SMP/BIOS Overview

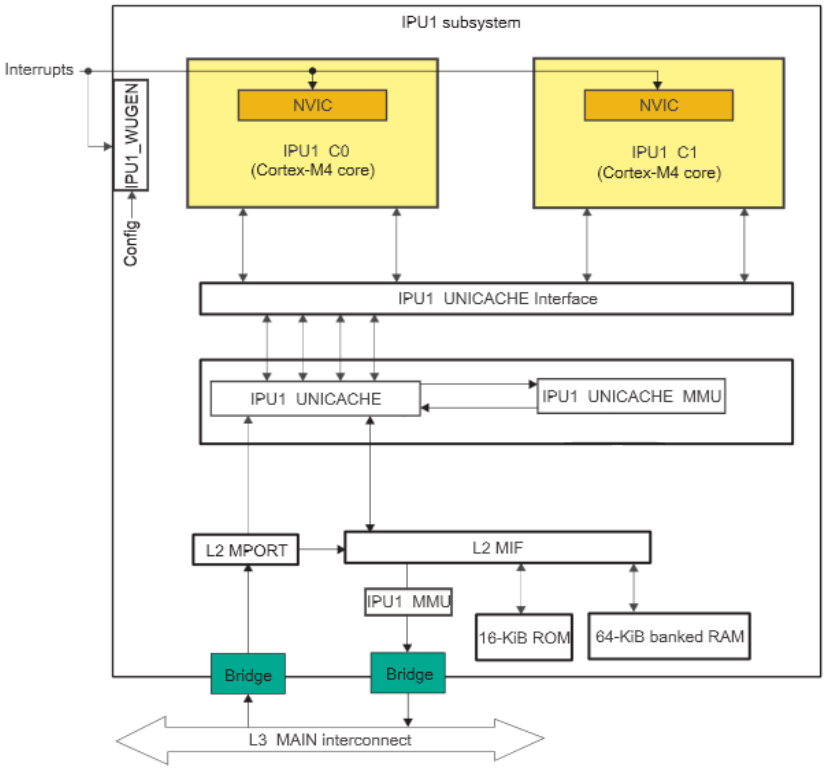
May 19, 2016

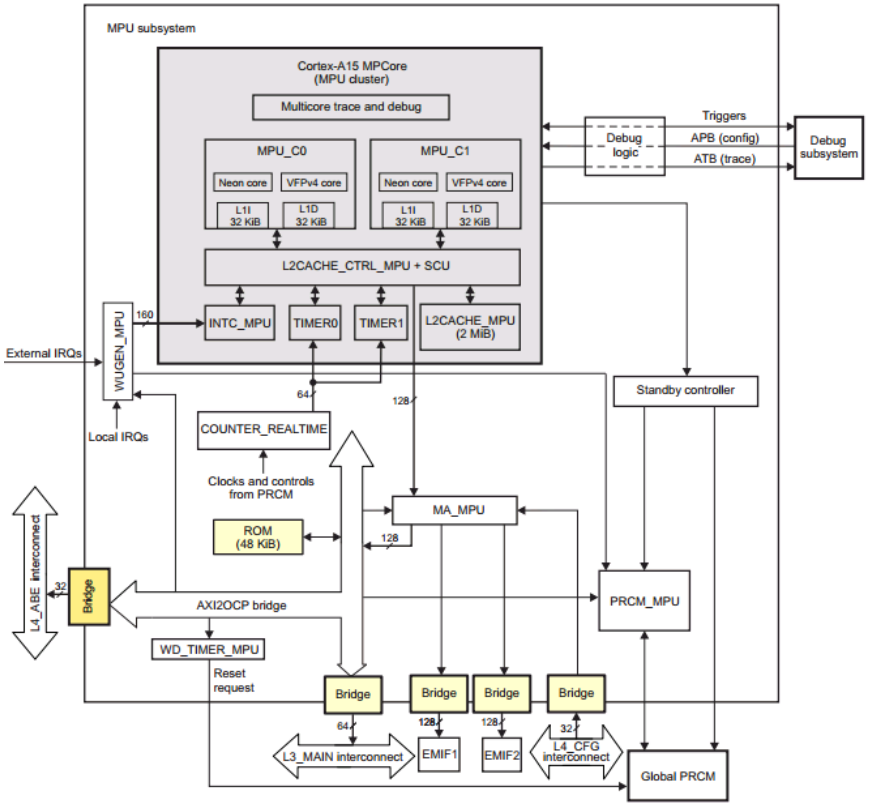
!!! SMP/BIOS is currently supported only on dual-core Cortex-M3/M4 subsystems and Cortex-A15 (J6/K2/AM57x) subsystems !!!

Agenda

- SMP/BIOS Overview
  - What is SMP ?
  - What is SMP/BIOS ?
  - SMP/BIOS Benefits
  - New APIs & Config Params
  - Task Scheduling in SMP mode
  - Hwi/Swi Scheduling in SMP mode
  - Inter-core locking
  - Performance and Benchmarking
  - Getting Started
  - CCS setup for SMP debug (Sync Group feature)
  - Summary
- Q&A

