Freescale Semiconductor
Application Note

Document Number: AN3572
Rev. 0, 11/2007

Recommendations for Creating a Multi-Core Application for the MSC8144 Architecture

by *Mihail Nistor*
Freescale Semiconductor, Inc.

This application note provides recommended guidelines for developing multi-core applications for the MSC8144 architecture using the CodeWarrior® IDE tools.

Contents

freescale™ semiconductor

1 Considerations

The MSC8144 DSP has two new features that must be considered when developing applications:

- Defining the static configuration for the core subsystem memory management unit (MMU) task descriptors.
- Defining which information is private or shared among the cores by using directives from the CodeWarrior linker control file (LCF).

In addition, you may want to consider the following options:

2 MMU Task Descriptor Static Configuration

To define the static configuration for task descriptors of the MMU, consider that a segment is equivalent to a descriptor (.concatenate directive) from the MMU.

2.1 Using an LCF to Define the Descriptors

Use an LCF for a single core application without MMU support to allow the SC3400 architecture to compile/link the application to define the component of each descriptor. You can rename a section or place a variable in a specified section by using the application configuration file or by using the “pragma” directives in C files. Define each MMU descriptor, which the information can be added as a prefix in the name of each section that is a component of the descriptor:

1. Information about the hierarchy of memory when the descriptor is placed in the physical memory space. The MMU descriptor can be placed in one of the following: M2, M3 or DDR memory (for example, “m2” is the prefix for M2 memory, “m3” is the prefix for M3 memory, and “ddr” is the prefix for DDR memory).
2. Information about type of memory space. The MMU in the MSC8144 has two types of memory space:
   a) Program memory space. (for example, “`text`” is the prefix for program memory).
   b) Data memory space. The Data memory space can split in three parts:
      – Ready only data (for example, “`rom`” is the prefix).
      – Initialized data (for example, “`data`” is the prefix).
      – Uninitialized data (for examplem “`bss`” is the prefix).
3. Information about type of cache. For each type of memory space the MMU has lists of possible configuration of the cache.
   a) For data memory space:
      – `DATA NONCACHEABLE WRITE THROUGH STALL` (that is, “`non_cacheable_wts`” is the prefix).
      – `DATA NONCACHEABLE WRITE THROUGH` (for example, “`non_cacheable_wt`” is the prefix).
      – `DATA CACHEABLE WRITE BACK` (for example, “`cacheble_wb`” is the prefix).
      – `DATA CACHEABLE WRITE THROUGH` (for example, “`cacheable_wt`” is the prefix).

b) For program space:
   - `PROGRAM CACHEABLE` (for example, "`cacheable`" is the prefix)
   - `PROGRAM NONCACHEABLE` (for example, "`non_cacheable`" is the prefix)

NOTE

The cacheable property for program descriptor can always be set.

4. Information about task identification. The MMU supports two types:
   a) The system task. The value of ID is zero. This value can be changed in the LCF by using the `_SYSTEM_TASK_ID` symbol. The system task always runs in super user mode and the shared attribute is set. If the shared attribute is set, this descriptor cannot be overlapped in the virtual memory space by another descriptor without enabling the priority mechanism. (the prefix is "`sys`").
   b) The user task. The value if ID is between ranges 1 to 255. The user task usually runs in the user mode and the shared attribute is not set. The shared attribute needs to be set when the descriptor is shared between tasks. (the prefix is "`user1`" for task one).
5. Information about dependence among cores. The code or data can be private or shared among cores. If the code is only shared among cores and the data can be private or shared among cores, we have a Single Instruction Multi Data (SIMD) model of application. This is the common model that is used. If the code and data are private and shared among cores, we have the Multi Instruction Multi Data (MIMD) model of application. For the SIMD model of application, we can have the following cases:
   a) Private data not shared between tasks; data can be cacheable.
   b) Private data shared between tasks; data can be cacheable.
   c) Shared data can be cacheable when the data are read only, or each core can access a sub-range from the data and these sub-ranges do not overlap. Otherwise, the shared data must be non-cacheable, because the hardware does not support cache coherence among cores.
   d) Shared program can always be cacheable.

   The prefix for private is "private" and the prefix for shared is "shared".

All sections must be mentioned in a `.segment` directive. Therefore, the user must use one of the following options to force the linker to generate a error or warning message if a section is not explicitly mentioned in a segment directive:

- `-Xlnk "-enable-error-placing-section-on-first-fit-basis"`
- `-Xlnk "-enable-warn-placing-section-on-first-fit-basis"`

2.2 Using the ELF Utilities and Linker-Generated Map File to Evaluate Sections and Segments

The executable and linkable format (ELF) utilities allow you evaluate the sections and segments generated by the LCF. These utilities include the following:

- `sc100-size`. This utility generates information about size for each section/segment.
- `sc100-elfdump`. This utility generates information about header of ELF file, header of segment/section. By using the information from header of section, you can discover the property of the section:
  - If the type of section is `SHT_NOBITS`, indicates uninitialized data.
  - If the type of section is `SHT_PROGBITS` and the flags attribute include:
    - `SHF_ALLOC`, indicates read only data
    - `SHF_ALLOC` and `SHF_WRITE`, indicates initialized data
    - `SHF_ALLOC` and `SHF_EXECINSTR`, indicates an executable program.
- `sc100-elfinfo`. This utility generates for each section:
  - MMU descriptor types (program or data)
  - Section size
  - Virtual address
  - Physical address.

NOTE

The virtual address is different from the physical address if there are overlays defined in the linker control file.

NOTE

The overlay function is not supported in the MSC8144 devices. Some of the overlay support is provided by the hardware cache support.

- Use the map file generated by the linker to find information about the size of segment or section and any other information about the placement of the object in the memory.

