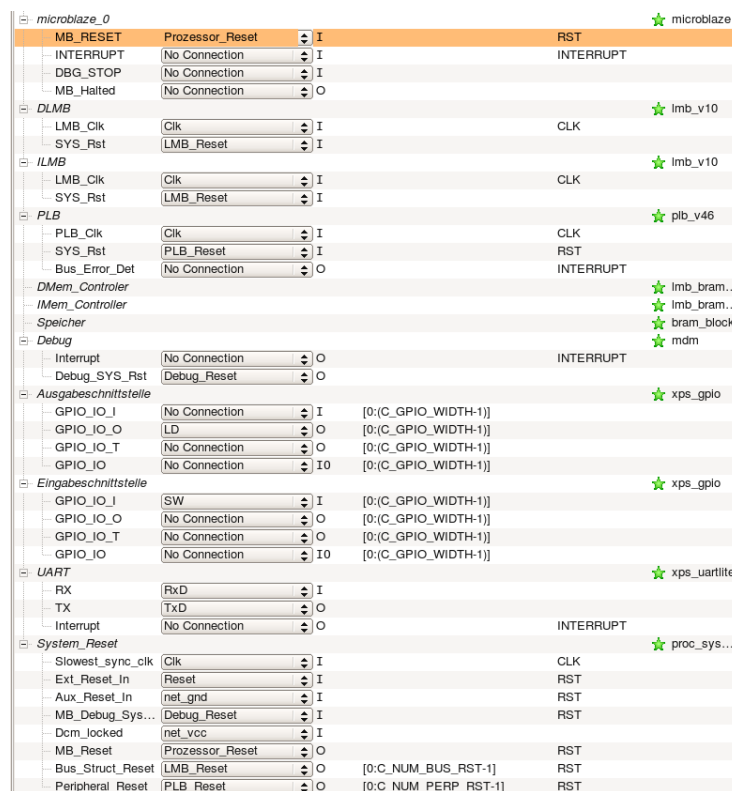
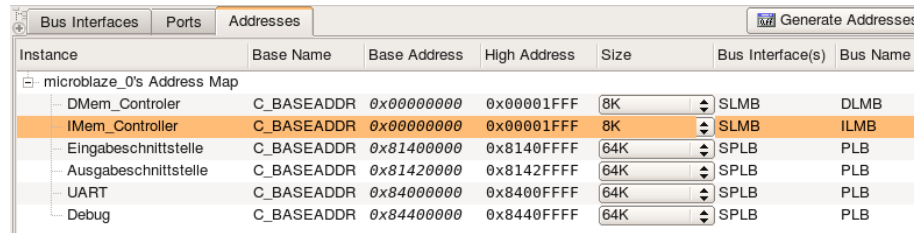


Figure 8: Connecting the external Ports with the System Components

generated. The smallest allowed size for block memories is 8 kByte. The address ranges for instructions and data should be the same. The peripheral units need non-overlapping address ranges. For the parallel interfaces the smallest selectable »Size« value is big enough. For the UART a reservation of an address range smaller than 2 kByte causes malfunctions. The default values of 64 kByte per device can also be retained. For the total address range is 2^{32} Byte, so that the oversized device address ranges will not cause a shortage.



Instance	Base Name	Base Address	High Address	Size	Bus Interface(s)	Bus Name
microblaze_0's Address Map						
DMem_Controller	C_BASEADDR	0x00000000	0x00001FFF	8K	SLMB	DLMB
IMem_Controller	C_BASEADDR	0x00000000	0x00001FFF	8K	SLMB	ILMB
Eingabeschnittstelle	C_BASEADDR	0x81400000	0x8140FFFF	64K	SPLB	PLB
Ausgabeschnittstelle	C_BASEADDR	0x81420000	0x8142FFFF	64K	SPLB	PLB
UART	C_BASEADDR	0x84000000	0x8400FFFF	64K	SPLB	PLB
Debug	C_BASEADDR	0x84400000	0x8440FFFF	64K	SPLB	PLB

Figure 9: Defining address ranges

1.5 Creating the Constraint File

As in the designs with »Xilinx-ISE« the (user) constraint file (UCF) assigns package pins to interface signals, timing and other constraints. It is important to specify the clock period, so that the synthesis can check, whether the circuit is fast enough. The constraint file will be selected via the tab »Project Files« and select »UCF File data/system.ucf«. Figure 10 shows the required entries for the example design.

