OpenMP™ Runtime 2.x

User's Guide

Publication Date: October 1, 2014

Document License

Contents

List of Figures

List of Tables

Listings

Glossary

Application configuration file
The .cfg file is a RTSC file used to configure the application. It typically includes OpenMP configuration as well as configuration for other RTSC modules.

Platform file
The Platform.xdc file that describes memory regions available on the target platform. The regions in the platform file are included in the linker command file that is created as the result of a RTSC build of the platform file.

Acronyms

BIOS Short for SYS/BIOS, a scalable real-time kernel

CCS Code Composer Studio

IPC Inter Process Communication

MCSDK Multi-Core Software Development Kit

PDK Platform Development Kit

QMSS Queue Manager Sub System

RTSC Real Time Software Components

1 Overview

The OpenMP Runtime 2.1 implements support for the OpenMP 3.0 API specification. OpenMP is the de facto industry standard for shared memory parallel programming. It enables incremental parallelization of existing code bases and is portable across shared memory architectures. More information on the OpenMP API (including the API specification) is available at `http://www.openmp.org`.

The two main components of an OpenMP implementation are a compiler and a corresponding runtime. This User's Guide describes the OpenMP Runtime 2.1 for KeyStone and KeyStone II devices. The runtime is based on the GCC OpenMP runtime (libgomp) and takes advantage of hardware features available on KeyStone SoCs to implement OpenMP constructs with low overheads (e.g. Hardware Queues, Semaphores).

1.1 Devices Supported

- KeyStone (TMS320C6678, TMS320C6670, and TMS320C6657)
- KeyStone II (66AK2H)

□ Note: The OpenMP runtime libraries are available in little-endian mode only.

1.2 Modes of Operation

The runtime supports two modes of operation on KeyStone and KeyStone II - bare-metal mode and RTSC mode. The bare-metal mode is the default. Both modes are described in the table below.

Table 1: Comparison of bare-metal and RTSC modes

	Bare-metal mode	**RTSC mode**
When is it applicable?	The application using OpenMP does not require any RTSC components	The application uses other RTSC components (e.g. EDMA LLD)
Configuration	Via __TI_omp_reset and __TI_omp_configure function	Via OpenMP module parameters
Startup	Custom boot routine (rts6000/boot.c)	XDC runtime reset and startup hooks
Dynamic memory management	Heap mapped to '.sysmem' section	IPC/SharedRegion/HeapMemMP
Dependencies on RTSC modules	None	BIOS and IPC

On heterogenous devices such as 66AK2H, the OpenMP runtime supports a third mode: **OpenCL mode**. For details on this mode, refer Section 6.

1.3 Software Architecture

Figure 1 describes the software architecture of the runtime in bare-metal mode. In this mode, the only external dependency for the runtime is the PDK for the device.

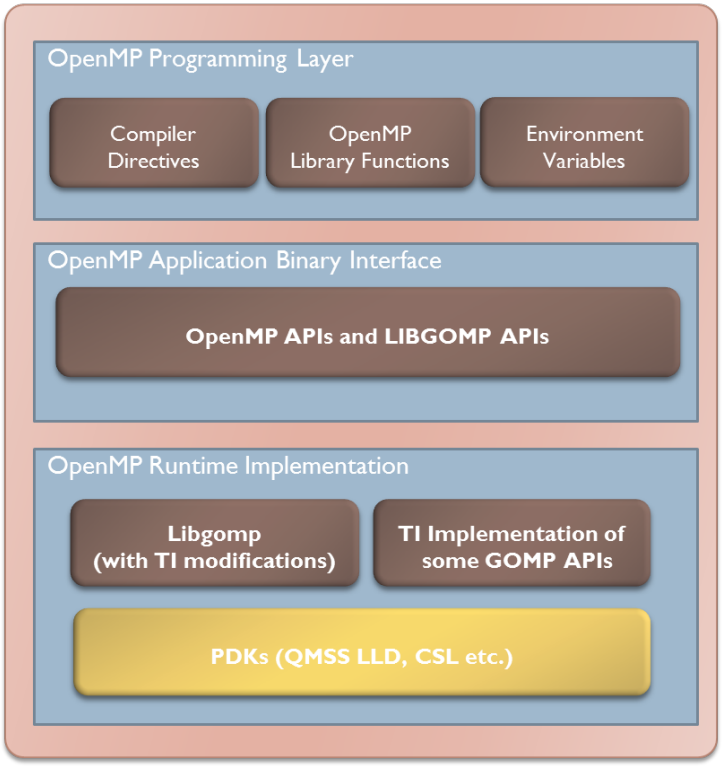

Figure 1: Software architecture (Bare-Metal)

2 Building an OpenMP Application

This section describes the steps for building OpenMP applications for suppoted devices. Section 2.1 discusses additional software components that are required for building OpenMP applications. Sections 2.2 and 2.3 describe building OpenMP applications within CCS and using makefiles.

□ Note: Running OpenMP applications on similators is not supported at this time.

2.1 Build Prerequisites

TMS320C6000 Optimizing Compiler v 7.4.2 or greater is required to build OpenMP applications. In addition, Table 2 lists versions of various software components required with the OpenMP runtime. The OpenMP runtime has been validated against these versions of the components. The components, except for the compiler, are bundled in the Muticore Software Development Kit (MCSDK).

2.2 Building Applications Within CCS

CCS and MCSDK installations are required in order to build applications within CCS. For directions on downloading and installing CCS, see `processors.wiki.ti.com/index.php/Download_CCS`. Various versions of the MCSDK are available at `www.ti.com/tool/bioslinuxmcsdk`. Note that the required software components listed in Table 2, including the OpenMP runtime, must be discovered as RTSC products within CCS. Also, the correct compiler version must be discovered in CCS. The following instructions are based on CCSv5.4.

- Create a new RTSC project (Empty RTSC project under Project templates and examples. Refer Figure 2)

Table 2: RTSC Component Versions

PDK (TMS320C6678)	2.1.3.7	MCSDK 2.1.3.7
PDK (TMS320C6670)	2.1.3.7	MCSDK 2.1.3.7
PDK (TMS320C6657)	2.1.3.7	MCSDK 2.1.3.7
PDK (66AK2H)	3.0.3.15	MCSDK 3.0.3.15
BIOS	6.35.04.50	MCSDK 3.0.x or download from `http://downloads.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/bios/sysbios/index.html`
IPC	3.00.04.29	MCSDK 3.0.x or download from `http://downloads.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/ipc/index.html`
XDC	3.25.02.70	MCSDK 3.0.x or download from `http://downloads.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/rtsc/index.html`

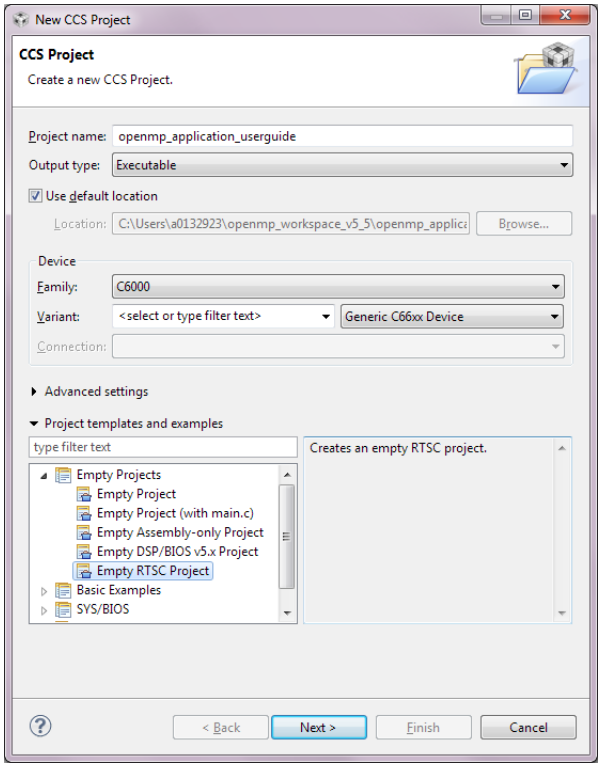

Figure 2: Creating a new RTSC project

  - Family: C6000
  - Variant: TMS320C66XX, TCI6636K2H (or) C66xx Multicore DSP, TMS320C6678, TMS320C6670, TMS320C6657
  - Set Output Format under Advanced Settings to eabi (ELF)

