OMP (OpenMP Runtime for SYS/BIOS) Users Guide

Table of Contents

1. Directory structure of OMP product

`/etc`	Contains sample config.bld.default file that can be used to build OMP applications outside CCS
`/docs`	Documentation including this Users Guide
`/packages`	Repository of OMP libraries
`/ti/omp`	Package containing libgomp port (source + libraries)
`/config/sysbios`	SYS/BIOS-specific helper code used by ti.omp package
`/navigator`	QMSS specific module used by ti.omp package
`/utils`	RTSC modules used internally by OMP (SemaphoreMP, ThreadLocal) and used to configure OpenMP application (OpenMP module)
`/examples`	Examples used for CCS project templates
`/runscript`	Script used to run build OMP images without the use of CCSv4/5 IDE

2. Building OpenMP examples within CCSv4/5

You can use the "Project Wizard" in CCS to create a project using some of the examples that accompanies the OMP product. NOTE: some minor variations of these steps might be needed for CCSv5.

1) Click on File→ New → CCS Project
2) Choose a name for your project
3) Choose 'C6000' as the Project Type
4) Simply click 'Next' on the 'Additional Project Settings' dialog
5) Ensure that the above configuration exists in the 'Project Settings' dialog.
6) Choose an OpenMP example
7) Ensure that the OpenMP, SYS/BIOS, IPC and PDK products are selected in "Products and Repositories." Ensure that the following RTSC build settings are chosen:
    - Target: ti.targets.elf. C66
    - Platform: ti.omp.examples.platforms.evm6678/evm6670
    - Build-profile: debug or release
8) Click 'Finish'. You should now see your project in the Projects pane. At this point you can build and run the executable.

3. Building OpenMP examples without CCSv4/5

You may either use an XDC-build approach or a traditional Makefile-based approach to build an OMP application.

Using an XDC-build approach:

1) Create a repository (directory) on your filesystem where you will place the following
   a. Your applications (.c, .cfg files)
   b. A 'config.bld' file that tells XDCs how to build your application. NOTE: ensure that the --omp flag is passed to the C6x compiler.
   c. Build scripts (package.bld, package.xdc) that tell 'xdc' what to build.

   # mkdir /home/myRepo

2) Set the XDCPATH environment variable to point to the following repositories:
   a. XDC
   b. SYS/BIOS
   c. IPC
   d. PDK
   e. OMP
   f. 'myRepo'

```
# setenv XDCPATH
/path/to/xdctools_xx/packages;/path/to/bios_xx/packages;/path/to/
ipc_xx/packages;/path/to/pdk_xx/packages;/path/to/omp_xx/packages
;/home/myRepo
```

3) Create a package.xdc file within 'myRepo' with the following contents:

```
package myRepo [1,0,0,0] {
}
```

4) Create a package.bld file with the following contents:

```
var testBld = xdc.loadCapsule("ti/sysbios/build/test.bld");

Pkg.uses = [
    "ti/omp",
];

/* Include the header files in ti/omp */
Pkg.attrs.incs= "-I ..";

/*
 *  ======== testArray ========
 */
var testArray = [
    {name: 'myApp-1'},  /* Assume that myApp-1.c/cfg and   */
    {name: 'myApp-2'},  /* myApp-2.cfg exist               */
];
```

```
arguments = ["profile=release"];

/* Generate the makefile goals. 'arguments' is XDCARGS */
testBld.buildTests(testArray, arguments);
```

NOTE: The 'Pkg.uses' configuration primarily exists to ensure that 'omp.h' is in the compiler path.

5) Type 'xdc' within the myRepo directory. This should build the tests.

Using a Makefile approach:

You may alternatively generate prebuilt C libraries against which an OpenMP application can be compiled and linked. A typical build flow involves building the prebuilt library once for a given device (i.e. evm6678) and for a specific RTSC configuration.

The files needed to generate the prebuilt libraries can be found in [OMP_INSTALL_DIR]\preconfig directory.

1) Edit 'ompdefault.cfg' as needed to match your desired RTSC configuration.

2) Edit the 'makeomplibs' file as needed:
   a. Point to your BIOS, IPC, PDK, XDCTools and OMP products
   b. Change the build profile as needed
   c. Change the build platform as needed

3) Build the prebuilt libraries

```
$ make -f makeomplibs omp-evm6678
```

4) Edit Makefile as follows:
   a. Edit the path to the C6x OpenMP-aware codegen tools
   b. Add application build goals to the Makefile using the example for 'omp_hello' provided as a guideline.

