SYS/BIOS with GCC (CortexA)

SYS/BIOS 6.35.00.20 product release, combined with XDC Tools 3.24.06.63 introduced support for building SYS/BIOS and SYS/BIOS applications with the GNU compiler (GCC) toolchain for ARM Cortex-A8 (AM35xx and TI81xx devices) and Cortex-A15 targets (DRA7xx, TDA2xx and OMAP54xx devices). SYS/BIOS 6.37.00.20 (XDCTools 3.25.04.88) product release added support for Cortex-A9 target (OMAP4 and AM437x devices).

As of SYS/BIOS 6.41.01 product, we also support running in SMP mode on multi-core Cortex-A15 devices. For more info on this please visit the SMP/BIOS wiki (http://processors.wiki.ti.com/index.php/SMP/BIOS) page.

This page describes two aspects of working with SYS/BIOS:

1. Building SYS/BIOS from source
2. Building applications using GCC with SYS/BIOS

Building SYS/BIOS from source is straightforward, whether using the TI compiler toolchain or the GNU GCC toolchain. There are a few different ways to build applications. Users can choose what fits their development model.

To make the discussion concrete, a simple directory structure is used below. This is only an example and names and paths can easily be changed to fit other models. The ... below will be referred to as the $BASE directory in the following discussion.

```
.../A15GccExample/bios_6_35_02_45     - the SYS/BIOS product distribution
.../A15GccExample/rebuilt-sysbios     - SYS/BIOS rebuilt locally for desired target but not configured (optional)
.../A15GccExample/sysbios.cfg         - Project's SYS/BIOS configuration (.cfg) (optional)
.../A15GccExample/reentrancyTest.c    - App that builds with configuration as part of the project
```

An implementation of the proposal is attached for Linux - File:A15GccExampleLinux.tar.gz and Windows - File:A15GccExampleWindows.zip. You will need to add your own copies of:

1. SYS/BIOS
2. XDC Tools
3. GCC (arm-none-eabi)

Note: In general, avoid installing the various tools and source distributions in directories that have spaces in their pathnames.

Contents

Supported devices

Cortex-A8, Cortex-A9 and Cortex-A15 based devices. See device support (http://downloads.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/bios/sysbios/6_41_00_26/exports/bios_6_41_00_26/docs/cdoc/ti/sysbios/family/doc-files/devices.html) for a complete list of supported devices. The supported device list can also be obtained from the SYS/BIOS release notes and is more accurate for the downloaded product version.

Build SYS/BIOS

SYS/BIOS ships with a top-level makefile that enables users to easily (re)build SYS/BIOS using their choice of compilers and desired "targets". (A target incorporates a particular ISA and runtime model. For example, Cortex-A15 and gcc compiler with specific options).

One-time Setup

- Install SYS/BIOS product release
- Clean it

<syntaxhighlight lang="javascript">

```
linux% cd $BASE/A15GccExample/bios_6_35_02_45
linux% make -f bios.mak XDC_INSTALL_DIR=<path-to-xdctools> clean
```

</syntaxhighlight>

- Remove top-level docs directory (optional)
- Add to SCM (optional)

Build

To build SYS/BIOS, the top-level bios.mak makefile is used, specifying 2 directories:

- Path to XDC Tools
- Path to GCC toolchain

You could edit bios.mak and set the variables directly or specify via command-line options when invoking make. (Obviously, you could skip the clean step if already cleaned from step above.)

<syntaxhighlight lang="javascript">

```
linux% cd $BASE/A15GccExample/bios_6_35_02_45
linux% make -f bios.mak XDC_INSTALL_DIR=<path-to-xdctools> clean
linux% make -f bios.mak XDC_INSTALL_DIR=<path-to-xdctools> gnu.targets.arm.A15F=<path-to-gcc-arm-cross-compiler>
linux% make -f bios.mak XDC_INSTALL_DIR=<path-to-xdctools> gnu.targets.arm.A15F=<path-to-gcc-arm-cross-compiler> DESTDIR=$BASE/rebuilt-sysbios install-packages
```

</syntaxhighlight>

Note: Use gnu.targets.arm.A8F for Cortex-A8 and gnu.targets.arm.A9F for Cortex-A9.

The last step is optional, it simply copies out the (re)built SYS/BIOS libraries and associated source code to the directory specified by DESTDIR. You can also just use the (re)built libraries in-place in the original SYS/BIOS directory.

Configure SYS/BIOS

SYS/BIOS needs to be *configured* before applications can be compiled and linked with it. Configuration is primarily used for specifying which modules to include and setting default values for the many tunable parameters in SYS/BIOS. The input to the configuration tool, configuro, is a text file with a *.cfg* extension written in JavaScript (ECMAScript). This file can be authored using a text editor or graphically generated using a plugin in CCS/Eclipse. The output of configuration is a set of options for the C compiler to be used when compiling user applications, and a set of linker options for use when linking applications.

For compiling, the output of the configuration process is a set of include options (-I<dir-to-include>) and pre-processor defines (-D). These are contained a file called compiler.opt that can be provided to gcc via the @ option. For example, =gcc -c @cfg-out-dir/compiler.opt file.c=. The include directories are referenced by absolute paths, so you should probably not copy the values into a makefile since they will change based on where SYS/BIOS and XDCtools are installed.

For linking, a list of libraries and object files is produced by configuro and output in a file called linker.cmd. This file should be passed to the linker along with the user's linker command file specifying memory placement and any other linker directives. When using the compiler front-end for linking, the option for specifying the linker command file is -Wl,-T,cfg-out-dir/linker.cmd.

See the SYS/BIOS User's Guide for more details on the configuration process.

Examples of integrating configuro with various toolchains using make is covered here (http://rtsc.eclipse.org/docs-tip/Consuming_Configurable_Content/makefiles).

GCC Compiler and Linker Options

The following compiler options must be used when compiling application code that will be linked with SYS/BIOS libraries.

- -mcpu=cortex-a8 (for Cortex-A8) and -mcpu=cortex-a15 (for Cortex-A15)
- -mabi=aapcs (use ARM Procedure Call Standard)
- -mfpu=neon -mfloat-abi=hard (enables Hardware FP support)

The following linker options must be used when linking application code with SYS/BIOS libraries.

- -nostartfiles (dont link with default compiler startup files)
- -static (fully bound executable, i.e. no shared libraries or no DLLs)
- -Wl,--gc-sections (discard unused code/data sections)
- -Wl,-T,<path-to-cfg-dir>/linker.cmd (SYS/BIOS linker command file)

Build Application with Configuration

In this approach, the SYS/BIOS configuration is built in the same directory as the application(s) using that configuration. This approach keeps the configuratino as a local setting for the application. The basic model remains the same: a SYS/BIOS configuration is linked with the application to form the embedded executable.

The Makefile in the A15GccExample/ directory illustrates the approach, which follows the article referenced above.

```
A15GccExample/
   makefile                - makefile to build configuration and app
   sysbios.cfg             - project-specific configuration source file
   reentrancyTest.c        - example application
```

There are three user-specific variables at the top of the Makefile that can be overridden via command-line arguments to make, or the Makefile can be edited.