- Add OpenMP runtime to list of RTSC modules. Also add BIOS and IPC if building applications with RTSC components. Refer Section 4.1.
- If building for C6678/C6670/C6657, add the C6678/C6670/C6657 PDK. If building for TCI6636, add the KeyStone2 PDK

- Target: ti.targets.elf.C66
- Platform:
  - C6678: ti.runtime.openmp.platforms.evm6678
  - C6670: ti.runtime.openmp.platforms.evm6670
  - C6657: ti.runtime.openmp.platforms.evm6657
  - TMS320TCI6636: ti.runtime.openmp.platforms.evmTCI6636K2H
- Build Profile: release or debug
- Enable the –openmp compiler option (under Advanced options → Advanced Optimizations)
- Enable the –priority linker option (under C6000 Linker → File Search Path → Search libraries in priority order)
- Add the source files (e.g. packages/examples/hello/omp_hello.c)
- Add a configuration file (packages/examples/hello/omp_config.cfg)
- Build the application
- Connect to the DSP Cores
  - Creating a group with all 8 DSP cores makes it easier to connect to all cores
  - C6678/C6670/C6657: After connecting, run the Global_Default_Setup script on Core 0 to initialize the device (Select Core 0 in the Group, then Scripts → EVM6678L Init Function → Global_Default_Setup)

    ☐ Note: Target-specific GEL files must be loaded in order for any scripts to be displayed in the Scripts menu. See `processors.wiki.ti.com/index.php/Creating_Custom_Target_Configurations` for more information on specifying a startup GEL file.
- Load on all cores

  ☐ Note: Disable Run to symbol in debug configuration for all cores (Software breakpoints used to enable this functionality can cause instability when they are set in code shared across cores). Refer Figure 3.

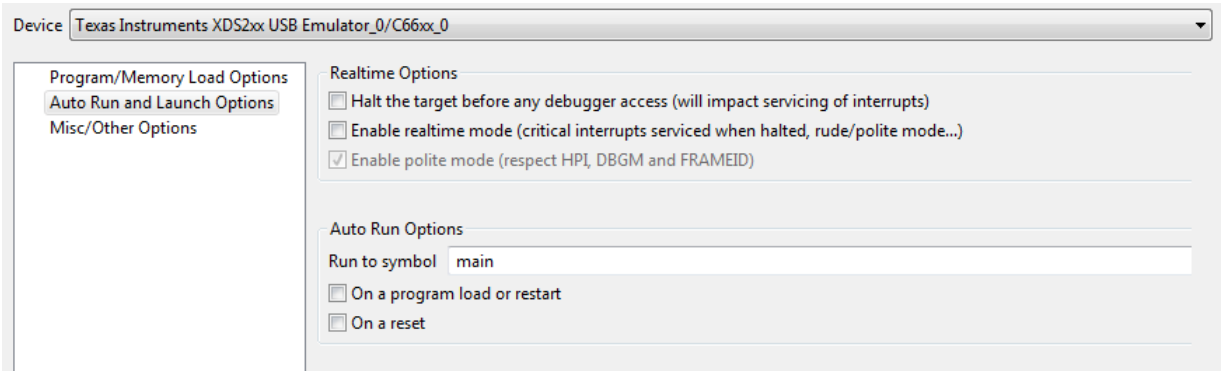

Figure 3: Disable run to symbol ”main”

- Run

2.3 Building Applications Using Makefiles

It is possible to build OpenMP applications outside of CCS using Makefiles. An example is located at packages/examples/hello_with_make. This example uses 2 Makefiles:

Makefile.libomp

- Used for RTSC build. This build takes a configuration file (e.g. packages/examples/hello_with_make /omp_config.cfg) and platform file (e.g. ti.runtime.openmp.platforms.evm6678) as input and generates a compiler options file (omp_config/compiler.opt) and linker command file (omp_config/linker.cmd).
- Set BUILD_TYPE to release or debug to include the appropriate OpenMP runtime libraries
- Set the following environment variables to the locations of corresponding components:
  - OMP_DIR
  - C6678_PDK_DIR/C6670_PDK_DIR/C6657_PDK_DIR/C6636_PDK_DIR based on the device
  - XDC_DIR
  - XDCCGROOT (path to C6000 compiler tools)
  - BIOS_DIR and IPC_DIR (if in BIOS mode)

Makefile

- Used to build the OpenMP application. Includes Makefile.libomp and adds the appropriate dependencies to run the RTSC build before building the application.
- Set OMP_TARGET to one of C6678, C6670, C6657 or C6636
- Set USE_BIOS to 1 if application uses BIOS or other RTSC modules, 0 otherwise

omp_config.cfg

- A sample RTSC mode configuration file. Refer Section 4 for details.

omp_config_bm.cfg

- A sample bare-metal mode configuration file. Refer Section 4 for details.

omp_config_api.c

- A sample implementation of the configuration APIs. Refer Section 4 for details.

3 Heap Management API

3.1 Heaps in Shared Memory (DDR or MSMC)

The DSP runtime provides the following APIs to initialize and manage heaps in shared memory.

3.1.1 Heap Initialization API

The heap initialization functions specified in listing 1 must be called by one of the DSP cores to initialize internal heap data structures before making any memory management calls. Once initialized, the heaps are accessible by all the DSP cores. These APIs are thread safe under the OpenMP and OpenCL programming models on the DSP (Each DSP is running a single thread of execution).

□ Note: If data allocated on the heap is shared across DSP cores, the programmer is responsible for cache consistency of allocated memory across cores . If OpenMP is used to parallelize the program, cache consistency is managed by the OpenMP runtime.

Listing 1: Heap Initialization APIs

```
void  __heap_init_ddr(void *ptr, int size);
void  __heap_init_msmc(void *ptr, int size);
```

3.1.2 Dynamic Memory Management APIs

After the DDR and/or MSMC heap is initialized by one of the DSP cores using the API specified in Section 3.1.1, the following APIs are available from all DSP core for dynamic memory management:

Listing 2: Allocation from DDR

```
void *__malloc_ddr   (size_t size);
void *__calloc_ddr   (size_t num, size_t size);
void *__realloc_ddr  (void *ptr,  size_t size);
void  __free_ddr     (void *ptr);
void *__memalign_ddr (size_t alignment, size_t size);
```

Listing 3: Allocation from MSMC

```
void *__malloc_msmc   (size_t size);
void *__calloc_msmc   (size_t num, size_t size);
void *__realloc_msmc  (void *ptr, size_t size);
void  __free_msmc     (void *ptr);
void *__memalign_msmc (size_t alignment, size_t size);
```

3.2 Heap in Local Memory (L2SRAM)

The DSP runtime provides a simplistic API to initialize a heap in L2 and allocate from it. This heap is local to the core which initialized it.