1.6 Finishing the Hardware Design

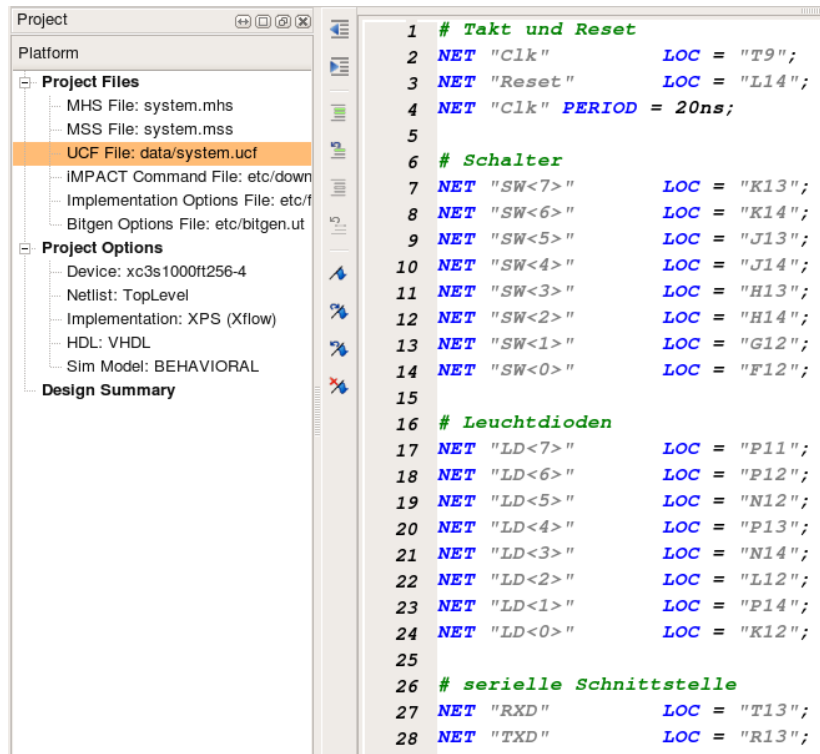
The software design should be carried out by the »eclipse« based development environment »SDK«:

- »Project« ▷ »Export Hardware Design to SDK ...«
- »Export and Launch SDK«

The synthesis, placement and wiring lasts several minutes. After finishing successfully a sub-directory »SDK/SDK_Export/hw/« is created, containing all hardware specific information including the bit-file for programming, software driver for peripheral components etc. SDK can also be launched directly from the menu bar under linux by

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

After launch the working directory has to be changed to »SDK/SDK_Workspace« of the current »EDK« project, where »SDK« will find the file ».metadata« of the project.



```

1 # Takt und Reset
2 NET "Clk" LOC = "T9";
3 NET "Reset" LOC = "L14";
4 NET "Clk" PERIOD = 20ns;
5
6 # Schalter
7 NET "SW<7>" LOC = "K13";
8 NET "SW<6>" LOC = "K14";
9 NET "SW<5>" LOC = "J13";
10 NET "SW<4>" LOC = "J14";
11 NET "SW<3>" LOC = "H13";
12 NET "SW<2>" LOC = "H14";
13 NET "SW<1>" LOC = "G12";
14 NET "SW<0>" LOC = "F12";
15
16 # Leuchtdioden
17 NET "LD<7>" LOC = "P11";
18 NET "LD<6>" LOC = "P12";
19 NET "LD<5>" LOC = "N12";
20 NET "LD<4>" LOC = "P13";
21 NET "LD<3>" LOC = "N14";
22 NET "LD<2>" LOC = "L12";
23 NET "LD<1>" LOC = "P14";
24 NET "LD<0>" LOC = "K12";
25
26 # serielle Schnittstelle
27 NET "RXD" LOC = "T13";
28 NET "TXD" LOC = "R13";

```

Figure 10: : Required entries in the constraint file

2 Software Development under SDK

Create a software platform by:

- »File« ▷ »New« ▷ »Software Plattform ...« ▷ »Add Software Application Project ...«

As in figure 11 a, the software platform should be named »SW_Plattform«. During finishing in the sub-directory »processor/include« of the software platform the hardware specific libraries for software development are created (figure 11 b).