After running make, the directory will contain the following:

```
A15GccExample/
   makefile                - makefile to build configuration and app
   sysbios                 - output directory from configuro tool
   sysbios.cfg             - project-specific configuration source file
   sysbios_pa15fg.rov.xs   - ROV (object viewer) file for debugging in CCS
   reentrancyTest.c        - example application
   reentrancyTest.out      - example built applications
```

The Makefile for this example:

<syntaxhighlight lang="javascript">

```
A15TOOLS = /db/toolsrc/library/tools/vendors/linaro/gcc-arm-none-eabi-4_7-2012q4
SYSBIOS = /home/local/Projects/A15GccExample/bios_6_35_02_45
XDCTOOLS = /home/local/Projects/A15GccExample/xdctools_3_25_01_65

XDCPATH = $(SYSBIOS)/packages

CONFIGURO = $(XDCTOOLS)/xs --xdcpath="$(XDCPATH)" xdc.tools.configuro -b config.bld
TARGET = gnu.targets.arm.A15F
PLATFORM = ti.platforms.evmDRA7XX

CONFIG = sysbios

CC = $(A15TOOLS)/bin/arm-none-eabi-gcc
CFLAGS = -Wall -mcpu=cortex-a15 -mabi=aapcs -mapcs -mfpu=neon -mfloat-abi=hard \
       @$(CONFIG)/compiler.opt -O3 -g

LD = $(A15TOOLS)/bin/arm-none-eabi-gcc
LFLAGS = -nostartfiles -static -Wl,--gc-sections -Wl,-T,$(CONFIG)/linker.cmd \
         -Wl,-Map,reentrancyTest.map -mfloat-abi=hard

# Starting with SYS/BIOS 6.40.xx.yy, the targets are shipped with the SYS/BIOS
# product and therefore the C runtime libraries are in the SYS/BIOS product.
# Please use the following path instead if on SYS/BIOS 6.40.xx :
# $(SYSBIOS)/packages/gnu/targets/arm/libs/install-native/arm-none-eabi/lib/fpu
REENTLIBPATH = $(XDCTOOLS)/packages/gnu/targets/arm/libs/install-native/arm-none-eabi/lib/fpu

.PRECIOUS: %/compiler.opt %/linker.cmd

%/compiler.opt %/linker.cmd : %.cfg
    $(CONFIGURO) -c $(A15TOOLS) -t $(TARGET) -p $(PLATFORM) -r release $<
    cp $*/package/cfg/$*_pa15fg.rov.xs .

%.o : %.c

%.o : %.c $(CONFIG)/compiler.opt
    $(CC) $(CFLAGS) -c $<

%.out : %.o $(CONFIG)/linker.cmd
    $(CC) -o $@ $< $(LFLAGS) -lgcc -lc -lm -lrdimon -L$(REENTLIBPATH)

all: reentrancyTest.out

clean:
    -rm -rf *.o *.out *.map *.rov.xs $(CONFIG)
```

</syntaxhighlight>

Note: The same makefile can be used to build an example for Cortex-A8/A9 by simply replacing the target and platform with Cortex-A8/A9's target and platform. The Compiler options should remain the same except -mcpu option.

Migrating existing GCC projects to SYS/BIOS v6.51 or newer

SYS/BIOS v6.51 migrated to GCC v6.3.1 codegen tools and switched from newlib to newlib-nano C runtime library. newlib-nano was selected as it is optimized for embedded applications.

In order to migrate an existing GCC project to use SYS/BIOS v6.51, the following updates need to be made:

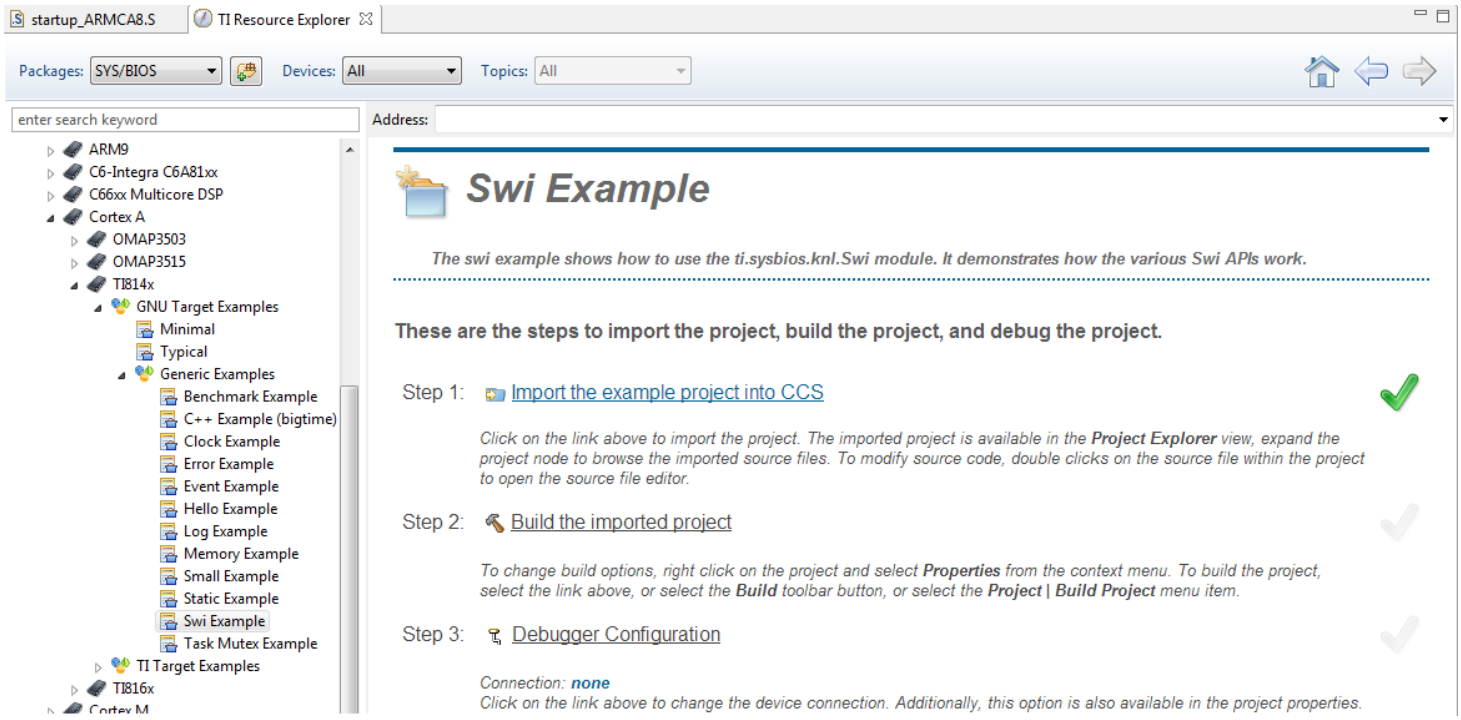
- Point to latest GCC v6 compiler shipped with CCS (GCC v6.3.1 shipped with CCS v7.2).
- Add "<GCC_INSTALL_DIR>/gcc-arm-none-eabi-6-2017-q1-update/arm-none-eabi/include/newlib-nano" to the compiler include path.
- Replace "-L$(SYSBIOS)/packages/gnu/targets/arm/libs/install-native/arm-none-eabi/lib/fpu" with "-L$(SYSBIOS)/packages/gnu/targets/arm/libs/install-native/arm-none-eabi/lib/hard --specs=nano.specs" on the link line. "nano.specs" selects newlib-nano as the C runtime library.
- Rebuild the project.

See this FAQ for a list of limitations with newlib-nano.