3.2.1 Heap Initialization API

A heap can be initialized in L2 using the API in listing 5. The storage associated with the heap must be start on a 64bit boundary.

Listing 4: Heap Initialization API

```
void  __heap_init_l2(void *ptr, int size);
```

3.2.2 Dynamic Memory Management APIs

After the L2 heap is initialized by the DSP cores using the __heap_init_l2 call, the only API available is a malloc:

Listing 5: Allocation from L2SRAM

```
void *__malloc_l2 (size_t size); /* Pointer returned is aligned to 64 bit boundary */
```

4 Configuring the Runtime

Section 4.1 describes how to configure the runtime and integrate it into applications that use RTSC components such as BIOS. Section 4.2 does the same for bare-metal applications that do not use any RTSC components.

4.1 RTSC mode Configuration

RTSC mode is enabled by setting usingRtsc to true in the application configuration file. In this mode, the OpenMP runtime uses a boot routine supplied by XDC.

Listing 6: Configuring the OpenMP runtime mode

```
var ompSettings = xdc.useModule("ti.runtime.openmp.Settings");
ompSettings.usingRtsc = true;
```

☐ Note: Unlike OpenMP Runtime 1.x, this runtime does not invoke main() in the context of a BIOS Task nor does it start the BIOS scheduler via BIOS_start().

The OpenMP module provides various parameters that are used to configure the runtime. These parameters are described in the sections below.

4.1.1 Configuring Cores

Listing 7 configures the index of the master core and the number of cores available to the runtime. The set of cores must be contiguous and can be smaller than the number of cores available on the device. E.g. Cores 0-3, Cores 1-6 etc.

Listing 7: Configuring Cores

```
var OpenMP = xdc.useModule('ti.runtime.ompbios.OpenMP');
OpenMP.masterCoreIdx = 0;
OpenMP.numCores      = 8;
```

4.1.2 Configuring Memory Regions

The OpenMP runtime needs to know the memory ranges corresponding to the various memory regions described in the platform file (Section 4.3). It uses this information to set the appropriate caching attributes for the regions.

Listing 8: Configuring Memory Regions

```
// Pull in memory ranges described in Platform.xdc to configure the runtime
var ddr3       = Program.cpu.memoryMap["DDR3"];
var msmc       = Program.cpu.memoryMap["MSMCSRAM"];
var msmcNcVirt = Program.cpu.memoryMap["OMP_MSMC_NC_VIRT"];
var msmcNcPhy  = Program.cpu.memoryMap["OMP_MSMC_NC_PHY"];

// Initialize the runtime with memory range information
OpenMP.msmcBase = msmc.base;
OpenMP.msmcSize = msmc.len;

OpenMP.msmcNoCacheVirtualBase  = msmcNcVirt.base;
OpenMP.msmcNoCacheVirtualSize  = msmcNcVirt.len;
```

```
13
14 OpenMP.msmcNoCachePhysicalBase  = msmcNcPhy.base;
15
16 OpenMP.ddrBase          = ddr3.base;
17 OpenMP.ddrSize          = ddr3.len;
```

4.1.3 Configuring the Heap

In RTSC mode, the OpenMP runtime uses the HeapOMP module to configure and initialize heap memory. This module handles memory allocation requests:

- From BIOS components
- From malloc and memalign

HeapOMP maintains a core local heap (HeapMem) and a shared heap (HeapMemMP). The core local heap is used to satisfy memory allocation requests before the shared heap is intialized and available. The shared heap is implemented via a heap in a Shared Region that is created during IPC startup.

Listing 9: Configuring the Heap

```
1  var HeapOMP = xdc.useModule('ti.runtime.ompbios.HeapOMP');
2
3  // Shared Region 0 must be initialized for IPC
4  var sharedRegionId = 0;
5
6  // Size of the core local heap
7  var localHeapSize  = 0x8000;
8
9  // Size of the heap shared by all the cores
10 var sharedHeapSize = 0x8000000;
11
12 // Initialize a Shared Region & create a heap in the DDR3 memory region
13 var SharedRegion   = xdc.useModule('ti.sdo.ipc.SharedRegion');
14 SharedRegion.setEntryMeta( sharedRegionId,
15                            {   base: ddr3.base,
16                                len:  sharedHeapSize,
17                                ownerProcId: 0,
18                                cacheEnable: true,
19                                createHeap: true,
20                                isValid: true,
21                                name: "DDR3_SR0",
22                            });
23
24 // Configure and setup HeapOMP
25 HeapOMP.configure(sharedRegionId, localHeapSize);
```

4.1.4 Configuring Reset and Startup functions

__TI_omp_reset_rtsc_mode The reset function sets up memory attributes for the various regions specified in the platform file using corresponding parameters (Section 4.1.2) from the OpenMP module. The reset function is invoked via the XDC Reset hook:

Listing 10: Configuring Reset function

```
1 var Reset = xdc.useModule('xdc.runtime.Reset');
2 Reset.fxns.$add('&__TI_omp_reset_rtsc_mode');
```

__TI_omp_initialize_rtsc_mode This function configures the runtime and initializes QMSS hardware queues required by the runtime. It also starts the runtime on worker cores. This function is invoked via the XDC Startup hook mechanism (xdc.runtime.Startup).

Listing 11: Configuring Startup function

```
1 // __TI_omp_initialize_rtsc_mode configures the runtime and calls main
2 var Startup = xdc.useModule('xdc.runtime.Startup');
3 Startup.lastFxns.$add('&__TI_omp_initialize_rtsc_mode');
```

This function is added in the application configuration file to give the programmer flexibility to order Startup.lastFxns appropriately. For example, if the application is initializing QMSS, the call to initialize QMSS must be added to lastFxns before __TI_omp_initialize_rtsc_mode is added.

4.2 Bare-metal Mode Configuration

This is the default mode (ti.runtime.openmp.Settings.useRtsc is false).

The OpenMP runtime requires certain parameters to be initialized before the program starts (i.e. reaches main). The runtime provides callback functions that are invoked during boot (_c_int00). Default implementations of these functions are available in packages/ti/runtime/openmp/src/tomp_config.c. Typically, the programmer will use the __TI_omp_reset and __TI_omp_configure functions in tomp_config.c as a template and modify them if required by their configuration.

4.2.1 __TI_omp_reset

The default implementation is located in packages/ti/runtime/openmp/src/tomp_config.c. The memory addresses used in the default version of __TI_omp_reset correspond to memory regions defined in the platform file (ti.runtime.openmp.platforms.evm6678). If the user modifies the Platform file or provides their own, __TI_omp_reset must be updated to correspond to the new memory ranges. This function is called from _c_int00 (packages/ti/runtime/rts6000/boot.c) before C initialization occurs and is used to:

- Set L1 and L2 cache sizes (Lines 22, 23, 24)
- Make a region of MSMC SRAM non-cacheable using MPAX and disabling caching (Lines 37, 38, 39)
- Enabling caching for MSMC and DDR regions (Lines 42, 43)

