Freescale Semiconductor
Application Note

Document Number: AN4260
Rev. 0, 03/2011

Using Asymmetric DSP Application Projects with CodeWarrior v10.1.8 or Later

by *Devtech Customer Engineering*
Freescale Semiconductor, Inc.
Austin, TX

This application note is a companion document to the application note AN4063 “Configuring an Asymmetric Multicore Application for StarCore DSPs” and AN4155 “Configuring a Mixed Asymmetric Multicore Application for StarCore DSPs”. It contains information specific to managing asymmetric DSP application projects using CodeWarrior for StarCore DSPs v10.1.8 or later.

Customers using CodeWarrior for DSPs v10.1.5 or earlier can use the information contained in application note AN4063.

Contents

1 New Features

CodeWarrior for StarCore DSPs v10.1.8 and later includes several enhancements that require some changes in the asymmetric sample projects that accompany application note AN4063 or AN4155. These are:

- Remote System Explorer (RSE): RSE defines a “container” called a *remote system* that stores connection properties. The remote system settings can be easily shared among multiple launch configurations. This is a key feature for multicore DSP applications where each core is managed by its own launch configuration. For more information on RSE, consult the application note AN4253, “Converting Earlier Versions of CodeWarrior for StarCore DSPs Projects to v10.1.8”.
- Flexible Startup: This allows better support for the asymmetric memory map used by most DSP applications.
- Messages involving linker symbol redefinitions can be disabled.

These features are described in the sections that follow.

1.1 RSE

All launch configurations have been updated to use either a **MSC8156 ADS USB** or a **MSC8156 ISS** remote system configuration.

The first time a CodeWarrior project using v10.1.5 or earlier is imported, make sure to also import the remote system configurations **MSC8156 ADS USB** and **MSC8156 ISS**. These remote system configurations can be used for importing future projects.

Alternatively, if there are already some remote systems configurations defined, they can be used here as well.

1.2 Flexible Startup

Some of the restrictions formerly present in the definition of the asymmetric memory map have been removed.

The linker `.l3k` files that are automatically generated by the CodeWarrior New Project wizard have been modified to support this new capability. New projects can now have a separate descriptor for the `.att_mmu` and the stack sections. See section 3 for more details on the remaining limitations.

Refer to the SmartDSP OS demo project `asymmetric_demo` or sample projects for V10.1.8 delivered with AN4063 for an example of linker files with separate descriptors for the `.att_mmu` and the stack sections.

1.3 Disable Message About Linker Symbol Redefinition

All asymmetric SmartDSP OS projects are linked with the option `-disable-warn-redef-linker-sym`. This prevents generation of the messages:

```
"Redefinition of linker predefined symbol ..."
```

2 Warnings

Attempts to build SmartDSP OS asymmetric projects display the following informative messages for each core:

```
"Could not locate stack section for core c0. Stack not set. It must be set by user."
"Could not locate heap section for core c0. Heap not set. It must be set by user. "
```

This is the expected behavior, as the `heap` and `stack` sections are not explicitly specified in the linker command files. Using the linker files generated by the wizard, stack (actually symbol `_StackStart`) is allocated in section `.oskernel_local_data_bss` and RT .Lib heap is placed into the reserved memory area allocated at the beginning of DDR0 memory.

To remove these messages, add definitions for the stack and heap sections to the `.l3k` files. Consult the next two sections for details.

2.1 Define a Stack Section

To define a real stack section, apply the following modifications to the project:

1. In the file `msc815x_iit.c`, define the symbol `StackStart` as follows:

```
uint8_t  StackStart[ALIGN_SIZE(OS_STACK_SIZE,8)]
__attribute__((section("stack")));

#pragma align StackStart            8
```

In the local_map.l3k file, add the stack section to `descriptor_stack_heap` as follows:

```
descriptor_stack_heap{
  .oskernel_local_data
  .oskernel_local_data_bss
  stack
} > stack_heap_descriptor
```