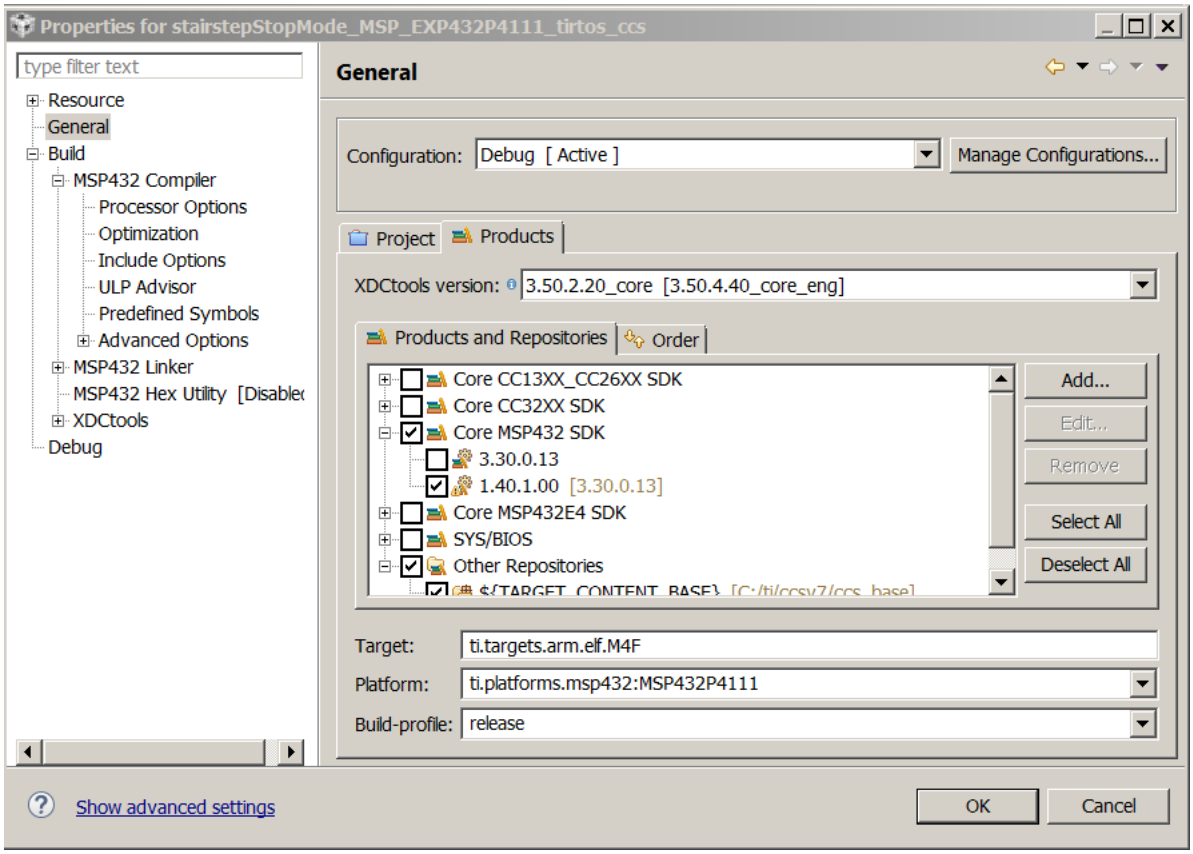
FAQs

How do I enable Semi-Hosting for Cortex-A GNU targets ?

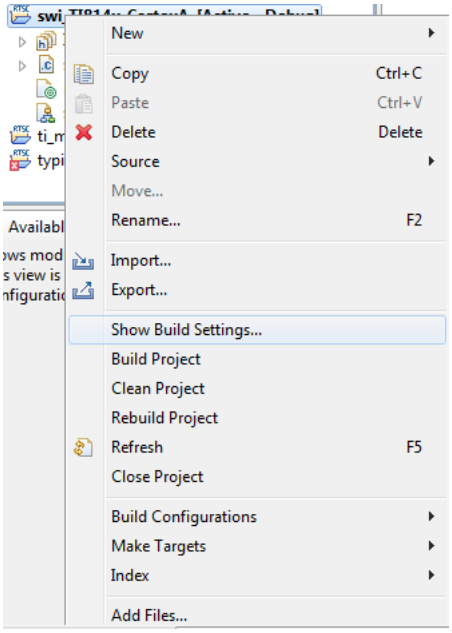
Step1: Import a GNU Example for any Cortex-A device from TI Resource Explorer in CCS. In the below example, a TI814x (Cortex-A8) Swi example for GNU targets is being imported.



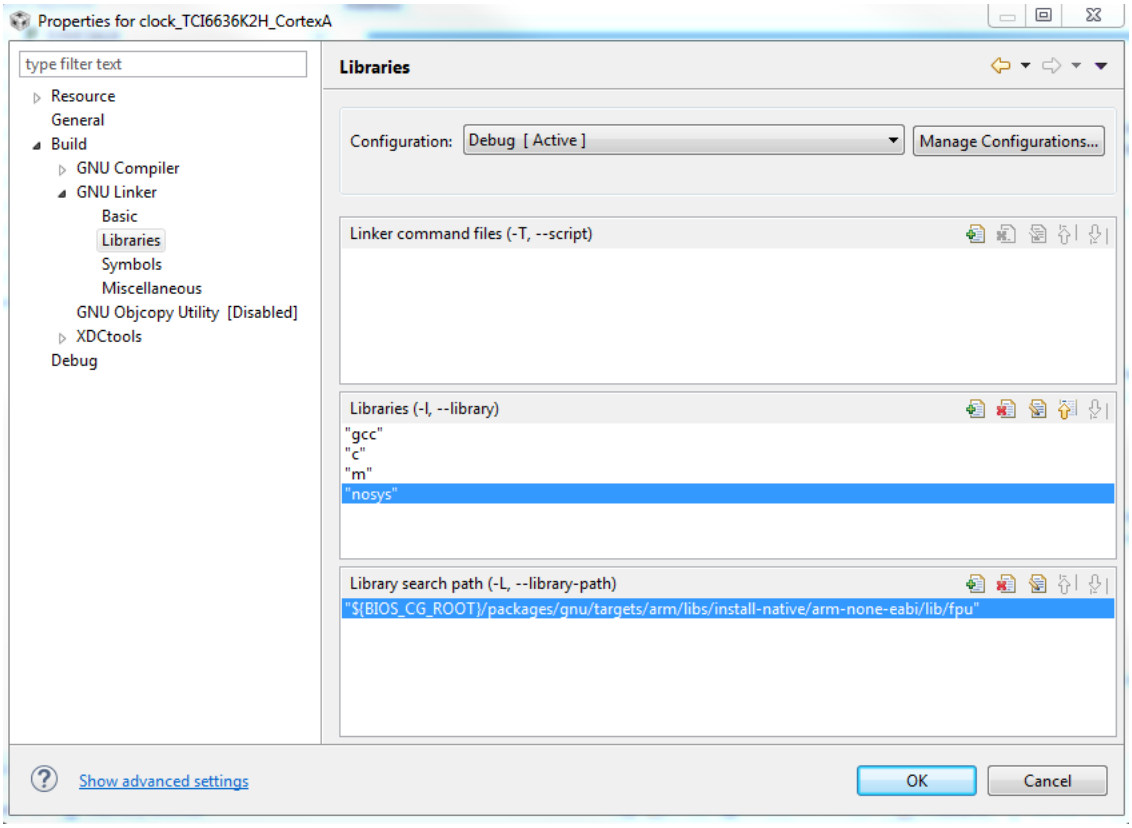
Step2: Click on the **"Products"** tab (**"RTSC"** tab on older CCS) and ensure the latest SYS/BIOS and XDCTools are being used. Semi-Hosting requires **SYS/BIOS 6.35.2.45 or newer** and **XDCTools 3.25.1.65 or newer**. After making the selection click **"OK"**. The example will be imported and will show up in the project explorer window.