2.3 Placing Each Object in the Descriptor

The generation and evaluation processes yield the following information for each descriptor:

- Size of each descriptor
- The list of sections that defines each descriptor
- All properties for each descriptor

The more difficult part is taking the defined information for each descriptor and deciding where each object must be placed in the descriptor. Example 1 shows how to configuring the MMU Task Static Descriptors using a typical set of section and segment data. The comments provide detailed descriptions.

Example 1. Configuring MMU Task Static Descriptors

```
.firstfit _M2_PRIVATE_start
; Private data in M2
; Descriptor properties:
;        -cacheable write back
;        -prefetch is enabled
;        -read & write access in both user and supervisor mode
;        -burst size 4
;        -system task (shared between tasks)
.segment descriptor__m2__cacheable_wb__sys__private__data, \
        ".zdata", \
        ".m2__cacheable_wb__sys__private__data",\
        ".m2__cacheable_wb__sys__private__rom", \
        ".m2__cacheable_wb__sys__private__bss"
; The descriptor__m2__cacheable_wb__sys__private__data is not defined in MMU
; if the value of
; size_required_by_mmu_for__descriptor__m2__cacheable_wb__sys__private__data
; symbol is zero.
.set size_required_by_mmu_for__descriptor__m2__cacheable_wb__sys__private__data, \
        @iif(@segsize(descriptor__m2__cacheable_wb__sys__private__data) == 0, 0, \
        @mmu_align(@segsize(descriptor__m2__cacheable_wb__sys__private__data)))


.firstfit _M3_PRIVATE_start
; Private data in M3
; Descriptor properties:
; -cacheable write back
; -prefetch is enabled
; -read & write access in both user and supervisor mode
; -burst size 4
; -system task (shared between tasks)
.segment descriptor__m3__cacheable_wb__sys__private__data, \
        ".m3__cacheable_wb__sys__private__data",\
        "reserved_crt_tls", \
        ".data", \
        ".m3__cacheable_wb__sys__private__rom", \
        ".m3__cacheable_wb__sys__private__bss", \
        ".bss"
; The descriptor__m3__cacheable_wb__sys__private__data is not defined in MMU
; if the value of
; size_required_by_mmu_for__descriptor__m3__cacheable_wb__sys__private__data
; symbol is zero.
.set size_required_by_mmu_for__descriptor__m3__cacheable_wb__sys__private__data, \
        @iif(@segsize(descriptor__m3__cacheable_wb__sys__private__data) ==0, 0, \
```

```
        @mmu_align(@segsize(descriptor__m3__cacheable_wb__sys__private__data)))


.firstfit _DDR_PRIVATE_start
; Private data in DDR
; Descriptor properties:
;       -cacheable write back
;       -prefetch is enabled
;       -read & write access in both user and supervisor mode
;       -burst size 4
;       -system task (shared between tasks)
.segment descriptor__ddr__cacheable_wb__sys__private__data, \
        ".ddr__cacheable_wb__sys__private__data",\
        ".ddr__cacheable_wb__sys__private__rom", \
        ".bsstab", ".init_table", ".rom_init", \
        ".rom_init_tables", ".exception", ".exception_index", ".staticinit", \
        ".ddr__cacheable_wb__sys__private__bss"
; The descriptor__ddr__cacheable_wb__sys__private__data is not defined in MMU
; if the value of
; size_required_by_mmu_for__descriptor__ddr__cacheable_wb__sys__private__data
; symbol is zero.
.set size_required_by_mmu_for__descriptor__ddr__cacheable_wb__sys__private__data, \
        @iif(@segsize(descriptor__ddr__cacheable_wb__sys__private__data), 0, \
        @mmu_align(@segsize(descriptor__ddr__cacheable_wb__sys__private__data)))


.firstfit _M2_SHARED_start
; Shared text in M2
; Descriptor properties:
;       -cacheable
;       -prefetch is enabled
;       -execute access in both user and supervisor mode
;       -burst size 4
;       -system task (shared between tasks)
.segment descriptor__m2__cacheable__sys__shared__text, \
        ".m2__cacheable__sys__shared__text", \
        ".text", ".default"
; The descriptor__m2__cacheable__sys__shared__text is not defined in MMU
; if the value of
; size_required_by_mmu_for__descriptor__m2__cacheable__sys__shared__text
; symbol is zero.
.set size_required_by_mmu_for__descriptor__m2__cacheable__sys__shared__text, \
        @iif(@segsize(descriptor__m2__cacheable__sys__shared__text) == 0, 0, \
        @mmu_align(@segsize(descriptor__m2__cacheable__sys__shared__text)))


; Shared data in M2
; Descriptor properties:
;       -noncacheable write through
;       -prefetch is enabled
;       -read & write access in both user and supervisor mode
;       -burst size 4
;       -system task (shared between tasks)
.segment descriptor__m2__non_cacheable_wt__sys__shared__data, \
        "reserved_crt_mutex", \
        ".m2__non_cacheable_wt__sys__shared__data",\
        ".m2__non_cacheable_wt__sys__shared__rom",\
```

```
        ".m2__non_cacheable_wt__sys__shared__bss"
; The descriptor__m2__non_cacheable_wt__sys__shared__data is not defined in MMU
; if the value of
; size_required_by_mmu_for__descriptor__m2__non_cacheable_wt__sys__shared__data
; symbol is zero.
.set size_required_by_mmu_for__descriptor__m2__non_cacheable_wt__sys__shared__data,\
        @iif(@segsize(descriptor__m2__non_cacheable_wt__sys__shared__data)==0, 0, \
        @mmu_align(@segsize(descriptor__m2__non_cacheable_wt__sys__shared__data)))


; Shared data in M2
; Descriptor properties:
;        -cacheable write back
;        -prefetch is enabled
;        -read & write access in both user and supervisor mode
;        -burst size 4 ; -system task (shared between tasks)
.segment descriptor__m2__cacheable_wb__sys__shared__data, \
        ".m2__cacheable_wb__sys__shared__data",\
        ".m2__cacheable_wb__sys__shared__rom",\
        ".m2__cacheable_wb__sys__shared__bss"
; The descriptor__m2__cacheable_wb__sys__shared__data is not defined in MMU
; if the value of
; size_required_by_mmu_for__descriptor__m2__cacheable_wb__sys__shared__data
; symbol is zero.
.set size_required_by_mmu_for__descriptor__m2__cacheable_wb__sys__shared__data, \
        @iif(@segsize(descriptor__m2__cacheable_wb__sys__shared__data) ==0, 0, \
