

# Lab 14 Writing Basic Software Applications Lab: MicroBlaze



# Writing Basic Software Applications Lab: MicroBlaze

# Introduction

This lab guides you through the process of writing a basic software application. The software will write to one of the OPB GPIOs; LEDs. Xilinx Platform Studio (XPS) will write the code and create an MSS file for LibGen.

# **Objectives**

After completing this lab, you will be able to:

- Write a basic application to access an IP peripheral
- Utilize XPS to generate a MSS file
- Generate a bit file
- Download the bit file and verify in hardware (if hardware is available)
- Develop a simple linker script

# Procedure

The first three labs defined the hardware for the processor system. This lab comprises several steps, including the writing of a basic software application to access one of the peripherals specified in Lab2mb. Below each general instruction for a given procedure, you will find accompanying step-by-step directions and illustrated figures providing more detail for performing the general instruction. If you feel confident about a specific instruction, feel free to skip the step-by-step directions and move on to the next general instruction in the procedure.

**Note:** If you are unable to complete the lab at this time, you can download the lab files for this module from the Xilinx University Program site at http://university.xilinx.com



# **Opening the Project**

Step 2

- Create a **lab14mb** folder in the **c:\xup\embedded\labs** directory. If you wish to continue with your completed design from lab3 then copy the contents of the **lab3mb** folder into the **lab4mb** folder, otherwise, if you wish to start with a known good design, then copy the contents of c:\xup\embedded\mb\_completed\lab3mb into the **lab4mb** directory.
- Open XPS by clicking Start → Programs → Xilinx Platform Studio → Xilinx Platform Studio
- Olick File → Open Project and browse to the project which in the directory: c:\xup\embedded\labs\lab14mb
- Click system.xmp to open the project

### **Creating a BSP**

Specify the MicroBlaze<sup>TM</sup> processor standalone operating system and driver interface level.

The BSP is created based upon the peripherals included in the design.

● Double-click microblaze\_0 from the System BSP hierarchy, as shown in Figure 10c-1. You can also open the same dialog box by clicking Project → Software Platform Settings



🚔 Xilinx Platform Studio - Ci\Users\Lab\Desktop\shxkh\system.xmp - [System Assembly View]				- 10 - X
🔶 File Edit View Project Hardware Software Device Configuration Debug Sime	alation Window	Help		_ 8 1
□ @ P @ P @ D D D X D D X @ P @ D @ P @ D D D D D D D D D D D D D		No 🔊 🗠 📩 😒 🗤 🕹	H 🛛 💥 🕞 🕅 🗉 😫	II N?
Applications ↔ □ & ×	PLI	Bus Interfaces Ports	Addresses	Bus Interface Filters
Software Projects	LMM	Name Bus Name	IP Type IP Version	By Connection
- Dia Add Software Application Project	<u> </u>	dimb	☆ Imb_v10 1.00.a	Connected
Default: microblaze_0_bootloop	0	ilmb	🙀 Imb_v10 1.00.a	Unconnected
Default: microblaze_0_xmdstub	0	mb_plb	🛊 plb_v46 1.04.a	E by bus standard
Project: TestApp_Memory_microblaze_0	1444	microblaze_0	microblaze 7.30.a	PLBV46
<ul> <li>Processor: microbiaze_0</li> <li>Everytable: (\) [rest\] ab\Deckton\shikb\Test\u00e0on Memory microblaze () everytable eff</li> </ul>		(1) diab calls	pram_block 1.00.a	FSL
Compiler Options	225	alimh catir	Imb hram i 210 h	Xilinx Point To Point
Sources		(i) mdm_0	🙀 mdm 1.00.g	By Interface Type
C:\Users\Lab\Desktop\shxkh\TestApp_Memory_microblaze_0\src\TestApp_Memory.c	5	BTNs_4Bit	🙀 xps_gpio 2.00.a	V Slaves
Headers	9	∃ DIPs_48it	🙀 xps_gpio 2.00.a	Master Slaver
Project: TestApp_Peripheral_microblaze_0	9	E LEDs_88it	xps_gpio 2.00.a	Monitors
Everytable: () Urger\Lab\Deckton\chukb\Tertänn Beinheral microblaze fileverytable eff		(i) PC232_DUE	W xps_uartite 1.01.a	Targets
Compiler Ontions		disck gener	clock gene 400 a	Initiators
Sources		proc_sys_re	proc_svs_re 2.00.a	
Headers			A 1	
	leterd	b		
e >	Master Slave	Master/Slave Target (Init License (paid)	iator Connected OUnconnected al) Recal Pre Production Beta III	Development
🗢 Project 🐟 Applications 🐟 IP Catalog	Start Up Pag	pe 🔝 🌋 Design Summary	🕒 🔶 Block Diagram 💽 🔶 System Ass	embly View 🔟
Console				+08
Saving bit stream in "system.bit". Bitstream generation is complete. Done!				
Console A Warnings O Errors				,
🚳 🖉 🚞 o 🐟 🖭 🧭				- 🐚 🌆 🐠 10:59 AM

Figure 10c-1. Opening S/W Settings for the microblaze\_0 Instance



This will open the Software Platform Settings dialog with the Software Platform tab.