Listing 12: Default reset function, __TI_omp_reset

```
1  /**
2   * Default reset routine. Invoked by all cores during boot, before cinit.
3   *
4   * Invoked before C initialization is performed - C init run addresses
5   * can be in regions mapped by MPAX.
6   * Annotated weak, can be overridden by a definition in application source
7   * @see c_int00
8   *
9   * Typically performs the following operations:
10  * - Sets up caches
11  * - Initializes the MPAX registers for mapping memory regions
12  * - Initializes the MAR registers to set attributes for memory regions
13  *
14  * NOTE: The addresses and sizes used here must correspond to those specified
15  * in the Platform or linker command file!
```

```
 */
#pragma WEAK(__TI_omp_reset)
#pragma CODE_SECTION(__TI_omp_reset, ".text:omp:reset")
void __TI_omp_reset(void)
{
    /* Configure caches */
    CACHE_setL1DSize(CACHE_L1_32KCACHE);
    CACHE_setL1PSize(CACHE_L1_32KCACHE);
    CACHE_setL2Size(CACHE_128KCACHE);

    /* OMP runtime requires a portion of MSMCSRAM to be non-cached. Since it is
     * not possible to disable caching in the MSMCSRAM address range, we need a
     * 2 step process:
     * 1. Map a portion of MSMCSRAM into a range that can be marked as
     *    non-cached. This is done using the MPAX register
     * 2. Annotate the mapped section as non-cached using the appropriate
     *    MAR register for that memory range
     * All accesses to MSMCSRAM through the mapped address range will not
     * be cached.
     */
    /* 0x10 => 128K, 0x13 => 1MB */
    __TI_setMPAX(3, MSMCSRAM_NC_START_ADDR,
                    MSMCSRAM_START_ADDR, 0x10 /* 128 KB */);
    __TI_omp_disable_caching(MSMCSRAM_NC_START_ADDR, MSMCSRAM_NC_SIZE);

    /* Annotate MSMCSRAM and DDR as cached + prefetch + write through */
    __TI_omp_enable_caching(MSMCSRAM_START_ADDR, MSMCSRAM_SIZE);
    __TI_omp_enable_caching(DDR_START_ADDR,      DDR_SIZE);

}
```

☐ Note: Any shared memory used by the application must be annotated write-through. The OpenMP runtime relies on write-through for correctness.

4.2.2 __TI_omp_configure

This function is called from _c_int00 (packages/ti/runtime/rts6000/boot.c) before the OpenMP runtime is initialized. The following configuration functions must be called from within __TI_omp_configure:

__TI_omp_config_cores Configure number of cores available to OpenMP runtime and the index of the master core.

In the example above, the runtime is configured to use cores 0, 1, 2 and 3 with 0 being the master core.

__TI_omp_config_clock_freq_in_mhz Specify the operating frequency of the cores in MHz. Core clock frequency information is used by the OpenMP runtime timing functions, omp_get_wtime() and omp_get_wtick().

__TI_omp_config_thread_stack The OpenMP configuration file provided with the examples (packages/examples/hello/omp_config.cfg) places the OpenMP thread stacks in L2SRAM and sizes the stack at 128KB. L2SRAM usage can be reduced by allocating the OpenMP thread stacks from the heap and sizing the stacks in L2SRAM to, say, 8KB. For example, the snippet of code allocates 0x400000 bytes of memory from the heap (using malloc) for each thread.

__TI_omp_config_hw_semaphores Specify the set of hardware semaphores available to the runtime. When the OpenMP runtime is used along with IPC, must ensure that the semaphores used by IPC do not conflict with those of the OpenMP runtime.

__TI_omp_config_hw_queues Most users will not have to change hardware queue settings. This API is mainly for internal use.

Listing 13: Default configuration function, __TI_omp_configure

```
/**
 * Default OpenMP Runtime configuration function.
 *
 * The OpenMP runtime requires the following hardware resources:
 *   - The set of cores (contiguous) that run the OpenMP runtime
 *   - Hardware Semaphores (6)
 *   - QMSS general purpose hardware queues (11)
 *   - QMSS memory region (1)
 *
 * The configuration function specifies the hardware resources that can be
 * used by the runtime.
 *
 * It is annotated weak and can be overridden by a user provided function
 * with the same name.
 * @see __TI_omp_config_thread_stack
 */
#pragma CODE_SECTION(__TI_omp_configure, ".text:omp:configure")
#pragma WEAK(__TI_omp_configure)
void __TI_omp_configure(void)
{
    __TI_omp_config_cores (OMP_MASTER_CORE_IDX, OMP_NUM_CORES);

    __TI_omp_config_hw_semaphores(/*hw_sem_base_idx=*/3);

    __TI_omp_config_clock_freq_in_mhz(CLOCK_RATE);

    /* The OpenMP runtime requires 11 hardware queues and uses the QMSS LLD
     * APIs to initialize these queues.
     * QMSS_PARAM_NOT_SPECIFIED
     *      QMSS LLD allocates queue numbers during call to Qmss_queueOpen
     * Qmss_MemRegion_MEMORY_REGION_NOT_SPECIFIED
     *      QMSS LLD allocates memory region used by Qmss_insertMemoryRegion
     */
#ifdef TI_C6636
    __TI_omp_config_hw_queues (/* init_qmss=*/                    1,
                               /* hw_queue_base_idx=*/          7332,
                               /* first_desc_idx_in_linking_ram=*/ 8000,
                               /* first_memory_region_idx=*/      32);
#else
    __TI_omp_config_hw_queues (/* init_qmss */ 1,
                               QMSS_PARAM_NOT_SPECIFIED,
                               /* first_desc_idx_in_linking_ram=*/0,
                               Qmss_MemRegion_MEMORY_REGION_NOT_SPECIFIED);
#endif

    /* Thread stacks in core local memory */
    __TI_omp_config_thread_stack(0, 0);
    //__TI_omp_config_thread_stack(1, 0x400000);
}
```

4.3 Platform file

A RTSC platform provides information about device memory, peripherals, clock speed and external off-chip memory.

4.3.1 Device Name

The supplied RTSC application configuration files rely on the device name to configure the number of cores used by the OpenMP runtime correctly. Device names are obtained from the platform file and should match the desired target.

4.3.2 CPU Clock Frequency

The OpenMP runtime timing functions, omp_get_wtime() and omp_get_wtick(), require information about the operating frequency of the CPUs (in MHz). In RTSC mode, the CPU clock frequency is obtained from the platform file. Therefore, users must ensure that the CPU clock frequency specified in the platform file is correct.

4.3.3 Memory Regions

Default sizes for level 1 program cache, level 1 data cache and level 2 cache are obtained from the platform file.

Platform files used with the OpenMP runtime must define the following memory regions:

L2SRAM Level 2 SRAM local to each core configured as scratch. At least 64K of L2 must be configured as scratch to hold core-specific runtime variables. If the application stack is placed in L2SRAM, it must be sized taking application stack requirements into account.

OMP_MSMC_NC_VIRT This region is mapped to shared on-chip memory (MSMC SRAM) and caching is disabled using the MAR registers corresponding to the memory region. The OpenMP runtime requires 128KB of MSMC SRAM to be non-cached to store internal data structures.

Since the MAR register corresponding to MSMC address range does not permit caching to be disabled, an MPAX register is used to map a portion of MSMCSRAM (in this case, 128KB) into an unused address space (0xa0000000 on the C6678 EVM) and the MAR register corresponding to 0xa0000000 is used to disable caching.

OMP_MSMC_NC_PHY An MPAX register is used to map the region specified by OMP_MSMC_NC_-VIRT to physical memory specified by OMP_MSMC_NC_PHY.

MSMCSRAM Shared on-chip memory. Can be used for code, data or heap

DDR3 Shared off-chip memory. Used to hold Thread Local Storage (TLS) initialization sections. Can also be used for code, shared data or the heap.

A sample C6678 Platform file is shown in the listing below.