NOTE

The code snippet above refers to linker files delivered with asymmetric sample projects. If you start from a project created using the New Project wizard, the `stack` section needs to be added at the end of `descriptor_local_data`.

```
descriptor_local_data {
      .oskernel_local_data
      .data
      .oskernel_rom
      .rom
      .exception
      .exception_index
      .ramsp_0
      .init_table
      .rom_init
      .rom_init_tables
      .staticinit
```

```
            LNK_SECTION(att_mmu, "rw", 0x200, 4, ".att_mmu");
            .bsstab
            reserved_crt_tls
            .oskernel_local_data_bss
            .bss
             stack
        } > local_data_descriptor;
```

2.2 Define a Heap Section

To define a real heap section, apply the following modification to `local_map.l3k`:

1. Remove the definitions `__BottomOfHeap` and `__TopOfHeap`.
   The following lines must be removed from the `.l3k` file:

```
#if (USING_USER_KA_STACK == 1)
  __BottomOfHeap  = _KernelAwareness_e;
#else
  __BottomOfHeap  = _VirtLocalDataDDR0_b;
#endif
__TopOfHeap       = __BottomOfHeap +__rtlibHeapSize;
```

2. Modify the definition of `reserved_size` as follows:

```
reserved_size = _KernelAwareness_size;
```

3. Add a `heap` section to `descriptor_local_data_ddr0`:

```
    descriptor_local_data_ddr0 {
      .local_data_ddr0
      .local_data_ddr0_bss
    #if (USING_RTLIB == 1)
      LNK_SECTION(heap, "rw", __rtlibHeapSize, 0x8, "heap");
    #endif
} > local_data_ddr0_descriptor;
```

3 Asymmetric Application Memory Map Limitations

The asymmetric application memory map still has the following restrictions:

3.1 General Purpose Limitations

1. The application entry code and startup code must be allocated in a memory area with one-to-one mapping between virtual and physical address.

This is a requirement and applications that do not follow this scheme will not start.

2. To generate bootable code, the applications entry point should be located at the same physical address on all cores.

This is a requirement and applications that do not follow this scheme will not work when attempting to boot the application over Ethernet, I2C, SPI, or any other interface.

3.2 Guidelines for SmartDSP OS Limitations

1. The section that contains the symbol `g_heap_nocache` must be allocated in the same MMU segment as the startup stack. That means the section `.oskernel_local_data` must be allocated in same MMU segment as `.oskernel_local_bss`.

This limitation comes from implementation of the SmartDSP OS hook function `__target_setting`. If this rule cannot be followed, the SMARTDSP OS function `__target_setting` must be rewritten.

2. The section `.os_shared_data` must be allocated in M3 non-cacheable shared memory. That is because this section contains spinlocks variables used within the OS code.

If this rule cannot be followed, multicore synchronization will not run correctly.

3. Due to the current startup code implementation, `_VBAddr` must be located at the same virtual address for all the cores running the SmartDSP OS application.

If this rule cannot be followed, revise the library module `startup__startup_msc8156_.asm`.

4. SmartDSP OS heaps must have the same size on all cores running SmartDSP OS.

This is an OS requirement.

5. If Kernel Awareness is enabled, the Kernel Awareness buffer size and virtual start address must be identical on all cores.

If this rule is not followed, `smartdsp_os.c` needs to be changed.

3.3 Guidelines for Bare Board Limitations

1. The section `.att_mmu` and startup stack label `_StackStart` must be located in the same MMU segment.

This is a runtime library requirement and applications that do not follow this scheme will not execute the startup code. If this rule cannot be followed, the function `___target_asm_start` must be rewritten.

2. Due to the current startup code implementation, `_VBAddr` must be located at the same virtual address for all the cores running the bare board application

If this rule cannot be followed, revise the library module `startup__startup_msc8156_.asm`.

3.4 Startup code

Figure 1 below depicts a flow diagram of the startup code delivered with the CodeWarrior IDE for SmartDSP OS applications. The gray blocks represent the functions that implement the various stages of this process.