5) Build the application:

```
$ make omp_hello.xe66
```

4. Creating a new OMP application

You need to make your application a RTSC application. This involves adding a .cfg file (i.e. myapp.cfg) whose minimum contents of this .cfg file need to be:

```
xdc.loadCapsule('ti/omp/common.cfg.xs');
```

By default, the number of cores used by an OMP application is equal to the maximum # of cores on the device (i.e. 4 on C6670 and 8 on C6678). However, this number of cores can be overridden. The number of processors used in the application can be set as follows:

```
var OpenMP = xdc.useModule('ti.omp.utils.OpenMP');
OpenMP.setNumProcessors(3); /* Only use CORE0, CORE1 and CORE2 */
```

Set the application shared memory heap. Set cache write through enable for the heap.

```
var HeapOMP = xdc.useModule('ti.omp.utils.HeapOMP');
HeapOMP.sharedRegionId = 2;
HeapOMP.localHeapSize  = 0x20000;
HeapOMP.sharedHeapSize = 0x1000000;
// Specify the Shared Region
SharedRegion.setEntryMeta( HeapOMP.sharedRegionId,
                           {   base: 0x90000000,
                               len:  HeapOMP.sharedHeapSize,
                               ownerProcId: 0,
                               createHeap: true,
                               isValid: true,
                               name: "HeapOMP",
                           }
                          );
var Cache        = xdc.useModule('ti.sysbios.family.c66.Cache');
Cache.setMarMeta(0x80000000, 0x20000000, Cache.PC | Cache.WTE );
```

These are the defaults for some optional configurations

```
OpenMP.stackRegionId = -1; // Stack region for dynamic tasks
OpenMP.qmssInit = true ;  // QMSS to be Initialized by OMP
OpenMP.qmssMemRegion = -1; // QMSS Mem region to be used by OMP
OpenMP.qmssStartIndex = 0; // QMSS Start index to be used by OMP
OpenMP.qmssDestQueueNum = -1; // QMSS Destination Queue to be
//                               used by OMP
OpenMP.internalOsal = true; // OMP OSAL to be used
OpenMP.cppiHwSem = 3; //if OMP OSAL, CPPI HW Semaphore
OpenMP.qmssHwSem = 4; //if OMP OSAL, QMSS HW Semaphore
OpenMP.autoDnld = true; //Slave cores load and run by Master core
OpenMP.mpaxIndex = 3; // MPAX Index for mapping DDR to MSMC
```

If ti/omp is not in the compiler's include path, the omp header files need to be specified using an absolute path

i.e.

```
#include "omp.h"
```

needs to change to

```
#include <ti/omp/omp.h>
```

5. Incompatibilities between SYS/BIOS OMP and Linux OpenMP runtime

Although the OMP runtime is designed to seamlessly execute OpenMP applications that work with GCC's libgomp and POSIX, it is important to note that a few incompatibilities exist:

- GCC intrinsic aren't supported by the Texas Instruments compiler.

6. Creating a RTSC Platform

Certain rules must be satisfied in your RTSC platform in order to successfully build & execute an OMP application. Be aware that breaking these rules might result in an application that builds fine but might (arbitrarily) malfunction at runtime. RTSC platforms contain configurations called 'dataMemory', 'codeMemory', and 'stackMemory' which determines where various linker sections are placed.

'dataMemory' rules

- Shared OpenMP state is contained within the 'dataMemory' portion of the RTSC platform. Since cache coherency is not performed upon this state, the 'dataMemory' must set to a non-cached shared memory segment.
- On C66 devices, L1D can set to a non-zero value as long as libgomp state is placed in non-cached shared memory. 'dataMemory' can be set to MSMC RAM as long as the RAM is accessed via a non-cached MPAX page. In order to allow this, the OMP runtime uses a reset function that is executed before c_init_00 and maps the entire range of MSMC RAM (0x0C000000) to a non-cached alias at 0xA0000000).
    - The sample C6678 platform that accompanies OMP (ti.omp.examples.platforms.evm6678) includes a memory segment named "MSMCSRAM_NOCACHE" that is placed within this non-cached page.
    - If 'dataMemory' is set to a memory segment that lies in the physical address range for MSMC RAM (0x0C000000), then L1D must be set to '0K' to make this address range non-cacheable.

'codeMemory' rules

- 'codeMemory' contains code & constants and should be set to cached shared memory. **L1P cache should always be set to '32K' to maximize performance.**
-

'stackMemory' rules

- 'stackMemory' contains data that is local to each core and must be set to a memory segment that is local to each core (i.e. LL2RAM). This memory should ideally be cached as well (i.e. by turning on L1D cache if possible).