Listing 14: Sample C6678 Platform file

```
metaonly module Platform inherits xdc.platform.IPlatform {

    config ti.platforms.generic.Platform.Instance CPU =
        ti.platforms.generic.Platform.create("CPU", {
            clockRate:      1000,
            catalogName:    "ti.catalog.c6000",
            deviceName:     "TMS320C6678",
            customMemoryMap: [
                ["L2SRAM",
                                {name: "L2SRAM",  base: 0x00800000,
                                len: 0x00060000, access: "RW"}],
                ["OMP_MSMC_NC_VIRT",
                                {name: "OMP_MSMC_NC_VIRT", base: 0xA0000000,
                                len: 0x00020000, access: "RW"}],
                ["OMP_MSMC_NC_PHY",
                               {name: "OMP_MSMC_NC_PHY", base: 0x0C000000,
                                len: 0x00020000, access: "RW"}],
```

```
                ["MSMCSRAM",
                                {name: "MSMCSRAM", base: 0x0C020000,
                                len: 0x003E0000, access: "RWX"}],
                ["DDR3",
                                {name: "DDR3", base: 0x80000000,
                                len: 0x20000000, access: "RWX"}],
            ],
            l2Mode:"128k",
            l1PMode:"32k",
            l1DMode:"32k",
    });

instance :

    override config string codeMemory  = "MSMCSRAM";
    override config string dataMemory  = "DDR3";
    override config string stackMemory = "L2SRAM";
```

The OpenMP runtime supplies platform files for the various devices it supports in the ti.runtime.openmp.platforms directory.

☐ Note: The addresses in the platform file must be consistent with those specified in the configuration API in Section 4.2. The linker command file template in the OpenMP runtime (ti/runtime/openmp/linkcmd.xdt) contains references to memory regions defined in the platform file with the following names: L2SRAM, DDR, OMP_MSMC_NC_VIRT and OMP_MSMC_NC_PHY. Consequently any platform file used with the OpenMP runtime must contain these regions. Alternatively, the linkcmd.xdt file can be modified to refer to the changed names.

5 Integrating Applications Using QMSS

OpenMP runtime uses Queue Manager Subsystem (QMSS) and initializes it by default. If applications use QMSS outside of OpenMP (e.g. to run the networking stack on one of the DSP cores using the NDK), the initialization must be performed by the application because the runtime is not aware of QMSS memory regions used by the application.

- Set openmp.Settings.runtimeInitializesQmss to false
- Configure QMSS related parameters in the openmp module:
  - qmssMemRegionIndex
  - qmssFirstDescIdxInLinkingRam

  Refer 14.1 for details on these parameters.

☐ Note: The application function that performs QMSS initialization (calls Qmss_init) must be added to Startup.lastFxns before __TI_omp_initialize_rtsc_mode.

On KeyStone devices (TMS3206657, TMS320C6670 and TMS3206678), memory regions must be added in increasing order of memory addresses. The openmp module configuration parameters qmssMemRegionIndex and qmssFirstDescIdxInLinkingRam can be used to enforce this restriction. Refer QMSS LLD API for details.

6 OpenCL Mode

In addition to the bare-metal and RTSC modes of operation, the OpenMP runtime also supports an OpenCL mode. OpenCL is an industry standard for programming heterogeneous computing systems. The OpenMP

runtime's OpenCL mode allows OpenMP C programs to be dispatched using OpenCL APIs, and is only available on 66AK2H. This execution mode is a Texas Instruments-specific extension to OpenCL. In this mode, the OpenMP runtime is embedded in the OpenCL monitor that runs on the DSP cores. Texas Instrument's OpenCL implementation, OpenCL v1.0, currently implements version 1.1 of the OpenCL specification. More information about OpenCL is available at `khronos.org/opencl`. For details on Texas Instrument's OpenCL v1.0, see the OpenCL User's Guide in the OpenCL package.

6.1 Dispatching OpenMP with OpenCL

An OpenCL program usually consists of a *host program* that executes on the host, and *kernels* that execute on OpenCL devices. In the OpenCL mode, a kernel acts as a wrapper that invokes another function which contains *OpenMP regions*. These three components — the OpenCL host program, the OpenCL kernel, and the OpenMP region — form the main parts of an OpenCL program capable of dispatching OpenMP regions.

The example in Figure 4 provides an overview of dispatching OpenMP regions with OpenCL APIs. The OpenCL runtime manages the execution of the OpenCL host program on the host, which is a quad-core ARM Cortex-A15 in this example. The embedded OpenMP runtime along with the OpenCL runtime manages the execution of the OpenCL kernel and OpenMP region on the device. In this example, the device is a DSP with 8 C66x cores. More example programs are included in the OpenCL package.

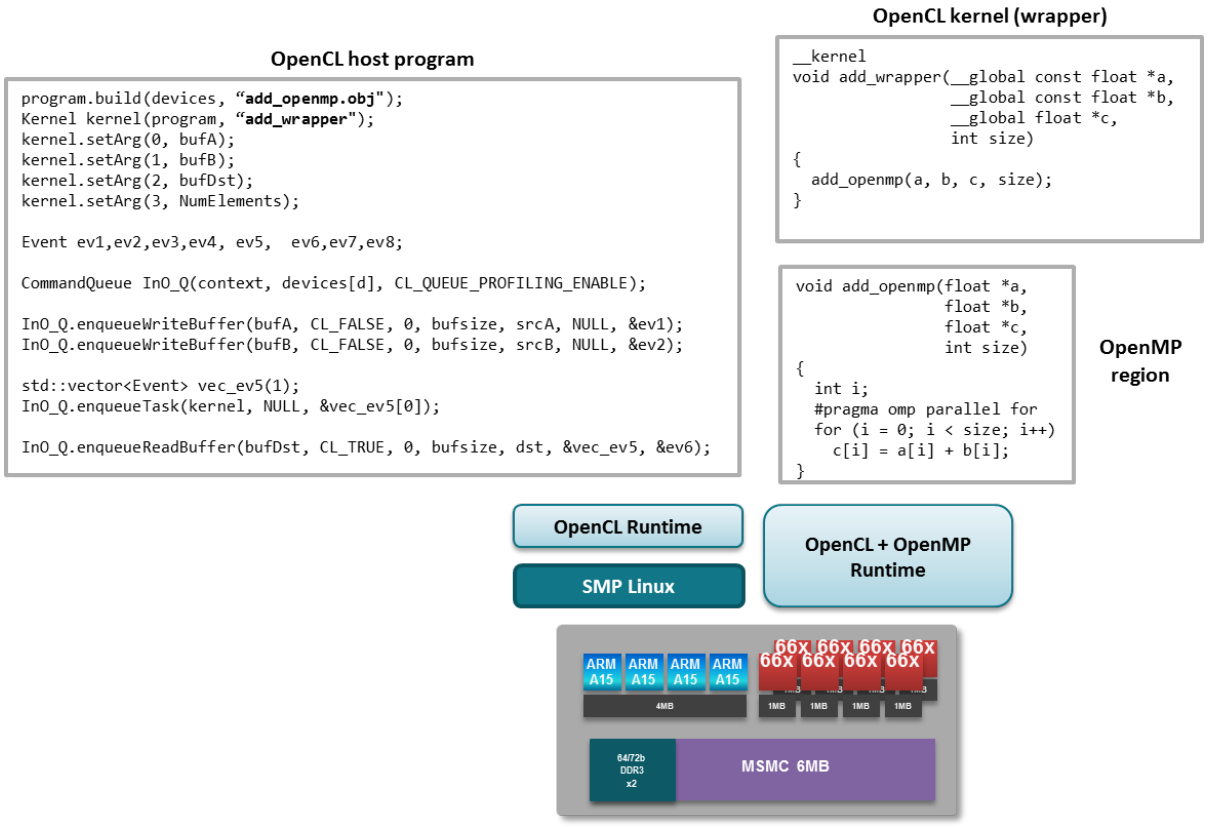

Figure 4: Dispatching OpenMP regions with OpenCL APIs

7 OpenMP 3.0 Implementation-Defined Behaviors

The OpenMP 3.0 specification identifies some features as implementation defined. The following documents the behaviour of the runtime with respect to implementation-defined features.