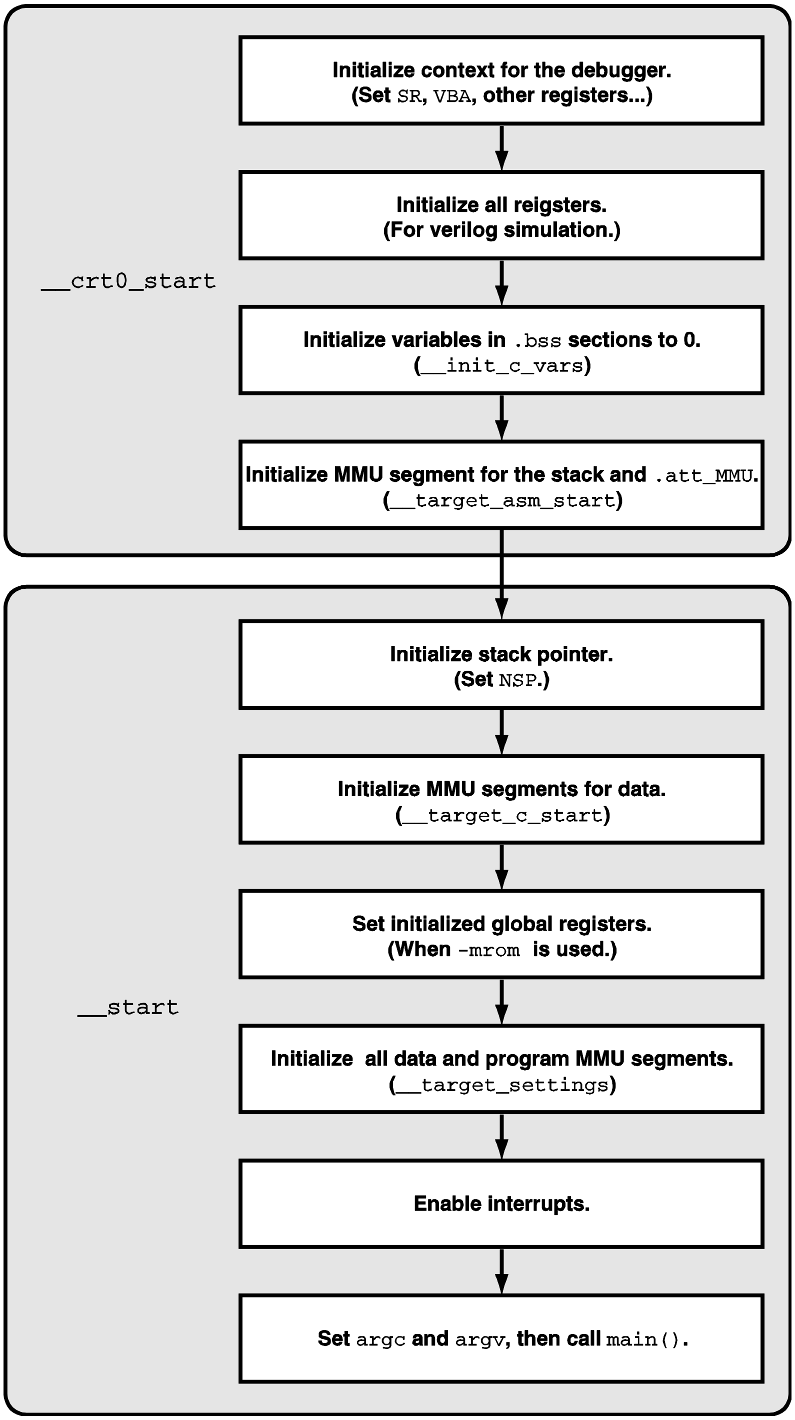

Figure 1. The Flow of Execution of an Application When It Starts.

4 Further Asymmetric Mapping Topics

Asymmetric SmartDSP OS projects use standard SmartDSP OS linker files. In those cases where the application asymmetric memory map must be heavily modified, the `.l3k` files generated for the project may need further modifications.

The project's `.l3k` files need modification if the following is required:

- Different size for VTB buffer on some cores.
- Different size for KA Buffer on some cores.
- Different size for startup and exception stack on some cores.
- Different size for RTlib heap on some cores.

4.1 Different Size for VTB Buffer

In the default SmartDSP OS linker files, the VTB buffer is allocated at the virtual base of private DDR1 memory. If `_VTB_size` is different on some cores, any symmetrical variables allocated in DDR1 will receive inconsistent addresses during linking. In order to avoid this problem, it is better to move VTB buffer at the virtual end of the private DDR1 memory.