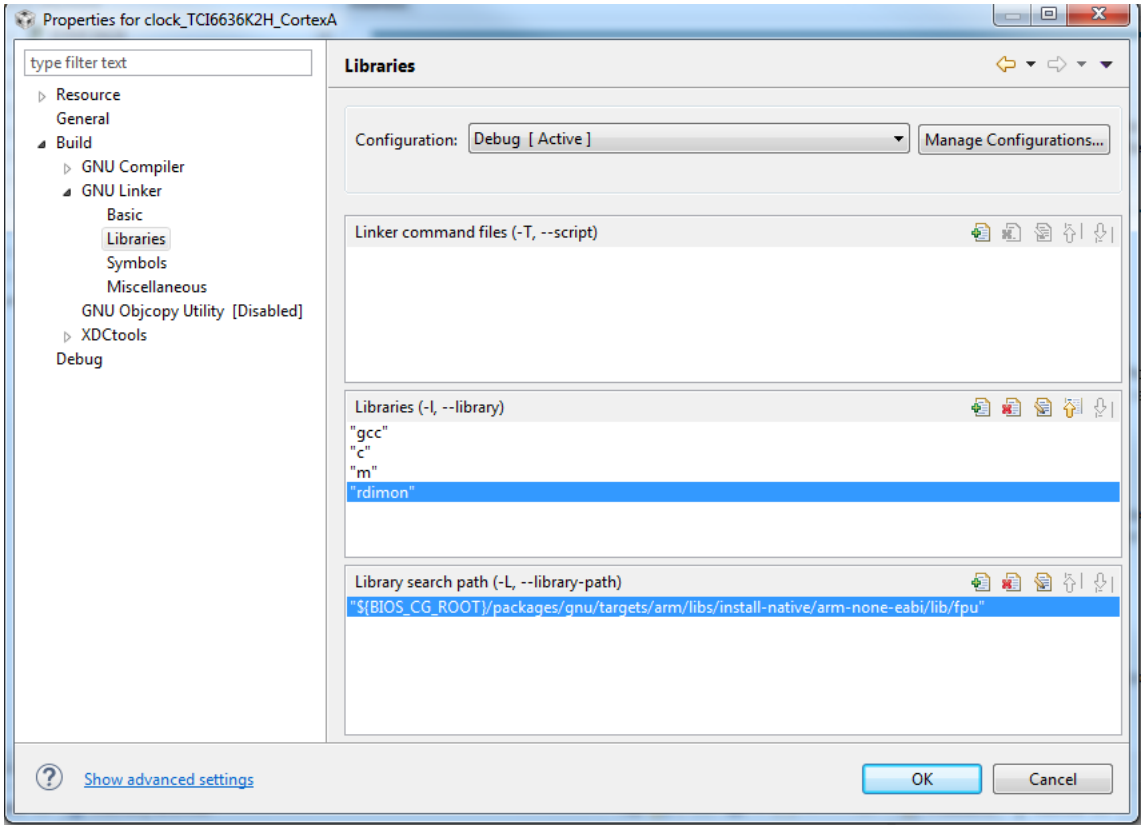


Step3: Open the build settings for this example.



Step4: Add **"rdimon"** library to the **"GNU Linker"** -> **"Libraries"** view. If the **"nosys"** library is already listed in the **"Libraries"** view then replace it with **"rdimon"**. This will cause the application to link with **librdimon.a** library which is a Semi-Hosting enabled BSP library.

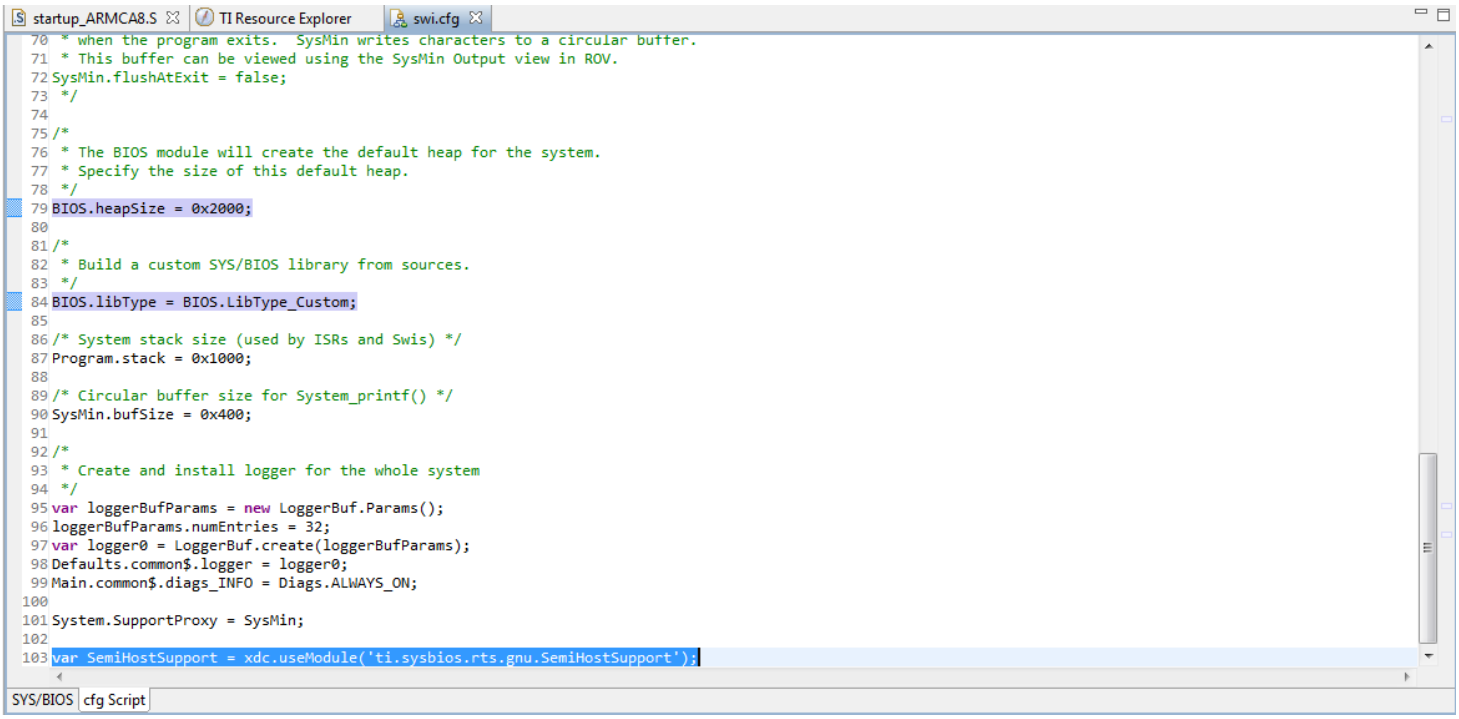




Step5: Open **swi.cfg** and manually edit the config script. Add the following line:

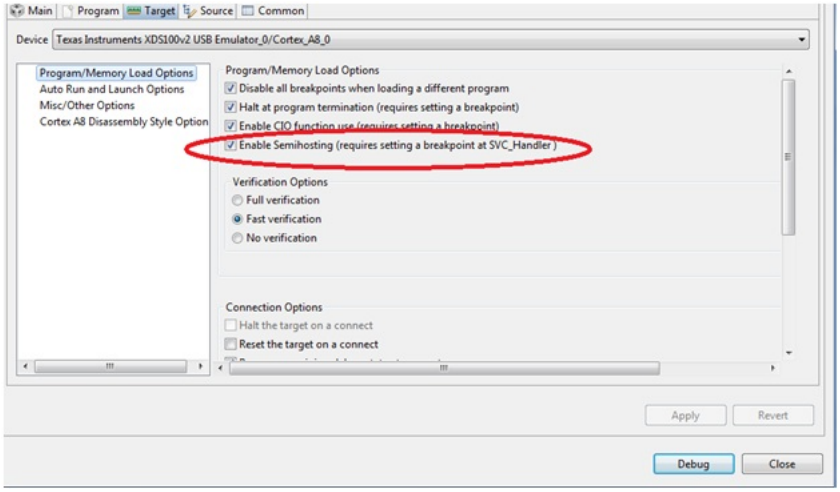
var SemiHostSupport = xdc.useModule('ti.sysbios.rts.gnu.SemiHostSupport');

This module does the required setup (install SVC_Handler and do the required file handle init) to support SemiHosting.