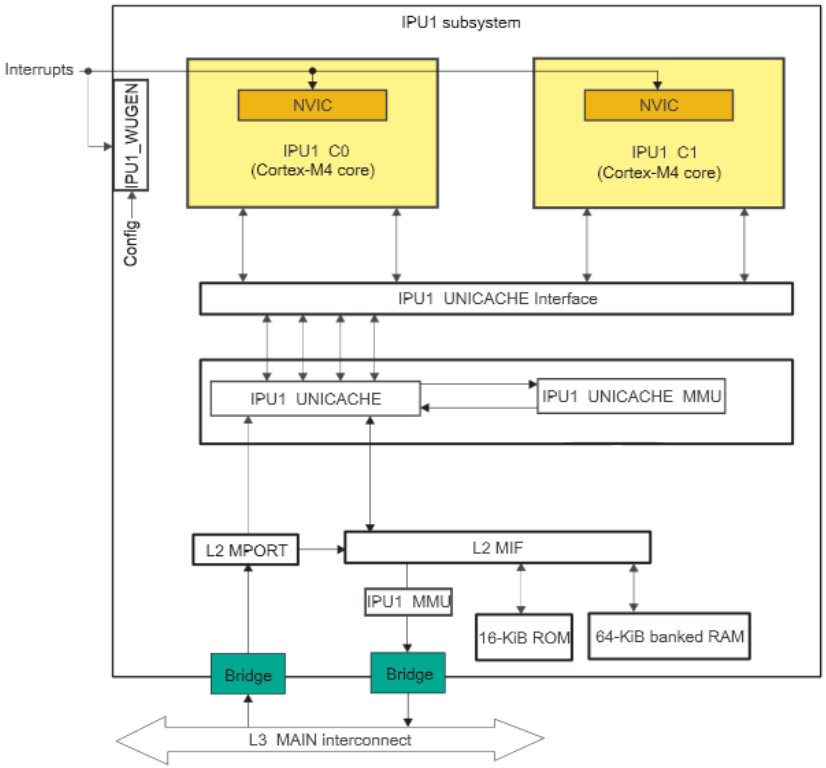
What is SMP ?

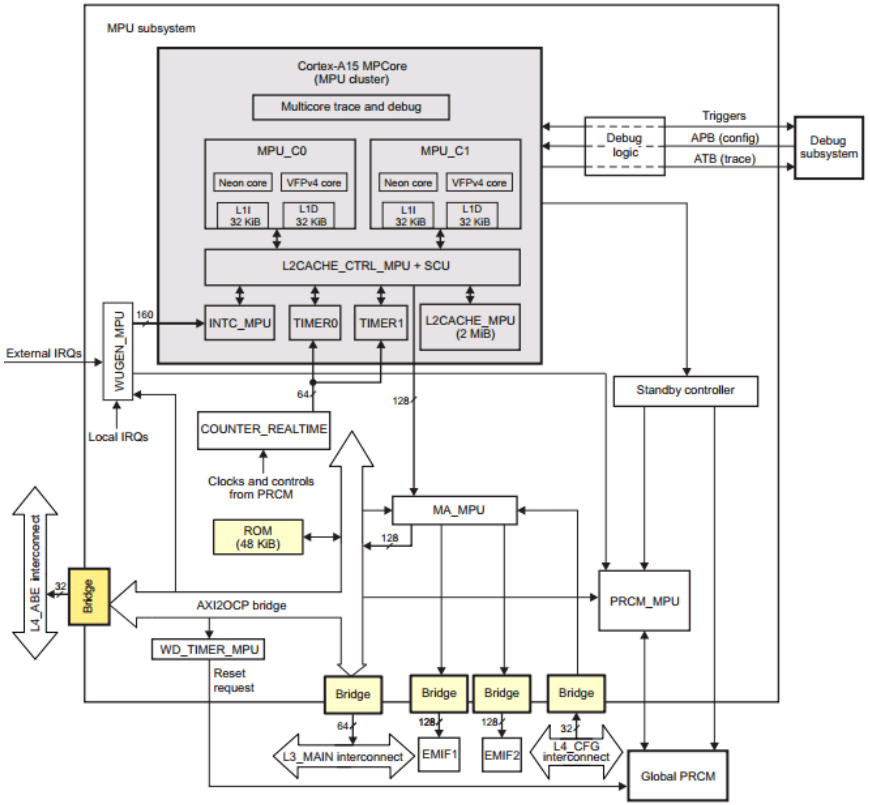




- Symmetric Multiprocessing (SMP) involves a single OS instance managing processing on two or more identical processor cores that share a common view of memory and peripherals.

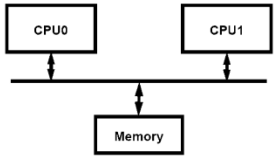
What is SMP/BIOS ?





- SMP/BIOS is an operational mode of SYS/BIOS which is supported on selected multi-core subsystems present on various TI SoC devices. As of this writing, SMP/BIOS is supported on TI's dual Cortex-M3/M4 and multi-core Cortex-A15 subsystems.

SMP/BIOS Benefits



- A single instance of SMP/BIOS manages the concurrent execution of tasks on shared cores.
- Simplified development of multi-core applications.
  - Easily maximize utilization of each core.
- Simple migration from multiple SYS/BIOS application instances to a single SMP/BIOS application instance.
- Single application image is loaded versus 2 separate images
  - Helps save boot time.
- Inherent load-balancing of Tasks.
  - Task affinity needs to be “DON’T CARE” to leverage load-balancing.
- Backwards compatible with SYS/BIOS applications with certain caveats.
  - Possible application behavioral differences (including application failure) if the application relied on task priorities and/or thread type to ensure exclusive access to system objects.

New APIs & Config Params

BIOS Module

- Bool BIOS.smpEnabled (BIOS_smpEnabled)
  - This flag is provided to manage building applications for both SMP and non-SMP versions of SYS/BIOS.

Task Module

- UInt Task_setAffinity(Task_Handle handle, UInt affinity);
  - Used to dynamically set a task's core affinity.
  - Can be used by a running task to move itself to another core.
- UInt Task_getAffinity(Task_Handle handle);
  - Used to dynamically get a task's core affinity.
- Task_Handle Task_getIdleTaskHandle(Uint coreId);
  - Returns a handle to the idle task object for the specified coreId.
- Task.defaultAffinity module config parameter
  - Used to globally define default Task affinity of user created tasks.
  - Defaults to Task_AFFINITY_NONE.
- Task.PARAMS.affinity instance config parameter
  - Used to define a task's affinity at create time.
  - Default is inherited from Task.defaultAffinity (i.e. Task_AFFINITY_NONE).