        @mmu_align(@segsize(descriptor__m2__cacheable_wb__sys__shared__data)))


.firstfit _M3_SHARED_start
; Shared text in M3
; Descriptor properties:
;        -cacheable
;        -prefetch is enabled
;        -execute access in both user and supervisor mode
;        -burst size 4
;        -system task (shared between tasks)
.segment descriptor__m3__cacheable__sys__shared__text, \
        ".m3__cacheable__sys__shared__text"
; The descriptor__m3__cacheable__sys__shared__text is not defined in MMU
; if the value of size_required_by_mmu_for__descriptor__m3__cacheable__sys__shared__text
; symbol is zero.
.set size_required_by_mmu_for__descriptor__m3__cacheable__sys__shared__text, \
        @iif(@segsize(descriptor__m3__cacheable__sys__shared__text) == 0, 0, \
        @mmu_align(@segsize(descriptor__m3__cacheable__sys__shared__text)))


; Shared data in M3
; Descriptor properties:
;        -noncacheable write through
;        -prefetch is enabled
;        -read & write access in both user and supervisor mode
;        -burst size 4
;        -system task (shared between tasks)
.segment descriptor__m3__non_cacheable_wt__sys__shared__data, \
        ".m3__non_cacheable_wt__sys__shared__data",\
        ".m3__non_cacheable_wt__sys__shared__rom",\
```

```
        ".m3__non_cacheable_wt__sys__shared__bss"
; The descriptor__m3__non_cacheable_wt__sys__shared__data is not defined in MMU
; if the value of
; size_required_by_mmu_for__descriptor__m3__non_cacheable_wt__sys__shared__data
; symbol is zero.
.set size_required_by_mmu_for__descriptor__m3__non_cacheable_wt__sys__shared__data, \
        @iif(@segsize(descriptor__m3__non_cacheable_wt__sys__shared__data)==0, 0, \
        @mmu_align(@segsize(descriptor__m3__non_cacheable_wt__sys__shared__data)))


; Shared data in M3
; Descriptor properties:
;       -cacheable write back
;       -prefetch is enabled
;       -read & write access in both user and supervisor mode
;       -burst size 4
;       -system task (shared between tasks)
.segment descriptor__m3__cacheable_wb__sys__shared__data, \
        ".m3__cacheable_wb__sys__shared__data",\
        ".m3__cacheable_wb__sys__shared__rom",\
        ".rom",\
        ".m3__cacheable_wb__sys__shared__bss"
; The descriptor__m3__cacheable_wb__sys__shared__data is not defined in MMU
; if the value of
; size_required_by_mmu_for__descriptor__m3__cacheable_wb__sys__shared__data
; symbol is zero.
.set size_required_by_mmu_for__descriptor__m3__cacheable_wb__sys__shared__data, \
        @iif(@segsize(descriptor__m3__cacheable_wb__sys__shared__data) ==0, 0, \
        @mmu_align(@segsize(descriptor__m3__cacheable_wb__sys__shared__data)))


.org _VBAddr
; Place all the shared data memory sections in
; "descriptor__m3__cacheable__sys__shared__text_boot"
; Shared text in M3 ; Descriptor properties:
;       -cacheable
;       -prefetch is enabled
;       -execute access in both user and supervisor mode
;       -burst size 4
;       -system task (shared between tasks)
.segment descriptor__m3__cacheable__sys__shared__text_boot, \
        ".intvec", ".text_boot"
; The descriptor__m3__cacheable__sys__shared__text_boot is not defined in MMU
; if the value of
; size_required_by_mmu_for__descriptor__m3__cacheable__sys__shared__text_boot
; symbol is zero.
.set size_required_by_mmu_for__descriptor__m3__cacheable__sys__shared__text_boot, \
        @iif(@segsize(descriptor__m3__cacheable__sys__shared__text_boot) ==0, 0, \
        @mmu_align(@segsize(descriptor__m3__cacheable__sys__shared__text_boot)))


.firstfit _DDR_SHARED_start
; Shared text in DDR
; Descriptor properties:
;       -cacheable
;       -prefetch is enabled
;       -execute access in both user and supervisor mode
```

```
;        -burst size 4
;        -system task (shared between tasks)
.segment descriptor__ddr__cacheable__sys__shared__text, \
        ".ddr__cacheable__sys__shared__text"
; The descriptor__ddr__cacheable__sys__shared__text is not defined in MMU
; if the value of
; size_required_by_mmu_for__descriptor__ddr__cacheable__sys__shared__text
; symbol is zero.
.set size_required_by_mmu_for__descriptor__ddr__cacheable__sys__shared__text, \
        @iif(@segsize(descriptor__ddr__cacheable__sys__shared__text) ==0, 0, \
        @mmu_align(@segsize(descriptor__ddr__cacheable__sys__shared__text)))


; Shared data in DDR
; Descriptor properties:
;        -noncacheable write through
;        -prefetch is enabled
;        -read & write access in both user and supervisor mode
;        -burst size 4
;        -system task (shared between tasks)
.segment descriptor__ddr__non_cacheable_wt__sys__shared__data, \
        ".ddr__non_cacheable_wt__sys__shared__data",\
        ".ddr__non_cacheable_wt__sys__shared__rom",\
        ".ddr__non_cacheable_wt__sys__shared__bss"
; The descriptor__ddr__non_cacheable_wt__sys__shared__data is not defined in MMU
; if the value of
; size_required_by_mmu_for__descriptor__ddr__non_cacheable_wt__sys__shared__data
; symbol is zero.
.set size_required_by_mmu_for__descriptor__ddr__non_cacheable_wt__sys__shared__data, \
        @iif(@segsize(descriptor__ddr__non_cacheable_wt__sys__shared__data) == 0, 0, \
        @mmu_align(@segsize(descriptor__ddr__non_cacheable_wt__sys__shared__data)))


; Shared data in DDR
; Descriptor properties:
;        -cacheable write back
;        -prefetch is enabled
;        -read & write access in both user and supervisor mode
;        -burst size 4
;        -system task (shared between tasks)
.segment descriptor__ddr__cacheable_wb__sys__shared__data, \
        ".ddr__cacheable_wb__sys__shared__data",\
        ".ddr__cacheable_wb__sys__shared__rom",\
        ".ddr__cacheable_wb__sys__shared__bss"
; The descriptor__ddr__cacheable_wb__sys__shared__data is not defined in MMU
; if the value of
; size_required_by_mmu_for__descriptor__ddr__cacheable_wb__sys__shared__data
; symbol is zero.
.set size_required_by_mmu_for__descriptor__ddr__cacheable_wb__sys__shared__data,\
        @iif(@segsize(descriptor__ddr__cacheable_wb__sys__shared__data) ==0, 0, \
        @mmu_align(@segsize(descriptor__ddr__cacheable_wb__sys__shared__data)))
```