Step6: Build the example.

Step7: Open **"Program/Memory Load Options"** and ensure semihosting is enabled.



That's it! If you now load and run this example, it will use Semi-Hosting to do IO.

What do I need to do to make the C runtime library re-entrant when building SYS/BIOS applications for Cortex-A GNU targets ?

The C runtime APIs are not inherently re-entrant. In order to make them re-entrant, there are 2 options:

1. The user should call the re-entrant version of the API (i.e. API appended with a _r) and pass a re-entrancy structure as an argument, OR
2. The OS implements __getreent() which will return a different re-entrancy structure based on which thread (Task) is currently executing. With this solution, the user can call the standard C runtime APIs (not the _r version) and is guaranteed that the APIs will be re-entrant. This solution however requires re-compiling the C runtime libraries (i.e. libc, libm, etc) provided with the GNU tools. The libraries need to be recompiled with a special _DYNAMIC_REENT_ option. This flag tells the library that the OS will implement a __getreent() function and to use __getreent() to get a handle to the re-entrancy structure. Re-entrancy structure is basically thread local storage for static variables that are required by the C runtime libraries.

SYS/BIOS implements the 2nd solution mentioned above and the re-compiled libraries are distributed as part of the SYS/BIOS product. Based on the build flow you are using, here are the extra steps you need to take at build time:

- package.bld build flow:

If the user is using package.bld, then XDC Tools will ensure the right libraries are picked up from XDC Tools. The user does not have to do anything special, just selecting gnu.targets.arm.M3/M4/M4F/A8F/A9F/A15F target for the build is enough. Please note that by default, the libnosys.a library is linked (does not support IO). In order to support semi-hosting (to make IO work), the following lines need to be added to a config.bld file:

<syntaxhighlight lang="javascript">

```
/* GCC bare metal targets */
var gccArmTargets = xdc.loadPackage('gnu.targets.arm');

gccArmTargets.A15F.bspLib = "rdimon";
```

</syntaxhighlight>

- Using configuro in a makefile based project or building a CCS project:

If using SYS/BIOS 6.51.00 or newer, you need to add "-lgcc -lc -lm -lnosys -L$(SYSBIOS)/packages/gnu/targets/arm/libs/install-native/arm-none-eabi/lib/hard --specs=nano.specs" to the link line. This tells the linker that the libc, libm, ... libraries are present in the SYS/BIOS package in the specified directory and that the application should be built with newlib_nano. If you plan to build a semi-hosted application, replace -lnosys with -lrdimon. For more info on how to enable semi-hosting please refer this FAQ -> How do I enable Semi-Hosting for Cortex-A GNU targets ? (http://processors.wiki.ti.com/index.php/SYS/BIOS_with_GCC_(CortexA)#How_do_I_enable_Semi-Hosting_for_Cortex-A_GNU_targets_.3F)

If using SYS/BIOS 6.40.00 through SYS/BIOS 6.50.01, you need to add "-lgcc -lc -lm -lnosys -L$(SYSBIOS)/packages/gnu/targets/arm/libs/install-native/arm-none-eabi/lib/fpu" to the link line. This tells the linker that the libc, libm, ... libraries are present in the SYS/BIOS package in the specified directory. If you plan to build a semi-hosted application, replace -lnosys with -lrdimon. For more info on how to enable semi-hosting please refer this FAQ -> How do I enable Semi-Hosting for Cortex-A GNU targets ? (http://processors.wiki.ti.com/index.php/SYS/BIOS_with_GCC_(CortexA)#How_do_I_enable_Semi-Hosting_for_Cortex-A_GNU_targets_.3F)

If using an older SYS/BIOS with XDC Tools 3.25 or older, you need to add "-lgcc -lc -lm -lnosys -L$(XDCTOOLS)/packages/gnu/targets/arm/libs/install-native/arm-none-eabi/lib/fpu" to the link line. Starting with SYS/BIOS 6.40, the pre-built C runtime libraries were moved from XDC Tools to SYS/BIOS product.

How does variable initialization work in SYS/BIOS applications built for GNU targets ?

SYS/BIOS applications built using GNU tools use load time initialization i.e. variables are initialized at load time instead of run time. If the application is going to be loaded once but will be reset and run multiple times, the ".data" section should have a different load and run address and be copied into the RAM (run address) during startup. The ".data" section can be loaded to read-only memory like FLASH or some other non-volatile memory available on the device. SYS/BIOS startup routine checks to see if the load and run address of the ".data" section is different. If it is different, it copies the ".data" section contents from the load address to the run address.

On most platforms, the load and run addresses are the same by default. They can be changed through the config script. Here's an example "*.cfg" code snippet to do this:

<syntaxhighlight lang="javascript">

```
Program.sectMap[".data"] = new Program.SectionSpec();

/* Set the load address for .data section */
Program.sectMap[".data"].loadAddress = 0x82000000;

/* Set the run address for .data section */
Program.sectMap[".data"].runAddress = 0x82010000;
```

</syntaxhighlight>

loadAddress & runAddress can be replaced with loadSegment & runSegment in the above example.

How do I change the address where the entry point function (_c_int00) gets placed ?

In SYS/BIOS 6.40.03.39 or newer versions, the entry point function (i.e. _c_int00) is put in its own section called ".c_int00". In order to change the address of this function, the ".c_int00" section needs to be placed at the desired address.

Here's an example showing how to place this section by editing the application's *.cfg file: Program.sectMap[".c_int00"] = new Program.SectionSpec(); Program.sectMap[".c_int00"].loadAddress = 0x80000000; Program.sectMap[".c_int00"].runAddress = 0x80000000;

Alternately, you can create a template file, say MyApp.xdt, which has the section placement code (in GNU linker format) and set Program.sectionsTemplate to this file. This will cause XDC to copy the contents of the template file into SECTIONS {} in the generated linker command file.

- .cfg:

Program.sectionsTemplate = "./MyApp.xdt";

MyApp.xdt: .c_int00 0x80000000 : AT (0x80000000) {*(.c_int00)}

In older versions of SYS/BIOS, the entry point function (i.e. _c_int00) is not put in its own section and therefore cannot be placed. In older releases, a workaround to place the entry point function at a fixed address is to create your own entry point function that internally calls _c_int00 and place it. Here are some code snippets showing how the entry point function can be placed at address 0x80000000:

Step1: Create a new entry point function that calls _c_int00 and put this function in its own section <syntaxhighlight lang="javascript">

```
extern void _c_int00();

void __attribute__ ((section (".myEntryPoint"))) myEntryPoint()
{
    _c_int00();
}
```

</syntaxhighlight>

Step2: Update the *.cfg file to place ".myEntryPoint" section. <syntaxhighlight lang="javascript">

```
// Place the section at address 0x80000000
Program.sectMap[".myEntryPoint"] = new Program.SectionSpec();
Program.sectMap[".myEntryPoint"].loadAddress = 0x80000000;
Program.sectMap[".myEntryPoint"].runAddress = 0x80000000;
```

</syntaxhighlight>

Step3: Build the app with " -e myEntryPoint " linker option. This command line option will override the "ENTRY(_c_int00)" in the generate linker script.

The SYS/BIOS app should now have the myEntryPoint() function placed at 0x80000000.

How do I boot SYS/BIOS using U-Boot ?