This can be done applying following changes in `local_map.l3k`:

- Define a core-dependent symbol, `VTB_size`.

```
#if (USING_VTB == 1)
  _VTB_size = (core_id() == 0) ? 0x4000 :
              (core_id() == 1) ? 0x4000 :
              (core_id() == 2) ? 0x8000 :
              (core_id() == 3) ? 0x8000 :
              (core_id() == 4) ? 0x8000 :
              (core_id() == 5) ? 0x4000 :
              0x0;

  _ENABLE_VTB = 1;
#else
  _VTB_size = 0;
#endif
_VTB_end   = _VirtLocalDataDDR1_e;
_VTB_start = _VTB_end - _VTB_size + 1;
```

- Using the appropriate address translation block, reserve memory at the end of DDR1.

```
address_translation (*) {
...
#if (USING_VTB == 1)
  reserve (SYSTEM_DATA_MMU_DEF): _PhysLocalDataDDR1_e -_VTB_size + 1 ,
_VTB_start, _VTB_size, "rw";
#endif
...
}
```

NOTE

On hardware that lacks DDR1 memory, the VTB buffer can be allocated in another memory block (DDR0 or M3).

4.2 Different Size for KA Buffer

In the default SmartDSP OS linker files, the KA buffer is allocated at the virtual base of the private DDR0 memory. If `_KernelAwareness_size` is different on some cores, any symmetrical variables allocated in DDR0 receive inconsistent address during linking. In order to avoid this problem, it is better to move KA buffer at the virtual end of the private DDR0 memory.

This can be done applying following changes in `local_map.l3k`:

- Define a core-dependent symbol, `_KernelAwareness_size`.

```
#if  (USING_USER_KA_STACK == 1)
  _KernelAwareness_size = (core_id() == 0) ? 0x4000 :
                          (core_id() == 1) ? 0x4000 :
                          (core_id() == 2) ? 0x8000 :
                          (core_id() == 3) ? 0x8000 :
                          (core_id() == 4) ? 0x8000 :
                          (core_id() == 5) ? 0x4000 :
                           0x0;
  _KernelAwareness_e = _VirtLocalDataDDR0_e;
  _KernelAwareness_b = (_KernelAwareness_size == 0) ? _
              KernelAwareness_e :_KernelAwareness_e - _KernelAwareness_size +
1;
#else
  _KernelAwareness_size = 0;
#endif
```

- For the appropriate address translation block, reserve memory at the end of DDR0.

```
address_translation (*) {
...
#if  (USING_USER_KA_STACK == 1)
  reserve (SYSTEM_DATA_MMU_DEF): _PhysLocalDataDDR0_e - reserved_size+1,
                                 _KernelAwareness_b, reserved_size, "rw";
#endif
...
}
```

- Change `smartdsp_os.c` source file to support KA buffers with different size and/or start address. First, define symmetrical variables to hold the virtual start address and the size of the KA buffer. For example:

```
uint32_t  kaBufferSize  = (uint32_t)&KernelAwareness_size;

uint32_t  kaBufferStart = (uint32_t)&KernelAwareness_b;
```

- For the call of function `osLogInitialize` inside of function `osInitialize`, make sure to use the newly defined variables instead of the macros from `os_config.h`.

```
status = osLogInitialize((uint8_t*)kaBufferStart, kaBufferSize,
OS_NUM_OF_CORES);
```

4.3 Different Size for Startup Stack

In the default SmartDSP OS linker/source files, `StackStart` is allocated inside of the section `.oskernel_local_data_bss` as a normal array. If `_StackSize` is different on some cores, this approach cannot be used. The solution is to define a stack section in `descriptor_stack_heap`.

This can be done by applying the following changes:

- In `local_map.l3k`, define a core-dependent symbol, `_StackSize`.

```
_StackSize = (core_id() == 0) ? 0x1e00 :
             (core_id() == 1) ? 0x1e00 :
             (core_id() == 2) ? 0x1800 :
             (core_id() == 3) ? 0x1800 :
             (core_id() == 4) ? 0x1800 :
             (core_id() == 5) ? 0xa00 :
              0x0;
```

- In `local_map.l3k`, place the stack section at the end of `descriptor_stack_heap`.

```
descriptor_stack_heap{
  .oskernel_local_data
  .oskernel_local_data_bss
  LNK_SECTION(stack, "rw", _StackSize, 0x8, "stack");
} > stack_heap_descriptor
```

- In `msc815x_init.c`, remove the definition of the variable, `StackStart`.