Software Platform Settings Depreca	A E B PO DA DE A A E BR B X E ted		
Processor Information			
pplicat laze_0_ Processor Instance: microblaze_0 💌			
aze_0_: p_Mer Software Platform			
oblazeOS and Lib Configuration	Processor Settings		
sers\La Drivers	CPU Driver: cpu	CPU Driver Ve	rsion: 1.12.b
)\Deskt	Processor Parameters:		
	Name Current Value	Default Value Type	Description
p_Peri	imicroblaze_0		
oblaze_	compiler mb-gcc	mb-gcc string	Compiler used to compile both BSP
15	extra coa	-a string	Extra compiler flags used in BSP and
	xmdstu none	none peripheral_in	sta Debug peripheral to be used with x
	OS & Library Settings		
	OS & Library Settings OS: standalone v Version: skarnet	3.00.a 💌 Standalone i	s a simple, low-level software platform. It
	OS & Library Settings OS: standalone v Version Silkernel Use standalone Correry	3.00.a 💌 Standalone i Version Description	s a simple, low-level software platform. It
	OS: 8.Lbrary Settings OS: standalone Version Vikernel Use standalone vitersion vitersi	3.00.a V Standalone i Version Description 1.00.a V Xilinx Memori	s a simple, low-level software platform. It
	OS & Library Settings OS: standalone Version Vikernel Use standalone vilimfs Xilliff Standalone vilimfs	3.00.a  Version Description 100.a  Xilinx Memor 200.a  Version	y File System em and Serial Flash Library
	OS & Library Settings OS: standalone v Version: Xilennel Use standalone villa xillash xillash xillash	3.00.a Standalone i Version Description 1.00.a Xilimx Memor 2.00.a Xilimx In-syst 2.00.a Xilimx In-syst 1.00.a Provides rea	s a simple, low-level software platform. It ry File System em and Serial Flash Library Drary for Intel/AMD CFI compliant parallel Vivrite routines to access files stored on a
	OS & Library Settings OS: standalone v Version <u>wikened</u> Use standalone in wilinfs wilinfs wilifash wilifash wilifash wilifash	3.00.a           Standalone i           Version         Description           1.00.a         Xilinx Memory           2.00.a         Xilinx In-syst           2.00.a         Xilinx In-syst           2.00.a         Xilinx In-syst           2.00.a         Willinx In-syst	s a simple, low-level software platform. It y File System em and Serial Flash Library brary for Intel/AMD CFL compliant parallel //write routines to access files stored on a tack library: WiP v1.3.0, Xillinx adapter v2.0
	OS & Lbrary Settings OS: standalone v Version Miterned Use silmfs xiliaf xiliaf xiliaf kvij130	3.00.a V Standalone i Version Description 1.00.a V Xilinx Memor 2.00.a Xilinx Flash li 1.00.a Provides read 2.00.a W Provides read 2.00.a W Provides read	y File System and Serial Flash Library mand Serial Flash Library for Intel/AMD CFI compliant parallel //write routines to access files stored on a tack library: IwIP v1.3.0, Xilinx adapter v2.0
	OS & Library Settings OS: standalone version Use standalone version vilifar vilifar vilifar vilifarfs lwip130	3.00.a V Standalone i Version Description 1.00.a V Jilnx Memor 2.00.a Jilnx In-syste 2.00.a Jilnx In-syste 1.00.a Provides reas 2.00.a V INP TCP/IP S	y File System m and Serial Flash Library Dary for Intel/AMD CFI compliant parallel //write routimes to access files stored on a tack library: Iw/P v1.3.0, Xilinx adapter v2.0
	OS & Ubrary Settings OS: standalone Version: Nikernel Use Standalone villef villef villef villef villes kvifafs kvip130	3.00.a     Standalone i       Version     Description       1.00.a     Xilinx Memorization       2.00.a     Xilinx Ihasin       1.00.a     Provides real       2.00.a     IwiP TCP/IP S	y File System em and Serial Flash Library brary for Intel/AMD CFI compliant parallel Varite routines to access files stored on a tack library: IwJP v1.3.0, Xilinx adapter v2.0
catoone	OS & Library Settings OS: standalone Version Standalone Version Use standalone very Sillinf standalon Sillifsh Sillifsh Sillifsh Iwip130	3.00.a Standalone i Version Description 1.00.a Xilinx Memoi 2.00.a Xilinx In-syst 2.00.a Xilinx In-syst 1.00.a Nilinx Fissh 1.00.a Nilinx Fissh 2.00.a Nilinx Fissh 2.00.a Nilinx Fissh 2.00.a Nilinx Fissh 2.00.a Nilinx Fissh 3.00.a Nilinx Fissh 3.	s a simple, low-level software platform. It y File System em and Serial Flash Library brary for Intel/AMD CFI compliant parallel /write routines to access files stored on a tack library: IwIP v1.3.0, Xilinx adapter v2.0
cations	OS & Lbrary Settings OS: standalone version witcerice Use standalone version xiliaf xiliaf xiliaf wiffash kvip130	3.00.a V Standalone i Version Description 1.00.a Xilinx Memor 2.00.a Xilinx Tesh li 1.00.a Provides read 2.00.a W MP TCP/IP S	y File System m and Serial Flash Library brary for Intel/AMD CFI compliant parallel //write routines to access files stored on a tack library: IwIP v1.3.0, Xilinx adapter v2.0
ators	OS & Library Settings OS: standalone Version Use standalone vitings Use standalone vitings Standalone viting	3.00.a V Standalone i Version Description 1.00.a V Xilinx Memor 2.00.a Xilinx Fissh i 1.00.a Provides reas 2.00.a IwiP TCP/IP S	y File System m and Serial Flash Library bary for Intel/AMD CFI compliant parallel //write routimes to access files stored on a tack library: IwIP v1.3.0, Xilinx adapter v2.0