The steps to be followed to boot SYS/BIOS using u-boot are similar for different boards/devices. In order to highlight the procedure, here's a step-by-step procedure to boot SYS/BIOS on evmDRA7xx's Cortex-A15:

Step1: Download and install the latest ARM cross-compiler. You can get the latest ARM bare-metal cross compiler from Linaro (https://launchpad.net/gcc-arm-embedded/4.7/4.7-2013-q3-update).

Step2: Clone the mainline u-boot git tree and checkout the tag for the latest release: <syntaxhighlight lang="javascript">

```
$ git clone git://git.denx.de/u-boot.git uboot
$ git checkout v2014.01
$ git checkout -b my_dev_branch
```

</syntaxhighlight>
Step3: Build u-boot for DRA7xx target: <syntaxhighlight lang="javascript">

```
$ export CROSS_COMPILE=arm-linux-gnueabihf-
$ cd $uboot
$ ./MAKEALL dra7xx_evm
```

</syntaxhighlight>
Step4: At this point we have a u-boot.img & MLO (in u-boot tree) targeting DRA7xx evm and are ready to create a SYS/BIOS uimage. Here are the commands to create a SYS/BIOS uimage: <syntaxhighlight lang="javascript">

```
// Using objcopy in the ARM bare-metal tools generate a binary
// out file from the elf executable
$ ~/gcc-arm-none-eabi-4_7-2013q3/bin/arm-none-eabi-objcopy -I elf32-little -O binary <mySysBiosApp executable/out file> sysbios.bin
// Create a SYS/BIOS uimage. The kernel_start_addr is the starting
// address of the application. This can be obtained from the
// application's map file.
// Example: If our test was placing the entire app in EXT RAM at
// address 0x80000000, the kernel_start_addr would be 0x80000000.
//
// The kernel_entry_point_addr is the address of the first
// function that runs when the SYS/BIOS kernel gets control.
// The name of this function is _c_int00 and it's address
// can be obtained from the map file.
$ $uboot/tools/mkimage -A arm -T kernel -C none -a <kernel_start_addr> -e <kernel_entry_point_addr> -d sysbios.bin sysbios.ub
// Create uEnv.txt file to set u-boot environment. We will set
// uenvcmd environment variable to load the kernel from MMC into
// memory and then boot the kernel from memory.
//
// Example uEnv.txt contents:
// uenvcmd=echo "Booting SYS/BIOS kernel from SD card...";setenv loadaddr 0x81000000;fatload mmc 0 ${loadaddr} sysbios.ub;bootm ${loadaddr}
//
// The above uenvcmd command will work for a sys/bios app that is linked
// to address 0x80000000.
```

</syntaxhighlight> It is possible to compress the sysbios.bin before generating a uimage. If that is done, then the -C option to mkimage needs to specify the type of compression used.

Step5: Copy MLO, u-boot.img, uEnv.txt and sysbios.ub to the boot partition of the sdcard.

Step6: Power cycle the evm or hit the reset button and the device should boot from sdcard.

Note: If you do not have a UART driver in your app and want to get the System_printf() messages, you can use SysMin as the System provider and set SysMin.flushAtExit to false. Once the app completes, you can connect to the board via CCS, load the program symbols and look at ROV SysMin view to read the print messages.

Why is System_printf()/printf() not working ?

If you are using semi-hosting then you need to ensure that semi-hosting is enabled and you are linking with the right bsp library. Please read the semihosting (http://processors.wiki.ti.com/index.php/SYS/BIOS_with_GCC_(CortexA)#How_do_I_enable_Semi-Hosting_for_Cortex-A_GNU_targets_.3F) FAQ for more info on how to enable semi-hosting.

If semi-hosting support is setup correctly, it is also possible that the problem is related to a low heap. With C runtime library's re-entrancy support enabled, the first call to printf() from within a task thread will try to allocate a buffer. If the system heap is running very low, the Memory_alloc() for this buffer will fail and raise an error (which will try to print an error msg). But since C runtime needs to allocate the buffer before it can print anything, the error msg print will raise yet another error msg and get stuck in a loop forever. This problem only occurs when the heap is running very low. The solution is to increase the heap size or switch to SysMin (xdc.runtime.SysMin) as the System.SupportProxy with flush on exit disabled.

Should a SYS/BIOS (TI-RTOS kernel) application use SYS/BIOS Cache and MMU APIs or use the Starterware version ?

A SYS/BIOS (TI-RTOS kernel) application should use SYS/BIOS Cache and MMU APIs instead of using the Starterware version. This is because the Starterware Cache/MMU APIs may conflict with SYS/BIOS Cache/MMU startup code.

Cache and MMU are enabled by default (only for targets that support Caches/MMUs) in all SYS/BIOS applications. The MMU translation table is initialized with cache-enabled entries for every memory segment defined in the platform. Cache-disabled entries are also added for the peripheral addresses used by SYS/BIOS (i.e. Timers, Interrupt controller).

It is possible to modify (add/remove) the default MMU table entries at build (config) or run time using setFirstLevelDesc()/setSecondLevelDesc() APIs. Typically, the application would need to add entries for the memory mapped registers of peripherals that are being used in the application. For example, if the app uses UART, the memory mapped regs of UART need to be mapped as non-cacheable device memory in the MMU table. The MMU module cdoc describes the APIs in more detail and also shows examples of how to program the MMU.

I combined a Starterware project into a SYS/BIOS (TI-RTOS kernel) project and am getting a Data Abort exception. What could be the problem ?

In SYS/BIOS (TI-RTOS kernel) applications the hardware's Memory Management Unit (MMU) is enabled by default. This is done so Caching can be enabled. Since the MMU is enabled, one common reason for a data abort is that the driver code in the application is attempting to read/write peripheral memory that has not been mapped. The application needs to map the peripheral memory as strongly ordered in the MMU before accessing it.

SYS/BIOS allows programming the MMU at build or run time. The MMU module cdoc lists all MMU APIs provided by SYS/BIOS. It also includes example C and cfg code for adding custom MMU entries for peripheral memory - A8 MMU cdoc (http://downloads.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/bios/sysbios/6_41_02_41/exports/bios_6_41_02_41/docs/cdoc/ti/sysbios/family/arm/a8/Mmu.html#xdoc-desc) and A15 MMU cdoc (http://downloads.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/bios/sysbios/6_41_02_41/exports/bios_6_41_02_41/docs/cdoc/ti/sysbios/family/arm/a15/Mmu.html#xdoc-desc).

If combining a Starterware project into a SYS/BIOS project, it is likely that the project is making calls to several Starterware MMU and Cache APIs. These APIs may conflict with SYS/BIOS's MMU/Cache management code and not behave as expected. It is recommended to replace all Starterware MMU/Cache APIs with SYS/BIOS MMU/Cache APIs. Please see this FAQ (http://processors.wiki.ti.com/index.php/SYS/BIOS_with_GCC_%28CortexA%29#Should_a_SYS.2FBIOS_.28TI-RTOS_kernel.29_application_use_SYS.2FBIOS_Cache_and_MMU_APIs_or_use_the_Starterware_version_.3F) for more info.

What is the recommended procedure to reload a program on the Cortex-A15s (J6, AM57xx & Keystone2 devices) ?

See Debugging Cortex A15 (http://processors.wiki.ti.com/index.php/Debugging_Cortex_A15) wiki page for more info on this.

Why does the Cache_inv() API flush the cache on Cortex-A15 ?

The invalidate instruction is treated by A15 as a clean/invalidate instruction. Therefore, calls to Cache_inv()/Cache_invAll() will behave like Cache_wbInv()/Cache_wbInvAll() on A15.

In CPU revision r3p0 or newer, ACTLR2 register bit0 control whether data cache clean is treated as a clean and invalidate.