After creating the software platform a project has to be created:

- »File« ▷ »New« ▷ »New Managed Make C Application Project« ▷ select a name, select »Empty Application« ▷ continue clicking until finishing

In figure 12 left the name of the new project is »Einfuehrung« (engl. introduction). Within the project a programming file has to be created:

- »File« ▷ »New« ▷ »Source File« ▷ select a name ending with ».c« and finish

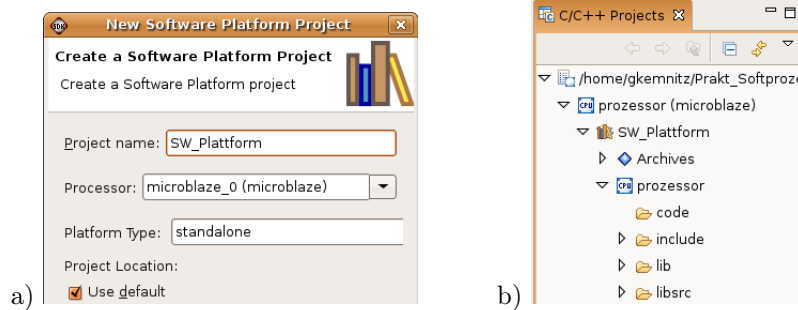


Figure 11: a) Creating a software platform b) Object tree of the software platform

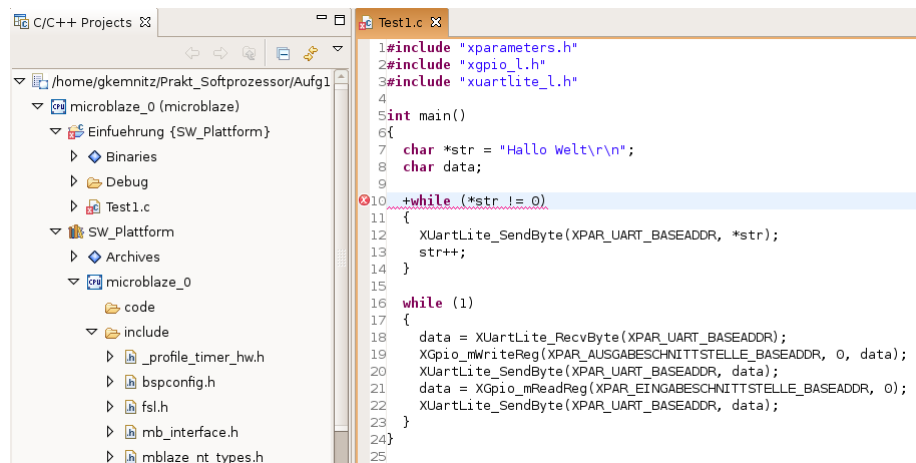


Figure 12: : Project example with a programming fault

The newly created file, in the example »Test1.c«, will, as shown in figure 12 left, be stored in the project directory. Open it by double click to edit the program. By storing, the program will be compiled automatically. Detected syntax faults are displayed in the source code. In figure 12 line 11 contains a fault – the »+« in front of »while« – that has to be removed before an executable program will be produced.

2.1 The programming Example

The example program in figure 12 starts with the include-instructions:

```

#include "xparameters.h"
#include "xgpio_1.h"
#include "xuartlite_1.h"

```

to import the required constants, makros and subroutines to control the peripheral interfaces. The project uses from »xparameters.h« the base addresses of the input and output interfaces

```

XPAR_AUSGABESCHNITTSTELLE_BASEADDR
XPAR_EINGABESCHNITTSTELLE_BASEADDR
XPAR_UART_BASEADDR

```

from »xgpio_1.h« the functions and macros for parallel input and output

```

XGpio_mWriteReg(BaseAddress, RegOffset, Data)
XGpio_mReadReg(BaseAddress, RegOffset)

```

and from »xuartlite_1.h« the functions and macros for serial input and output

```

XUartLite_RecvByte(XPAR_UART_BASEADDR)
XUartLite_SendByte(XPAR_UART_BASEADDR, data)