Figure 10c-2. Software Platform Settings for the microblaze\_0 Instance

• In the **Software Platform** tab, the **Driver** can be selected for each of the peripherals in the system. You can also select the **Kernel and Operating Systems** for each of the processor instances. In addition, supporting libraries can be selected if they will be used. Make sure that the settings are as displayed in Figure 10c-2



• Click on the **Processor and Driver Parameters** tab and specify the following parameters:

#### **Processor Parameters: Instance**

compiler – mb-gcc archiver – mb-ar EXTRA\_COMPILER\_FLAGS – -g xmdstub\_peripheral – none CORE\_CLOCK\_FREQ\_HZ – 50000000 Driver Parameters: Instance

Leave Blank

ocessor Instance: microblaze_0 v						
tware Platform and Lib Configuration	Drivers Configuration:					
vers	Peripheral	Hw Version	Instance	Driver	Version	
	Imb bram i	2.10.b	dimb_cntir	bram	- 2.00.a	
	Imb bram i	2.10.b	ilmb cntlr	bram	▼ 2.00.a	
	xps uartlite	1.01.a	RS232 DTE	uartlite	▼ 2.00.a	Ţ
	xps_uartlite	1.01.a	RS232_DCE	uartlite	▼ 2.00.a	
	xps_qpio	2.00.a	LEDs_8Bit	gpio	▼ 3.00.a	
	xps_gpio	2.00.a	DIPs_4Bit	gpio	▼ 3.00.a	
	xps_qpio	2.00.a	BTNs_4Bit	gpio	▼ 3.00.a	
	mdm	1.00.g	mdm_0	uartlite	🗶 2.00.a	
	Driver Parameters:					
	Name					
	bram : dlmb_	cntlr				
	bram : ilmb_o	ntlr				
	uartlite : RS23	2_DTE				
	uartlite : RS23	2_DCE				
	ED O	Bit				
	gpio:LEDs_8					
	gpio : LEDS_8	Sit Di+				
	gpio : DIPs_4 gpio : DIPs_4 gpio : BTNs_4	Bit D				
	gpio : LEUS_8 gpio : DIPs_4 gpio : BTNs_4 uartlite : mdn	Sit Bit n_0				
	gpio : LED5_4 gpio : DIPs_4t gpio : BTNs_4 uartlite : mdn	sit Bit n_0				
	gpio : LEDS 8 gpio : DIPs_4t gpio : BTNs_4 uartlite : mdn	sit Bit n_0				
	gpio : LEUS_a gpio : DIPs_4t gpio : BTNs_4 uartlite : mdn	sit Bit n_0				
	gpio : LEVS_0 gpio : DIPS_4t gpio : STNS_4 uartlite : mdn	sit Bit n_0				
	gpio : LDVs_4 gpio : DVPs_4 gpio : BTNs_4 uartlite : mdn	sit Bit 1_0				
	gpio : EUS_6 gpio : DIS_4 gpio : BTNs_4 uartlite : mdn	sit Bit 1_0				
	gpio : LDVs_0 gpio : DDVs_4 gpio : BTNs_4 uartlite : mdn	sit Bit n_0				

