**Freescale Semiconductor**
Application Note

Document Number: AN3203
Rev. 0, 12/2006

# CodeWarrior™ Linker Control File (LCF) for MSC8144 DSP

*by Duberly Mazuelos*
*NCSD*
*Freescale Semiconductor, Inc.*
*Austin, TX*

This application note describes how to use the linker control file (LCF) to define the memory layout for an application developed with CodeWarrior™ for StarCore DSP and executing on the MSC8144 DSP ADS board. The document focuses on understanding the LCF used in the MSC8144 ADS stationery provided with the CodeWarrior for StarCore DSP, version 3.0 beta integrated development environment (IDE). Throughout this document this version of the tools is simply called CW.

The LCF is a text file created by the application developer and used by the CW IDE linker to define the placement of data and code in memory for a given application. To accomplish this, the LCF needs to define the initial setup of the MSC8144 memory management unit (MMU). The MMU offers a level of sophistication that may prove challenging for the first-time user. Therefore, this application note is provided to help you understand how to use the LCF to set up the MSC8144 MMU to define an application's memory map. Send questions and comments to Freescale Support using the contact information on the back cover of this document.

**Contents**

1 MSC8144 Memory Management Basics . . . . . . . . . . .2
1.1 MSC8144 Memory Map . . . . . . . . . . . . . . . . . . . . .2
1.2 Memory Management Unit (MMU) . . . . . . . . . . . .2
2 LCF Directives . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .3
2.1 Memory Devices and Physical Memory . . . . . . . . .4
2.2 Sections and Segments . . . . . . . . . . . . . . . . . . . . . .6
2.3 Virtual Memory . . . . . . . . . . . . . . . . . . . . . . . . . . . .6
2.4 Shared Memory Among Cores . . . . . . . . . . . . . . . .8
3 CW MSC8144 Startup Code . . . . . . . . . . . . . . . . . . . . .9
3.1 First Hook . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .9
3.2 Second Hook . . . . . . . . . . . . . . . . . . . . . . . . . . . . .10
3.3 Third Hook . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .11
4 LCF Example . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .11
4.1 Define Virtual Memory . . . . . . . . . . . . . . . . . . . . .11
4.2 Define Physical Memory . . . . . . . . . . . . . . . . . . .14
5 Debugging the LCF . . . . . . . . . . . . . . . . . . . . . . . . . . 15

APPENDIXES:
Appendix A
MSC8144 Physical Memory and Peripherals . . . . . . 17
Appendix B
Requirements for Default Sections . . . . . . . . . . . . . . .19
Appendix C
msc8144ADS_common.txt File . . . . . . . . . . . . . . . . . .20
Appendix D
msc8144ADS.lcf File for Four MSC8144 Cores . . . .23

© Freescale Semiconductor, Inc., 2006. All rights reserved.

freescale™ semiconductor

This application note assumes knowledge in the following areas:

- CodeWarrior IDE for StarCore DSP. Preferably hands-on experience with development on previous Freescale DSPs such as the MSC8122. Refer to the SC linker-related documentation in the CW tools.
- MSC8144 DSP, particularly the memory subsystem. Be sure to read the chapter on the internal memory subsystem in the MSC8144 reference manual.

# 1 MSC8144 Memory Management Basics

This section introduces the main concepts involved in creating an LCF for the MSC8144 using the CW IDE. For further details, consult the MSC8144 reference manual.

## 1.1 MSC8144 Memory Map

This application note is concerned with the MSC8144 memory map as it is visible to an application executing on the SC3400 cores. From this point of view, you must consider two distinct memory maps when developing an MSC8144 application. These two maps are defined by the use of either physical addresses or virtual addresses. A physical address is the actual address of a device (memory or peripheral) within the MSC8144 DSP. The physical addresses are fixed and cannot be changed by the user application.The physical address memory map of the MSC8144 is shown in Appendix A, "MSC8144 Physical Memory and Peripherals." A virtual address is the address used by the SC3400 core at run time to access the various memory and peripheral devices in the MSC8144. A virtual address can be different from or the same as the physical address.

A useful way to relate to these two memory maps is to think of virtual addresses as used exclusively by an application at run-time to access the MSC8144 resources, whereas the physical addresses are used outside the SC3400 extended core to access the physical device itself. The level 1 (L1) caches also use virtual address. The run-time process of converting the (virtual) address visible to the core to the physical address used by the hardware is called address translation and is performed by the MMU in the extended core.

## 1.2 Memory Management Unit (MMU)

The significant performance advantages of an MMU are beyond the scope of this document. The MMU in the MSC8144 serves multiple purposes, but the address translation operation, depicted in Figure 1, is the main concern here. In essence, the translation operation replaces the most significant portion of the virtual address to generate a physical address. The lower portion or the address remains intact. The translation process is programmed through a memory attributes and translation table (MATT) in the MMU. Segment descriptors in the MATT define how to replace the upper portion of the virtual address with the corresponding portion of the physical address. The segment descriptors also define memory attributes that provide cache and bus controls for efficient memory management.

Address translation occurs for both data and program accesses independently, so there are two MATT tables:

- A program MATT, which has 12 segment descriptors
- A data MATT, which has 20 segment descriptors

This arrangement implies that data and code virtual addresses can use the same values without conflict. For example, you can define global data to begin at address 0x0 and also use address 0x0 for interrupt vectors. This is possible because the MMU translates these data and code virtual addresses to different physical addresses in memory.

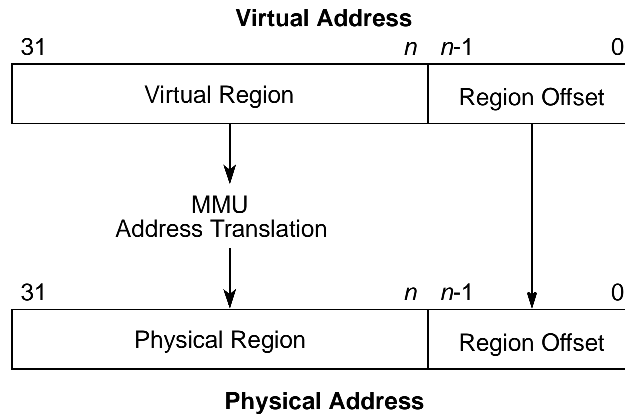


**Figure 1. MMU Address Translation Operation**

The developer uses the LCF to define the virtual memory map for an application and also to define the mapping between these virtual addresses and the corresponding fixed physical addresses on the MSC8144 DSP. The developer must be concerned only with the memory map for program and data accesses to M2, M3, and DDR, and not with accesses to control and status registers. The MMU has fixed descriptors for translating the addresses for the extended core registers and the peripheral memory space. These descriptors map these virtual addresses to be the same as the physical addresses. Therefore, the application uses the addresses shown in Appendix A, "MSC8144 Physical Memory and Peripherals" to access the various registers on the MSC8144 DSP.

**NOTE**

In the LCF, the base address of a region defined by a segment descriptor in a MATT must be aligned to a power of 2. Furthermore, the size of the region is chosen so that the base of the region is always a multiple of the region size.

# 2 LCF Directives

Directives in the LCF are used to define the memory for an MSC8144 application. There are several ways to accomplish the same task. This application note describes only one of the ways to write the LCF to create a particular memory map for an application. This example is the one used in the MSC8144ADS stationery of CW. The LCF in the CW project for this stationery consists of four files: `mmu_attributes.txt`, `common.txt`, `descriptors.txt`, and `crtscbmm.cmd`. The first three files are included in the fourth file, which is the LCF proper.

## 2.1 Memory Devices and Physical Memory

Table 1 shows the memory devices on the MSC8144ADS and their fixed physical addresses.

**Table 1. MSC8144 ADS Memory**

| Memory | Start Address | End Address | Size |
|---|---|---|---|
| MSC8144 M2 | 0xC000 0000 | 0xC007 FFFF | 512 Kbytes |
| MSC8144 M3 | 0xD000 0000 | 0xD09F FFFF | 10 Mbytes |
| External DDR | 0x4000 0000 | 0x4FFF FFFF | 256 Mbytes |

To define the memory available to the linker in an MSC8144 application, the LCF uses the `.memory` directive, which may be familiar if you have used earlier Freescale DSPs, such as the MSC8122. In Example 1, the LCF in the CW MSC8144ADS stationery defines an available (physical) memory region starting from _M2_PRIVATE_start to _M2_PRIVATE_end with read, write, and executable properties.