New APIs & Config Params

Idle Module

- Existing metaonly Idle.addFunc (Function);
  - Add idle functions only to Core 0
- New metaonly Idle.addCoreFunc (Function, CoreId);
  - Add idle functions to a specific core

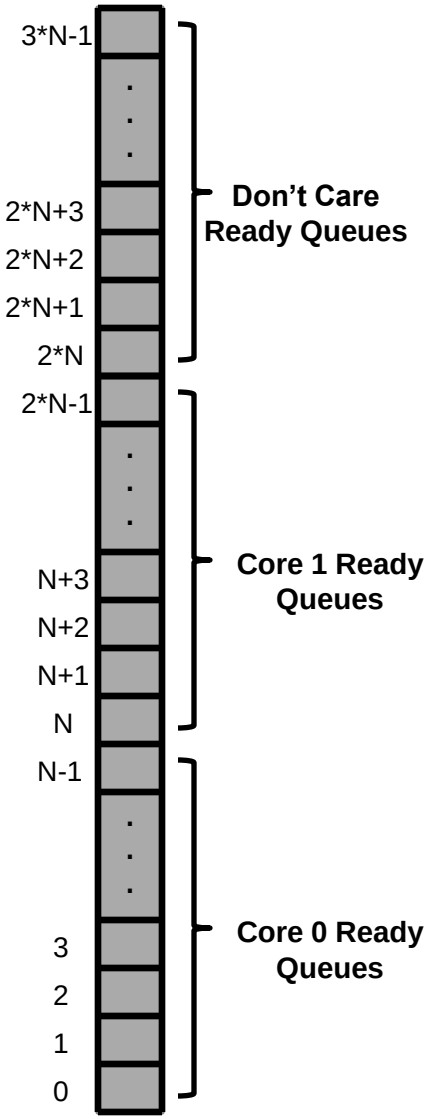
Core Module (new)

- ti.sysbios.hal.Core module
  - UInt Core_getCoreId();
    - returns the current core Id
  - const UInt Core_numCores (Core.numCores)
    - number of SMP cores
- Core is a proxy for target/device specific delegate module
  - Currently bound to ti.sysbios.family.arm.ducati.smp.Core module on Cortex-M3/M4 devices and ti.sysbios.family.arm.a15.smp.Core module on cortex-A15 devices.

Hwi Module

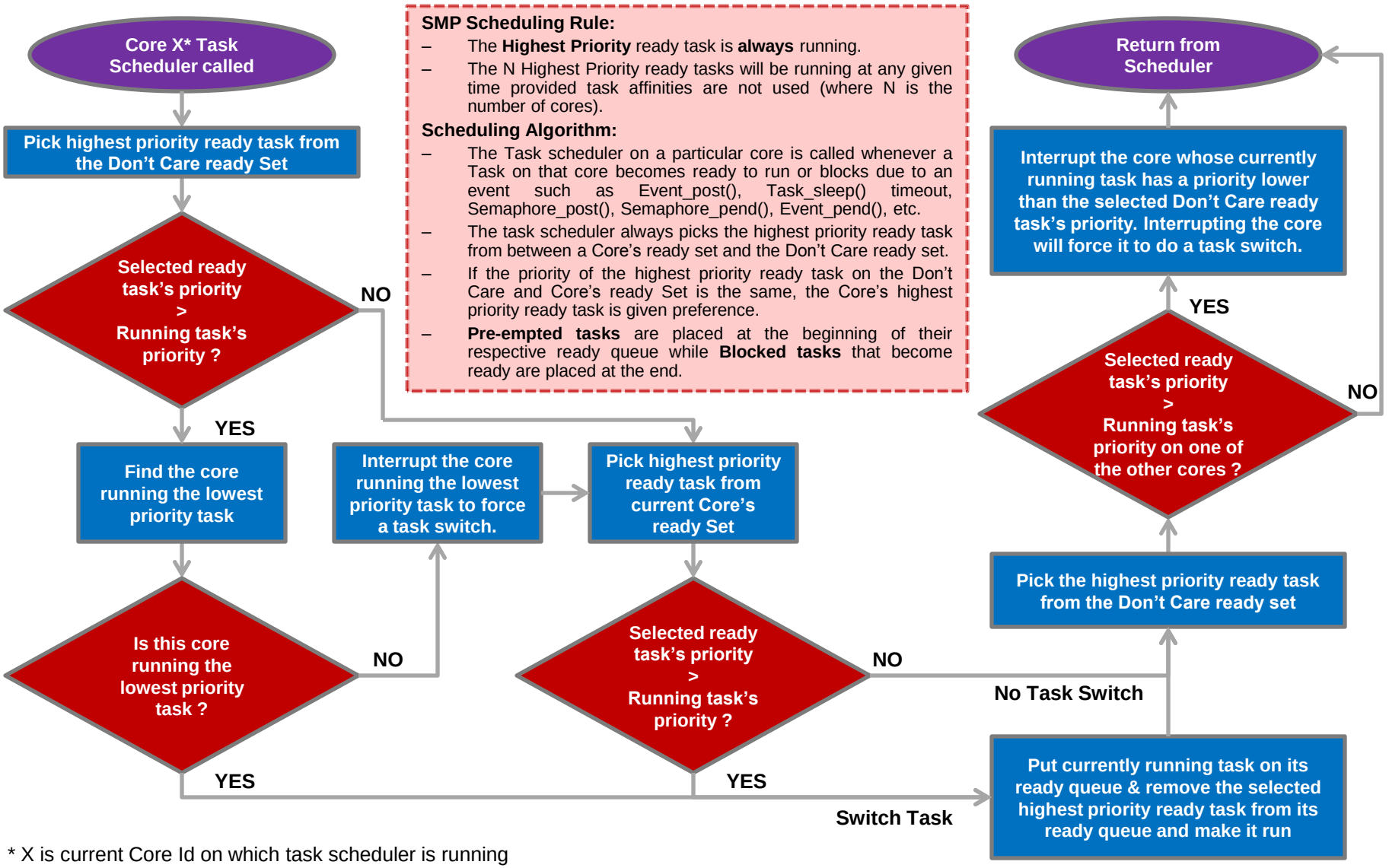
- New metaonly config UInt8 intAffinity[];
  - An array that maps an interrupt number to a particular coreId (default is core 0 for all Hwis).
  - Allows the application to control which core runs each Hwi.