```

The name of the main program that takes control after starting the embedded system is in »c« always »main«. In the example at the begin of the programm the string constant »Hallo Welt\r\n« is created in the instruction memory and a pointer to the begin of the string and a variable to store a single character are declared:

```

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

```

A string in »c« always ends with a true »0«. The two control characters »\r\n« have the values »0c0D« and »0x0A« (see ASCII table in figure 13). All up the string »Hallo Welt\r\n« corresponds to the number sequence »0x48 0x61 0x6c 0x6c 0x6F 0x20 0x57 0x65 0x6c 0x74 0x0D 0x0A 00«⁵. The following while loop sends characters and increases the pointer until the character value is zero. In this way the whole character sequence including line feed is sended to the PC:

```

while (*str != 0)
{
    XUartLite_SendByte(XPAR_UART_BASEADDR, *str);
    str++;
}

```

In each run of the following infinite loop the program waits on a byte from the serial interface, displays the byte value on the LEDs, sends the byte back to the PC, reads a byte value from the switches and sends it also to the PC. To return the space character between two received characters, the switches must be set to 0x20:

⁵Please check it with the ASCII table in figure 13 an later during step operation (pausing execution after each instruction).

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

Source: www.LookupTables.com

Figure 13: ASCII table for translation between characters and character values

```

while (1)
{
    data = XUartLite_RecvByte(XPAR_UART_BASEADDR);
    XGpio_mWriteReg(XPAR_AUSGABESCHNITTSTELLE_BASEADDR, 0, data);
    XUartLite_SendByte(XPAR_UART_BASEADDR, data);
    data = XGpio_mReadReg(XPAR_EINGABESCHNITTSTELLE_BASEADDR, 0);
    XUartLite_SendByte(XPAR_UART_BASEADDR, data);
}
}

```

2.2 Program Test

Before testing the program

- connect the test board to the power supply and the programming cable and load the hardware configuration in the programmable logic circuit by
- »Tools« ▷ »Program FPGA« ▷ »Save and Program«

Before starting the program it is also useful to start the serial port terminal on the PC:

- »Linux Menu bar« ▷ »Anwendungen« ▷ »Zubehör« ▷ »Serial port terminal«

Within the terminal program adapt the port settings via:

- »Configuration« ▷ »Port«

The port settings must be the same as for the soft processor, in the example speed »9600«, parity »odd«, bits »8«, stopbits »1« and flow control »none«. Connecting the testboard to the PC via a normal serial cable the port name is generally »/dev/ttyS0«. Using an USB to serial adapter look before and after plugging in the cable what the newly created device port is. If no transmission can be established, the wrong port may be selected or an other program has locked the port. In the last case the other program has to be closed and the serial port monitor started again. The reason of eye-catching data corruptions is generally wrong baud rate setting. Alternatively also a wrong clock frequency of the UART in figure 6 b may be the source of the malfunction.

After a successful download and the start of the serial port monitor start the program with

- »Run« ▷ »Run ...« ▷ ... ▷ select »Debug/Einfuehrung.elf«

After start and pushing the reset button »BTN3« the program sends to the serial port terminal the text »Hallo Welt«, a newline and a carriage return, waits in each iteration on a character from the PC, displays the character value on the LEDs, returns the received character to the PC, reads the 8-bit value from the switches and sends it as a character also to the PC. Figure 14 shows as an example the monitor output for the input sequence »abzd<Enter>a109« and switch setting 0x20 (space character).

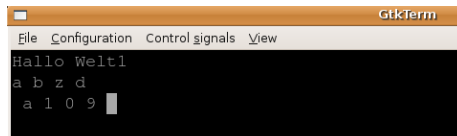


Figure 14: Example of the test output displayed by the serial port terminal

Transmitter and receiver of UART, each have an 8 character buffer, witch by chance may be filled with characters. This may cause that the program may receive or transmit unexpected characters after (re-) starting the program.