Figure 10c-3. Processor and Driver Parameters Tab of the Software Platform Settings for the microblaze\_0 Instance

Click the Library/OS Parameters tab



Processor Information						
Processor Instance: microblaze_0 💌						
Software Platform						
OS and Lib Configuration	Drivers Configuration:					
Drivers						
	Peripheral	Hw Version	Instance	Driver	Version	
	Imb_bram_if_cntIr	2.10.b	dlmb_cntlr	bram		
	Imb_bram_if_cntlr	2.10.b	ilmb_cntlr	bram	2.00.a	-
	xps_uartite	1.01.a 1.01.5	R5232_DTE	uartite	2.00.a	
	- xps_datate	2.00 a	LEDs 8Bit	anio	= 3.00 a	
	xps gpio	2.00.a	DIPs 4Bit	gpio	¥ 3.00.a	
	xps_gpio	2.00.a	BTNs_4Bit	gpio	🖵 3.00.a	
	mdm	1.00.g	mdm_0	uartlite	🗨 2.00.a	
	Driver Parameters:					
	Driver Parameters:					
	Driver Parameters: Name - bram : dlmb_entir					
	Driver Parameters: Name bram : ilmb_cntir bram : ilmb_cntir					
	Driver Parameters: Name bran: ilmb_cntr bran: ilmb_cntr uartice R5232_DTE uartice R5232_DTE					
	Driver Parameters: Name bram: dimb_cntir bram: ilmb_cntir curitire: rR523, DTE uaritire: R523, DCE corio: IEDe R84					
	Driver Parameters: Name bram : ilmb_cntir bram :					
	Driver Parameters: Name bram: ilmb_cntlr bram: ilmb_cntlr uaritite: RS232_DTE uaritite: RS232_DCE grio: LID>_8Bit grio: LID>_8Bit grio: LID>_4Bit					
	Driver Parameters: Name bram: ilmb_cntlr bram: ilmb_cntlr - uaritite: R3222 DTE - gpio: LBD_28Bit - gpio: DB_24Bit - gpio: TIN2_4Bit - uaritite: m40_0					
	Driver Parameters: Name bram : ilmb_cntlr bram : ilmb_cntlr bram : ilmb_cntlr bramitire : R5222, DTE gio: LDD_x88it gpio: LDD_x88it gpio: DID_x48it brance : mdm_0					
	Driver Parameters: Name bram: dimb_cntir bram: ilmb_cntir curifice :R532, DTE ourifice :R532, DTE gric: LED_s Bit gric: LED_s Bit gric: EDS_Bit gric: BTNs_46it uartilite : mdm_0					
	Driver Parameters: Name bram : ilmb_cntir bram : ilmb_cntir uaritie : R5232_DTE grite : R5232_DTE grite : R5232_DCE grite : LEDs_8Bit grite : LEDs_8Bit grite : DPs_4Bit brance : mdm_0					
	Driver Parameters: Name bram : Ilmb_cntir bram : Ilmb_cntir cutifie : R5232_DTE quitifie : R5232_DTE gpio : LEDs_88it gpio : EIDs_88it gpio : EIDs_48it uartitie : mdm_0					
	Driver Parameters: Name - bram: dlmb_cntlr - bram: ilmb_cntl - uaritite: RS322_DCE - gpio: LB0_aBit gpio: DB7_4Bit - gpio: ETNs_4Bit - uaritite: mdm_0					
	Driver Parameters: Name bram : ilmb_cntir bram : ilmb_cntir bram : ilmb_cntir bramitire : R5323_DTE bramitire : R532_DTE bramitire : R532_DTE bra					
	Driver Parameters: Name bram: dimb_cntir bram: ilmb_cntir uaritire: R532.0TE gpio: LED_R Bit gpio: DIP_4Bit gpio: BIP_4Bit uaritire: mdm_0					
	Driver Parameters: Name bram : ilmb_cntir bram : ilmb_cntir uaritie : R5232_DTE grite : R5232_DTE grite : R5232_DCE grite : LEDs_8Bit grite : LEDs_8Bit grite : LEDs_8Bit uaritie : mom_0					

Figure 10c-4. Library/OS Parameters Tab of the Software Platform Settings for the microblaze\_0 Instance

- Click the Current Value field for stdin and select RS232. Similarly, click the Current Value field for stdout and select RS232. This will assign the uart device as the stdin and stdout. Base System Builder already set these when the RS232 peripheral was included. If a system does not have any stdin/stdout devices, then keep the current value as None. Leave the Current Value for need\_xil\_malloc as false because your application does not use any malloc function call
- Click **OK** to accept the settings



Generate the BSP.