**Example 1. .memory Directive**

```
.provide _M2_PRIVATE_start, _M2_start + _ID_CORE * _VIRTUAL_PRIVATE_M2_DATA_size
.provide _M2_PRIVATE_end, _M2_PRIVATE_start + _VIRTUAL_PRIVATE_M2_DATA_size -1
.memory _M2_PRIVATE_start, _M2_PRIVATE_end, "rwx"
```

The `.provide` directive defines a global symbol that can be used elsewhere in the LCF (or executable). In the LCF of the CW MSC8144ADS stationery, the use of the `.provide` and `.memory` directives is straightforward (in the `common.txt` file). These directives define all the memory regions available to the linker on the MSC8144ADS as shown in Table 2.

**Table 2. Physical Memory Layout for CW MSC8144ADS Stationery**

| Memory Region | Start Address | End Address | Size |
|---|---|---|---|
| M2 (private core 0) | 0xC000 0000 | 0xC000 FFFF | 64 Kbyte |
| M2 (private core 1) | 0xC001 0000 | 0xC001 FFFF | 64 Kbyte |
| M2 (private core 2) | 0xC002 0000 | 0xC002 FFFF | 64 Kbyte |
| M2 (private core 3) | 0xC003 0000 | 0xC003 FFFF | 64 Kbyte |
| M2 (shared by all cores) | 0xC004 0000 | 0xC005 FFFF | 128 Kbyte |
| M2 (boot core 0) | 0xC006 0000 | 0xC006 07FF | 2 Kbyte |
| M2 (boot core 1) | 0xC006 8000 | 0xC006 87FF | 2 Kbyte |
| M2 (boot core 2) | 0xC007 0000 | 0xC007 07FF | 2 Kbyte |
| M2 (boot core 3) | 0xC007 8000 | 0xC007 87FF | 2 Kbyte |
| M3 (private to core 0) | 0xD000 0000 | 0xD003 FFFF | 256 Kbyte |
| M3 (private to core 1) | 0xD004 0000 | 0xD007 FFFF | 256 Kbyte |
| M3 (private to core 2) | 0xD008 0000 | 0xD00B FFFF | 256 Kbyte |
| M3 (private to core 3) | 0xD00C 0000 | 0xD00F FFFF | 256 Kbyte |
| M3 (shared) | 0xD010 0000 | 0xD09F FFFF | 9 Mbyte |

**Table 2. Physical Memory Layout for CW MSC8144ADS Stationery (continued)**

| Memory Region | Start Address | End Address | Size |
|---|---|---|---|
| DDR (private to core 0) | 0x4000 0000 | 0xD007 FFFF | 512 Kbyte |
| DDR (private to core 1) | 0x4008 0000 | 0x400F FFFF | 512 Kbyte |
| DDR (private to core 2) | 0x4010 0000 | 0x4017 FFFF | 512 Kbyte |
| DDR (private to core 3) | 0x4018 0000 | 0x401F FFFF | 512 Kbyte |
| DDR (shared) | 0x4020 0000 | 0x4FFF FFFF | 254 Mbyte |

The shared regions refer to memory that is accessible to all cores on the MSC8144 and thus can be used for code and data that is common to all cores. A private region is used exclusively by one core and can contain items such as a stack or data buffers it is processing. A portion of the M2 boot memory private to each core is excluded from the memory definitions shown in Table 2. This memory is reserved for the stack of each core. Figure 2 shows the physical memory defined in the CW LCF. Recall that all memories (M2, M3, and DDR) on the MSC8144ADS are physically accessible to all cores. However, the LCF allows you to partition these memories into shared and private regions.

M2 (512 Kbytes)

| End Address | Region | Start Address |
|---|---|---|
| 0xC007 FFFF | private_boot Core 3 | 0xC007 8000 |
| 0xC007 7FFF | private_boot Core 2 | 0xC007 0000 |
| 0xC006 FFFF | private_boot Core 1 | 0xC006 8000 |
| 0xC006 7FFF | private_boot Core 0 | 0xC006 0000 |
| 0xC005 FFFF | m2_shared_data m2_shared_text | 0xC004 0000 |
| 0xC003 FFFF | m2_private_data Core 3 | 0xC003 0000 |
| 0xC002 FFFF | m2_private_data Core 2 | 0xC002 0000 |
| 0xC001 FFFF | m2_private_data Core 1 | 0xC001 0000 |
| 0xC000 FFFF | m2_private_data Core 0 | 0x0000 0000 |

M3 (10 Mbytes)

| End Address | Region | Start Address |
|---|---|---|
| 0xD09F FFFF | m3_shared_text | 0xD010 0000 |
| 0xD00F FFFF | m3_private_data Core 3 | 0xD00C 0000 |
| 0xD00B FFFF | m3_private_data Core 2 | 0xD008 0000 |
| 0xD007 FFFF | m3_private_data Core 1 | 0xD004 0000 |
| 0xD003 FFFF | m3_private_data Core 0 | 0xD000 0000 |

DDR (256 Mbytes)

| End Address | Region | Start Address |
|---|---|---|
| 0x4FFF FFFF | ddr_shared_text ddr_shared_data | 0x4020 0000 |
| 0x401F FFFF | ddr_private_data Core 3 | 0x4018 0000 |
| 0x4017 FFFF | ddr_private_data Core 2 | 0x4010 0000 |
| 0x400F FFFF | ddr_private_data Core 1 | 0x4008 0000 |
| 0x4007 FFFF | ddr_private_data Core 0 | 0x4000 0000 |

**Figure 2. Physical Memory Layout for CW MSC8144ADS Stationery**

## 2.2 Sections and Segments

In processors, such as the MSC8122, that do not have an MMU, the addresses used by the cores in an application are the actual physical addresses of the memory and peripheral devices being addressed. Thus, the concept of a virtual address is absent. Example 2 shows the customary method used by the linker to define where code and data are placed into memory using a combination of `.org` and `.segment` directives.

**Example 2. .org and .segment directives**

```
.org MEMORY_START
.segment CODE, ".text", ".secret_code"
.segment DATA, ".data", ".double_secret_data"
```

A .segment directive combines all the sections that match the specified section names into a new memory segment. In Example 2, a segment named CODE is constructed that contains all sections named .text followed by all sections named `.secret_code`. A second segment named DATA is constructed that contains all sections named `.data` followed by all sections named `.double_secret_data`.

A section is simply a relocatable block of code or data that is encapsulated by the SECTION and ENDSEC assembler directives and has an associated section name and type. Although you can create any name for a section, some section names are reserved by the debugger and the Smart DSP operating system (SDOS). The application must not use these reserved names (refer to the assembler user's guide and the corresponding SDOS documentation). In addition, the assembler recognizes conventional ELF sections such as `.text`, `.data`, `.rodata`, and **.BSS**. Appendix B, "Requirements for Default Sections" presents information and allocation requirements for the default sections generated by the CW tools. These sections are used in the LCF for the CW MSC8144ADS stationery.

The linker places segments in memory at the current location counter, which is defined by the `.org` directive. Example 2 begins linking the CODE segment at the address value represented by the MEMORY_START symbol followed by the segment DATA. It is possible to use `.org` and `.segment` directives (in combination with other LCF directives) to define the placement of code and data sections in physical memory and their corresponding mapping to virtual memory for an MSC8144 application. However, a different method used in the CW stationery LCF is presented in the following section.

## 2.3 Virtual Memory

One straightforward method to define where code and data are placed into the virtual memory, and how it is mapped to a physical memory location, is to use a combination of `.concatenate` and .att_mmu linker directives in the LCF. The .concatenate directive simply groups a list of sections together under a new section name. Using the `.concatenate` directive before an `.att_mmu` directive simplifies the organization of the `.att_mmu` directives.