3 Defining Private and Shared Information

Select an LCF template for single core or multi core to define which information is private or shared among the cores. Use the MSC8144 architecture so that you compile/link the file for your application. The LCF template uses the following five files to define the application:

- `mmu_attributes.txt` defines:
  - MMU attributes that are used in the attribute field from MMU directive for each descriptor.
  - MMU configuration that includes:
    - Minimum size of a region
    - Maximum size of a region
    - System task identifier
    - Maximum counter of data descriptor
    - Maximum counter of program descriptor
    - A means to specify that the descriptors cannot overlap in the virtual memory space
    - A means to force a descriptor to overlap in virtual memory in the linker control file.
  - the protection mode is selected
  - the system task identification is selected
  - the user task identification is selected. This task will be set in the MMU register by default in the second hook implemented in the runtime library
- `common.txt` defines:
  - virtual and physical memory layout available for core
  - value for the status register (SR)
  - Range of stack and heap
- `descriptors.txt` defines all the sections of a MMU descriptor.
- `mmu_private_data.txt` defines the setting of the MMU private data descriptors.
- `crtscbmm.cmd` defines the settings of the MMU shared data or program descriptors (base address, virtual address, properties) and the application layout.

Place all sections in one of the available descriptors in `descriptors.txt`, based on the information defined using the process described in Section 2, “MMU Task Descriptor Static Configuration.” Example 2 shows how to assign the task descriptors as private or shared. The comments provide programming details.