• Double-click the **system.mss** file under the **System** tab in XPS, as shown in Figure 10c-5

This will open the MSS file for this project.



							-				
🖹 File	Edit	View	Project	Har	dware	Softwa	re	Device (	Config	uratio	n
🗋 🏓		7 🕹	5	• <b>F</b>	Ж	Ð B	$\times$	۱	2	A	
Project								↔ 🗆	₽×	$\overline{{ \exists \exists}}$	Г
Platform										ÞE	
🖨 Proje	ct File	s								_	
N	IHS Fil	e: systen	n.mhs								
···· N	ISS File	: system	n.mss							_	
	ADA CT	: data/sy	stem.uci	tc/do	wolese	d croad				=	
In	nplem	entation	Options	File: ef	tc/fast	runtime.	opt			2	
В	itgen (	Options F	File: etc/b	itgen.	ut						
🖨 Proje	ct Opt	tions								7.0	
D	evice:	xc3s700a	fg484-4							76	
- N	etlist:	TopLevel								*	
	npiem IDI • VE	entation IDI	: 762 (211	ow)						*	
- Si	im Mo	del: BEH	AVIORAL								
Desig	jn Sun	nmary								r 7 7	
										r 	
										r 	
										r 	
										r 	
										- - - -	
										-	

Figure 10c-5. System Tab



• XPS has written the parameters that are specified in the **Peripheral Options** to the **system.mss** file

?	1.	Lis
---	----	-----

List the assigned driver to each of the following peripheral instances:

mb_opb:	
debug_module:	
dlmb_cntlr:	
ilmb_cntlr:	
LEDs_8Bit:	
Push_Buttons_3Bit:	
DIP_Switches_8Bit:	
delay:	
RS232:	
opb_7segled_0:	

2. Why do some of the above mentioned devices not have a specific driver?

- Close the **system.mss** file
- Generate the **BSP** by clicking **Tools**  $\rightarrow$  **Generate Libraries** or click the button

This will run LibGen on the system.mss file to generate the BSP library files.



3. List the created subfolders, their contents, and their possible purposes under the **microblaze\_0** folder in the lab4mb folder.



### Creating a Basic C File

Add a software project and create a basic C file to write data to the **LEDs\_8Bit** peripheral.

• In the Applications Tab double Click on Software Projects

This will open a dialog box to create a new project



Figure 10c-6. New Project

- Enter the name **MyProj** and click **OK**.

This will open a dialog box to create a new project

This will open a new document in the XPS editor



Xilinx Platform Studio - C:\xup\embedded\labs\lab4mb\ - [System1]	
🙀 File Edit View Project Tools Options Window Help	- 8 ×
□[\$P\$ \$P\$ \$P\$ \$P\$ \$P\$ \$P\$ \$P\$ \$P\$ \$P\$ \$P\$	
	100
Bysteen Applications Cydons Symbols	100
Right Click for Options	
R System BSP	
b an microbiace_0 b Driver.cpu_v1_00_a	
Ostardelone v1.00.a	
B Generated Header: microblaz	
e — ma_oppo e — ≣ debug_module	
-T_L imb	
(e) 着 dimb_catir	
e → umo_com	
Push_Buttons_38t	
⊕ DP_SWitches_dbit     ⊕ E deley	
in ∰ opb_7segled_0 R M Project Files	
Inflis File: system.mhs	
MSS Fee: system.mss     BPD File: system.pbd	
G UCF File: deta/system.ud     MPACT Command File: etcl/dow	
Implementation Options File: etc\;	
a bargen Oppons File: enclosigen_si B GE Project Options	
Device:xc3e2001255-4     Notice: Tool aveit     Commentation aveit	
System1	
Compilina apia	_
Compiling functr Compiling sou	_
Libraria apparated in Chaus) apparticed labes labered principalizes (Nith) directory	
Eventues generated on experimentation of the set of the	
Valming Sales, garletale for 0.5 es. Dirvers and Eurones	
Done.	
U V Varings Errors	
Ready	1 4

Figure 10c-7. New Document in the XPS Editor

The first step is to add the header files for the required functions. All of the header files related to this project were placed in the microblaze\_0/include directory when LibGen was run.