The `.concatenate` directives used in the LCF of CW MSC8144ADS stationery are listed in the `descriptors.txt` file. Example 3 shows how the first .concatenate directive creates a new section called m2_shared_data that contains the single section named reserved_crt_mutex. The second `.concatenate` directive creates another section named m2_shared_text that is a concatenation of the sections `.intvec`, `.text`, and `.default`. The developer can add or move sections to the list in the `.concatenate` directive.

**Example 3. .concatenate directive**

```
.concatenate "m2_shared_data" ,"reserved_crt_mutex"
.concatenate "m2_shared_text", ".intvec",".text", ".default"
```

The `.att_mmu` directive defines the virtual address range for an existing section (or a group of sections if a `.concatenate` is defined). A corresponding physical address is also defined with each of these sections. With this information the linker creates an address translation table (ATT) for use by the application startup code that programs the MSC8144 MMU. Example 4 shows the `.att_mmu` directive. It comes directly from the LCF file `crtscbmm.cmd` file in the CW MSC8144ADS stationery where all these statements are declared.

**Example 4. .att_mmu directive for shared M2 memory**

```
.att_mmu "M2_shared_mmu",  _M2_SHARED_start, _M2_SHARED_end,\
         "m2_shared_text", \
                  base_address:_M2_SHARED_start, \
                  attribute: SYSTEM_PROG_MMU_DEF, \
                  physical_address: _M2_SHARED_start, \
         "m2_shared_data",\
                  attribute: SHARED_DATA_MMU_DEF, \
                  base_address: @vsecend("m2_shared_text"), \
                  physical_address: @secend("m2_shared_text")
```

In Example 4, the .att_mmu directive creates an entry in the ATT table for the two sections created by the concatenate directives of the preceding example (m2_shared_data and m2_shared_text). The virtual address range for these sections starts at the address represented by the _M2_SHARED_start symbol and goes to the address represented by the _M2_SHARED_end symbol. These symbols are defined in the `common.txt` file and evaluate to the M2 memory region shared by all cores shown in Table 2. Finally, the physical address used to locate these sections is defined by the _M2_SHARED_start symbol. This is the same symbol used for the virtual address, indicating that there is a 1-to-1 mapping of virtual address to physical address. The memory attributes for the cache and bus controls for these sections is defined by the SYSTEM_PROG_MMU_DEF and SHARED_DATA_MMU_DEF symbols, which are located in the `mmu_attributes.txt` file of the CW stationery. A similar approach is used to create entries in the ATT table for shared M3 and DDR sections in the CW MSC8144ADS stationery LCF.

Example 5 is also from the `crtscbmm.cmd` file of the CW MSC8144ADS stationery LCF and is used to create the MMU mapping for the private data for each core. It works like Example 4 except that there is not a 1-to-1 mapping between virtual and physical addresses. The virtual address range is defined by the symbols VIRTUAL_PRIVATE_MEM_DATA_start and _VIRTUAL_PRIVATE_MEM_DATA_end + VIRTUAL_BOOT_size, which evaluate to the range 0x0000 0000 to 0x000D 7FFF (see `common.txt` LCF in the stationery). Each core thus accesses its private memory on this address range. However, the physical memory addresses are defined by the symbols _VIRTUAL__BOOT_start, _M2_PRIVATE_start, _M3_PRIVATE_start, and _DDR_PRIVATE_start, which evaluate to different locations in physical memory indicated in Table 2.

**Example 5. .att_mmu Directive for Private Memory**

```
.att_mmu  "Data_private_mmu",        _VIRTUAL_PRIVATE_MEM_DATA_start,
_VIRTUAL_PRIVATE_MEM_DATA_end + _VIRTUAL_BOOT_size, \
         "private_boot", attribute : SYSTEM_DATA_MMU_DEF, \
                           base_address:_VIRTUAL_BOOT_start, \
                           physical_address: _M2_PRIVATE_BOOT_start, \
         "m2_private_data", attribute : SYSTEM_DATA_MMU_DEF, \
                           physical_address: _M2_PRIVATE_start, \
         "m3_private_data", attribute : SYSTEM_DATA_MMU_DEF, \
                           physical_address: _M3_PRIVATE_start, \
         "ddr_private_data", attribute : SYSTEM_DATA_MMU_DEF, \
                           physical_address: _DDR_PRIVATE_start
```

Note that the linker performs several checks of the memory regions defined by the user in the LCF to ensure that virtual and physical addresses are valid and that the regions map appropriately. As Figure 3 shows, you can verify that the LCF included with the CW MSC8144ADS stationery creates a virtual memory map for each core.

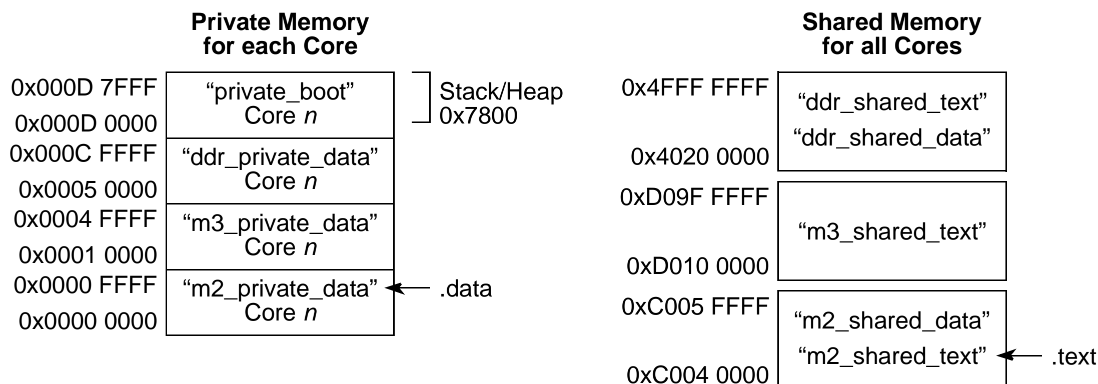


**Figure 3. Virtual Memory Map per Core for the MSC8144ADS Stationery LCF**

## 2.4 Shared Memory Among Cores

Although all memories (M2, M3, and DDR) on the MSC8144ADS are physically accessible to all cores, the LCF allows you to partition these memories into shared and private regions. Shared regions are memory that is accessible to all cores, and a private region is used exclusively by one core.

The function of the .space directive is very similar to the combination of .org and .segment directives The .space directive defines a memory space for a group of segments that corresponds to a physical memory device in a multiple-core environment. This directive is used to specify the address range where these sections are to be mapped. The .space directive is useful because, in combination with `.import` and .export directives, it easily defines shared regions of memory among cores, as shown in Example 6.

**Example 6. .space and .export Directives**

```
.space    m2_shared, _M2_SHARED_start, _M2_SHARED_end, "m2_shared_data", "m2_shared_text"
.space m3_shared, _M3_SHARED_start, _M3_SHARED_end, "m3_shared_text"
.space    ddr_shared, _DDR_SHARED_start, _DDR_SHARED_end, "ddr_shared_data", "ddr_shared_text"
.export "m2_shared", "m3_shared", "ddr_shared"
```

In Example 6, which also comes from the CW MCS8144ADS stationery LCF, the `.space` directive is used to define three memory spaces corresponding to the shared regions in M2, M3, and DDR memory. Notice that the segments allocated to each region are the same segments defined by the .concatenate directives and used in the .att_mmu directives in the LCF.

After the shared memory space is defined, the .export directive is used to indicate that this memory is to be shared with other cores. In the CW MSC8144ADS LCF, it is the description for core 0 that defines the shared M2, M3, and DDR memory and exports it to the other cores on the MSC8144. Example 7 shows how the other cores use the .import directive to access the segments placed in the shared spaces.

**Example 7. .import Directive**

```
.import "c0`m2_shared","c0`m3_shared", "c0`ddr_shared"
```

# 3 CW MSC8144 Startup Code

After you have created the LCF to define the virtual memory map and the corresponding translation to physical addresses, the linker uses the LCF to map the application in memory. The linker also generates several tables and variables based on the LCF for use by the C startup code to set up various aspects of the MSC82144 device and C run-time environment before the application C `main()` function executes. To set up the application properly, you must be aware of what the startup code requires of the LCF.