How do I add a 4KB granularity MMU pages on Cortex-A8, Cortex-A9 and Cortex-A15 devices ?

On Cortex-A8 and Cortex-A9 devices, the MMU hardware supports upto 2 levels of translation tables, but, SYS/BIOS only supports the first level translation table. Adding a second level translation table will allow creating smaller granularity (4KB) pages. See this section for more info and an example showing how to add another level of translation table(s).

On Cortex-A15 devices, the MMU hardware supports upto 3 levels of translation tables, but, SYS/BIOS only support 2 levels of translation tables. Adding a third level translation table will allow creating smaller granularity (4KB) pages. See this section for more info and an example showing how to add another level of translation table(s).

How do I add second level MMU translation tables on Cortex-A8 and Cortex-A9 devices ?

Here's some example C code showing how to add a second level translation table with 4KB granularity on Cortex-A8 and Cortex-A9 devices: <syntaxhighlight lang="javascript">

1. include <xdc/std.h>
2. include <xdc/runtime/System.h>

1. include <ti/sysbios/BIOS.h>
2. include <ti/sysbios/family/arm/a8/Mmu.h>

typedef enum {

```
  SecondLevelDesc_Type_FAULT = 0,      /*! Invalid entry */
  SecondLevelDesc_Type_LARGE_PAGE = 1, /*! Large page section descriptor */
  SecondLevelDesc_Type_SMALL_PAGE = 2  /*! Small page section descriptor */
```

} SecondLevelDesc;

typedef struct {

```
  SecondLevelDesc type;   /*! second level descriptor type           */
  Bool  bufferable;       /*! is memory section bufferable           */
  Bool  cacheable;        /*! is memory section cacheable            */
  Bool  shareable;        /*! is memory section shareable            */
  Bool  noexecute;        /*! is memory section not executable       */
  Bool  notGlobal;        /*! not Global bit                         */
  UInt8 accPerm;          /*! access permission bits value 0-7       */
  UInt8 tex;              /*! memory region attr type extension field */
```

} SecondLevelDescAttrs;

UInt32 secondLevelTable[256] __attribute__ ((aligned(4096)));

Void initSecondLevelDescAttrs(SecondLevelDescAttrs *attrs) {

```
  /* Default attributes */
  attrs->type = SecondLevelDesc_Type_SMALL_PAGE;
  attrs->bufferable = FALSE;
  attrs->cacheable = FALSE;
  attrs->shareable = FALSE;
  attrs->noexecute = FALSE;
  attrs->notGlobal = FALSE;
  attrs->accPerm = 3;
  attrs->tex = 1;
```

}

/*

```
* Generate a second level table descriptor using given page base address
* and attributes.
*/
```

UInt32 constructSecondLevelDesc(Ptr pageBaseAddr, SecondLevelDescAttrs *attrs) {

```
  UInt32 desc = 0;
```

```
  /* Determine which kind of descriptor. */
  switch (attrs->type) {
      case SecondLevelDesc_Type_LARGE_PAGE:
          /* Page table descriptor */
          desc = (attrs->type) |
                 ((attrs->bufferable & 0x1) << 2) |
                 ((attrs->cacheable & 0x1) << 3) |
                 ((attrs->accPerm & 0x3) << 4) |
                 ((attrs->accPerm & 0x4) << 7) |
                 ((attrs->shareable & 0x1) << 10) |
                 ((attrs->notGlobal & 0x1) << 11) |
                 ((attrs->tex & 0x7) << 12) |
                 (attrs->noexecute  & 0x15) |
                 ((UInt32)pageBaseAddr & 0xFFFF0000);
          break;
```

```
      /* Section descriptor */
      case SecondLevelDesc_Type_SMALL_PAGE:
          desc = (attrs->type) |
                 (attrs->noexecute  & 0x1) |
                 ((attrs->bufferable & 0x1) << 2) |
                 ((attrs->cacheable & 0x1) << 3) |
                 ((attrs->accPerm & 0x3) << 4) |
                 ((attrs->tex & 0x7) << 6) |
                 ((attrs->accPerm & 0x4) << 7) |
                 ((attrs->shareable & 0x1) << 10) |
                 ((attrs->notGlobal & 0x1) << 11) |
                 ((UInt32)pageBaseAddr & 0xFFFFF000);
          break;
```

```
      default:
          break;
  }
```

```
  return desc;
```

}

/*

```
*  ======== main ========
*/
```

Void main(Int argc, Char * argv[]) {

```
  /*
   * Note: The MMU init code can be moved into a
   *       last function if MMU needs to be setup
   *       earlier.
   */
  UInt idx, baseAddr;
  Mmu_FirstLevelDescAttrs attrs1;
  SecondLevelDescAttrs attrs2;
```

```
  Mmu_disable();
```

```
  /*
   * Initialize first level table attributes.
   */
  Mmu_initDescAttrs(&attrs1);
  attrs1.type = Mmu_FirstLevelDesc_PAGE_TABLE;
```

```
  /*
   * Map virtual address 0x80000000 to second level MMU table.
   */
  Mmu_setFirstLevelDesc((Ptr)0x80000000, (Ptr)(&secondLevelTable[0]), &attrs1);
```

```
  /*
   * Initialize second level table attributes.
   */
  initSecondLevelDescAttrs(&attrs2);
  attrs2.bufferable = TRUE;
  attrs2.cacheable = TRUE;
```

```
  /* Construct second level table */
```

```
  /* ALL SMALL PAGES */
  attrs2.type = SecondLevelDesc_Type_SMALL_PAGE;
  for (baseAddr = 0x80000000, idx = 0; baseAddr < 0x80100000; baseAddr += 0x1000, idx++) {
      secondLevelTable[idx] = constructSecondLevelDesc((Ptr)baseAddr, &attrs2);
  }
```

```
  /* ALL LARGE PAGES */
  //attrs2.type = SecondLevelDesc_Type_LARGE_PAGE;
  //for (baseAddr = 0x80000000, idx = 0; baseAddr < 0x80100000; baseAddr += 0x1000, idx++) {
      /*
       * The first Large page descriptor occurs on a sixteen-word boundary
       * and is then repeated 16 times.
       */
  //    secondLevelTable[idx] = constructSecondLevelDesc((Ptr)baseAddr, &attrs2);
  //}
```

```
  /* It is safe to enable the MMU now */
  Mmu_enable();
```

```
  BIOS_start();
```

} </syntaxhighlight>

How do I add third level MMU translation tables on Cortex-A15 devices ?

Here's some example C code showing how to add a third level translation table with 4KB granularity on Cortex-A15 devices: <syntaxhighlight lang="javascript">

1. include <xdc/std.h>
2. include <xdc/runtime/System.h>

1. include <ti/sysbios/BIOS.h>
2. include <ti/sysbios/family/arm/a15/Mmu.h>

typedef enum {

```
  ThirdLevelDesc_Type_FAULT0 = 0,     /*! Invalid entry */
  ThirdLevelDesc_Type_FAULT1 = 1,     /*! Invalid entry */
  ThirdLevelDesc_Type_FAULT2 = 2,     /*! Invalid entry */
  ThirdLevelDesc_Type_SMALL_PAGE = 3  /*! Page section descriptor */
```

} ThirdLevelDesc;

typedef struct {

```
  ThirdLevelDesc type;    /*! third level descriptor type             */
  Bool  noExecute;        /*! execute-never bit                       */
  Bool  privNoExecute;    /*! privileged execute-never bit            */
  Bool  contiguous;       /*! hint bit indicating 16 adjacent table
                              entries point to contiguos memory       */
  Bool  notGlobal;        /*! not global bit                          */
  Bool  accessFlag;       /*! access flag                             */
```

```
  UInt8 shareable;        /*! shareability field value 0-3           */
  UInt8 accPerm;          /*! access permission bits value 0-3        */
  Bool  nonSecure;        /*! non-secure bit                          */
  UInt8 attrIndx;         /*! stage 1 memory attributes index field for
                              the indicated MAIRn register value 0-7  */
  UInt8 reserved;         /*! Bits[58:55] reserved for software use   */
```

