Address Management

Objectives

After completing this module, you will be able to:

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

Outline

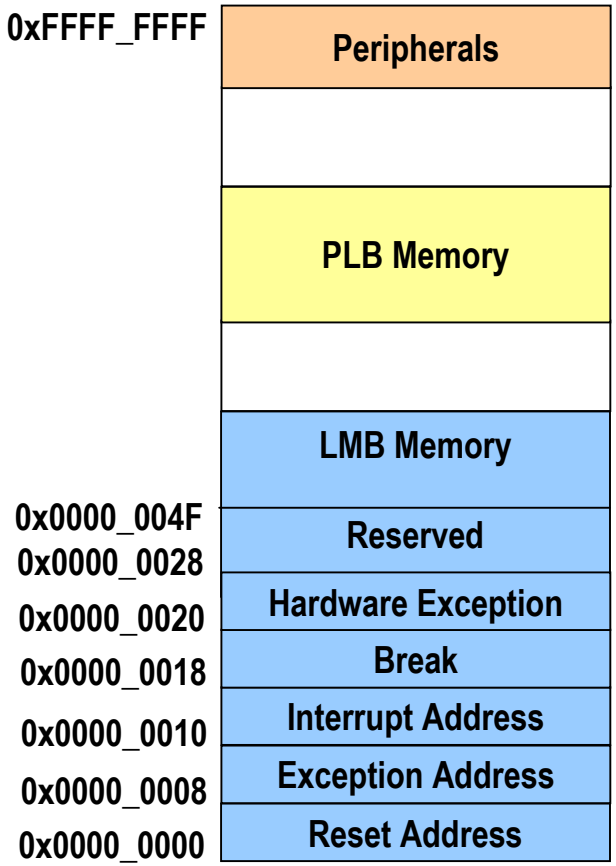
- **Address Management**
- System Address Space
- Advanced User Address Space
- Object File Sections
- Linker Scripts

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

- Memory and peripherals
  - The MicroBlaze™ processor uses 32-bit addresses
- Special addresses
  - MicroBlaze processors must have user-writable memory from 0x00000000 through 0x0000004F
  - Each vector consists of two instructions IMM followed by a BRAI instruction to address full memory range



Outline

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

Startup Files

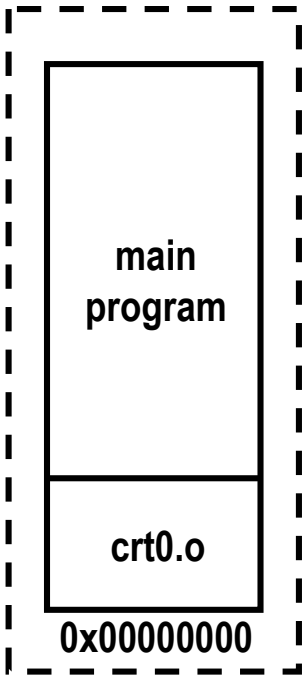
The compiler includes pre-compiled startup and end files when forming the executable

- 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 sub-system)

System Address Space

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

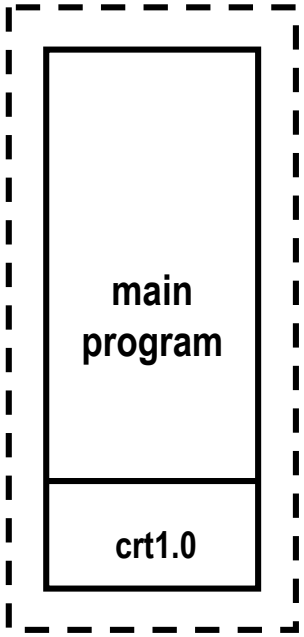
- The C runtime file crt0.o is linked with the user program
  - Starts at address location 0x0, immediately followed by the user program
  - Populates reset, interrupt, exception and hardware exception vectors



System Address Space

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

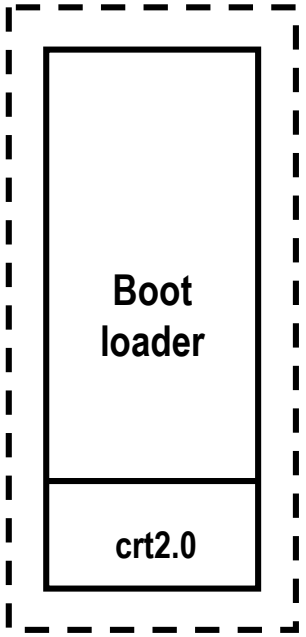
- The C runtime file crt1.o is linked with the user program
  - Starts at address location 0x0, immediately followed by the user program
  - Populates all vectors except the breakpoint and reset vectors