- Add the following to the C file:
  - #include "xparameters.h"
  - #include "xgpio.h"

This will open the Save As dialog.

• Create a new directory named **code** in the **lab4mb** directory and save the file as **system.c**, as shown in Figure 10c-8

Save As		? 🛛
Save in: 🗀	code 💌 🗲 🔁	📸 🎟 -
File name:	system.c	Save
Save as type:	Text Files ( *.c *.cpp *.cc *.c++ *.cxx *.h *.s *	Cancel



#### Figure 10c-8. Save As Dialog

Add the following to the **C** file

main() {

The first step in main is to initialize the GPIO peripheral



Write a **main**() function that will initialize and set the data direction for the **LEDs\_8Bit** peripheral to output.

- From the Windows start menu Click Start → Programs → Xilinx Platform Studio 6.2i → EDK 6.2 Documentation
- Click the **Documents** link
- Scroll down and click Driver Reference Guide

This will open the **xilinx\_drivers.pdf** file.

- In Acrobat Reader, click and search for the XGpio\_Initialize function. You may have to click Find Next to observe the function description page
- This documentation contains a detailed description of the **XGpio\_Initialize** function
  - The documentation outlines two parameters that XGpio\_Initialize requires:
  - *InstancePtr* is a pointer to an XGpio instance. The memory that the pointer references must be preallocated by the caller. Further calls to manipulate the component through the XGpio API must be made with this pointer.
  - *DeviceId* is the unique ID of the device controlled by this XGpio component. Passing in a device ID associates the generic XGpio instance to a specific device, as chosen by the caller or application developer.
- Define an **XGpio** type variable named **gp\_out**. This variable will be used as the first parameter in the **Xgpio\_Initialize** function call
- Add the variable to the function call. It should now look like the following:

XGpio\_Initialize(&gp\_out,

The second parameter is the device ID for the device that you want to initialize. This information can be found in the **xparameters.h** file.

• Under the System BSP microblaze\_0 instance, double-click the **Generated Header:** microblaze\_0/include/xparameters.h entry, as shown in Figure 10c-9



Right Click for Options			
E System BSP			
i⊒⊷CPU microblaze_0			
Driver: cpu_v1_00_a			
Debug Peripheral: debug_module		I	
OS: standalone_v1_00_a			
Generated Header: microblaze_0/include/xparam	iete	rs	h
i∃			Γ
🗄 📲 debug_module		I	
→ imb		I	
→ dlmb			

Figure 10c-9. Double-click the Generated Header File

- LibGen writes the **xparameters.h** file, and the file provides critical information for driver function calls.
- This function call initializes the GPIO that is used as an output for the LEDs found on the board. In the *xparameters.h* file, find the following #define used to identify the LEDS\_8BIT peripheral:

*#define XPAR\_LEDS\_8BIT\_DEVICE\_ID 0* **••••••** Note: The number might be different

**Note:** The LEDS\_8BIT matches the instance name assigned in the MHS file for this peripheral.

• This #define can be used as the device ID in the function call.



• Add the **device ID** to the **function call** so that it looks like the following:

XGpio\_Initialize(&gp\_out, XPAR\_LEDS\_4BIT\_DEVICE\_ID);

The file should now look like Figure 10c-10.

```
00 #include "xparameters.h"
01 #include "xgpio.h"
02
03 main()
04 {
05 XGpio gp_out;
06
07 XGpio_Initialize(&gp_out, XPAR_LEDS_8BIT_DEVICE_ID);
```

Figure 10c-10. Partially Completed C File

Refer to the documentation to determine how to set all of the bits of the GPIO bus as outputs. This will involve using the XGpio\_SetDataDirection function call. Add code to perform this function, and then save the file.

XGpio\_SetDataDirection (&gp\_out, 1, 0x00);



Write a counter that continuously counts from 0 to 255 to drive the **LEDs\_8Bit** peripheral output. Output the current value of the counter by using the appropriate function. Write a software pause for loop to pause the output between each count.

• Write code to implement a counter that continuously counts from 0 to 255. The count will be used to drive the output of the LEDs\_8Bit peripheral

The following code is provided as an example:

while(1) { j = (j+1) % 256; }

You will also need to include the declaration of the variable j.

• Output the current value of **j** to the **LEDs\_8Bit** 



4. Using the information in the **gpio.h** documentation (the gpio information in the PDF file that is open), which function can be used?