General platform rules

- Care must be taken to ensure that platform segments placed in the 0xA0000000 address range don't overlap with platform segments placed in the 'native' (0x0C000000) address range.

```
/* BAD (overlapping segments) */
["MSMCSRAM", {name: "MSMCSRAM",  base: 0x0C000000, len:
0x00200000}],
["MSMCSRAM_NOCACHE", {name: "MSMCSRAM_NOCACHE", base: 0xA0100000,
len: 0x00300000}],

/* GOOD (no overlapping segments) */
```

```
["MSMCSRAM", {name: "MSMCSRAM",  base: 0x0C000000, len:
0x00100000}],
["MSMCSRAM_NOCACHE", {name: "MSMCSRAM_NOCACHE", base: 0xA0100000,
len: 0x00300000}],
```

Optimizing of OMP applications

The 'debug' build profile is best used when debugging OMP applications, but using the 'release' build profile will yield best performance. **The whole_program[_debug] profile has been deprecated and is no longer supported**. Please refer to the IPC Users Guide and the BIOS Users guide for ways to optimize memory usage and improve runtime performance.

7. Running Built OMP applications

Running within CCS IDE

It is recommended to disable auto-run-to-main in CCS before running OMP applications.

1) Load the image onto CORE0 only and load symbols on the remaining cores being used in the application. If ***OpenMP.autoDnld*** is set to true, the master core will reset the slave core, and run the slave core from **_c_int00**.
2) Please make sure the slave core are either disconnected or are running. If the slave core are disconnected the CIO will not be available from salve core.
3) To see the output from slave core with ***OpenMP.autoDnld*** set to **true**, please load the symbols on slave cores.

Running using supplied run scripts

OMP contains scripts that utilize CCS's scripting capabilities (DSS) to allow running a built OMP executable and viewing CIO output within a Windows console.

To use these scripts supplied with the OMP product:

1) Edit 'DEBUGSERVER' in common_variables.bat to point to your CCS installation
2) Create a new platforms\[MYCONFIG].xs file using an existing one (i.e. evm6678_xds560.xs) as a template.
    a. Place your .ccxml file in a place where the scripts can access them (i.e. OMP_INSTALL_DIR\packages\regression\common\ccxml). Point your platforms\[MYCONFIG].xs file to that .ccxml file using the following syntax:

```
Main.ccxmlConfigFile =
"../regression/common/ccxml/evm6678_xds560.ccxml";
```

    b. Make sure that the platform name in the first argument to 'Main.addCpu' matches your platform name.

3) Run the script as follows (the -n flag should be used to specify the number of cores that the tests runs on)
a) *OpenMP.autoDnld = false;*

```
V:\ti\omp_1_01_02_xxx\packages\runscript>run.bat
platforms\evm6678_xds560.xs
    v:\ti\omp\tests\release\omp_hello.xe66 -n 4

Connected to cores #0-3
[CORE0]
    Hello World from thread = 0
    Number of threads = 4
[CORE1]
    Hello World from thread = 1
[CORE2]
    Hello World from thread = 2
[CORE3]
    Hello World from thread = 3

V:\ti\omp_1_01_02_xx\packages\runscript>
```

b) *OpenMP.autoDnld = true;*

```
V:\ti\omp_1_01_02_xx\packages\runscript>run.bat
platforms\evm6678_xds560.xs
    v:\ti\omp\tests\release\omp_hello.xe66 -n 1

Connected to cores #0-0
[CORE0]
    Hello World from thread = 0
    Number of threads = 4

V:\ti\omp_1_01_02_xx\packages\runscript>
```

8. Rebuilding 'OMP' product

The following steps can be used to rebuild the OMP runtime. You need to have compatible versions of the following products in order to rebuild OMP. Check the release notes for specific versions needed for this version of OMP.

- SYS/BIOS 6
- XDCTools
- IPC

http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/index.html

- PDK, C667x MCSDK

http://software-dl.ti.com/sdoemb/sdoemb_public_sw/bios_mcsdk/latest/index_FDS.html

Once the products have been installed on your system, follow steps similar to "Rebuilding IPC" in the IPC Users Guide (SPRUGO6C), Appendix A. Instead of using the config.bld.default supplied by default, use the version in [INSTALLED_OMP_PRODUCT]/packages/etc/config.bld.default. Your XDCPATH will need to contain all the products listed above (including the packages/ subdirectory) as well as the OMP product and your own repository.