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

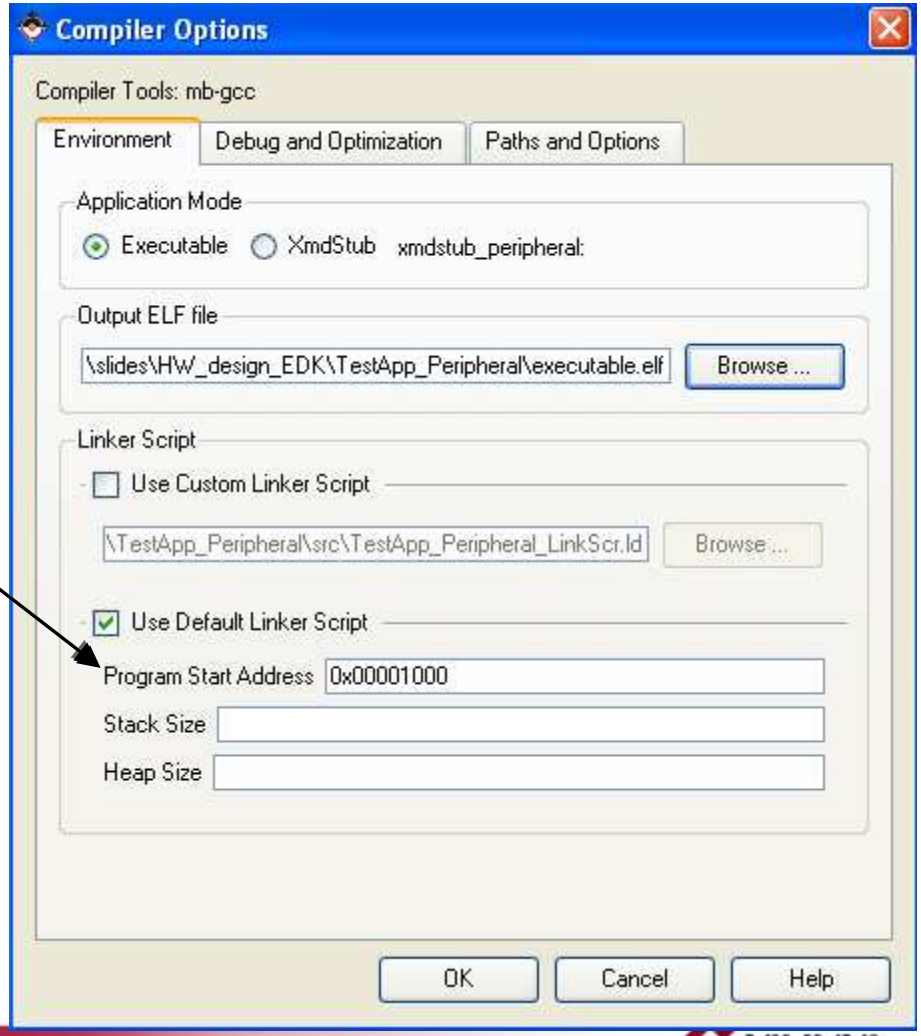


Outline

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

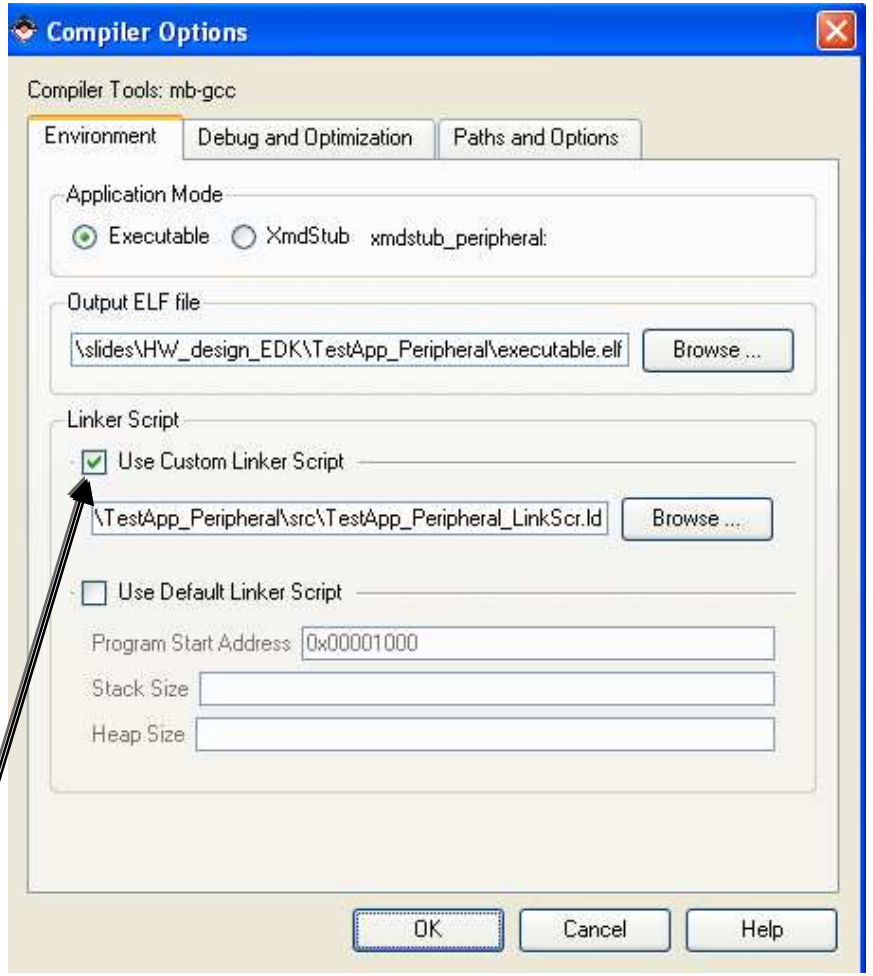
Advanced User
Address Space

- 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



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



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

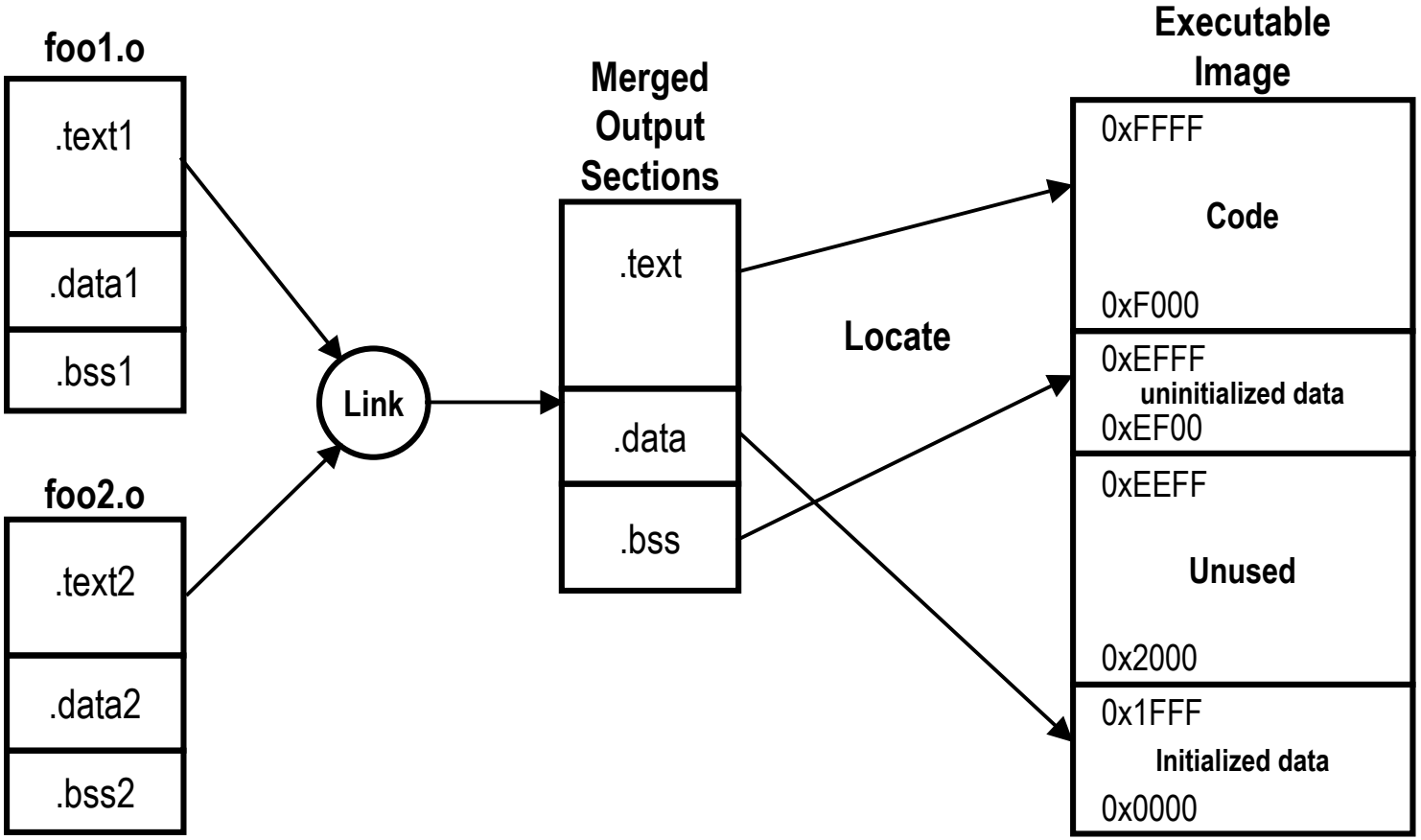
Outline

- 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 = 0x8000, LENGTH = 0x5000
}

SECTIONS
{
  .text : { *(.text)  } > lmb
  . = ALIGN(4);
  _heap = .;