Example 2. Assigning Private and Shared Information

```
; Private boot data in M2/M3/DDR (MMU tables and stack)
; Descriptor properties:
;        -cacheable write back
;        -prefetch is enabled
;        -read & write access in both user and supervisor mode
;        -burst size 4
;        -system task (shared between tasks)
; Change the properties in .att_mmu directive for
; descriptor__xxx__cacheable_wb__sys__private__data__boot in local_data.txt
.concatenate "descriptor__xxx__cacheable_wb__sys__private__data__boot", \
".ovltab", ".att_mmu"


; Private data in M2
; Descriptor properties:
;        -cacheable write back
;        -prefetch is enabled
;         -read & write access in both user and supervisor mode
;        -burst size 4
;        -system task (shared between tasks)
; Change the properties in .att_mmu directive for
; descriptor__m2__cacheable_wb__sys__private__data in local_data.txt
.concatenate "descriptor__m2__cacheable_wb__sys__private__data", \
        ".zdata", ".m2__cacheable_wb__sys__private__data",\
        ".m2__cacheable_wb__sys__private__rom", \
        ".m2__cacheable_wb__sys__private__bss"


; Private data in M3
; Descriptor properties:
;        -cacheable write back
;        -prefetch is enabled
;        -read & write access in both user and supervisor mode
;        -burst size 4
;        -system task (shared between tasks)
; Change the properties in .att_mmu directive for
; descriptor__m3__cacheable_wb__sys__private__data in local_data.txt
.concatenate "descriptor__m3__cacheable_wb__sys__private__data", \
        ".m3__cacheable_wb__sys__private__data",\
        "reserved_crt_tls", ".data", \
        ".m3__cacheable_wb__sys__private__rom", \
        ".m3__cacheable_wb__sys__private__bss", \
        ".bss"


; Private data in DDR
; Descriptor properties:
;        -cacheable write back
;        -prefetch is enabled
;        -read & write access in both user and supervisor mode
;        -burst size 4
;        -system task (shared between tasks)
; Change the properties in .att_mmu directive for
; descriptor__ddr__cacheable_wb__sys__private__data in local_data.txt
.concatenate "descriptor__ddr__cacheable_wb__sys__private__data", \
        ".ddr__cacheable_wb__sys__private__data",\
```

```
        ".ddr__cacheable_wb__sys__private__rom", \
        ".bsstab", ".init_table", ".rom_init", \
        ".rom_init_tables", ".exception", ".exception_index", ".staticinit", \
        ".ddr__cacheable_wb__sys__private__bss"


; Place all the shared data memory sections in
; "descriptor__m2__non_cacheable_wt__sys__shared__data"
; Shared data in M2
; Descriptor properties:
;       -noncacheable write through
;       -prefetch is enabled
;       -read & write access in both user and supervisor mode
;       -burst size 4
;       -system task (shared between tasks)
; Change the properties in .att_mmu directive for
; descriptor__m2__non_cacheable_wt__sys__shared__data in crtscbmm.cmd
.concatenate "descriptor__m2__non_cacheable_wt__sys__shared__data", \
        "reserved_crt_mutex", \
        ".m2__non_cacheable_wt__sys__shared__data",\
        ".m2__non_cacheable_wt__sys__shared__rom",\
        ".m2__non_cacheable_wt__sys__shared__bss"


; Place all the shared data memory sections in
; "descriptor__m2__cacheable_wb__sys__shared__data"
; Shared data in M2
; Descriptor properties:
;       -cacheable write back
;       -prefetch is enabled
;       -read & write access in both user and supervisor mode
;       -burst size 4
;       -system task (shared between tasks)
; Change the properties in .att_mmu directive for
; descriptor__m2__cacheable_wb__sys__shared__data in crtscbmm.cmd
.concatenate "descriptor__m2__cacheable_wb__sys__shared__data", \
        ".m2__cacheable_wb__sys__shared__data",\
        ".m2__cacheable_wb__sys__shared__rom",\
        ".m2__cacheable_wb__sys__shared__bss"


; Place all the shared data memory sections in
; "descriptor__m3__non_cacheable_wt__sys__shared__data"
; Shared data in M3
; Descriptor properties:
;       -noncacheable write through
;       -prefetch is enabled
;       -read & write access in both user and supervisor mode
;       -burst size 4
;       -system task (shared between tasks)
; Change the properties in .att_mmu directive for
; descriptor__m3__non_cacheable_wt__sys__shared__data in crtscbmm.cmd
.concatenate "descriptor__m3__non_cacheable_wt__sys__shared__data", \
        ".m3__non_cacheable_wt__sys__shared__data",\
        ".m3__non_cacheable_wt__sys__shared__rom",\
        ".m3__non_cacheable_wt__sys__shared__bss"
```

```
; Place all the shared data memory sections in
; "descriptor__m3__cacheable_wb__sys__shared__data"
; Shared data in M3
; Descriptor properties:
;        -cacheable write back
;        -prefetch is enabled
;        -read & write access in both user and supervisor mode
;        -burst size 4
;        -system task (shared between tasks)
; Change the properties in .att_mmu directive for
; descriptor__m3__cacheable_wb__sys__shared__data in crtscbmm.cmd
.concatenate "descriptor__m3__cacheable_wb__sys__shared__data", \
        ".m3__cacheable_wb__sys__shared__data",\
        ".m3__cacheable_wb__sys__shared__rom",\
        ".rom",\
        ".m3__cacheable_wb__sys__shared__bss"


; Place all the shared data memory sections in
; "descriptor__ddr__non_cacheable_wt__sys__shared__data"
; Shared data in DDR
; Descriptor properties:
;        -noncacheable write through
;        -prefetch is enabled
;        -read & write access in both user and supervisor mode
;        -burst size 4
;        -system task (shared between tasks)
; Change the properties in .att_mmu directive for
; descriptor__ddr__non_cacheable_wt__sys__shared__data in crtscbmm.cmd
.concatenate "descriptor__ddr__non_cacheable_wt__sys__shared__data", \
        ".ddr__non_cacheable_wt__sys__shared__data",\
        ".ddr__non_cacheable_wt__sys__shared__rom",\
        ".ddr__non_cacheable_wt__sys__shared__bss" \


; Place all the shared data memory sections in
; "descriptor__ddr__cacheable_wb__sys__shared__data"
; Shared data in DDR
; Descriptor properties:
;        -cacheable write back
;        -prefetch is enabled
;        -read & write access in both user and supervisor mode
;        -burst size 4
;        -system task (shared between tasks)
; Change the properties in .att_mmu directive for
; descriptor__ddr__cacheable_wb__sys__shared__data in crtscbmm.cmd
.concatenate "descriptor__ddr__cacheable_wb__sys__shared__data", \
        ".ddr__cacheable_wb__sys__shared__data",\
        ".ddr__cacheable_wb__sys__shared__rom",\
        ".ddr__cacheable_wb__sys__shared__bss"


; Place all the shared data memory sections in
; "descriptor__m2__cacheable__sys__shared__text"
; Shared text in M2
; Descriptor properties:
```

```
;        -cacheable
;        -prefetch is enabled
;        -execute access in both user and supervisor mode
;        -burst size 4
;        -system task (shared between tasks)
; Change the properties in .att_mmu directive for
; descriptor__m2__cacheable__sys__shared__text in crtscbmm.cmd
.concatenate "descriptor__m2__cacheable__sys__shared__text", \
        ".m2__cacheable__sys__shared__text", \
        ".text", ".default"


; Place all the shared data memory sections in
; "descriptor__m3__cacheable__sys__shared__text"
; Shared text in M3
; Descriptor properties:
;        -cacheable
;        -prefetch is enabled
;        -execute access in both user and supervisor mode
;        -burst size 4
;        -system task (shared between tasks)
; Change the properties in .att_mmu directive for
; descriptor__m3__cacheable__sys__shared__text in crtscbmm.cmd
.concatenate "descriptor__m3__cacheable__sys__shared__text", \
        ".m3__cacheable__sys__shared__text"


; Place all the shared data memory sections in
; "descriptor__m3__cacheable__sys__shared__text_boot"
; Shared text in M3
; Descriptor properties:
;        -cacheable
;        -prefetch is enabled
;        -execute access in both user and supervisor mode
;        -burst size 4 ; -system task (shared between tasks)
; Change the properties in .att_mmu directive for
; descriptor__m3__cacheable__sys__shared__text in crtscbmm.cmd
.concatenate "descriptor__m3__cacheable__sys__shared__text_boot", \
        ".intvec", ".text_boot"


; Place all the shared data memory sections in
; "descriptor__ddr__cacheable__sys__shared__text"
; Shared text in DDR
; Descriptor properties:
;        -cacheable
;        -prefetch is enabled
;        -execute access in both user and supervisor mode
;        -burst size 4
;        -system task (shared between tasks)
; Change the properties in .att_mmu directive for\
; descriptor__ddr__cacheable__sys__shared__text in crtscbmm.cmd
.concatenate "descriptor__ddr__cacheable__sys__shared__text", \
        ".ddr__cacheable__sys__shared__text"
```

Adjust the virtual and physical addresses for each descriptor, if needed. The virtual address of a descriptor is set using `base_address` attribute. The physical address of a descriptor is set using the physical address attribute. Example 3 lists sample code. This example uses a 1:1 translation for `descriptor__m3__cacheable__sys__shared__text_boot`. The virtual address (base address) and the physical address are the same because the interrupt vector (`.intvec`) and boot code (`.text_boot`) are mentioned in this descriptor.