Task Scheduling in SMP mode



- Core Affinity Support
  - Each task has a core affinity (Hard CPU Affinity)
  - 0, 1, or Task_AFFINITY_NONE
  - Default is Task_AFFINITY_NONE
- Task Scheduler Ready Queues
  - 3 ready queue sets in total, 1 per core affinity
  - Each ready queue set contains N queues, N being the number of supported task priorities (16 by default).
  - Each ready queue maintains a list of ready tasks that share the same priority and affinity.
  - For coding efficiency, the three sets of ready queues are placed contiguously in memory
  - A ready task is removed from its ready queue when it is made to run.

Task Scheduling Algorithm



SMP Scheduling Rule:

- The **Highest Priority** ready task is **always** running.
- The N Highest Priority ready tasks will be running at any given time provided task affinities are not used (where N is the number of cores).

Scheduling Algorithm:

- The Task scheduler on a particular core is called whenever a Task on that core becomes ready to run or blocks due to an event such as Event_post(), Task_sleep() timeout, Semaphore_post(), Semaphore_pend(), Event_pend(), etc.
- The task scheduler always picks the highest priority ready task from between a Core's ready set and the Don't Care ready set.
- If the priority of the highest priority ready task on the Don't Care and Core's ready Set is the same, the Core's highest priority ready task is given preference.
- **Pre-empted tasks** are placed at the beginning of their respective ready queue while **Blocked tasks** that become ready are placed at the end.

* X is current Core Id on which task scheduler is running

Hwi and Swi Scheduling in SMP mode

Hwi Scheduling

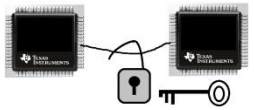
- Hwi's can be configured to run on either cores in a SMP application
- The Hwi module has a new module wide config parameter called *intAffinity* to manage which core runs a Hwi in SMP mode.
- intAffinity[] is an array that maps an interrupt number to a particular coreId.
  - By default, all interrupts are mapped to core 0.
- The core affinity cannot be changed for Interrupt numbers below 16 on Cortex-M3/M4. Such interrupts will always be serviced by core 0 in a SMP application.
- The core affinity cannot be changed for Interrupt numbers below 32 on Cortex-A15.
- Each core has its own Hwi stack on which the ISR routine runs.

Swi Scheduling

- For design simplification, Swi's are forced to run on Core 0.
- Swi's can be posted from either core but will only be run on Core 0.

NOTE: Unlike Non-SMP BIOS, Swis can be running on Core 0 <u>while</u> a Task is running on Core 1. Additionally, Hwis can be running on one core <u>while</u> a Task is running on the other core.

Inter-core Locking



- An Inter-core Lock guarantees exclusive access to Hwi/Swi/Task critical section code/data.
  - Accessed through these Core module APIs:
    - Core_lock()
    - Core_unlock().
  - These APIs are spec'd in ICore but must only be used internally by BIOS.
    - Their implementations are hardware specific.
- Current Design
  - The Inter-core lock is acquired whenever any of the three schedulers (Task, Swi or Hwi) are disabled by Task_disable(), Swi_disable(), or Hwi_disable(), and released only when all 3 schedulers are enabled.
  - *A Task or Swi disable on one core effectively disables Hwi's on the other core.*

Performance

Simulated 1080P load with 6 tasks, 14% task, 1% Hwi, 2% Swi loads running on J6 dual Cortex-M4s

	SYS/BIOS 6.42.00	SMP/BIOS affinity=0	SMP/BIOS affinity=X Hwis on Core0	SMP/BIOS affinity=X Distributed Hwis
Task Switches	70385	70089	89113	90104
Swis	19901	19903	19901	19900
Hwis	19901	19903	34653	37129
tsk0Count[0]	10002	10003	8912	8219
tsk0Count[1]	0	0	1089	1782
tsk1Count[0]	10003	10004	1485	5048
tsk1Count[1]	0	0	8517	4954
tsk2Count[0]	9904	9905	8418	3368
tsk2Count[1]	0	0	1485	6534
tsk3Count[0]	9905	9907	8021	8218
tsk3Count[1]	0	0	1883	1685
tsk4Count[0]	9905	9906	1982	199
tsk4Count[1]	0	0	7921	9702
tsk5Count[0]	9906	9908	990	4556
tsk5Count[1]	0	0	8914	5346
idleCount[0]	639269	287083	1840631	1854199
idleCount[1]	0	13353403	7121866	6713442
Hwi load	1	1	2	2
Swi load	2	2	2	2
tsk0 load	14	14	15	15
tsk1 load	14	14	15	15
tsk2 load	14	14	15	16
tsk3 load	14	14	14	14
tsk4 load	14	14	15	15
tsk5 load	14	14	14	14
Core 0 cpu load	**89**	**92**	**51**	**49**
Core 1 cpu load	**0**	**0**	**46**	**49**