The CW MSC8144 default startup code resides in the `startup__startup_msc8144_.asm` file of the CW `\compiler\src\rtlib\expanded` directory. The startup code executes three hooks, or functions, that proceed to set up items such as the MMU, stack, and heap. You can skip execution of these hooks or redefine these hooks by adding code with the same function names to the CW project.

## 3.1 First Hook

The CW MSC8144 default startup code first initializes the status register (SR), vector base address (VBA) register, and the core register file (for Verilog simulation purposes). Next, the code calls the first hook, which is an assembly function called `___target_asm_start` in the `target_asm_start__common_.asm` file of the CW `\compiler\src\rtlib\expanded` directory. This hook is executed before the stack pointer and the C/C++ environment are initialized. Thus, at this point only assembly code (without jump-to-subroutine instructions) can execute.

The code in the first hook enables the MMU and defines the translation for the stack and heap. After this hook the stack pointer register is initialized to allow execution of C/C++ code. Thus, it is critical to set up the LCF correctly so that the MMU segment descriptors for the stack pointer and heap are properly defined in the MMU MATT registers. Table 3 shows the symbols used by the first hook of the CW MSC8144 default startup code. It also indicates the LCF file in the CW ADS stationery where the symbol is defined.

**Table 3. Symbols Used the First Hook**

| Symbol | Stationery LCF | Description |
|---|---|---|
| _ENABLE_MMU_TRANSLATION | mmu_attributes.txt | MMU cache and bus controls |
| SYSTEM_DATA_MMU_DEF | mmu_attributes.txt | MMU cache and bus controls |
| _LocalData_b | common.txt | Virtual address base of private data memory for each core |

**Table 3. Symbols Used the First Hook (continued)**

| Symbol | Stationery LCF | Description |
|---|---|---|
| _LocalData_size | common.txt | Size of private data memory for each core |
| _LocalData_Phys_b | common.txt | Physical address of the private data memory for *all* cores |

Deciphering the symbols in the `mmu_attributes.txt` file of the stationery LCF is straightforward. The _LocalData_b and _LocalData_size symbols define the *virtual* base address and size, respectively, for the private memory to each core where the stack is to reside. _LocalData_Phys_b defines the starting *physical* address for this memory. This startup hook assumes that this physical memory for all cores is placed one after the other and thus the _LocalData_Phys_b symbol defines the base address of the corresponding memory for core 0. In the LCF for the CW MSC8144ADS stationery, this memory region corresponds to the private boot located in M2 (see Table 2, "Physical Memory Layout for CW MSC8144ADS Stationery," on page 4).

These symbols are important because the first hook uses them to set up the MMU for access to the stack. Thus, if you plan to modify the stationery LCF or create a new LCF and use the default startup code and hooks, be sure to define these symbols in the LCF you are using. The application project generates a linker error if these symbols are omitted from the LCF.

## 3.2 Second Hook

After the first hook, the startup code initializes the stack pointer using the _StackStart symbol or the LCF. Then it calls the second hook, which is in a C function called `__target_c_start` in the `target_c_start__common.c` file of the CW `\StarCore_Support\compiler\src\rtlib\expanded` directory. This function is used to configure the rest of the MMU according to the memory map defined by the user in the LCF. C/C++ run-time initializations have not occurred at this point, so C code in the second hook cannot use uninitialized global variables. Table 4 shows the symbols used by the second hook of the CW MSC8144 default startup code. It also indicates the LCF file in the stationery where the symbol is defined.

**Table 4. Symbols Used the Second Hook**

| Symbol | Stationery LCF File |
|---|---|
| _ENABLE_MMU_TRANSLATION | mmu_attributes.txt |
| _ENABLE_MMU_PROTECTION | mmu_attributes.txt |
| _ENABLE_DEFAULT_TASK_ID | mmu_attributes.txt |
| _SYSTEM_TASK_ID | mmu_attributes.txt |
| _MMU_HIGH_PRIORITY | mmu_attributes.txt |
| _MMU_PROG_DEF_SYSTEM | mmu_attributes.txt |
| _MMU_DATA_DEF_SYSTEM | mmu_attributes.txt |
| _LocalData_Phys_b | common.txt |
| _LocalData_size | common.txt |

Thus, if you plan to modify the stationery LCF or create a new LCF and use the default startup code and hooks, be sure to define these symbols in the LCF files. Understanding the symbols in the `mmu_attributes.txt` file of the stationery LCF is straightforward. The symbols for _LocalData_Phys_b and _LocalData_size simply define the physical address and size of the memory that is private to each core where the stack and MMU tables reside.

## 3.3 Third Hook

After the second hook executes, the startup code initializes the C/C++ run-time environment and then executes the third hook by calling a C function called `__target_setting()` located in the `target_setting__common_.c` file of the CW `\compiler\src\rtlib\expanded` directory. This function is currently empty, but it can be used to perform other initializations specific to the target. After the third hook the startup code calls the C `main()` function.

# 4 LCF Example

Using the concepts presented thus far, we create an LCF for an MSC8144ADS application. We must carefully consider the memory requirements for the application and then define the LCF appropriately. The memory devices for the ADS are fixed as indicated in Table 1. Example 8 presents the symbol declarations that define this memory.

**Example 8. Symbols for MSC8144ADS Memory for LCF**

```
;--------------------------------
;   MSC8144 ADS memory addresses
;--------------------------------
.provide _M2_start, 0xC0000000
.provide _M2_size,  0x00080000              ; M2 size = 512K
.provide _M2_end,   _M2_start + _M2_size - 1
.provide _M3_start, 0xD0000000
.provide _M3_size,  0x00a00000              ; M3 size = 10M
.provide _M3_end, _M3_start + _M3_size - 1
.provide _DDR_start,0x40000000
.provide _DDR_size, 0x10000000              ; DDR size = 256MB
.provide _DDR_end, _DDR_start + _DDR_size -1
```

The next task is to define how to partition the available memory into local (private) and shared memory and how each memory region is to be used.

A system-level analysis of this hypothetical application provides the following general guidelines. Because all cores perform the same task, most of the program is placed into M3 and shared by all cores. M3 is also used for shared data. M2 is used for the local data for each core, including the stack and global variables. M2 also holds critical shared data and code, such as interrupt and key DSP routines. DDR is used for shared program and data that is less critical, such as initialization routines and other infrequently used data and code.

## 4.1 Define Virtual Memory

We define the virtual memory required by each core to execute the application. Specifically, the requirements for M2 are as follows:

- 64 Kbyte of memory in M2 for each core to use for local data, including the stack and global variables
- 256 Kbyte of memory in M2 for critical shared data and code

The requirements for M3 are as follows:

- 128 Kbyte of memory in M3 for each core to be used for local data
- Remaining memory in M3 to be used as critical shared code and data

Figure 4 shows the virtual map for each core.

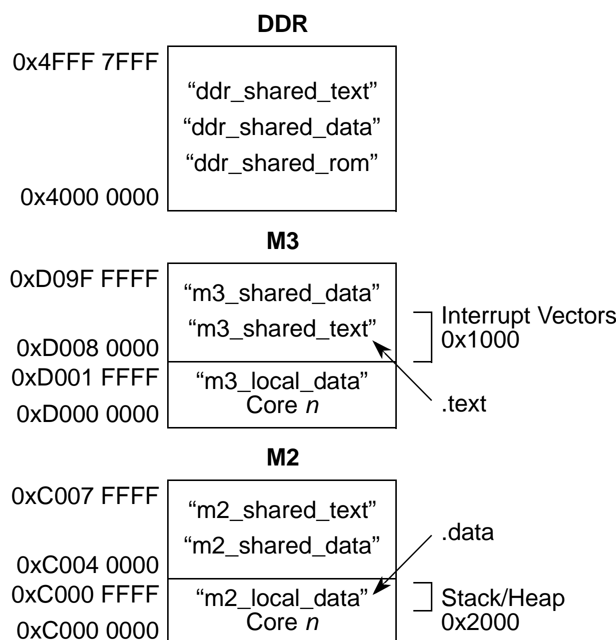


**Figure 4. Virtual Memory Example**

Note that the addresses used for the virtual memory are similar to the physical addresses of the device. In particular, the shared regions map 1:1 with the physical addresses. However, the virtual local memory for each core, m2_local_data, and m3_local_data is mapped to the base (beginning) of M2 and M3, respectively. Since these local memory regions are not shared among cores, they are mapped to different addresses of physical memory. Example 9 shows the linker directives that define the symbols for this virtual memory.