- Nested Parallelism: Only one level of parallelism is supported. Nested parallel regions will be executed by teams comprising only one thread.
- Task Scheduling Points in Untied Tasks: Only tied tasks are currently supported and untied tasks are treated as tied tasks. Therefore, task scheduling points in untied task regions occur at the same points as in tied task regions.
- Memory Model: Memory accesses by multiple threads to the same variable without synchronization may not be atomic with respect to each other.
- Dynamic adjustment of threads: Dynamic adjustment of the number of threads is not supported. When a thread encounters a parallel construct, the number of threads delivered by this implementation is determined according to Algorithm 2.1 in the OpenMP 3.0 Specification. When more threads are requested than are available, the available threads are supplied.
- Loop directive: The effect of `schedule(runtime)` clause when the *run-sched-var* ICV is set to auto is static with no chunk size.
- Constructs
  - `sections` construct: The structured blocks in a `sections` construct are assigned to the members of the team executing the sections region, such that the threads execute an approximately equal number of sections.
  - `single` construct: The first thread that encounters a `single` construct executes its structured block.
- Runtime library routines
  - omp_set_num_theads: If the argument is not a positive integer, then the call is ignored.
  - omp_set_schedule: Only valid OpenMP schedule kinds are accepted.
  - omp_set_max_active_levels: When called from within an explicit parallel region, the binding thread set is all threads. However, since nested parallelism is not supported, this implementation has little use for this function. If the argument is a negative integer or a positive integer greater than one, then the call is ignored.
  - omp_get_max_active_levels: This routine may be called anywhere in the program. It always returns the value of the gomp_max_active_levels_var internal control variable.

8 Migration Guide

8.1 Key Differences between OpenMP Runtime 1.x and 2.x

In bare-metal mode, OpenMP runtime 2.x:

- Does not require BIOS/IPC components.
- Does not require the RTSC build tool (xdc) to build OpenMP applications.

- Is configured via API functions __TI_omp_reset and __TI_omp_configure instead of parameters in a RTSC OpenMP module.

In RTSC mode, OpenMP runtime 2.x is configured by setting parameters in the OpenMP RTSC module. Refer to the sample application configuration file for details.

8.2 Porting an OpenMP Runtime 1.x Application to 2.x

1. Switch to a platform file (Platform.xdc) supplied with 2.x
2. Use the provided example application configuration file as a starting point to port OpenMP runtime 1.x configuration to corresponding 2.x parameters.
3. If you are modifying the memory ranges specified in Platform.xdc:

   RTSC mode: update OpenMP module parameters in the application configuration file to correspond to the new Platform file.

   Bare-metal mode: Write your own __TI_omp_reset function by using the one supplied in tomp_config.c as a template.
4. OpenMP Runtime 1.x invokes the application main function as a BIOS Task and calls BIOS_start before main. OpenMP runtime 2.x does not call main in the context of a BIOS Task. If the application uses Tasks, it must call BIOS_start().

9 Resource Usage

Table 3 lists resource usage of the runtime with the default configuration.

Table 3: Resource Usage

Resource	Description	OMP runtime usage on C6678
L1P	Level 1 Program cache	Configured as cache (32KB)
L1D	Level 1 Data cache	Configured as cache (32KB)
L2 (cache)	Level 2 Cache	128K configured as cache in __TI_omp_reset
L2 (memory)	Level 2 Scratch	Default configuration uses 131KB as scratch in bare-metal mode. Out of this, 128KB is stack (configured via program.stack). In RTSC mode, 171KB is used as scratch. 32KB is used for the core local heap(configured via localHeapSize) and 128KB is stack.
MSMCSRAM Cached	On-chip shared memory	Not used by the runtime. Application can use for code/data. Must be marked write-through (Refer __TI_omp_reset)
MSCMSRAM Non-cached	On-chip shared memory	128KB must be non-cached using a combination of MPAX and MAR registers. (Refer __TI_omp_reset)
DDR	Off-chip shared memory	Application heap is mapped to DDR in omp_config.cfg. The size of the heap if controlled by program.heap and the section it is mapped to depends on the mapping of section .sysmem. Any DDR regions annotated cached by the application must also be annotated write-through (see __TI_omp_reset)
Hardware Semaphores	Semaphore module	6 semaphores, starting at hwSemBaseIdx
Hardware Queues	QMSS General purpose queues	11 hardware queues starting at qmssHwQueueBaseIdx
Memory Regions	QMSS Memory Regions	1 memory region, specified by qmssMemRegionIdx

10 Reducing Memory Footprint in L2SRAM

This section details various techniques for reducing L2SRAM usage. The examples shown here make the assumption that the application requires 64K of stack space per core.

10.1 Stacks in MSMCSRAM

The default configuration places the stack in each core's L2SRAM and sizes them to 128K. A technique for reducing L2SRAM usage is to place the thread stacks in MSMCSRAM. For example, with 8 cores and 128KB of stack per core, the trade-off is to use 1MB of MSMCSRAM for stacks and free up 128KB on each core's local L2SRAM. Steps:

1. Update the application configuration file to create the heap in MSMCSRAM

Bare-Metal mode

Update application configuration file to place the heap in MSMCSRAM

```
program.sectMap[".sysmem"] = new Program.SectionSpec();
program.sectMap[".sysmem"].loadSegment      = "MSMCSRAM";
```

RTSC mode

Setup shared region 0 on MSCMSRAM instead of DDR3. This will create the heap in MSMCSRAM.

```
// 64K per core for stack + 64K for other mallocs
var sharedHeapSize = 0x90000;

var msmcmem = Program.cpu.memoryMap["MSMCSRAM"];

// Configure a Shared Region with a heap in MSMC memory region
var SharedRegion    = xdc.useModule('ti.sdo.ipc.SharedRegion');
SharedRegion.setEntryMeta( sharedRegionId,
                           {   base: msmcmem.base,
                               len:  sharedHeapSize,
                               ownerProcId: 0,
                               cacheEnable: true,
                               createHeap: true,
                               isValid: true,
                               name: "MSMC_SR0",
                           });
```

2. Update omp_config.c, __TI_omp_configure to allocate thread stacks from the heap Replace

```
__TI_omp_config_thread_stack(0, 0);
```

With

```
__TI_omp_config_thread_stack(1, 0x10000);
```

3. Reduce program.stack to 4K in the configuration file. This stack is only used by OpenMP runtime during initialization. The program's main thread starts execution in the stack configured in steps 1, 2.

```
program.stack = 0x1000;
```

☐ Note: Placing thread stacks in DDR has potential to significantly degrade performance due to register spills to slow DDR stack within frequently executed loops in the application.

11 Building the Runtime

The OpenMP runtime ships with all the sources required to build the runtime. Use the Makefile provided in the utils directory after setting the following environment variables: XDCCGROOT, XDC_DIR, C6670_PDK_DIR, C6657_PDK_DIR, C6678_PDK_DIR, C6636_PDK_DIR. XDCCGROOT points to an installation of the C6000 compiler tools, version 7.4.6.


```
make –f utils/Makefile
```

The build creates libraries in the following directories:

1. ti/runtime/openmp/lib
2. ti/runtime/argsmain/lib
3. ti/runtime/argsmain/lib
4. ti/runtime/device/lib
5. ti/runtime/ompbios/lib
6. ti/runtime/rts6000/lib
7. ti/runtime/cio/lib

It will also update the platform files located in ti/runtime/openmp/platforms.