Performance

Simulated 1080P load with 6 tasks, 14% task, 1% Hwi, 2% Swi loads running on J6 dual Cortex-A15s

	SYS/BIOS 6.42.00	SMP/BIOS affinity=0	SMP/BIOS affinity=X Hwis on Core0	SMP/BIOS affinity=X Distributed Hwis
Task Switches	69750	69529	88279	89921
Swis	19507	19507	19506	19210
Hwis	19507	19508	34277	37575
tsk0Count[0]	9907	9607	8590	7964
tsk0Count[1]	0	0	1016	1642
tsk1Count[0]	9908	9608	1495	3016
tsk1Count[1]	0	0	8112	6591
tsk2Count[0]	9902	9902	8507	2765
tsk2Count[1]	0	0	1395	7137
tsk3Count[0]	9903	9903	8248	9536
tsk3Count[1]	0	0	1655	366
tsk4Count[0]	9902	9903	1790	178
tsk4Count[1]	0	0	8112	9429
tsk5Count[0]	9903	9904	834	5338
tsk5Count[1]	0	0	9069	4270
idleCount[0]	2827442	694952	3078394	3057151
idleCount[1]	0	163667444	89533250	93948426
Hwi load	1	1	1	1
Swi load	2	2	2	2
tsk0 load	13	13	14	14
tsk1 load	13	13	14	14
tsk2 load	14	14	15	16
tsk3 load	13	14	14	14
tsk4 load	14	14	15	14
tsk5 load	13	14	14	14
Core 0 cpu load	**87**	**89**	**49**	**46**
Core 1 cpu load	**0**	**0**	**45**	**48**

Benchmarks

M4 Benchmark results on a evmDRA7XX (J6) board running @212.8MHz

BENCHMARK	NON-SMP	SMP (DUAL)
Hwi_enable	1	62
Hwi_disable	2	68
Hwi dispatcher prolog	103	336
Hwi dispatcher epilog	238	311
Hwi dispatcher	335	643
Hardware Interrupt to Blocked Task (Core0)	541	1610
Hardware Interrupt to Blocked Task (Core1)	N.A.	2253
Hardware Interrupt to Software Interrupt	375	854
Swi_enable	79	248
Swi_disable	13	76
Post Software Interrupt Again	36	131
Post Software Interrupt without Context Switch	101	366
Post Software Interrupt with Context Switch (Core 0)	199	582
Post Software Interrupt with Context Switch (Core 1)	N.A.	1163
Create a New Task without Context Switch	2031	3243
Set a Task Priority with a Context Switch (Same Core)	N.A.	801
Set a Task Priority without a Context Switch (Same Core)	167	385
Set a Task Priority with a Context Switch (Other Core)	N.A.	1392
Set a Task Priority without a Context Switch (Other Core)	N.A.	385
Task_yield	207	919
Change Task Affinity from 0 to 1 for a running Task	N.A.	1886
Change Task Affinity from 1 to 0 for a running Task	N.A.	1862
Change Task Affinity for a ready Task (no Context Switch)	N.A.	387
Post Semaphore, No Waiting Task	51	190
Post Semaphore No Task Switch	195	192
Post Semaphore with Task Switch	257	1093
Pend on Semaphore, No Context Switch	73	203
Pend on Semaphore with Task Switch	278	897
Clock_getTicks	8	10

* Using SYS/BIOS 6.45.01.29 product ** In CPU Cycles

Benchmarks

A15 Benchmark results on a evmDRA7XX (J6) board running @750MHz using DMTimer (@19.2MHz) for timestamps

BENCHMARK	NON-SMP	SMP (DUAL)
Hwi_enable	80	234
Hwi_disable	78	117
Hwi dispatcher prolog	442	820
Hwi dispatcher epilog	328	351
Hwi dispatcher	759	1406
Hardware Interrupt to Blocked Task (Core0)	1382	3046
Hardware Interrupt to Blocked Task (Core1)	N.A.	4687
Hardware Interrupt to Software Interrupt	920	1523
Swi_enable	310	820
Swi_disable	2	0
Post Software Interrupt Again	153	117
Post Software Interrupt without Context Switch	321	468
Post Software Interrupt with Context Switch (Core 0)	394	1054
Post Software Interrupt with Context Switch (Core 1)	N.A.	2109
Create a New Task without Context Switch	2286	6328
Set a Task Priority with a Context Switch (Same Core)	N.A.	1289
Set a Task Priority without a Context Switch (Same Core)	328	820
Set a Task Priority with a Context Switch (Other Core)	N.A.	2929
Set a Task Priority without a Context Switch (Other Core)	N.A.	820
Task_yield	710	1406
Change Task Affinity from 0 to 1 for a running Task	N.A.	3867
Change Task Affinity from 1 to 0 for a running Task	N.A.	3515
Change Task Affinity for a ready Task (no Context Switch)	N.A.	820
Post Semaphore, No Waiting Task	170	468
Post Semaphore No Task Switch	531	468
Post Semaphore with Task Switch	873	2109
Pend on Semaphore, No Context Switch	156	468
Pend on Semaphore with Task Switch	749	1523
Clock_getTicks	0	0

* Using SYS/BIOS 6.45.01.29 product ** In CPU Cycles

Benchmarks

A15 Benchmark results on a K2H board running @1GHz using Timer64 (@225.28MHz) for timestamps

