

#### **Address Management**



This material exempt per Department of Commerce license exception TSU

# **Objectives**

#### After completing this module, you will be able to:

- Describe address management for MicroBlaze<sup>™</sup> processor
- Define a system address space
- Define an advanced user address space
- Describe the object file sections
- Describe what a linker script does



# Outline

#### Address Management

- System Address Space
- Advanced User Address Space
- Object File Sections
- Linker Scripts



# **Address Management**

- Embedded processor design requires you to manage the following:
  - Address map for the peripherals
  - Location of the application code in the memory space
    - Block RAM
    - External memory
- Memory requirements for your programs are based on the following:
  - The amount of memory required for storing the instructions
  - The amount of memory required for storing the data associated with the program



## **MicroBlaze Processor**





# Outline

- Address Management
- System Address Space
  - Advanced User Address Space
  - Object File Sections
  - Linker Scripts



# **Startup Files**

The compiler includes pre-compiled startup and end files when forming the executable

- Startup files setup the language and platform environment before your application code executes
  - Sets up vectors as required (reset, interrupt, exception, etc.)
  - Sets up registers (stack pointer, small data anchors, etc.)
  - Clears .bss memory region to zero
  - Invokes language initialization functions, such as C++ constructors
  - Initializes the hardware sub-system (ie. initialize profiling timers)
  - Sets up arguments for the main procedure and invokes it
- End files include code that must execute after the program ends
  - Invoke language cleanup functions, such as C++ destructors
  - De-initialize the hardware sub-system (ie. clean profiling system subsystem)



# System Address Space

Crt0.o initialization file is used when the executable is executed in standalone mode (no debug)

- The C runtime file crt0.o is linked with the user program
  - Starts at address location 0x0, immediately followed by the user program
  - Populates reset, interrupt, exception and hardware exception vectors





#### **System Address Space**

Crt1.o initialization file is used when the executable is executed in standalone mode (with debug)

- The C runtime file crt1.o is linked with the user program
  - Starts at address location 0x0, immediately followed by the user program
  - Populates all vectors except the breakpoint and reset vectors





#### System Address Space

Crt2.o initialization file is used when the executable is loaded using a boot loader

- The C runtime file crt2.o is linked with the boot loader
  - Starts at address location 0x0, immediately followed by the user program
  - Populates all vectors except the reset vector





# Outline

- Address Management
- System Address Space
- Advanced User Address Space
  - Object File Sections
  - Linker Scripts



#### Advanced User Address Space

- Different base address, contiguous user address space
  - The user program can run from any memory PLB, OPB, or LMB
  - To execute a program from any address location other than default, you must provide the compiler gcc with a different Program Start Address
  - Enter this option in the Compiler Settings dialog box

Application Mode                     Executable	vironment	Debug and Optimization	Paths and Options	
<ul> <li>Executable XmdStub xmdstub_peripheral:</li> <li>Dutput ELF file</li> <li>\slides\HW_design_EDK\TestApp_Peripheral\executable.elf Browse</li> <li>inker Script</li> <li>Use Custom Linker Script</li> <li>\TestApp_Peripheral\src\TestApp_Peripheral_LinkScr.ld Browse</li> <li>Use Default Linker Script</li> <li>Program Start Address 0x00001000</li> <li>Stack Size</li> </ul>	Application M	Mode		
Dutput ELF file          \slides\HW_design_EDK\TestApp_Peripheral\executable.elf       Browse         inker Script       Use Custom Linker Script         \TestApp_Peripheral\src\TestApp_Peripheral_LinkScr.ld       Browse         Use Default Linker Script       Program Start Address         Stack Size	Executa	able 🔘 XmdStub xmdstu	b_peripheral:	
\slides\HW_design_EDK\TestApp_Peripheral\executable.elf       Browse         inker Script       Use Custom Linker Script         \TestApp_Peripheral\src\TestApp_Peripheral_LinkScr.ld       Browse         We Default Linker Script       Program Start Address         Ox00001000       Stack Size	Output ELF I	file		
inker Script Use Custom Linker Script TestApp_Peripheral\src\TestApp_Peripheral_LinkScr.ld Browse Use Default Linker Script Program Start Address 0x00001000 Stack Size	\slides\HW	_design_EDK\TestApp_Peri	pheral\executable.elf	
Use Custom Linker Script  TestApp_Peripheral\src\TestApp_Peripheral_LinkScr.ld Browse  Use Default Linker Script  Program Start Address 0x00001000 Stack Size	Linker Script			
\TestApp_Peripheral\src\TestApp_Peripheral_LinkScr.ld       Browse         Image: Start Address       0x00001000         Stack Size       Stack Size	- 🔲 Use Ci	ustom Linker Script		
Use Default Linker Script Program Start Address 0x00001000 Stack Size	TURNA	Perinheral\stc\TestAnn_Pe	visharal LinkCorld	T
Use Default Linker Script  Program Start Address 0x00001000  Stack Size	1 estApp	i onpriorantere in courpe i s	inprietal_LINKSCI.IU	
Program Start Address 0x00001000 Stack Size				
Stack Size	- VI estApp	efault Linker Script		
	Use D Program S	efault Linker Script		
Heap Size	ViestApp Use Di Program S Stack Siz	efault Linker Script Start Address 0x00001000		
	ViestApp Vise Di Program S Stack Siz Heap Size	efault Linker Script Start Address 0x00001000 e		
	Program S Stack Siz	efault Linker Script Start Address 0x00001000 e		
	Program S Stack Siz	efault Linker Script Start Address 0x00001000 e		
	ViestApp Vise D Program S Stack Siz Heap Size	efault Linker Script		