**Example 9. Symbols for Virtual Memory in LCF Example**

```
;--------------------------------
;   Application virtual memory
;--------------------------------
; Virtual space for private data will translate to start of M2 memory
.provide _VIRTUAL_LOCAL_M2_DATA_start,  _M2_start
.provide _VIRTUAL_LOCAL_M2_DATA_size,   0x00010000
.provide _VIRTUAL_LOCAL_M2_DATA_end,    _VIRTUAL_LOCAL_M2_DATA_start +
_VIRTUAL_LOCAL_M2_DATA_size - 1
```

```
; Virtual space for shared data and code in M2 will have a 1:1 virtual to physical address
mapping
```

```
.provide _VIRTUAL_SHARED_M2_start,      _M2_start + _NUMBER_OF_CORES *
_VIRTUAL_LOCAL_M2_DATA_size
.provide _VIRTUAL_SHARED_M2_end,        _M2_end
```

```
; Virtual space for private code will translate to start of M3 memory
.provide _VIRTUAL_LOCAL_M3_DATA_start,  _M3_start
.provide _VIRTUAL_LOCAL_M3_DATA_size,   0x00020000
.provide _VIRTUAL_LOCAL_M3_DATA_end,    _VIRTUAL_LOCAL_M3_DATA_start +
_VIRTUAL_LOCAL_M3_DATA_size - 1
```

```
; Virtual space for shared data and code in M3 will have a 1:1 virtual to physical address
mapping
.provide _VIRTUAL_SHARED_M3_start,      _M3_start + _NUMBER_OF_CORES *
_VIRTUAL_LOCAL_M3_DATA_size
.provide _VIRTUAL_SHARED_M3_end,        _M3_end
```

```
; Virtual space for shared data and code in DDR will have a 1:1 virtual to physical address
mapping
.provide _VIRTUAL_SHARED_DDR_start,     _DDR_start
.provide _VIRTUAL_SHARED_DDR_end,       _DDR_end
```

## 4.2 Define Physical Memory

Figure 5 defines the corresponding physical memory map. Here we see how the physical addresses of the shared regions map 1:1 with the virtual addresses shown in Figure 4.

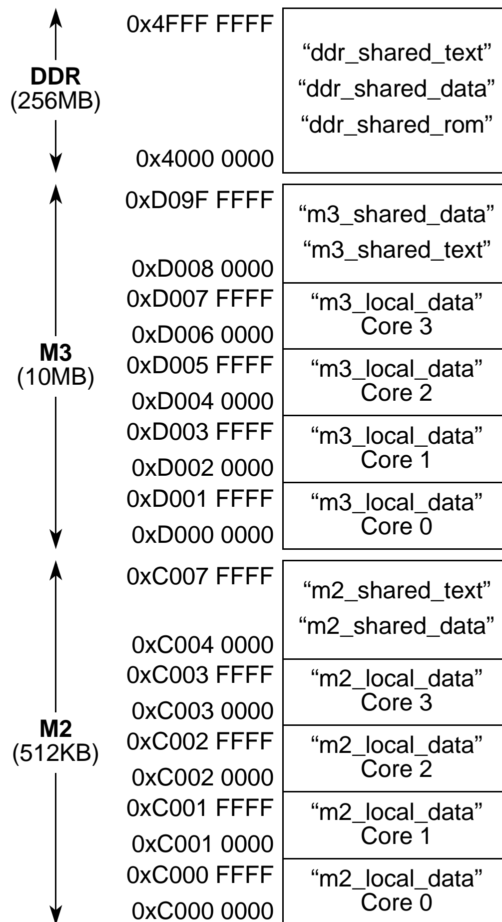


**Figure 5. Physical Memory Example**

We also see how the local memory regions are mapped to different physical addresses in memory. Example 10 lists the linker directives that define the symbols for this physical memory.

**Example 10. Symbols for Physical Memory in LCF Example**

```
;--------------------------------
;   Application physical memory
;--------------------------------
; Private data sections for each core in M2 memory
.provide _PHYSICAL_LOCAL_M2_DATA_start, _M2_start + _ID_CORE * _VIRTUAL_LOCAL_M2_DATA_size
.provide _PHYSICAL_LOCAL_M2_DATA_end,   _PHYSICAL_LOCAL_M2_DATA_start +
_VIRTUAL_LOCAL_M2_DATA_size - 1
```

```
; Shared code and data sections in M2 memory
.provide _PHYSICAL_SHARED_M2_start,     _M2_start + _NUMBER_OF_CORES *
_VIRTUAL_LOCAL_M2_DATA_size
.provide _PHYSICAL_SHARED_M2_end,       _M2_end
```

```
; Private code sections for each core in M3 memory
.provide _PHYSICAL_LOCAL_M3_DATA_start, _M3_start + _ID_CORE * _VIRTUAL_LOCAL_M3_DATA_size
.provide _PHYSICAL_LOCAL_M3_DATA_end,   _PHYSICAL_LOCAL_M3_DATA_start +
_VIRTUAL_LOCAL_M3_DATA_size - 1
```

```
; Shared code and data sections in M3 memory
.provide _PHYSICAL_SHARED_M3_DATA_start, _M3_start + NUMBER_OF_CORES *
_VIRTUAL_LOCAL_M3_DATA_size
provide _PHYSICAL_SHARED_M3_end,      _M3_end
```

```
; Shared code and data sections in DDR memory
.provide _PHYSICAL_SHARED_DDR_start,    _DDR_start
provide _PHYSICAL_SHARED_DDR_end,      _DDR_end
```

Because we use the C startup code provided with the CW tools, we must also define the symbols used by this startup code and map them to the corresponding memory as indicated in Section 3, "CW MSC8144 Startup Code." Example 11 shows the symbols for startup C code.

**Example 11. Symbols for Startup C Code in LCF Example**

```
;--------------------------------
;         Startup Code
;--------------------------------
; These symbols are used in the default C startup code in CW
; These symbols must be present in the LCF to use this default C startup code
.provide _LocalData_b,       _VIRTUAL_LOCAL_M2_DATA_start
.provide _LocalData_size,    _VIRTUAL_LOCAL_M2_DATA_size
.provide _LocalData_e,       _VIRTUAL_LOCAL_M2_DATA_end
.provide _LocalData_Phys_b, _M2_start
```

```
.provide _StackSize, 0x2000
.provide _StackStart, _LocalData_e + 1 - _StackSize
.provide _TopOfStack, _LocalData_e + 1
```

The complete list of the LCF symbols with a few minor additions is shown in Appendix C, "msc8144ADS_common.txt File." Finally, we use symbols defined so far with the .concatenate and .att_mmu directives described in Section 2.3, "Virtual Memory" to map the translation from the virtual to physical addresses of local and private memory in the application. Also, we use the `.import` and `.export` directives described in Section 2.4, "Shared Memory Among Cores" to define the shared memory among the cores. This exercise is straightforward and is not presented here. The resulting LCF file is shown in Appendix D, "msc8144ADS.lcf File for Four MSC8144 Cores." You are encouraged to go through the exercise of understanding the directives in this file.

# 5 Debugging the LCF

The techniques for debugging a device with an MMU (and multiple caches) is beyond the scope of this application note. However, one simple, yet very useful, tool is the MMU Configurator tool available from the View → **MMU Config** menu of the CW IDE. The MMU Config tool generates C code for use by an application to program the MSC8144 MMU program and data MATTs for a desired mapping from virtual

to physical addresses. However, the MMU config tool can be used to confirm that the LCF in use is programming the MMU as intended.