• Add this function and the appropriate parameters to the C code

**Note:** Because the MicroBlaze<sup>TM</sup> processor does not have an internal timer or counter, a delay loop must be created in the software.

• Add the following code to create a software delay to pause between each count that gets displayed:



```
int i;
for(i=0; i<80000; i++);
```

• The final C program should look like Figure 10c-11

```
00
     #include
               "xparameters.h"
    #include "xgpio.h"
01
02
03
    main()
04
05
    XGpio gp_out;
06
    int i=0;
07
    int j=0;
08
09
         XGpio_Initialize (&gp_out, XPAR_LEDS_8BIT_DEVICE_ID);
10
         XGpio_SetDataDirection (&gp_out, 1, 0x00);
11
12
         while (1)
13
              ł
j
                = (j + 1) \% 256;
14
15
16
              //write the value of j to the LEDs
17
              XGpio_DiscreteWrite (&gp_out, 1, j);
18
19
20
21
22
              //software delay loop for pause
              for (i=0; i<10; i++);</pre>
              }
    }
23
24
```

Figure 10c-11. Final C Program

• Save and close the file



Select the **Application** Tab, add the source code (system.c from the code directory) and change the SW project TestApp so that **TestApp.c** is no longer initialized in BRAM. Compile the source code.

- Click the Application tab to view the current project's Compiler Options and Sources
  - Right-click Project: TestApp and unselect Mark to Initialize BRAMs
  - Verify that the small green arrow next to **Project: MyProj** does *not* have a red x through it. If it does, Right-click **Project: MyProj** and toggle **Mark to Initialize BRAMs**.





#### Figure 10c-12. Applications Tab for the Current Project

- Right-click Sources under Project: MyProj and select Add File
- Browse to the **code** directory under the current project (lab4mb) and select the **system.c** file
- **9** Double-click **Compiler Options** under **Project: MyProj** in the Application tab
- Click the **Directories** tab to set the directory options.
  - Change the **Output ELF File** entry to place the **executable.elf** file in the **microblaze\_0**\code directory under the current project directory.
  - For Linker Script Browse to the TestAppLinkScr file in the TestApp/Src directory.
  - The directory options should be set as shown in Figure 10c-13



Set Compiler :	settings for the project: MyProj 🛛 🛛 🔀
Environment 0	Dptimization Directories Advanced
All paths sho Multiple Optic Search Path	uld be relative to the Project Directory ons should be seperated by space s
Compiler (-B)	
Library (-L)	
Include (-I)	
Linker Option	ns
Libs to link (-l)	
Linker Script	b4mb\TestApp\src\TestAppLinkScr
Output Inform	nation
Output ELF F	s\lab4mb\microblaze_0\code\executable.elf
	OK Cancel

#### Figure 10c-13. Assigning the Destination Directory for the Executable File

- Click **OK** to accept the setting
- Compile the C code by clicking the button



5. After the program has compiled determine the sizes of the program sections:

.text section (code): .data section (variables): .bss section (heap and stack): Total size in decimal: Total size in hexadecimal:



# Linker Script

#### Step 4

Compile the C code by clicking the button. Change the directory to c:/xup/embedded/labs/lab4mb/microblaze\_0/ code. Execute the mb-objdump –h executable.elf command in the Xygwin shell and analyze the output.

- Start a Xygwin shell by clicking the button. This should place you in the project directory.
- Change the directory to c:/xup/embedded/labs/lab4mb/microblaze\_0/code by using the cd command
- Type **mb-objdump** –**h** executable.elf at the prompt in the **Xygwin** shell window to list various sections of the program, along with the starting address and size of each section



6. From the objdump output, complete the following memory map table:

Note: There are many sections are listed in the output, only some of them are being asked below

Section	Starting Address	Ending Address	Size in Hex
.text			
.rodata			
.sdata2			
.data			
.sdata			
.sbss			
.bss			
.bss stack			



7. Looking at the objdump output, list the sections that consume no memory.



Open the **TestAppLinkScr** file and change the stack size to **0x100**. Recompile the code, re-execute the objdump command, and analyze the output.

- In XPS, double-click the **TestAppLinkScr** file under **Compiler Options** in the **Applications** tab
- Change the stack size to **0x100**
- Save the linker script
- Recompile the program



• Execute the **mb-objdump** command in the **Xygwin** shell

?

8. From the object dump output, complete the following table:

Section	Starting Address	Ending Address	Size in Hex
.bss			
.bss_stack			



Define a heap space of **256** bytes, keeping the stack size to **256** in the linker script. Add the heap space in the linker script after the stack allocation, as shown below. Recompile the code, re-execute the objdump command, and analyze the output.