© 2007 Xilinx, Inc. All Rights Reserved

#### Advanced User Address Space

- Different base address, noncontiguous user address space
  - You can place different components of your program in different memories
    - For example, you can target code to instruction LMB memory and the data to external DDR memory
  - Noncontiguous executables that represent the application must be created
    - To do this, a linker script must be used

Application Mode	wironment	Debug and Optimization	Paths and Options	
Output ELF file          \slides\HW_design_EDK\TestApp_Peripheral\executable.elf       Browse         Linker Script       Image: Script         Image: Use Custom Linker Script       Browse         Image: Use Default Linker Script       Browse	Application I	Mode able 🔿 XmdStub xmdstu	b_peripheral:	
\slides\HW_design_EDK\TestApp_Peripheral\executable.elf       Browse         Linker Script       Image: Script         Image: Use Custom Linker Script       Image: Script Script         Image: Use Default Linker Script       Browse	Output ELF	file		
Linker Script Use Custom Linker Script \TestApp_Peripheral\src\TestApp_Peripheral_LinkScr.Id Use Default Linker Script	\slides\HW	/_design_EDK\TestApp_Peri	pheral\executable.elf	Browse
Program Start Address 0x00001000		efault Linker Script		
Stack Size	- 🔲 Use D Program :	Start Address 0x00001000		
Heap Size	Program Stack Siz	Start Address 0x00001000		



# Outline

- Address Management
- System Address Space
- Advanced User Address Space
- Object File Sections
  - Linker Scripts



# **Object File Sections**

- What is an object file?
  - An object file is an assembled piece of code
    - Machine language: li r31,0 = 0x3BE0 0000
  - Constant data
  - There may be references to external objects that are defined elsewhere
  - This file may contain debugging information



# **Object File Sections**

Sectional Layout of an Object or an Executable file

.text	Text section
.rodata	Read-only data section
.sdata2	Small read-only data section (less than eight bytes)
.data	Read-write data section
.sdata	Small read-write data section
.sbss	Small uninitialized data section
.bss	Uninitialized data section



#### **Sections Example**

int ram\_data[10] = {0,1,2,3,4,5,6,7,8,9}; /\* DATA \*/

const int rom\_data[10] = {9,8,7,6,5,4,3,2,1}; /\* RODATA \*/

int I; /\* BSS \*/

main(){

}

... I = I + 10; /\* TEXT \*/ ...



# **Object File Sections**

Reserved sections that you typically would not modify

.init Language initialization code .fini Language cleanup code .ctors List of functions to be invoked at program startup .dtors List of functions to be invoked at program end .got Pointers to program data .got2 Pointers to program data .eh\_frame Frame unwind information for exception handling



# Outline

- Address Management
- System Address Space
- Advanced User Address Space
- Object File Sections
- Linker Scripts



# **Linker Scripts**

- Linker scripts control the linking process
  - Map the code and data to a specified memory space
  - Set the entry point to the executable
  - Reserve space for the stack
- Required if the design contains a discontinuous memory space
- GNU GCC linker scripts will not work for the WindRiver Diab compiler