    .bss : {   _STACK_SIZE = 0x400; . += _STACK_SIZE; . =
   ALIGN(4);   } > lmb
  _stack = .;
  . = ALIGN(4);
  .rodata : { *(.rodata)   } > lmb
  .data : { *(.data)  } > lmb
}
```

Sections Command

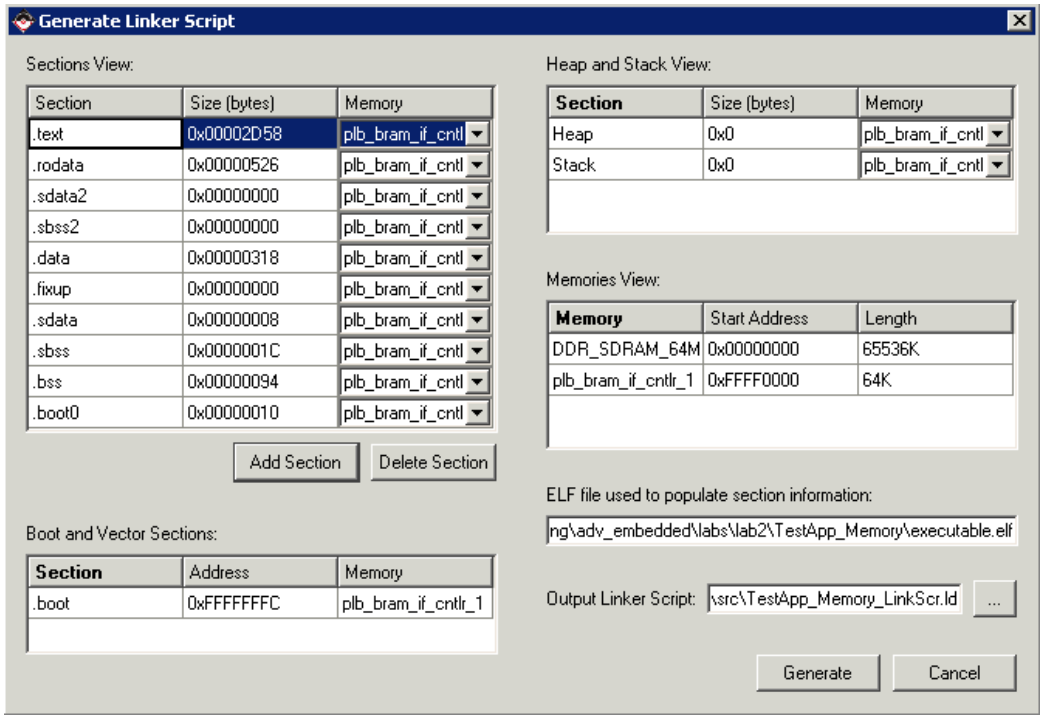
- This is where most of the work takes place
- Output sections are named, and the input sections are grouped and linked together into the output sections
- Example:

```
.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 Linker Script Generator
- Table-based GUI allows you to define the memory space for any section
- Launch from **Software → Generate Linker Script**, or from the Applications Tab, right-click on <project> → **Generate Linker Script**
- The tool will create a new linker script (the old script is backed up)



Knowledge Check

- When do you need to use a linker script?
- What does a linker script do?
- List some MicroBlaze™ 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™ 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 → Address Management*
  - *Embedded Systems Tools Guide → Stand-Alone Board Support Package*
  - *Embedded Systems Tools Guide → GNU Compiler Tools*
- Support Website
  - EDK Website: www.xilinx.com/edk
  - GNU Website: http://gcc.gnu.org/onlinedocs/gcc-3.4.4/gcc