  Comment out the following lines in the `msc815x_init.c` source file:

```
//uint8_t  StackStart[ALIGN_SIZE(OS_STACK_SIZE,8)];
//#pragma align StackStart            8
```

NOTE

If the application is not using a stack of identical size on all cores, the use of a dedicated virtual memory area to hold `.att_mmu` and stack sections is recommended. The programmer should also ensure that symmetrical sections are not allocated after `descriptor_stack_heap`.

4.4 Different Size for RTLib Heap

In the default SmartDSP OS linker files, RTLib Heap is allocated right after KA Buffer in the private DDR0 memory. If `__rtlibHeapSize` is different on some cores, the symmetric variables allocated in DDR0 receive inconsistent addresses during linking. To avoid this problem, it is better to move the heap to the end of the application private DDR0 memory.

This can be done by applying following changes:

- In `local_map.l3k`, define a core-dependent symbol, `__rtlibHeapSize`.

```
__rtlibHeapSize =  (core_id() == 0) ? 0x10000 :
                   (core_id() == 1) ? 0x10000 :
                   (core_id() == 2) ? 0x20000 :
                   (core_id() == 3) ? 0x20000 :
                   (core_id() == 4) ? 0x20000 :
                   (core_id() == 5) ? 0x40000 :
                    0x0;
```

- In system*.l3k, place the heap section at the end of the used DDR0 memory. In each core private unit, add the heap section at the end of the used DDR0 memory. For core 0, this is done as follows:

```
unit private (task0_c0) {
  memory {
    private_text_0 ("rx"): org = _VirtPrivate_M2_b;
    private_data_0 ("rw"): AFTER(private_text_0);
    ddr0_private   ("rw"): AFTER(ddr0_SYS0_data);
  }

  sections{
    ...
    ddr0_data {
#if (USING_RTLIB == 1)
      LNK_SECTION(heap, "rw", __rtlibHeapSize, 0x8, "heap");
#endif
    } > ddr0_private;
  }
}

address_translation  (task0_c0) {
  private_text_0 (SYSTEM_PROG_MMU_DEF): PRIVATE_M3;
  private_data_0 (SYSTEM_DATA_MMU_DEF): PRIVATE_M3;
  ddr0_private   (SYSTEM_DATA_MMU_DEF): LOCAL_DDR0;
}
```

The same approach can be applied to the other cores.

4.5 Memory Map

The linker files generated by default with the CodeWarrior New Project wizard place the local memory partitions first in either DDR or M3 memory, as shown in Figure 2.