12 Defect Reporting

All OpenMP Runtime 2.x defects should be reported via the TI Compiler forum at the following URL: `http://e2e.ti.com/support/development_tools/compiler/default.aspx`

13 Version History

Version	Description	Release Date
2.0.0.1	Initial version, bare-metal, C6678 only	05/07/2013
2.0.0.3	Added RTSC mode - support for including BIOS and other RTSC modules along with OpenMP runtime	06/13/2013
2.1.0	Added 66AK2H support. Requires OpenEM 1.3.0.2	06/03/2013
2.1.2	Added OpenCL integration - using OpenCL APIs to dispatch OpenMP worksharing programs. Requires OpenEM 1.5.0.1	07/15/2013
2.1.6	Removed OpenEM requirement. Added dispatching OpenMP tasking programs with OpenCL	10/30/2013
2.1.7	Added support for C6670 and C6657. Requires MCSDK 3.0.3.15 for 66AK2H and MCSDK 2.1.3.7 (Built after 11/13/2013) for C6678/C6670/C6657	11/19/2013
2.1.8	Beta 1	12/10/2013
2.1.9	Beta 2. In RTSC mode, the OpenMP runtime did not honor L1/L2 cache sizes specified in the platform file (Platform.xdc). Fixed.	12/16/2013
2.1.10	GA. Data Page (DP) register not set for worker cores when dispatching OpenMP programs with near data from OpenCL. Fixed	01/17/2014
2.1.11	Enabled Debian package creation. No changes to OpenMP runtime functionality.	01/28/2014
2.1.12	Added a section on OpenCL mode to the User's Guide. No changes to OpenMP runtime functionality.	02/14/2014
2.1.13	Fixed issue with runtime timing functions in OpenCL mode.	03/20/2014
2.1.14	Updated internal heap management API for use with the OpenMP accelerator model.	04/03/2014
2.1.15	Added a heap management API for use by OpenMP applications dispatched from OpenCL. These APIs are also available to standalone OpenMP applications.	06/17/2014
2.1.16	Integrated TI's Usage and Load Monitor library (ULMLib). ULMLib collects specific OpenMP runtime events into DSP trace buffers.	09/08/2014

14 Appendix

14.1 RTSC Mode Configuration Parameters

The set of parameters available for configuring the OpenMP runtime in RTSC mode are specified in the OpenMP module.

Listing 15: OpenMP.xdc

```
/*
 * Copyright (c) 2013, Texas Instruments Incorporated
 * http://www.ti.com
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * *  Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * *  Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 * *  Neither the name of Texas Instruments Incorporated nor the names of
 *    its contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
 * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
 * EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
/*
 *  ======== OpenMP.xdc ========
 */

import xdc.runtime.Types;


/*!
 *  ======== OpenMP ========
 *  OpenMP RTSC Configuration Module
 *
 *  This module must be used (via xdc.useModule) by all OpenMP RTSC
 *  applications. This module is used to setup a configuration for the
 *  runtime. The configuration is applied by the functions
 *  __TI_omp_reset_bios_mode and __TI_omp_start_bios_mode before main
 *  is called.
 *
 */
module OpenMP
```

```
{
    /*! Number of participating cores.
     *
     *  Can be fewer than number of cores available on device.
     */
    config Int32    numCores            = 8;

    /*! Index of the core running the main thread */
    config Int32    masterCoreIdx       = 0;

    /*! Core clock rate as defined in Platform.xdc */
    config Int32    clockFreq           = 0;

    /*! Base address of MSMC region as defined in Platform.xdc */
    config UInt32   msmcBase            = 0;

    /*! Size of MSMC region as defined in Platform.xdc */
    config UInt32   msmcSize            = 0;

    /*! Base address of OMP_MSMC_NC_VIRT region as defined in Platform.xdc */
    config UInt32   msmcNoCacheVirtualBase     = 0;

    /*! Size of OMP_MSMC_NC_VIRT region as defined in Platform.xdc */
    config UInt32   msmcNoCacheVirtualSize     = 0;

    /*! Base address of OMP_MSMC_NC_PHY region as defined in Platform.xdc */
    config UInt32   msmcNoCachePhysicalBase     = 0;

    /*! Base address of DDR region as defined in Platform.xdc */
    config UInt32   ddrBase             = 0;

    /*! Size of DDR region as defined in Platform.xdc */
    config UInt32   ddrSize             = 0;


    /*! MPAX register used for mapping OMP_MSMC_NC_VIRT addresses to
     *  corresponding addresses in the OMP_MSMC_NC_PHY region
     */
    config UInt32   mpaxForMsmcMapping  = 3;


    /*! Base index of Hardware Semaphore.
     *
     *  The runtime uses hwSemCount semaphores starting at base index.
     *  E.g. if hwSemBaseIdx is 3 and hwSemCount is 5, semaphores 3,4,5,6,7
     *  are used.
     */
    config Int32    hwSemBaseIdx        = 3;

    /*! Number of Hardware Semaphores
     *
     *  The user must not modify this field. The runtime uses 5 semaphores.
     */
    config Int32    hwSemCount          = 6;


    /*!
     *  @_nodoc
     *  Set via openmp.Settings.runtimeInitializesQmss. Do not set
     *  directly.
     */
    config bool     qmssInit = true;
```

```
114
115     /*! Index of the QMSS memory region
116      *
117      *  Defaults to -1 (Qmss_MemRegion_MEMORY_REGION_NOT_SPECIFIED).
118      *  If the application is initializing QMSS and setting up memory regions,
119      *  it can use qmssMemRegionIndex to indicate the memory region to be used
120      */
121     config Int32    qmssMemRegionIndex = -1;
122
123     /*! The first descriptor index used in linking RAM.
124      *
125      *  If the application is initializing QMSS, it typically provides an
126      *  index.
127      */
128     config UInt32   qmssFirstDescIdxInLinkingRam = 0;
129
130     /*! Base index of QMSS general purpose hardware queue.
131      *
132      *  The runtime uses 11 queues starting at this index. When the runtime
133      *  intializes QMSS, the default of -1 allows the QMSS LLD to pick the
134      *  queues allocated.
135      */
136     config Int32    qmssHwQueueBaseIdx  = -1;
137
138
139     /*! If true, stack for each OpenMP thread is malloc'ed from the heap
140      *
141      *  By default, the stacks are placed in L2SRAM.
142      */
143     config bool     allocateStackFromHeap    = false;
144
145     /*! If thread stack is allocated from heap, size of the stack on each
146      *  core. Defaults to 64K per-core
147      */
148     config Int32    allocateStackFromHeapSize = 0x10000;
149
150
151     /*!
152      *  @_nodoc
153      * Set number of processors for IPC to use
154      */
155     metaonly Void configureIpc(UInt16 masterCoreIdx, UInt16 numCores);
156
157     /*!
158      *  @_nodoc
159      *  Used by HeapOMP to switch to using Shared Region heap for allocation
160      */
161     Bool useSharedHeap();
162
163 internal:
164
165     struct Module_State {
166         Bool                useSharedHeap;
167     };
168 }
```

14.2 Bare-metal Configuration Functions

The default reset and configuration functions provided with the OpenMP runtime. The addresses for the memory regions used in these functions coresond to the memory regions defined in Platform.xdc.