After the debugger is started and execution has stopped at the `main()` function, select the CW debugger window for the core in question and invoke the MMU Config tool using the View → **MMU Config** menu of the CW IDE. Then select either the **Program MATT** or **Data MATT** tab. Figure 6 shows the MMU Config tool window for core 0 of the CW ADS stationery project with the **Data MATT** tab selected. The two entries correspond to the .m2_private_data and .private_boot sections for core 0 as defined in the corresponding LCF. Entry [0] indicates that virtual address 0xd0000 is mapped to physical address 0xC0060000 (corresponding to the .private_boot section) and entry [1] indicates that virtual address 0x0 is mapped to physical address 0xC0000000 (corresponding to the .m2_private_data section).

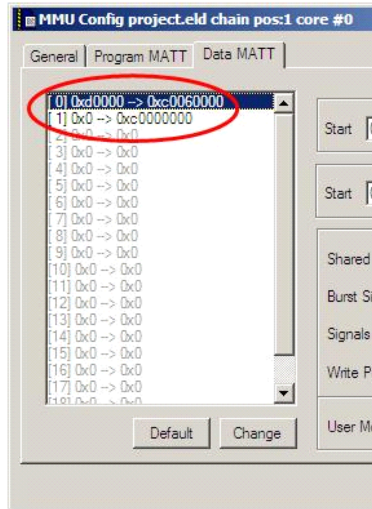


**Figure 6. MMU Config Tool**

This information can be used to verify that the core 0 MMU is programmed as expected. Note that several other sections defined in the LCF described in section Section 4, “LCF Example” do not appear in the window. This is because the application, in this case the MSC8144 ADS stationery example, does not use any of the memory in these sections.

This is a simple way to ascertain that the MMU is properly programmed by the startup code based on the LCF at the point when the application is to start execution.

# Appendix A MSC8144 Physical Memory and Peripherals

Table 5 shows the physical memory and peripheral devices on the MSC8144 DSP (DDR on ADS).

**Table 5. Physical Memory and Peripheral Device on the MSC8144 DSP**

| Device | Physical Address | Size (Bytes) |
|---|---|---|
| **Physical Memory** | | |
| DDR | 0x4000 0000 – 0x5FFF FFF | 512 M |
| M2 | 0xC000 0000 – 0xC007 FFFF | 512 K |
| M3 | 0xD0000 0000 – 0xD09F FFFF | 10 M |
| PCI | 0xE000 000 – 0xE7FF FFFF | 128 M |
| Packet Processor | 0xFEE0 0000 – 0xFEE3 FFFF | 256 K |
| Boot ROM | 0xFEF0 0000 – 0xFEF1 7FFF | 96 K |
| **SC3400 Extended Core** | | |
| OCE30 | 0xFFEF FE00 – 0xFFEF FFFF | 512 |
| EPIC | 0xFFF0 0400 – 0xFFF0 07FF | 1 K |
| DCache Registers | 0xFFF0 0800 – 0xFFF0 0BFF | 1 K |
| ICache Registers | 0xFFF0 0C00 – 0xFFF0 0FFF | 1 K |
| MMU | 0xFFF0 6000 – 0xFFF0 9FFF | 16 K |
| DPU | 0xFFF0 A000 – 0xFFF0 A2FF | 768 |
| Core Timers | 0xFFF0 A300 – 0xFFF0 A3FF | 256 |
| **Control Configuration and Status Registers (CCSR)** | | |
| DMA | 0xFFF1 0000 – 0xFFF1 03FF | 1 K |
| CLASS | 0xFFF1 8000 – 0xFFF1 AFFF | 12 K |
| DDR Controller | 0xFFF2 0000 – 0xFFF2 1FFF | 4 K |
| Clock | 0xFFF2 4000 – 0xFFF2 407F | 128 |
| Reset | 0xFFF2 4800 – 0xFFF2 4BFF | 1 K |
| I2C | 0xFFF2 4C00 – 0xFFF2 4FFF | 1 K |
| Watchdog Timers | 0xFFF2 5000 – 0xFFF2 54FF | 1.2 K |
| Timers | 0xFFF2 6000 – 0xFFF2 63FF | 1 K |
| GIC | 0xFFF2 7000 – 0xFFF2 70FF | 256 |
| HW Semaphores | 0xFFF2 7100 – 0xFFF2 71FF | 256 |
| GPIO | 0xFFF2 7200 – 0xFFF2 72FF | 256 |
| L2 ICache Registers | 0xFFF2 A000 – 0xFFF2 C01F | ~20 K |
| TDM | 0xFFF3 000 – 0xFFF4 FFFF | 128 K |

**Table 5. Physical Memory and Peripheral Device on the MSC8144 DSP (continued)**

| Device | Physical Address | Size (Bytes) |
|---|---|---|
| General configuration | 0xFFF7 8000 – 0xFFF7 803F | 64 |
| PCI | 0xFFF7 A000 – 0xFFF7 A0FF | 256 |
| UART | 0xFFF7 F000 – 0xFFF7 F03F | 64 |
| SRIO | 0xFFF8 0000 – 0xFFF9 FFFF | 128 K |
| OCeaN | 0xFFFA 0000 – 0xFFFA 00FF | 256 |
| OCeaN (system bus) | 0xFFFA 1000 – 0xFFFA 103F | 64 |

# Appendix B Requirements for Default Sections

Table 6 lists the information and allocation requirements for the default sections generated by the CW compiler linker. These sections are used in the LCF for the CW MSC8144 ADS stationery.

**Table 6. Requirement List for the Default Sections**

| Section Name | Usage | Private/Shared | MMU Descriptor |
|---|---|---|---|
| .att_mmu | MMU descriptors (used when mmu descriptors are defined in .lcf file) | Private | DATA |
| .bsstab | Table containing information about data to be initialized to 0 at startup. | Private | DATA |
| .bss | Uninitialized data | Private | DATA |
| .data | Initialized data | Private | DATA |
| .default | Section created by the assembler for code that is not put between section <name> and endsec directives. | Private if the user has private code in the application. Shared otherwise. | PROGRAM |
| .exception | C++ only. Exception table | Private | DATA |
| .exception_index | C++ only. Exception table index | Private | DATA |
| .intvec | Vector table | Private or Shared | PROGRAM |
| .ovltab | Overlay table (used overlays are defined in .lcf file) | Private | DATA |
| reserved_crt_tls | For reentrant run-time library support. The context local data variable for each core | Private | DATA |
| reserved_crt_mutex | For reentrant run-time library support. The MUTEX variables that are used by the critical region. | Shared | DATA<br>Non-cacheable |
| .rom | Read-only initialized data (or constants). Used when the –mrom option is used while linking. | Private | DATA |
| .rom_init | Initialization data to copy to RAM. Used when the –mrom option is used while linking. | Private | DATA |
| .rom_init_tables | Table containing information for copying initialization data to RAM | Private | DATA |
| .staticinit | C++ only. Table containing info for C++ static initializes. | Private | DATA |
| .text | Application code | Shared | PROGRAM |
| .zdata | Zero data Area initialized data | Yes | DATA |