#### \_HEAP\_SIZE = 256; /\* heap size definition \*/

• Change the linker script file to include a heap definition of 256 bytes near the top of the linker script as shown below. To do this, double click on the **Linker Script** under **Project: MyProj** as follows:



Add the \_HEAP\_SIZE definition near the top as shown below:



• Add the three lines shown below to the .bss\_stack section of the linker script as shown below

83	.bss stack : {
84	= ALIGN(8);
85	heap = .;
86	heap start = heap;
87	. += HEAP SIZE;
88	$= A\overline{L}IGN(\overline{1}6);$
89	$heap_end = .;$
90	. += _STACK_SIZE;
91	$= A\overline{L}IGN(8);$
92	_stack = .;
93	
94	$\rightarrow ilmb_cntlr$
95	}
96	-

• Save the linker script



- Recompile the code
- Execute the **mb-objdump** command in the **Xygwin** shell



9. From the object dump output, complete the following table:

Section	Starting Address	Ending Address	Size in Hex
.bss			
.bss_stack			



Edit the **system.c** file to define a local variable, **k**, initialized with a value of **0**, and increment it by **1** inside the **for loop**. Recompile the code, re-execute the objdump command, and analyze the output.

• Change the **system.c** file to define a local variable and increment it in the **for loop** 

The code should look like the following:

main() {

XGpio gp\_out; int j=0; int i=0; int k=0; /\* add this statement \*/

XGpio\_Initialize(&gp\_out, XPAR\_LEDS\_8BIT\_DEVICE\_ID); XGpio\_SetDataDirection(&gp\_out, 1, 0x00);

- **❸** Save the **system.c** file
- **3** Recompile the code
- Execute the **mb-objdump** command in the **Xygwin** shell

?

10. From the object dump output, complete the following table:

Section	Starting Address	Ending Address	Size in Hex
.text			



11. Has the .text section changed? Why or why not?





Edit the **system.c** file to add a statement that will output the value of the variable **k** to the LEDs. Recompile the code, re-execute the objdump command, and analyze the output.

0 Change the **system.c** file to add a statement that outputs the value of variable **k** to the LEDs

The code should look like the following:

```
main() {
        XGpio gp_out;
        int j=0;
        int i=0;
        int k=0;
        XGpio_Initialize(&gp_out, XPAR_LEDS_4BIT_DEVICE_ID);
        XGpio_SetDataDirection(&gp_out, 1, 0x00);
        while(1) {
                 j = (j + 1) \% 16;
                 k = k + 1;
                 // write the value of j to the LEDs
                 XGpio DiscreteWrite(&gp out, 1, j);
                 XGpio_DiscreteWrite(&gp_out, 1, k);
                 // software delay loop for pause
                 for(i=0; i<10; i++);
                 }
```

Save the **system.c** file 0

}

- Recompile the code €
- 4 Execute the mb-objdump command in the Xygwin shell

10. From the object dump output, complete the following table:

Section	Starting Address	Ending Address	Size in Hex
.text			



?

11. Has the .text section changed? Why or why not?



# Verifying in Hardware (Optional)

#### Step 5

In the **system.c** file, comment out all lines which refer to or use the variable **k** and save the file. Connect the hardware board for which you have developed this lab (Spartan3 Starter Kit board). Create a bitstream file.

- Connect the included programming cable to the PC parallel port and the Spartan3 Board
- Connect a serial cable between the PC and the DB-9 connector on the board.
- Attach the included power supply to the board.
- **9** In the source code file, comment out all lines which refer to variable  $\mathbf{k}$
- Change i<10 to i<400000 in the for loop and save the file
- Recompile the source file
- Click Tools → Update Bitstream and implement and generate the BIT file

Download the generated bitstream file into the hardware board by using a Parallel programming cable. Verify that the board is programmed and the LEDs are turning ON and OFF in proper sequence.

- Download the generated bit file by clicking  $Tools \rightarrow Download$
- After the board is programmed, you will see that the LEDs are turning ON and OFF in the desired sequence
  - Note: The LEDs are connected in such a way that "1" will turn OFF the LED
- Turn off the power when done

### Conclusion

XPS can be used to define, develop, and integrate the software components of the embedded system. A device driver interface can be defined for each of the peripherals and the processor. XPS creates an MSS file that represents the software side of the processor system. The peripheral-specific functional software can be developed and compiled. The executable file can be generated from the compiled object codes and libraries. The linker script can be edited to control placement of various code segments into target memory.