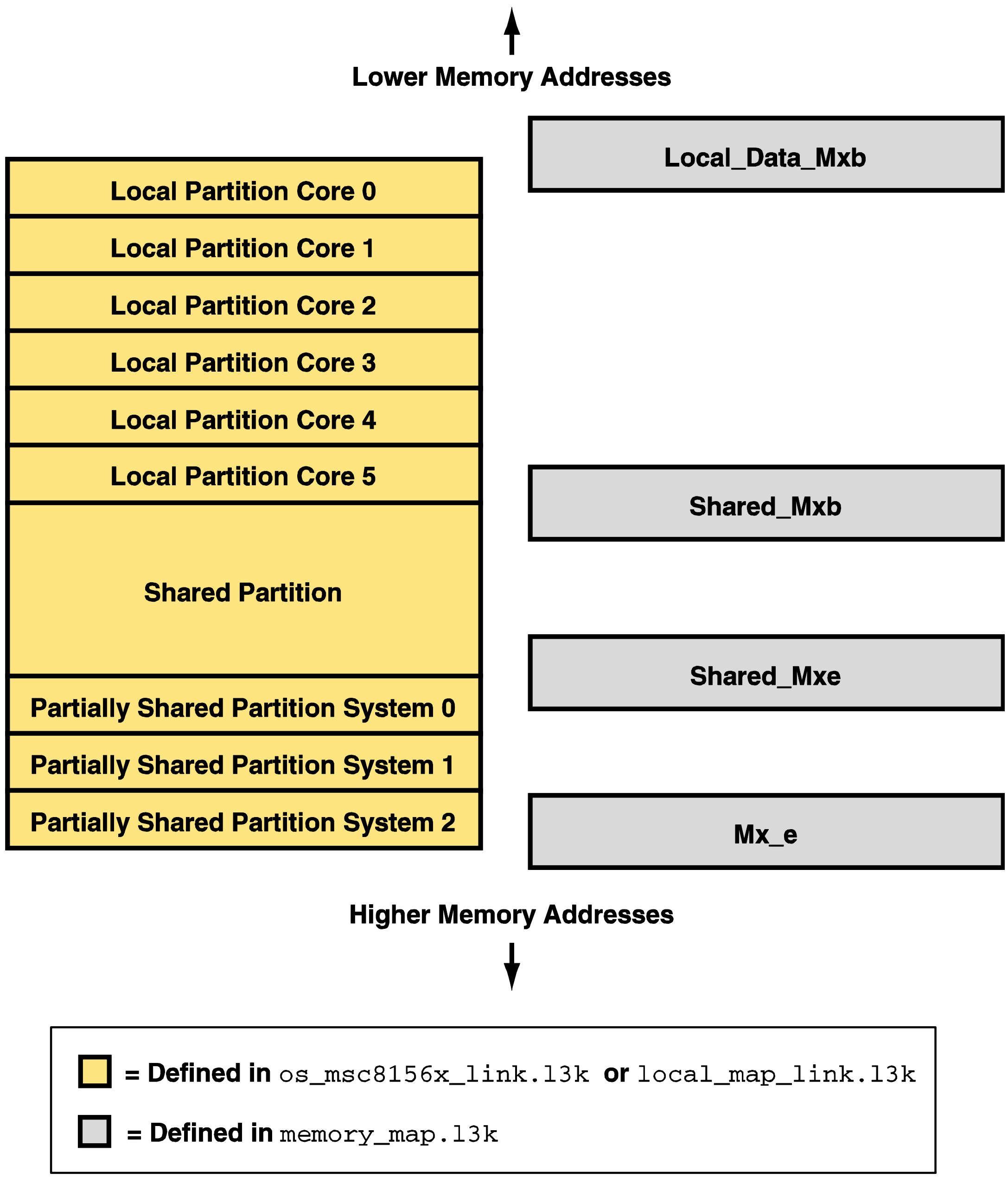

Figure 2. Default Arrangement of Objects in Memory.

To optimize local memory usage for each core, it is good practice to place the shared and partially shared partitions first in memory and then place local partitions in the higher portions of memory (Figure 3).