# Appendix C msc8144ADS_common.txt File

Following is the `msc8144ADS_common.txt` file showing linker directives for symbol declarations used in the example described in Section 4, "LCF Example."

```
;*****************************************************************************
;********************          Symbols          ********************
;*****************************************************************************
.provide _NUMBER_OF_CORES, 4
```

```
;--------------------------------
;   MSC8144 ADS memory addresses
;--------------------------------
.provide _M2_start, 0xC0000000
.provide _M2_size,  0x00080000               ; M2 size = 512K
.provide _M2_end,    _M2_start + _M2_size - 1
.provide _M3_start, 0xD0000000
.provide _M3_size,  0x00a00000               ; M3 size = 10M
.provide _M3_end, _M3_start + _M3_size - 1
.provide _DDR_start,0x40000000
.provide _DDR_size, 0x10000000               ; DDR size = 256MB
.provide _DDR_end, _DDR_start + _DDR_size -1
```

```
;--------------------------------
;   Application virtual memory
;--------------------------------
; Virtual space for private data will translate to start of M2 memory
.provide _VIRTUAL_LOCAL_M2_DATA_start,   _M2_start
.provide _VIRTUAL_LOCAL_M2_DATA_size,    0x00010000
.provide _VIRTUAL_LOCAL_M2_DATA_end,     _VIRTUAL_LOCAL_M2_DATA_start +
_VIRTUAL_LOCAL_M2_DATA_size - 1
; Virtual space for shared data and code in M2 will have a 1:1 virtual to physical address
mapping
.provide _VIRTUAL_SHARED_M2_start,       _M2_start + _NUMBER_OF_CORES *
_VIRTUAL_LOCAL_M2_DATA_size
.provide _VIRTUAL_SHARED_M2_end,         _M2_end
; Virtual space for private code will translate to start of M3 memory
.provide _VIRTUAL_LOCAL_M3_DATA_start,   _M3_start
.provide _VIRTUAL_LOCAL_M3_DATA_size,    0x00020000
.provide _VIRTUAL_LOCAL_M3_DATA_end,     _VIRTUAL_LOCAL_M3_DATA_start +
_VIRTUAL_LOCAL_M3_DATA_size - 1
; Virtual space for shared data and code in M3 will have a 1:1 virtual to physical address
mapping
.provide _VIRTUAL_SHARED_M3_start,       _M3_start + _NUMBER_OF_CORES *
_VIRTUAL_LOCAL_M3_DATA_size
.provide _VIRTUAL_SHARED_M3_end,         _M3_end
; Virtual space for shared data and code in DDR will have a 1:1 virtual to physical address
mapping
.provide _VIRTUAL_SHARED_DDR_start,      _DDR_start
.provide _VIRTUAL_SHARED_DDR_end,        _DDR_end
```

```
;--------------------------------
; Application physical memory
;--------------------------------
```

```
; Private data sections for each core in M2 memory
.provide _PHYSICAL_LOCAL_M2_DATA_start, _M2_start + _ID_CORE * _VIRTUAL_LOCAL_M2_DATA_size
.provide _PHYSICAL_LOCAL_M2_DATA_end,   _PHYSICAL_LOCAL_M2_DATA_start +
_VIRTUAL_LOCAL_M2_DATA_size - 1
; Shared code and data sections in M2 memory
.provide _PHYSICAL_SHARED_M2_start,     _M2_start + _NUMBER_OF_CORES *
_VIRTUAL_LOCAL_M2_DATA_size
.provide _PHYSICAL_SHARED_M2_end,       _M2_end
; Private code sections for each core in M3 memory
.provide _PHYSICAL_LOCAL_M3_DATA_start, _M3_start + _ID_CORE * _VIRTUAL_LOCAL_M3_DATA_size
.provide _PHYSICAL_LOCAL_M3_DATA_end,   _PHYSICAL_LOCAL_M3_DATA_start +
_VIRTUAL_LOCAL_M3_DATA_size - 1
; Shared code and data sections in M3 memory
.provide _PHYSICAL_SHARED_M3_start,     _M3_start + _NUMBER_OF_CORES *
_VIRTUAL_LOCAL_M3_DATA_size
.provide _PHYSICAL_SHARED_M3_end,       _M3_end
; Shared code and data sections in DDR memory
.provide _PHYSICAL_SHARED_DDR_start,    _DDR_start
.provide _PHYSICAL_SHARED_DDR_end,      _DDR_end
```

---

```
;--------------------------------
;         Start up Code
;--------------------------------
; These symbols are used in the default C startup code in CW
; These symbols must be present in the LCF to use this default C startup code
.provide _LocalData_b,      _VIRTUAL_LOCAL_M2_DATA_start
.provide _LocalData_size,   _VIRTUAL_LOCAL_M2_DATA_size
.provide _LocalData_e,      _VIRTUAL_LOCAL_M2_DATA_end
.provide _LocalData_Phys_b, _M2_start
.provide _StackSize, 0x2000
.provide _StackStart, _LocalData_e + 1 - _StackSize
.provide _TopOfStack, _LocalData_e + 1
; A dynamic configuration for stack and heap is defined when the __BottomOfHeap,  _StackStart
symbols have the same value.
; A static configuration is defined when values for __BottomOfHeap and _StackStart symbols are
different.
; The value of __BottomOfHeap is the lowest address that is used by heap when a static
configuration is used.
.provide __BottomOfHeap,  _StackStart
; By default, this serves as the heap start address. The heap grows downwards.
.provide __TopOfHeap,  _TopOfStack
.assert (((__TopOfHeap == _TopOfStack) && (__BottomOfHeap == _StackStart)) ||  ((__TopOfHeap
!= _TopOfStack) && (__BottomOfHeap != _StackStart)))
; The value to set the SR in the startup code:
;        - exception mode
;        - interrupt level 31
;        - saturation on
;        - rounding mode: nearest even
.provide _SR_Setting, 0x3e4000c
; Vector base address
.provide _VBAddr, _PHYSICAL_SHARED_M3_start
```

---

```
;*****************************************************************************
;********************              Memory               ********************
;*****************************************************************************
; Private M2 data memory (reserve memory for stack)
```

```
.memory _PHYSICAL_LOCAL_M2_DATA_start, _PHYSICAL_LOCAL_M2_DATA_end - _StackSize, "rwx"
; Shared M2 code and data memory
.memory _PHYSICAL_SHARED_M2_start, _PHYSICAL_SHARED_M2_end, "rwx"
; Private M3 data memory
.memory _PHYSICAL_LOCAL_M3_DATA_start, _PHYSICAL_LOCAL_M3_DATA_end, "rwx"
; Shared M3 code and data memory
.memory _PHYSICAL_SHARED_M3_start, _PHYSICAL_SHARED_M3_end, "rwx"
; Shared DDR code and data memory
.memory _PHYSICAL_SHARED_DDR_start, _PHYSICAL_SHARED_DDR_end, "rwx"
```