BENCHMARK	NON-SMP	SMP (QUAD)
Hwi_enable	79	199
Hwi_disable	78	195
Hwi dispatcher prolog	621	1087
Hwi dispatcher epilog	319	488
Hwi dispatcher	935	1549
Hardware Interrupt to Blocked Task (Core0)	1560	3342
Hardware Interrupt to Blocked Task (Core1)	N.A.	4833
Hardware Interrupt to Software Interrupt	1096	1797
Swi_enable	309	785
Swi_disable	5	186
Post Software Interrupt Again	154	275
Post Software Interrupt without Context Switch	318	696
Post Software Interrupt with Context Switch (Core 0)	379	1020
Post Software Interrupt with Context Switch (Core 1)	N.A.	2525
Create a New Task without Context Switch	2267	6281
Set a Task Priority with a Context Switch (Same Core)	N.A.	1282
Set a Task Priority without a Context Switch (Same Core)	329	847
Set a Task Priority with a Context Switch (Other Core)	N.A.	2783
Set a Task Priority without a Context Switch (Other Core)	N.A.	847
Task_yield	705	1429
Change Task Affinity from 0 to 1 for a running Task	N.A.	3946
Change Task Affinity from 1 to 0 for a running Task	N.A.	3693
Change Task Affinity for a ready Task (no Context Switch)	N.A.	874
Post Semaphore, No Waiting Task	166	488
Post Semaphore No Task Switch	524	488
Post Semaphore with Task Switch	875	2104
Pend on Semaphore, No Context Switch	158	483
Pend on Semaphore with Task Switch	738	1535
Clock_getTicks	5	0

* Using SYS/BIOS 6.45.01.29 product ** In CPU Cycles

Getting Started

- Download and install the latest SYS/BIOS and XDCtools products
  - Link to SYS/BIOS download page:
    - http://downloads.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/bios/sysbios/index.html
- Porting existing SYS/BIOS applications to SMP/BIOS:
  - Merge existing separate applications into a single application.
  - Merge separate platform memory definitions as necessary.
  - Add this to your existing application's config script:
    - BIOS.smpEnabled = true;
  - Use these SMP-aware clone modules in place of their xdc.runtime equivalents:
    - SysMin, SysStd, LoggerBuf (in ti.sybios.smp package)
  - The existing Load module has been tweaked to provide minimal support for SMP.
  - For initial sanity testing, force all tasks to run on core 0:
    - Task.defaultAffinity = 0;
  - Once basic functionality of the merged applications has been demonstrated, either remove Task.defaultAffinity setting or replace it with:
    - Task.defaultAffinity = Task.AFFINITY_NONE;