Example 3. Setting the Virtual and Physical Address

```
; Shared data/program for M3
; The descriptor__m3__cacheable__sys__shared__text_boot descriptor
; need to be mapped 1:1 (physical and virtual shared the same
; value), because the boot code and interrupt vector are put in this
; descriptor.
.att_mmu "Shared_mmu_m3", \
         _M3_SHARED_start, _M3_SHARED_end, \
         "descriptor__m3__cacheable__sys__shared__text_boot", \
              attribute: SYSTEM_PROG_MMU_DEF, \
              base_address: _VBAddr, \
              physical_address: _VBAddr, \
         "descriptor__m3__cacheable__sys__shared__text", \
              attribute: SYSTEM_PROG_MMU_DEF, \
              after_physical_address: _M3_SHARED_start, \
         "descriptor__m3__non_cacheable_wt__sys__shared__data", \
              attribute: SHARED_DATA_MMU_DEF, \
              after_physical_address: _M3_SHARED_start, \
         "descriptor__m3__cacheable_wb__sys__shared__data", \
              attribute: SYSTEM_DATA_MMU_DEF, \
              after_physical_address: _M3_SHARED_start
```

Rebuild the application using the new linker control file for MSC8144.

4 Changing the Configuration for Stack and Heap

The static configuration for Stack and Heap is selected by default in the examples used in Section 2 and Section 3. The static configuration means stack and heap use a distinct range of memory.

4.1 Stack and Heap Configuration

Example 4 shows the directive that configures these elements in the `comman.txt` file used with the LCF.

Example 4. Configuring Stack and Heap

```
; Stack/Heap configuration:
; ------------------------
; Dynamic configuration -Heap and Stack overlap in memory. The Stack
; starts at the _StackStart address and grows upwards. The Heap
; starts at the _TopOfHeap address and grows downwards. In a
; dynamic configuration, only the total size of the Stack and Heap
; is known, but individual sizes depend on the application
; characteristics.
; In this configuration, Stack and Heap are in the same memory.
; Static configurations -Heap and Stack are separated. The stack
; starts at _StartStart and ends at _TopOfStack. The memory
```

```
; reserved for the Heap starts at _BottomOfHeap and ends at
; __TopOfHeap.
```

In this configuration, Stack and Heap can be placed in different memories (for example: M2, M3, and DDR). Table 4-1 illustrates the difference between dynamic and static Stack and Heap configuration.

Table 4-1. Dynamic Versus Static Stack/Heap Configuration.

Condition	Stack/Heap Configuration	_STACK_HEAP_CONFIG Value*
_BottomOfHeap = StackStart	Dynamic	–1
_BottomOfHeap ≠ StackStart	Static	1
***Defined by** `.provide __STACK_HEAP_CONFIG, 1`.		

4.2 Dynamic Configuration

Use the following two steps for dynamic configuration:

1. Set –1 as the value of the `__STACK_HEAP_CONFIG`. You can do this in two ways:
   - In the command line option by using the `scc -Xlnk "D__STACK_HEAP_CONFIG=-1"`
   - In LCF by changing the value for the `.provide` directive. (for example, `.provide __STACK_HEAP_CONFIG, -1`)

2. Comment out two directives from the LCF. You can comment out a directive by adding ; at the beginning of the line.
   - In the `command.txt` file:

```
; You need to comment this directive
; if the dynamic configurtion for Stack/Heak is selected.

;.memory _PRIVATE_HEAP_start, _PRIVATE_HEAP_end, "rw"
```

   - In the `local_data.txt` file:

```
; Private heap descriptor
; You need to comment this directive
; if the dynamic configurtion for Stack/Heak is selected.

;.att_mmu "Data_heap_private_mmu", \
      _VIRTUAL_HEAP_start, \
      _VIRTUAL_HEAP_start + _VIRTUAL_HEAP_size, \
      _RESERVED_, \
            size: _VIRTUAL_HEAP_size, \
            region_type: "data", \
            attribute: SYSTEM_DATA_MMU_DEF, \
            base_address: _VIRTUAL_HEAP_start, \
            physical_address: _PRIVATE_HEAP_start
```

4.3 Changing the Size and Location of Stack and Heap in Physical Memory for the Dynamic Configuration

The first data descriptor for data includes the MMU tables and the Stack and Heap. The descriptor size must be a power of 2. The start address in physical and virtual memory space must be multiple of the descriptor size. This limits are part of the MMU requirements.