## **Linker and Locator Flows**





# MicroBlaze Processor Script Example

```
STACKSIZE = 4k;
MEMORY
LMB : ORIGIN = 0x0, LENGTH = 0x1000
OPB : ORIGIN = 0 \times 8000, LENGTH = 0 \times 5000
}
SECTIONS
  .text : { *(.text) } > 1mb
  = ALIGN(4);
  heap = \cdot;
    .bss : { STACK SIZE = 0x400; . += STACK SIZE; . =
  ALIGN(4); \} > lmb
  stack = .;
  = ALIGN(4);
  .rodata : { *(.rodata) } > lmb
  .data : { *(.data) } > 1mb
}
```

**XILINX**°

# **Sections Command**

- This is where most of the work takes place
- Output sections are named, and the input sections are grouped and linked together into the output sections
- Example:

.text : { \*(.text) \*(.init) } > bram

- Explanation:
  - .text is the name of the output section
  - { \*(.text) \*.(init) } includes all input sections named text and init from the object files being linked
  - > bram locates the .text output section in the next available memory in the block RAM area



# **Linker Script Generator GUI**

- XPS contains a graphical lacksquareLinker Script Generator
- Table-based GUI allows you  $\bullet$ to define the memory space for any section
- Launch from **Software**  $\rightarrow$ • Generate Linker Script, or from the Applications Tab, right-click on <project>  $\rightarrow$ **Generate Linker Script**
- The tool will create a new • linker script (the old script is backed up)

🔶 Generate Li	nker Script		
Sections View:			Heap and Stack Vie
Section	Size (bytes)	Memory	Section
.text	0x00002D58	plb_bram_if_cntl 💌	Неар
.rodata	0x00000526	plb_bram_if_cntl 💌	Stack
.sdata2	0x00000000	plb_bram_if_cntl 💌	
.sbss2	0x00000000	plb_bram_if_cntl 💌	
.data	0x00000318	plb_bram_if_cntl 💌	
.fixup	0x00000000	plb_bram_if_cntl 💌	Memories View:
.sdata	0x0000008	plb_bram_if_cntl 💌	Memory
.sbss	0x0000001C	plb_bram_if_cntl 💌	DDR_SDRAM_64
.bss	0x00000094	plb_bram_if_cntl 💌	plb_bram_if_cntlr_1
.boot0	0x00000010	plb_bram_if_cntl 💌	
	Add Sec	ction Delete Section	ļ
			ELF file used to pop
Boot and Vecto	r Sections:		ng\adv_embedded

Memory

#### 4 0x00000000 65536K 0xFFFF0000 64K ulate section information: \labs\lab2\TestApp\_Memory\executable.elf Output Linker Script: \src\TestApp\_Memory\_LinkScr.Id plb bram if ontlr 1 Generate Cancel

Size (bytes)

Start Address

0x0

0x0

Memory

Length

plb bram if cntl 🔻

olb bram if ontl



×

Section

.boot

Address

0xFFFFFFFC

# Knowledge Check

- When do you need to use a linker script?
- What does a linker script do?
- List some MicroBlaze<sup>™</sup> processor address space restrictions



## Answers

- When do you need to use a linker script?
  - When you have software developed in multiple source files and the compiled object code needs to be placed in different memory structures or in nonstandard configurations
- What does a linker script do?
  - The linker script controls the placement of the object code, data, stack, and heap in specific memory locations
- List some MicroBlaze<sup>™</sup> processor address space restrictions
  - If you are not using xmdstub, ensure that crt0 is loaded in memory at 0
  - Must have writable memory from 0x00000000 to 0x0000004F



# Knowledge Check

- What does .sdata2 section contain?
- What does .sdata section contain?
- What does .sbss section contain?



## Answers

- What does .sdata2 section contain?
  - Small read-only data
- What does .sdata section contain?
  - Small read-write data
- What does .sbss section contain?
  - Small un-initialized data



# Where Can I Learn More?

#### Tool documentation

- Embedded System Tools Guide  $\rightarrow$  Address Management
- Embedded Systems Tools Guide  $\rightarrow$  Stand-Alone Board Support Package
- Embedded Systems Tools Guide  $\rightarrow$  GNU Compiler Tools
- Support Website
  - EDK Website: <u>www.xilinx.com/edk</u>
  - GNU Website: http://gcc.gnu.org/onlinedocs/gcc-3.4.4/gcc