Listing 16: tomp_config.c

```
/*
 * Copyright (c) 2013, Texas Instruments Incorporated - http://www.ti.com/
 *   All rights reserved.
 *
 *  Redistribution and use in source and binary forms, with or without
 *  modification, are permitted provided that the following conditions are met:
 *      * Redistributions of source code must retain the above copyright
 *        notice, this list of conditions and the following disclaimer.
 *      * Redistributions in binary form must reproduce the above copyright
 *        notice, this list of conditions and the following disclaimer in the
 *        documentation and/or other materials provided with the distribution.
 *      * Neither the name of Texas Instruments Incorporated nor the
 *        names of its contributors may be used to endorse or promote products
 *        derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */
/**
 * \file tomp_config.c
 *
 * \brief Implements default versions of configuration functions
 *
 * Functions implemented:
 *    __TI_omp_reset
 *    __TI_omp_configure
 */

#include "omp.h"
#include <ti/csl/csl_cache.h>
#include <ti/csl/csl_cacheAux.h>
#include <ti/csl/csl_msmc.h>
#include <ti/csl/csl_msmcAux.h>
#include <ti/drv/qmss/qmss_qm.h>

extern cregister volatile unsigned int DNUM;

/** \defgroup omp_config OMP Runtime Configuration Hooks */
/* @{ */

#define MSMCSRAM_START_ADDR      (0x0C000000)
#define MSMCSRAM_NC_SIZE         (0x00020000)     /* 128 KB */

#if defined (TI_C6678)
#define MSMCSRAM_SIZE            (0x00400000)     /* 4 MB */
#define MSMCSRAM_NC_START_ADDR   (0xA0000000)
#define DDR_START_ADDR           (0x80000000)
#define DDR_SIZE                 (0x20000000)     /* 512 MB */
#define OMP_NUM_CORES            (8)
#define CLOCK_RATE               (1000)           /* 1000 Mhz */
#elif defined (TI_C6670)
```

```
#define MSMCSRAM_SIZE                   (0x00200000)    /* 2 MB */
#define MSMCSRAM_NC_START_ADDR          (0xA0000000)
#define DDR_START_ADDR                  (0x80000000)
#define DDR_SIZE                        (0x20000000)    /* 512 MB */
#define OMP_NUM_CORES                   (4)
#define CLOCK_RATE                      (983)           /* 983 Mhz */
#elif defined (TI_C6657)
#define MSMCSRAM_SIZE                   (0x00100000)    /* 1 MB */
#define MSMCSRAM_NC_START_ADDR          (0xA0000000)
#define DDR_START_ADDR                  (0x80000000)
#define DDR_SIZE                        (0x20000000)    /* 512 MB */
#define OMP_NUM_CORES                   (2)
#define CLOCK_RATE                      (1000)          /* 1000 Mhz */
#elif defined (TI_C6636)
#define MSMCSRAM_SIZE                   (0x00600000)    /* 6 MB */
#define MSMCSRAM_NC_START_ADDR          (0xE0000000)
#define DDR_START_ADDR                  (0xA0000000)
#define DDR_SIZE                        (0x20000000)    /* 512 MB */
#define OMP_NUM_CORES                   (8)
#define CLOCK_RATE                      (1000)          /* 1000 Mhz */
#else
#error "Device not supported"
#endif

#define OMP_MASTER_CORE_IDX (0)

/**
 * Default reset routine. Invoked by all cores during boot, before cinit.
 *
 * Invoked before C initialization is performed - C init run addresses
 * can be in regions mapped by MPAX.
 * Annotated weak, can be overridden by a definition in application source
 * @see c_int00
 *
 * Typically performs the following operations:
 * - Sets up caches
 * - Initializes the MPAX registers for mapping memory regions
 * - Initializes the MAR registers to set attributes for memory regions
 *
 * NOTE: The addresses and sizes used here must correspond to those specified
 * in the Platform or linker command file!
 */
#pragma WEAK(__TI_omp_reset)
#pragma CODE_SECTION(__TI_omp_reset, ".text:omp:reset")
void __TI_omp_reset(void)
{
    /* Configure caches */
    CACHE_setL1DSize(CACHE_L1_32KCACHE);
    CACHE_setL1PSize(CACHE_L1_32KCACHE);
    CACHE_setL2Size(CACHE_128KCACHE);

    /* OMP runtime requires a portion of MSMCSRAM to be non-cached. Since it is
     * not possible to disable caching in the MSMCSRAM address range, we need a
     * 2 step process:
     * 1. Map a portion of MSMCSRAM into a range that can be marked as
     *    non-cached. This is done using the MPAX register
     * 2. Annotate the mapped section as non-cached using the appropriate
     *    MAR register for that memory range
     * All accesses to MSMCSRAM through the mapped address range will not
     * be cached.
     */
    /* 0x10 => 128K, 0x13 => 1MB */
```

```
123     __TI_setMPAX(3, MSMCSRAM_NC_START_ADDR,
124                     MSMCSRAM_START_ADDR, 0x10 /* 128 KB */);
125     __TI_omp_disable_caching(MSMCSRAM_NC_START_ADDR, MSMCSRAM_NC_SIZE);
126
127     /* Annotate MSMCSRAM and DDR as cached + prefetch + write through */
128     __TI_omp_enable_caching(MSMCSRAM_START_ADDR, MSMCSRAM_SIZE);
129     __TI_omp_enable_caching(DDR_START_ADDR,      DDR_SIZE);
130
131 }
132
133
134 /**
135  * Default OpenMP Runtime configuration function.
136  *
137  * The OpenMP runtime requires the following hardware resources:
138  *   - The set of cores (contiguous) that run the OpenMP runtime
139  *   - Hardware Semaphores (6)
140  *   - QMSS general purpose hardware queues (11)
141  *   - QMSS memory region (1)
142  *
143  * The configuration function specifies the hardware resources that can be
144  * used by the runtime.
145  *
146  * It is annotated weak and can be overridden by a user provided function
147  * with the same name.
148  * @see __TI_omp_config_thread_stack
149  */
150 #pragma CODE_SECTION(__TI_omp_configure, ".text:omp:configure")
151 #pragma WEAK(__TI_omp_configure)
152 void __TI_omp_configure(void)
153 {
154     __TI_omp_config_cores (OMP_MASTER_CORE_IDX, OMP_NUM_CORES);
155
156     __TI_omp_config_hw_semaphores(/*hw_sem_base_idx=*/3);
157
158     __TI_omp_config_clock_freq_in_mhz(CLOCK_RATE);
159
160     /* The OpenMP runtime requires 11 hardware queues and uses the QMSS LLD
161      * APIs to initialize these queues.
162      * QMSS_PARAM_NOT_SPECIFIED
163      *      QMSS LLD allocates queue numbers during call to Qmss_queueOpen
164      * Qmss_MemRegion_MEMORY_REGION_NOT_SPECIFIED
165      *      QMSS LLD allocates memory region used by Qmss_insertMemoryRegion
166      */
167 #ifdef TI_C6636
168     __TI_omp_config_hw_queues (/* init_qmss=*/                        1,
169                                /* hw_queue_base_idx=*/               7332,
170                                /* first_desc_idx_in_linking_ram=*/ 8000,
171                                /* first_memory_region_idx=*/          32);
172 #else
173     __TI_omp_config_hw_queues (/* init_qmss */ 1,
174                                QMSS_PARAM_NOT_SPECIFIED,
175                                /* first_desc_idx_in_linking_ram=*/0,
176                                Qmss_MemRegion_MEMORY_REGION_NOT_SPECIFIED);
177 #endif
178
179     /* Thread stacks in core local memory */
180     __TI_omp_config_thread_stack(0, 0);
181     //__TI_omp_config_thread_stack(1, 0x400000);
182 }
183
184 /* @} */
```