2.3 Debugger

As in section before, before starting the debugger the hardware configuration has to be downloaded and the serial port terminal on the PC has to be started. The debugger is started with :

- »Run« ▷ »Debug ...« ▷ ... ▷ select »Debug/Einfuehrung.elf« ▷ »Debug«

A usual way for a systematic initial test of a program is to run it in jog operation or little pieces. For this the program will be stopped after each instruction to check the intermediate data. The debugger has the following mechanism to stop a program:

- »Run« ▷ »Step Into«: stops after the next instruction, in case of a subroutine call after the first instruction of the subroutine.
- »Run« ▷ »Step Over«: stops after the next instruction, in case of a subroutine call after returning from the subroutine.
- set a break point: »right mouse click before the line number« ▷ »Toggle Breakpoint«
- start/continue programm (»Run« ▷ »Resume«): stops at the next breakpoint.

For break points also a condition and an ignore count can be added, that must be true or counted down to zero, respectively, to interrupt the program. Though, the program may pass break points in loops several time before stopping. To visualize the value of a variable during a halt of the program place the mouse pointer over the variable. In figure 14 in the yellow window, pointer (address value) »str« and the string constant starting at this address are displayed. During the debugging of the example program keep in mind, that the function »XUartLite_RecByte(...)« within the inner loop waits always on a character from the PC. Without this input or without running the serial port terminal the debugger waits eternally.

3 Exercises

Aufgabe 1.1: EDK-Referenzentwurf

Do the hardware design in the described ways with EDK.

Aufgabe 1.2: SDK-Referenzentwurf

Export the hardware design to SDK and carry out the software design in the described ways with SDK.

1. First test the program with »run«.
2. Repeat the test with the debugger in step operation.
 - Write down the value of the pointer »str« and the value it points to in each iteration of the first while-loop.
 - Set a break point in front of the instruction that read the switches. Check whether the program stops before or after the break point.

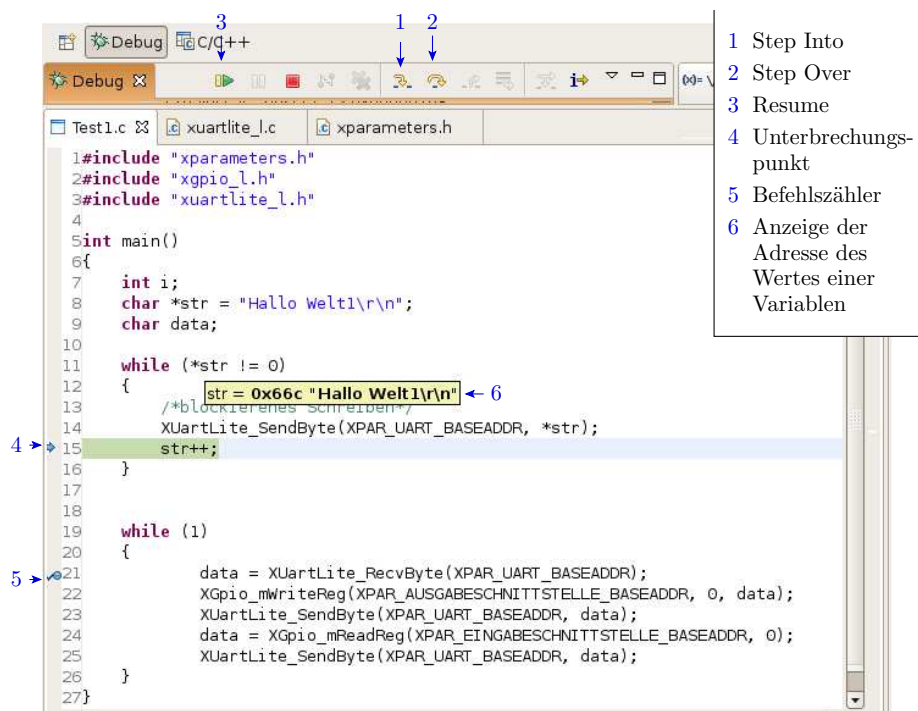


Figure 15: Program surface of the debugger

Aufgabe 1.3: Erstes eigenes Programm

Change the program so that the bytes received from the UART and displayed at the LEDs are processed in some way e.g. combined bitwise EXOR with the data read from the switches. Fill in the following tabular with test cases and carry out the tests.

character recieved by the UART	character value	switch input	LED-output
'a'		0x43	
'1'		0x65	

4 Questions for self-monitoring

- How long needs the transfer of the word »Hallo« with the UART-settings: 9600 baud, one start bit, eight data bits, one parity bit and one stop bit?
- Write your name, followed by a line feed as an ASCII-String.