} ThirdLevelDescAttrs;

UInt64 thirdLevelTable[512] __attribute__ ((aligned(4096)));

Void initThirdLevelDescAttrs(ThirdLevelDescAttrs *attrs) {

```
  /* Default attributes */
  attrs->type = ThirdLevelDesc_Type_SMALL_PAGE;
  attrs->noExecute = FALSE;
  attrs->privNoExecute = FALSE;
  attrs->contiguous = FALSE;
  attrs->notGlobal = FALSE;
  attrs->accessFlag = TRUE;
  attrs->accPerm = 0;
  attrs->nonSecure = FALSE;
  attrs->attrIndx = 0;
  attrs->reserved = 0;
```

}

/*

```
* Generate a third level table descriptor using given page base address
* and attributes.
*/
```

UInt64 constructThirdLevelDesc(UInt64 pageBaseAddr, ThirdLevelDescAttrs *attrs) {

```
  UInt64 desc = 0;

  /* Determine which kind of descriptor. */
  switch (attrs->type) {
      /* Section descriptor */
      case ThirdLevelDesc_Type_SMALL_PAGE:
          desc = ((UInt64)attrs->type & 0x3) |
                 ((UInt64)(attrs->attrIndx & 0x7) << 2) |
                 ((UInt64)(attrs->nonSecure & 0x1) << 5) |
                 ((UInt64)(attrs->accPerm & 0x3) << 6) |
                 ((UInt64)(attrs->shareable & 0x3) << 8) |
                 ((UInt64)(attrs->accessFlag & 0x1) << 10) |
                 ((UInt64)(attrs->notGlobal & 0x1) << 11) |
                 ((UInt64)pageBaseAddr & 0xFFFFE00000) |
                 ((UInt64)(attrs->contiguous & 0x1) << 52) |
                 ((UInt64)(attrs->privNoExecute & 0x1) << 53) |
                 ((UInt64)(attrs->noExecute & 0x1) << 54) |
                 ((UInt64)(attrs->reserved & 0xF) << 55);
          break;

      default:
          break;
  }

  return desc;
```

}

/*

```
*  ======== main ========
*/
```

Void main(Int argc, Char * argv[]) {

```
  /*
   * Note: The MMU init code can be moved into a
   *       last function if MMU needs to be setup
   *       earlier.
   */
  UInt idx;
  UInt64 baseAddr;
  Mmu_DescriptorAttrs attrs1;
  ThirdLevelDescAttrs attrs2;

  Mmu_disable();

  /*
   * Initialize first level table attributes.
   */
  Mmu_initDescAttrs(&attrs1);
  attrs1.type = Mmu_DescriptorType_TABLE;

  /*
   * Map 2MB block at virtual address 0x90000000 to third level MMU table.
   */
  Mmu_setSecondLevelDesc((Ptr)0x90000000, (UInt64)(SizeT)(&thirdLevelTable[0]), &attrs1);

  /*
   * Initialize third level table attributes.
   */
  initThirdLevelDescAttrs(&attrs2);
  attrs2.attrIndx = 0;      /* Non-cacheable normal memory */
```

```
    attrs2.shareable = 3;     /* Inner shareable            */
    attrs2.accPerm = 1;       /* RW at any privilege level  */

    /* Construct second level table */

    /* Create SMALL PAGE mapping from 0x90000000 to 0x90200000 (2MB) */
    attrs2.type = ThirdLevelDesc_Type_SMALL_PAGE;
    for (baseAddr = 0x90000000, idx = 0; baseAddr < 0x90200000; baseAddr += 0x1000, idx++) {
        thirdLevelTable[idx] = constructThirdLevelDesc(baseAddr, &attrs2);
    }
```

```
    /* It is safe to enable the MMU now */
    Mmu_enable();
```

```
    BIOS_start();
```

} </syntaxhighlight>

What are the limitations of newlib-nano libc compared to newlib libc ?

SYS/BIOS migrated to using newlib-nano starting SYS/BIOS 6.51.00 release. Newlib-nano is a runtime support C library with a very small footprint that is suited for use with MCUs. Here some of the limitations of using newlib-nano:

- A maximum of 32 functions can be registered with atexit()
- Programs requiring formatted floating-point input/output must explicitly reference the relevant support function during linking. For the scanf() family of routines the support function is "_scanf_float". For the printf() family of routines the support function is "_printf_float".
- Newlib-nano uses a light exit mechanism that skips calling C++ destructors unless the application is linked with "-u atexit" linker option.
- Newlib-nano has C++ exception handling disabled by default.

C++ destructors are not being called in my SYS/BIOS (TI-RTOS Kernel) C++ application ?

SYS/BIOS (TI-RTOS kernel) switched to using newlib-nano starting with v6.51.00 release. newlib-nano uses a light exit mechanism that skips calling C++ destructors unless the application is linked with "-u atexit" linker option.

Why is exception handling not working in my C++ application ?

SYS/BIOS (TI-RTOS kernel) switched to using newlib-nano starting with v6.51.00 release. newlib-nano's libc++ library is built without exception support to optimize the application size. If an exception is thrown by the application, the application will call _abort() and exit.

Training

The "Intro to TI-RTOS Kernel Workshop" is now available. Follow the link below to find out more. The TI-RTOS Kernel Workshop covers the SYS/BIOS operating system available for all TI embedded processors - C28x, MSP430, Tiva-C, C6000 and AM335x (Cortex A-8). You can take a LIVE workshop (scheduled at various sites around the U.S.) or download/stream the videos of each chapter online and watch at your own pace. All of the labs, solutions, powerpoint slides, student guides, installation instructions, lab procedures, etc., are all available to you. The workshop labs run on all MCU platforms and the C6000. Check it out...

Intro to TI-RTOS Kernel Workshop (http://processors.wiki.ti.com/index.php/TI-RTOS_Workshop)

{{

1. switchcategory:MultiCore=

- For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum
- For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum

Please post only comments related to the article **SYS/BIOS with GCC (CortexA)** here.

Keystone=

- For technical support on MultiCore devices, please post your questions in the C6000 MultiCore Forum
- For questions related to the BIOS MultiCore SDK (MCSDK), please use the BIOS Forum

Please post only comments related to the article **SYS/BIOS with GCC (CortexA)** here.

C2000=*For technical support on the C2000 please post your questions on The C2000 Forum. Please post only comments about the article **SYS/BIOS with GCC (CortexA)** here.*

DaVinci=*For technical support on DaVincoplease post your questions on The DaVinci Forum. Please post only comments about the article **SYS/BIOS with GCC (CortexA)** here.*

MSP430=*For technical support on MSP430 please post your questions on The MSP430 Forum. Please post only comments about the article **SYS/BIOS with GCC (CortexA)** here.*

OMAP35x=*For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article **SYS/BIOS with GCC (CortexA)** here.*

OMAPL1=*For technical support on OMAP please post your questions on The OMAP Forum. Please post only comments about the article **SYS/BIOS with GCC (CortexA)** here.*

MAVRK=*For technical support on MAVRK please post your questions on The MAVRK Toolbox Forum. Please post only comments about the article **SYS/BIOS with GCC (CortexA)** here.*

*For technical su please post you questions at http://e2e.ti.cor Please post on comments abo article **SYS/BIO GCC (CortexA***

}}

Links

Processors

- ARM Processors