The following directives in the `common.txt` file configure the size and virtual start address for this descriptor:

```
; Define virtual space for Data Boot (the Data Boot components are: MMU tables and Stack).
; Define the size of MMU_tables
.provide _MMU_TABLES_size,                      0x100
; Define the size of stack.
.provide _StackSize,                            0x7f00
; Define the virtual star address for Data Boot
.provide _VIRTUAL_DATA_BOOT_start,              0x20000000
; The size of Data Boot descriptor must be power of two.
.provide _VIRTUAL_DATA_BOOT_size,    _MMU_TABLES_size + _StackSize
; Verify the MMU constraint for Data Boot descriptor
.assert (_VIRTUAL_DATA_BOOT_size == @mmu_align( _VIRTUAL_DATA_BOOT_size))
```

You can change the size of the stack and heap in two ways:

- Use the command line option. (for example, `scc -Xlnk "-D_StackSize=0xff100"`)
- Change the value for `_StackSize` in the .provide directive. (for example, `.provide _StackSize, 0xff00`)

The following code helps to define the location where the descriptor is placed in physical memory:

```
; Define location for the STACK in physical memory space
; 2 -the STACK will be placed in the M2 memory
; 3 -the STACK will be placed in the M3 memory
; 4 -the STACK will be placed in the DDR memory

.provide __STACK_MEMORY, 3
.provide __M2_BOOT_size,@iif(__STACK_MEMORY == 2,_VIRTUAL_BOOT_size,0)
.provide __M3_BOOT_size,@iif(__STACK_MEMORY == 3,_VIRTUAL_BOOT_size,0)
.provide __DDR_BOOT_size,@iif(__STACK_MEMORY == 4,_VIRTUAL_BOOT_size,0)

; Checking the value where the STACK will be placed in memory

.assert ((__STACK_MEMORY == 2) || (__STACK_MEMORY == 3) || \
(__STACK_MEMORY == 4))
```

Change the location where the descriptor is placed in the physical memory using either of the following ways:

- Use the command line option. (for example, `scc -Xlnk "-D__STACK_MEMORY=4"`)
- Change the value for `__STACK_MEMORY` in the .provide directive (for example, `.provide __STACK_MEMORY, 4`)

4.4 Changing the Size and Location of Stack and Heap in Physical Memory for the Static Configuration

The first data descriptor for data includes the MMU tables and the Stack and Heap. The descriptor size must be a power of 2. The start address in physical and virtual memory space must be multiple of the descriptor size. This limits are part of the MMU requirements.

The following directives in the `common.txt` file configure the size and virtual start address for this descriptor:

```
; Define virtual space for Data Boot (the Data Boot components are: MMU tables and Stack).
; Define the size of MMU_tables
.provide _MMU_TABLES_size,                      0x100
; Define the size of stack.
.provide _StackSize,                            0x7f00
; Define the virtual star address for Data Boot
.provide _VIRTUAL_DATA_BOOT_start,              0x20000000
; The size of Data Boot descriptor must be power of two.
.provide _VIRTUAL_DATA_BOOT_size,     _MMU_TABLES_size + _StackSize
; Verify the MMU constraint for Data Boot descriptor
.assert (_VIRTUAL_DATA_BOOT_size == @mmu_align( _VIRTUAL_DATA_BOOT_size))
```

You can change the size of the stack and heap in two ways:

- Use the command line option. (for example, `scc -Xlnk "-D_StackSize=0xff100"`)
- Change the value for `_StackSize` in the .provide directive. (for example, `.provide _StackSize, 0xff00`)

The following code helps to define the location where the descriptor is placed in physical memory:

```
; Define location for the STACK in physical memory space
; 2 -the STACK will be placed in the M2 memory
; 3 -the STACK will be placed in the M3 memory
; 4 -the STACK will be placed in the DDR memory

.provide __STACK_MEMORY, 3
.provide __M2_BOOT_size,@iif(__STACK_MEMORY == 2,_VIRTUAL_BOOT_size,0)
.provide __M3_BOOT_size,@iif(__STACK_MEMORY == 3,_VIRTUAL_BOOT_size,0)
.provide __DDR_BOOT_size,@iif(__STACK_MEMORY == 4,_VIRTUAL_BOOT_size,0)

; Checking the value where the STACK will be placed in memory

.assert ((__STACK_MEMORY == 2) || (__STACK_MEMORY == 3) || \
(__STACK_MEMORY == 4))
```

Change the location where the descriptor is placed in the physical memory using either of the following ways:

- Use the command line option. (for example, `scc -Xlnk "-D__STACK_MEMORY=4"`)
- Change the value for `__STACK_MEMORY` in the .provide directive (for example, `.provide __STACK_MEMORY, 4`)

The heap puts in its own descriptor for data. The following code includes the directives that configure the size and location where the heap is placed in physical memory:

```
; Define location for the Heap only if the Stack/Heap configuration
; is static.
.provide __HEAP_MEMORY, 3
; 2 -the HEAP will be placed in the M2 memory
; 3 -the HEAP will be placed in the M3 memory
; 4 -the HEAP will be placed in the DDR memory
; Define the size of Heap to 4k if the static configuration
; for the Stack/Heap is used.
.provide _HeapSize, 0x1000
.provide _VIRTUAL_HEAP_size,@iif(__STACK_HEAP_CONFIG==1,_HeapSize,0)
; The size heap descriptor must be power of two.
.assert (_VIRTUAL_HEAP_size==@iif(_VIRTUAL_HEAP_size!=0, \
        @mmu_align(_VIRTUAL_HEAP_size), 0))
.provide __M2_HEAP_size,@iif(__HEAP_MEMORY == 2, _VIRTUAL_HEAP_size, 0)
.provide __M3_HEAP_size,@iif(__HEAP_MEMORY == 3, _VIRTUAL_HEAP_size, 0)
.provide __DDR_HEAP_size,@iif(__HEAP_MEMORY == 4,_VIRTUAL_HEAP_size, 0)
; Checking the value where the STACK will be placed in memory
.assert ((__HEAP_MEMORY==2)||(__HEAP_MEMORY==3)||(__HEAP_MEMORY==4))

.provide _VIRTUAL_HEAP_start, 0x30000000

.provide __BottomOfHeap,@iif(__STACK_HEAP_CONFIG==-1,_StackStart, \
        _VIRTUAL_HEAP_start)
.provide __TopOfHeap, @iif(__STACK_HEAP_CONFIG == -1,_TopOfStack, \
        ((__BottomOfHeap + _VIRTUAL_HEAP_size)-7)&0xFFFFFFF8)

; Verify Stack configuration:
.assert (((__TopOfHeap == _TopOfStack) && \
             (__BottomOfHeap == _StackStart)) || \
        ((__TopOfHeap != _TopOfStack) && \
             (__BottomOfHeap != _StackStart)))
```