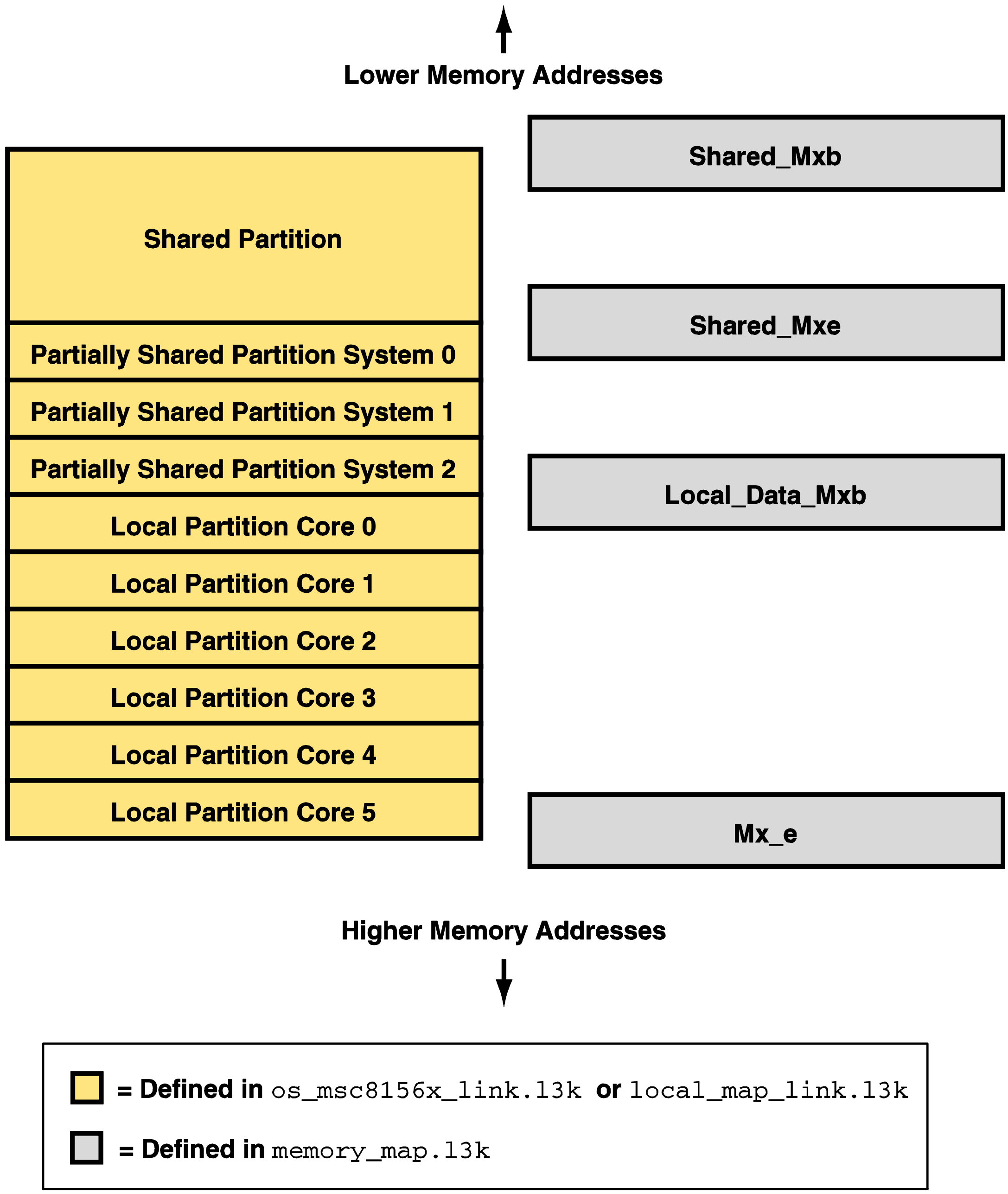

Figure 3. The Memory Map with Memory Organized for Optimized Use.

In order to achieve this, the start and end addresses of the shared, partially shared, and private areas must be adjusted. The sections that follow explain how to accomplish this.

4.5.1 Define the Memory Footprint

First, define the amount of memory required as partially shared memory for each of the subsystem. For example, the following code snippet defines amount of partially shared memory required by each subsystem on DDR0:

```
_DDR0_SHARED_SYS0_SIZE    = 0x20000;
_DDR0_SHARED_SYS1_SIZE    = 0x10000;
_DDR0_SHARED_SYS2_SIZE    = 0x10000;
_TotalSYS_DDR0_Size       = _DDR0_SHARED_SYS0_SIZE + _DDR0_SHARED_SYS1_SIZE +
                            _DDR0_SHARED_SYS2_SIZE;
```

Next, define the amount of local memory required on each core. The following snippet demonstrates how to define the amount of local memory required by each core on DDR0:

```
_LocalDataDDR0_c0_size = 0x2000000;
_LocalDataDDR0_c1_size = 0x2000000;
_LocalDataDDR0_c2_size = 0x1000000;
_LocalDataDDR0_c3_size = 0x1000000;
_LocalDataDDR0_c4_size = 0x1000000;
_LocalDataDDR0_c5_size = 0x3000000;
_TotalLocalDDR0Size    = _LocalDataDDR0_c0_size + _LocalDataDDR0_c1_size +
                         _LocalDataDDR0_c2_size + _LocalDataDDR0_c3_size +
                         _LocalDataDDR0_c4_size + _LocalDataDDR0_c5_size;
```