# Appendix D msc8144ADS.lcf File for Four MSC8144 Cores

Following is the `msc8144ADS.lcf` file for four MSC8144 cores showing the mapping of virtual to physical local and shared memory for the example described in Section 4, “LCF Example.”

```
;******************************************************************************
;******************** Core 0 ********************
;******************************************************************************
.unit c0,""
```

```
.set _ID_CORE, 0
.include "mmu_attributes.txt"
.include "msc8144ADS_common_2.txt"
;--------------------------------
;   Shared memory space
;--------------------------------
.space m2_shared, _PHYSICAL_SHARED_M2_start, _PHYSICAL_SHARED_M2_end, \
                 "m2_shared_data", "m2_shared_text"
.space m3_shared, _PHYSICAL_SHARED_M3_start, _PHYSICAL_SHARED_M3_end, \
                 "m3_shared_data", "m3_shared_text"
.space ddr_shared, _PHYSICAL_SHARED_DDR_start, _PHYSICAL_SHARED_DDR_end, \
                 "ddr_shared_rom" , "ddr_shared_data", "ddr_shared_text"
.export "m2_shared", "m3_shared", "ddr_shared"
;--------------------------------
; M2 Private data for this core
;--------------------------------
.concatenate "m2_local_data", ".rom_init_tables", ".bsstab", ".exception", \
                 ".exception_index",".staticinit",
"reserved_crt_tls",".data", ".zdata",\
                 ".ovltab", ".att_mmu", ".bss"
.att_mmu "m2_local", _VIRTUAL_LOCAL_M2_DATA_start, _VIRTUAL_LOCAL_M2_DATA_end, \
                 "m2_local_data", attribute : USER_DATA_MMU_DEF, \
                                   physical_address: _PHYSICAL_LOCAL_M2_DATA_start
;--------------------------------
; M2 shared data and code
;--------------------------------
.concatenate "m2_shared_data" ,".m2_shared_data"
.concatenate "m2_shared_text", ".m2_shared_text"
.att_mmu "m2_shared_mem",  _VIRTUAL_SHARED_M2_start, _VIRTUAL_SHARED_M2_end,\
                 "m2_shared_data",\
                          base_address: _PHYSICAL_SHARED_M2_start, \
                          attribute: SHARED_DATA_MMU_DEF, \
                          physical_address: _PHYSICAL_SHARED_M2_start, \
                 "m2_shared_text", \
                          attribute: SYSTEM_PROG_MMU_DEF, \
                          base_address: @vsecend("m2_shared_data"), \
                          physical_address: @secend("m2_shared_data")
;--------------------------------
; M3 Private data for this core
;--------------------------------
.concatenate "m3_local_data", ".m3_local_data"
.att_mmu "m3_local", _VIRTUAL_LOCAL_M3_DATA_start, _VIRTUAL_LOCAL_M3_DATA_end, \
                 "m3_local_data", attribute : USER_DATA_MMU_DEF, \
```

```
                  physical_address: _PHYSICAL_LOCAL_M3_DATA_start
;--------------------------------
; Shared data and code in M3
;--------------------------------
.concatenate "m3_shared_data" ,"reserved_crt_mutex", ".m3_shared_data"
.concatenate "m3_shared_text", ".intvec",".text", ".default"
.att_mmu "m3_shared",  _VIRTUAL_SHARED_M3_start, _VIRTUAL_SHARED_M3_end,\
                  "m3_shared_text", \
                           base_address: _PHYSICAL_SHARED_M3_start, \
                           attribute: SYSTEM_PROG_MMU_DEF, \
                           physical_address: _PHYSICAL_SHARED_M3_start, \
                  "m3_shared_data",\
                           attribute: SHARED_DATA_MMU_DEF, \
                           base_address: @vsecend("m3_shared_text"), \
                           physical_address: @secend("m3_shared_text")
;--------------------------------
; Shared data and code in DDR
;--------------------------------
.concatenate "ddr_shared_rom",".rom", ".init_table", ".rom_init"
.att_mmu DDR_mmu, _VIRTUAL_SHARED_DDR_start,_VIRTUAL_SHARED_DDR_end, \
                  "ddr_shared_rom", \
                           attribute : SYSTEM_DATA_MMU_DEF, \
                           after_physical_address: _PHYSICAL_SHARED_DDR_start
; DDR Data and Code Memory
.concatenate "ddr_shared_data", ".ddr_shared_data"
.concatenate "ddr_shared_text", ".ddr_shared_code"
.att_mmu DDR_shared, _VIRTUAL_SHARED_DDR_start,_VIRTUAL_SHARED_DDR_end, \
                  "ddr_shared_data", \
                           attribute : SHARED_DATA_MMU_DEF, \
                           after_physical_address: _PHYSICAL_SHARED_DDR_start, \
                  "ddr_shared_text", \
                           attribute : SYSTEM_PROG_MMU_DEF, \
                           after_physical_address: _PHYSICAL_SHARED_DDR_start
.entry ___crt0_start
```

---

```
;*****************************************************************************
;******************** Core 1 ********************
;*****************************************************************************
.unit c1
```

---

```
.set _ID_CORE, 1
.include "mmu_attributes.txt"
.include "msc8144ADS_common_2.txt"
;--------------------------------
;   Shared memory space
;--------------------------------
.import "c0`m2_shared", "c0`m3_shared", "c0`ddr_shared"
;--------------------------------
; M2 Private data for this core
;--------------------------------
.concatenate "m2_local_data", ".rom_init_tables", ".bsstab", ".exception", \
                  ".exception_index",".staticinit",
"reserved_crt_tls",".data", \
                  ".zdata", ".ovltab", ".att_mmu", ".bss"
.att_mmu "m2_local", _VIRTUAL_LOCAL_M2_DATA_start, _VIRTUAL_LOCAL_M2_DATA_end, \
                  "m2_local_data", attribute : USER_DATA_MMU_DEF, \
```

```
                          physical_address: _PHYSICAL_LOCAL_M2_DATA_start
;--------------------------------
; M3 Private data for this core
;--------------------------------
.concatenate "m3_local_data", ".m3_local_data"
.att_mmu "m3_local", _VIRTUAL_LOCAL_M3_DATA_start, _VIRTUAL_LOCAL_M3_DATA_end, \
                 "m3_local_data", attribute : USER_DATA_MMU_DEF, \
                          physical_address: _PHYSICAL_LOCAL_M3_DATA_start
.entry ___crt0_start
```

```
;*****************************************************************************
;******************** Core 2 ********************
;*****************************************************************************
.unit c2
```

```
.set _ID_CORE, 2
.include "mmu_attributes.txt"
.include "msc8144ADS_common_2.txt"
;--------------------------------
;   Shared memory space
;--------------------------------
.import "c0`m2_shared", "c0`m3_shared", "c0`ddr_shared"
;--------------------------------
; M2 Private data for this core
;--------------------------------
.concatenate "m2_local_data", ".rom_init_tables", ".bsstab", ".exception", \
                 ".exception_index",".staticinit",
"reserved_crt_tls",".data", \
                 ".zdata", ".ovltab", ".att_mmu", ".bss"
.att_mmu "m2_local", _VIRTUAL_LOCAL_M2_DATA_start, _VIRTUAL_LOCAL_M2_DATA_end, \
                 "m2_local_data", attribute : USER_DATA_MMU_DEF, \
                          physical_address: _PHYSICAL_LOCAL_M2_DATA_start
;--------------------------------
; M3 Private data for this core
;--------------------------------
.concatenate "m3_local_data", ".m3_local_data"
.att_mmu "m3_local", _VIRTUAL_LOCAL_M3_DATA_start, _VIRTUAL_LOCAL_M3_DATA_end, \
                 "m3_local_data", attribute : USER_DATA_MMU_DEF, \
                          physical_address: _PHYSICAL_LOCAL_M3_DATA_start
.entry ___crt0_start
```

```
;*****************************************************************************
;******************** Core 3 ********************
;*****************************************************************************
.unit c3
```

```
.set _ID_CORE, 3
.include "mmu_attributes.txt"
.include "msc8144ADS_common_2.txt"
;--------------------------------
;   Shared memory space
;--------------------------------
.import "c0`m2_shared", "c0`m3_shared", "c0`ddr_shared"
;--------------------------------
; ; M2 Private data for this core
```

```
;--------------------------------
.concatenate "m2_local_data", ".rom_init_tables", ".bsstab", ".exception", \
                 ".exception_index",".staticinit",
"reserved_crt_tls",".data", \
                 ".zdata", ".ovltab", ".att_mmu", ".bss"
.att_mmu "m2_local", _VIRTUAL_LOCAL_M2_DATA_start, _VIRTUAL_LOCAL_M2_DATA_end, \
                 "m2_local_data", attribute : USER_DATA_MMU_DEF, \
                          physical_address: _PHYSICAL_LOCAL_M2_DATA_start
;--------------------------------
; M3 Private data for this core
;--------------------------------
.concatenate "m3_local_data", ".m3_local_data"
.att_mmu "m3_local", _VIRTUAL_LOCAL_M3_DATA_start, _VIRTUAL_LOCAL_M3_DATA_end, \
                 "m3_local_data", attribute : USER_DATA_MMU_DEF, \
                          physical_address: _PHYSICAL_LOCAL_M3_DATA_start
.entry ___crt0_start
```

THIS PAGE INTENTIONALLY LEFT BLANK

***How to Reach Us:***


**Home Page:**
www.freescale.com

**Web Support:**
http://www.freescale.com/support

**USA/Europe or Locations Not Listed:**
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or
+1-480-768-2130
www.freescale.com/support

**Europe, Middle East, and Africa:**
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

**Japan:**
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku
Tokyo 153-0064
Japan
0120 191014 or
+81 3 5437 9125
support.japan@freescale.com

**Asia/Pacific:**
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

**For Literature Requests Only:**
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
+1-800 441-2447 or
+1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com



Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals” must be validated for each customer application by customer’s technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org. All other product or service names are the property of their respective owners.

© Freescale Semiconductor, Inc., 2006. All rights reserved.



Document Number: AN3203
Rev. 0
12/2006

freescale™
semiconductor