Use one of the following to change the size of the heap:

- Command line option (for example, `scc -Xlnk "-D_HeapSize =0x10000" …`)
- Change the value for `_HeapSize` in the `.provide` directive (for example, `.provide _HeapSize, 0x10000`)

Usee one of the following to change the location where the descriptor is placed in the physical memory:

- Command line option (for example, `scc -Xlnk "-D__HEAP_MEMORY=4"`)
- Change the value for `__HEAP_MEMORY` in the `.provide` directive (for example, `.provide __HEAP_MEMORY, 4`)

4.5 Changing the Location Where the .text Section is Defined in Physical Memory

The default location of the text memory implemented in the previous examples is in M2 memory. To move the section, you must first remove it from the M2 memory and then assign to the M3 memory.

4.5.1 Removing .text from M2 Memory Configuration

Remove the .text section from the `.concatenate descriptor__m2__cacheable__sys__shared__text` directive. The following steps describe this procedure:

1. The initial state depends on the LCF being used.
   a) In the `descriptor.txt` file of the LCF for Multi-core application:

```
; Place all the shared data memory sections in
; "descriptor__m2__cacheable__sys__shared__text"
; Shared text in M2
; Descriptor properties:
;     -cacheable
;     -prefetch is enabled
;     -execute access in both user and supervisor mode
;     -burst size 4 ; -system task (shared between tasks)
; Change the properties in .att_mmu directive for
; descriptor__m2__cacheable__sys__shared__text in crtscbmm.cmd
.concatenate "descriptor__m2__cacheable__sys__shared__text", \
     ".m2__cacheable__sys__shared__text", \
     ".text", ".default"
```

   b) In the `sc3400.cmd` file of the LCF for Single core application:

```
; Shared text in M2
; Descriptor properties:
;     -cacheable
;     -prefetch is enabled
;     -execute access in both user and supervisor mode
;     -burst size 4
;     -system task (shared between tasks)
.segment descriptor__m2__cacheable__sys__shared__text, \
     ".m2__cacheable__sys__shared__text", \
     ".text", ".default"
```

2. The file after the change depends on the LCF being used.
   a) In the `descriptor.txt` file of the LCF for Multi-core application:

```
; Place all the shared data memory sections in
; "descriptor__m2__cacheable__sys__shared__text"
; Shared text in M2
; Descriptor properties:
;     -cacheable
;     -prefetch is enabled
;     -execute access in both user and supervisor mode
;     -burst size 4
;     -system task (shared between tasks)
; Change the properties in .att_mmu directive for
; descriptor__m2__cacheable__sys__shared__text in crtscbmm.cmd
.concatenate "descriptor__m2__cacheable__sys__shared__text", \
     ".m2__cacheable__sys__shared__text", \
     ".default"
```

   b) In the `sc3400.cmd` file of the LCF for Single core application:

```
; Shared text in M2
; Descriptor properties:
;     -cacheable
;     -prefetch is enabled
;     -execute access in both user and supervisor mode
;     -burst size 4
```

```
;     -system task (shared between tasks)
.segment descriptor__m2__cacheable__sys__shared__text, \
     ".m2__cacheable__sys__shared__text", \
     ".default"
```

4.5.2 Adding .text to M3 Memory Configuration

Add the .text section to the `.concatenate descriptor__m3__cacheable__sys__shared__text` directive. The following steps describe this procedure:

1. The initial state depends on the LCF being used.
   a) In the `descriptor.txt` file in the LCF for Multi-core application:

```
; Place all the shared data memory sections in
; "descriptor__m3__cacheable__sys__shared__text"
; Shared text in M3
; Descriptor properties:
;     -cacheable
;     -prefetch is enabled
;     -execute access in both user and supervisor mode
;     -burst size 4
;     -system task (shared between tasks)
; Change the properties in .att_mmu directive for
; descriptor__m3__cacheable__sys__shared__text in crtscbmm.cmd
.concatenate "descriptor__m3__cacheable__sys__shared__text", \
     ".m3__cacheable__sys__shared__text"
```

   b) In the `sc3400.cmd` file in the LCF for Single core application:

```
; Shared text in M3
; Descriptor properties:
;     -cacheable
;     -prefetch is enabled
;     -execute access in both user and supervisor mode
;     -burst size 4
;     -system task (shared between tasks)
.segment descriptor__m3__cacheable__sys__shared__text, \
     ".m3__cacheable__sys__shared__text"
```

2. The file after the change depends on the LCF being used.
   a) In the `descriptor.txt` file in the LCF for Multi-core application:

```
; Place all the shared data memory sections in
; "descriptor__m3__cacheable__sys__shared__text"
; Shared text in M3
; Descriptor properties:
;     -cacheable
;     -prefetch is enabled
;     -execute access in both user and supervisor mode
;     -burst size 4
;     -system task (shared between tasks)
; Change the properties in .att_mmu directive for
; descriptor__m3__cacheable__sys__shared__text in crtscbmm.cmd
.concatenate "descriptor__m3__cacheable__sys__shared__text", \
     ".m3__cacheable__sys__shared__text", \
     ".text"
```

   b) In the `sc3400.cmd` file in the LCF for Single core application:

```
; Shared text in M3
; Descriptor properties:
```

```
;     -cacheable
;     -prefetch is enabled
;     -execute access in both user and supervisor mode
;     -burst size 4
;     -system task (shared between tasks)
.segment descriptor__m3__cacheable__sys__shared__text, \
     ".m3__cacheable__sys__shared__text", \
     "text"
```

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

Document Number: AN3572
Rev. 0
11/2007

freescale™
semiconductor