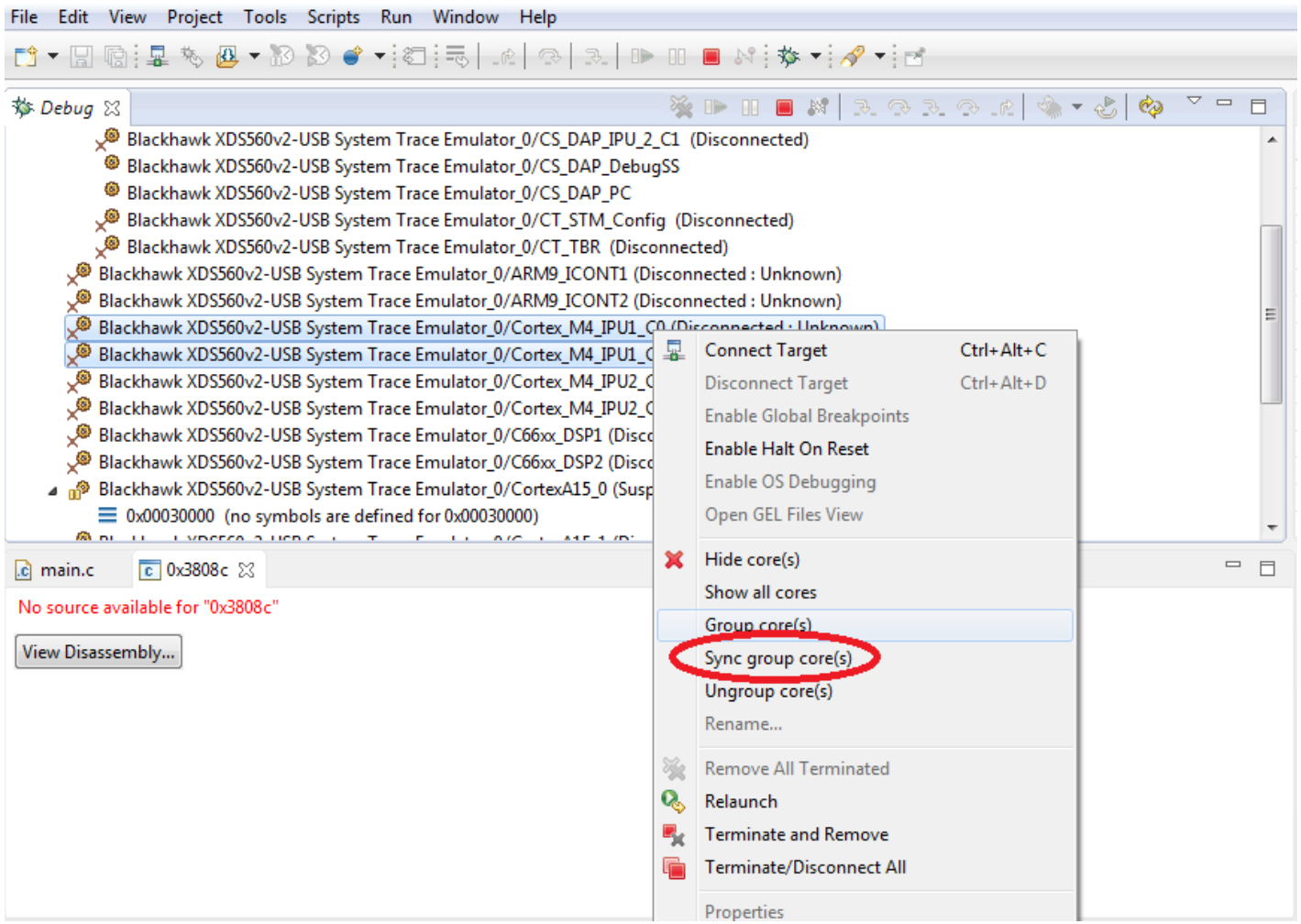
Getting Started

Loading and running applications using CCS

- Load the single image on both cores and release both cores from reset simultaneously.
- CCS 5.4+ adds a new “Sync Groups” feature that allows you to group the cores and treat them as a single debug entity.
  - See SMP Debug wiki for more info: http://processors.wiki.ti.com/index.php/SMP_Debug
- With “Sync Groups”, a Software breakpoint set on one core is automatically set on the other core. Also, starting/halting one core will start/halt the other core.
- If using an older version of CCS that does not support “Sync Groups”, to debug Cortex-M3/M4 SMP applications or if it is not possible to start both cores simultaneously, start Core1 before starting Core0.
  - There is no such ordering requirement for starting Cortex-A15 cores.

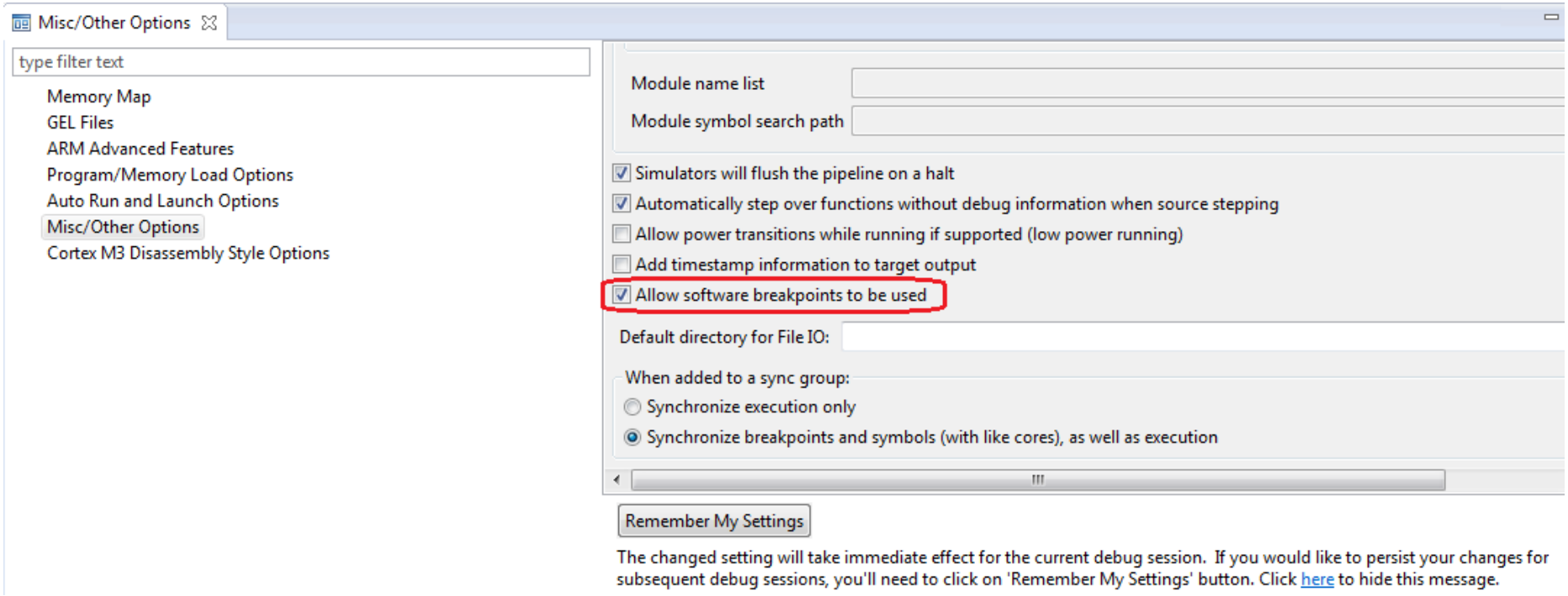
CCS setup for SMP Debug

Step1: Create a “Sync Group” that groups the cores in the SMP sub-system.



CCS setup for SMP Debug

Step2: Goto **Tools->Debugger Options->Misc/Other** Options and select "Allow software breakpoints to be used".