Now the amount of memory remaining as shared memory can be evaluated. This can be accomplished as follows:

```
_SharedDDR0_size  = _DDR0_e - _DDR0_b -  _TotalLocalDDR0Size -
                    _TotalSYS_DDR0_Size;
```

4.5.2 Define Memory Start and End Addresses

Shared memory is defined at the beginning of each memory block. The example below defines start and end address of shared memory on DDR0:

```
_SharedDDR0_b    = _DDR0_b;
_SharedDDR0_e    = _SharedDDR0_b + _SharedDDR0_size - 1;
```

Partially shared memory areas can be defined as:

```
_DDR0_SHARED_SYS0_Start =  _SharedDDR0_e + 1;
_DDR0_SHARED_SYS0_End = _DDR0_SHARED_SYS0_Start + _DDR0_SHARED_SYS0_SIZE -1;

_DDR0_SHARED_SYS1_Start =  _DDR0_SHARED_SYS0_End + 1;
_DDR0_SHARED_SYS1_End = _DDR0_SHARED_SYS1_Start +  _DDR0_SHARED_SYS1_SIZE - 1;

_DDR0_SHARED_SYS2_Start =  _DDR0_SHARED_SYS1_End + 1;
_DDR0_SHARED_SYS2_End = _DDR0_SHARED_SYS2_Start +  _DDR0_SHARED_SYS2_SIZE - 1;
```

Finally, the virtual and physical address for local data can be defined. In computing the physical start address for all cores, a symbol representing the offset for local data on each core, as compared to the core 0 address, must be computed. This can be done as follows:

```
_PhysLocalDDR0_Offset =
    (core_id() == 0) ? 0x0 :
    (core_id() == 1) ? _LocalDataDDR0_c0_size :
```

```
    (core_id() == 2) ? _LocalDataDDR0_c0_size + _LocalDataDDR0_c1_size :
    (core_id() == 3) ? _LocalDataDDR0_c0_size + _LocalDataDDR0_c1_size +
                       _LocalDataDDR0_c2_size :
    (core_id() == 4) ? _LocalDataDDR0_c0_size + _LocalDataDDR0_c1_size +
                       _LocalDataDDR0_c2_size + _LocalDataDDR0_c3_size:
    (core_id() == 5) ? _LocalDataDDR0_c0_size + _LocalDataDDR0_c1_size +
                        _LocalDataDDR0_c2_size + _LocalDataDDR0_c3_size +
                        _LocalDataDDR0_c4_size:
     0x0;
```

For each core the virtual and physical address for local data can then be computed:

```
_LocalDataDDR0_b = _DDR0_SHARED_SYS2_End + 1;

_VirtLocalDataDDR0_b = _LocalDataDDR0_b;
_VirtLocalDataDDR0_e = (_VirtLocalDataDDR0_b + _LocalDataDDR0_size -1);

_PhysLocalDataDDR0_b = _LocalDataDDR0_b + (_PhysLocalDDR0_Offset);
_PhysLocalDataDDR0_e = _PhysLocalDataDDR0_b + _LocalDataDDR0_size -1;
```

Finally, in order to be able to define the physical memory blocks available for each core in `local_map_link.l3k`, a new symbol is required to store the size of local memory on the core. This symbol is defined as follows:

```
_LocalDataDDR0_size =
    (core_id() == 0) ? _LocalDataDDR0_c0_size:
    (core_id() == 1) ? _LocalDataDDR0_c1_size:
    (core_id() == 2) ? _LocalDataDDR0_c2_size:
    (core_id() == 3) ? _LocalDataDDR0_c3_size:
    (core_id() == 4) ? _LocalDataDDR0_c4_size:
    (core_id() == 5) ? _LocalDataDDR0_c5_size:
     0x0;
```

5 Revision History

Table 1 provides a revision history for this application note.

Table 1. Revision History

Rev. Number	Date	Substantive Change
0	03/8/11	Initial creation

How to Reach Us:

Home Page:
www.freescale.com

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

USA/Europe or Locations Not Listed:
Freescale Semiconductor
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 China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 010 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN4260
Rev. 0
03/2011