CCS setup for SMP Debug

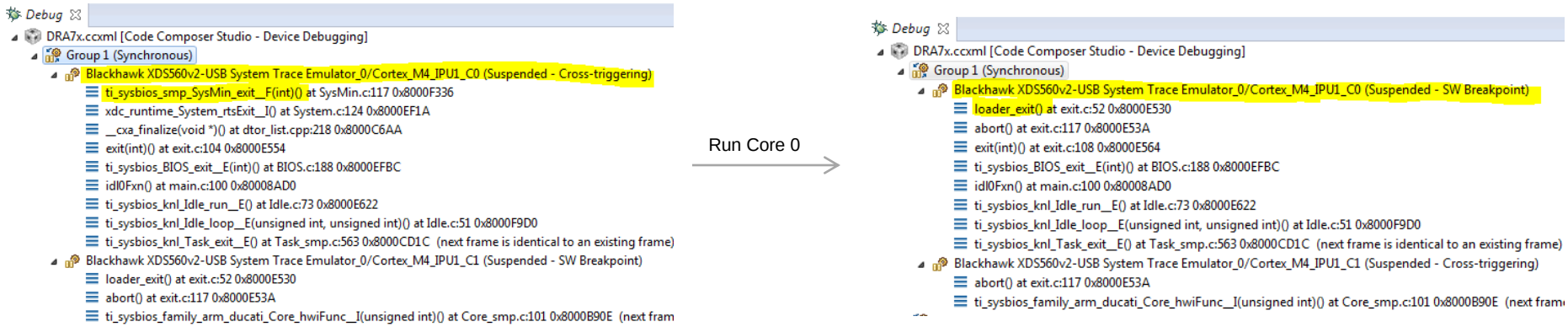
Step3: Shared memory needs to be setup next. We need to tell CCS what regions of memory are shared between the dual M3/M4/A15 cores. This is done by adding a shared attribute ("|SH#" where # is the shared memory block number). Here's an excerpt from a gel script that is used to configure the memory as shared between the two M4 cores:

```
hotmenu memorymap_init_shared()
{
    /* 64-bit OCP Master Port */
    GEL_MapAddStr(0x00000000, 0, 0x00004000, "R|W|AS4|SH1", 0);     /* Boot space - 16KBytes */
    GEL_MapAddStr(0x00004000, 0, 0x54FFC000, "R|W|AS4|SH2", 0);     /* L3 space - 1.5GBytes */


    /* Private memory map */
    GEL_MapAddStr(0x55000000, 0, 0x00004000, "R|W|AS4|SH3", 0);      /* ROM - 16KBytes */
    GEL_MapAddStr(0x55020000, 0, 0x00010000, "R|W|AS4|SH4", 0);      /* RAM - 64KBytes */
    GEL_MapAddStr(0x55040000, 0, 0x00020000, "R|W|AS4|SH5", 0);      /* ISP5 - 128KBytes */
    GEL_MapAddStr(0x55060000, 0, 0x00020000, "R|W|AS4|SH6", 0);      /* SIMCOP - 128KBytes */
    GEL_MapAddStr(0x55080000, 0, 0x00001000, "R|W|AS4|SH7", 0);      /* Unicache MMU - 4KBytes */
    GEL_MapAddStr(0x55081000, 0, 0x00001000, "R|W|AS4|SH8", 0);     /* Local PRCM register- 4KBytes */

    GEL_MapAddStr(0x55082000, 0, 0x00001000, "R|W|AS4|SH9", 0);     /* L2 MMU- 4KBytes */

    GEL_MapAddStr(0x56000000, 0, 0x0A000000, "R|W|AS4|SH10", 0);     /* L3 main space - 160MBytes */

    /* 64-bit OCP Master Port */
    GEL_MapAddStr(0x60000000, 0, 0x20000000, "R|W|AS4|SH11", 0);     /* Tiler space - 512MBytes */
    GEL_MapAddStr(0x80000000, 0, 0x60000000, "R|W|AS4|SH12", 0);     /* EMIF - 1.5GBytes */

    /* Private peripheral map (different view from each cortexM3 */
    GEL_MapAddStr(0xE0001000, 0, 0x00001000, "R|W|AS4", 0);     /* DWT- 4KBytes */
    GEL_MapAddStr(0xE0002000, 0, 0x00001000, "R|W|AS4", 0);     /* FPB- 4KBytes */
    GEL_MapAddStr(0xE000E000, 0, 0x00001000, "R|W|AS4", 0);     /* NVIC- 4KBytes */
    GEL_MapAddStr(0xE0042000, 0, 0x00001000, "R|W|AS4", 0);     /* IceCrusher Space- 4KBytes */
    GEL_MapAddStr(0xE00FE000, 0, 0x00001000, "R|W|AS4", 0);     /* RW table- 4KBytes */
    GEL_MapAddStr(0xE00FF000, 0, 0x00001000, "R|W|AS4", 0);     /* ROM table- 4KBytes */

    GEL_MapAddStr(0xE0100000, 0, 0x1FF00000, "R|W|AS4", 0);     /* Access to LPDDR2 SDRAMs */
}
```

CCS setup for SMP Debug

Known Issues: Occasionally users may observe that the SysMin log does not get flushed to the CIO console. This happens since execution cross-triggering has been enabled and when one of the cores exits and halts, it causes the other core to halt too. The solution is to run the other core that halted before flushing the SysMin buffer. Alternatively, the “Halt at program termination” feature can be disabled on all cores. This feature can be disabled from the “Tools” -> “Debugger Options” -> “Program/Memory Load Options” window. Once this feature is disabled, the cores will not halt at exit and need to be manually paused/halted.

The screenshots below show the “Debug” window view when such a situation occurs:



Summary

- With SMP/BIOS, two tasks may be RUNNING SIMULTANEOUSLY on two different cores.
  - You can not depend on Task priorities to guarantee critical section protection.
- Hwi's can be mapped to either CPU cores.
  - Core affinity can only be set for interrupt numbers >= 16 on M3/M4 and for interrupt numbers >= 32 on A15.
- All Swi's are run on core 0.
  - May be posted on either core.
- Swi's running on core 0 while tasks are running on core 1 may violate thread execution assumptions.
  - You can not depend on Swi/Task priorities to guarantee critical section protection
- SMP aware SysMin, SysStd, LoggerBuf modules (in ti.sysbios.smp package) are provided in place of corresponding xdc.runtime equivalents.
- ti.sysbios.Load module has been enhanced to support SMP/BIOS.

Q&A

Useful Links	
SYS/BIOS product download pages	http://downloads.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/bios/sysbios/index.html
SYS/BIOS wiki page	http://processors.wiki.ti.com/index.php/Category:SYSBIOS
SYS/BIOS FAQs	http://processors.wiki.ti.com/index.php/SYS/BIOS_FAQs
SMP/BIOS public wiki	http://processors.wiki.ti.com/index.php/SMP/BIOS
SMP/BIOS debug help	http://processors.wiki.ti.com/index.php/SMP_Debug