SITARA™ ARM® PROCESSORS

Boot camp

TEXAS INSTRUMENTS

U-Boot and Linux Kernel Debug using CCSv5

In this session we will cover fundamentals necessary to use CCSv5 and a JTAG to debug a TI SDK-based U-Boot and Linux kernel on an EVM platform.

LAB: http://processors.wiki.ti.com/index.php/Sitara_Linux_Training

July 2012

Creative Commons Attribution-ShareAlike 3.0 (CC BY-SA 3.0)

Pre-work Check List

- ❑ Installed and configured VMWare Player v4 or later
- ❑ Installed Ubuntu 10.04
- ❑ Installed the latest Sitara Linux SDK and CCSv5
- ❑ Within the Sitara Linux SDK, ran the setup.sh (to install required host packages)
- ❑ Using a Sitara EVM, followed the QSG to connect ethernet, serial cables, SD card and 5V power
- ❑ Booted the EVM and noticed the Matrix GUI application launcher on the LCD
- ❑ Pulled the ipaddr of your EVM and ran remote Matrix using a web browser
- ❑ Brought the USB to Serial cable you confirmed on your setup (preferable)

Agenda

- Sitara Linux SDK Development Components
- Example Development Environment
- U-Boot Debug Overview
- U-Boot Debug Lab
- Kernel Debug Overview
- Kernel Debug Lab

Background for this Workshop

- Understand the SPL/U-Boot/Kernel boot process
- Knowledge of the Sitara Linux SDK and that it contains a cross compiler for the target device, CCSv5.1 and the source code for SPL/U-Boot/Kernel
- Have run the setup scripts in the Sitara Linux SDK to configure the target to boot the Linux kernel from tftp.
- Some knowledge of CCSv5
- Techniques presented here are not the only way to do things.

SITARA LINUX SDK COMPONENTS

Where to get the Sitara SDK w/ CCS

- SDK Installer
- CCSv5 Installer

AM335xSDK Product Downloads			
	Title	Description	Size
AM335x SDK Essentials			
	ti-sdk-am335x-evm-05.04.01.00-Linux-x86-Install	AM335x EVM SDK	1414280K
AM335x SDK Optional Addons			
	CCS-5.1.1.00033-Sitara-ARM.tar.gz	Code Composer Studio for Sitara ARM	1087840K
	README.ccs	Code Composer Studio for Sitara ARM README	4K
AM335x SDK Individual Components			
	Sitara Linux SDK Release Notes	Link to Release Notes for Sitara Linux SDK	
	am335x-evm-qsg.pdf	AM335x EVM Quick Start Guide	2316K
	beaglebone-qsg.pdf	BeagleBone Quick Start Guide	176K
	sitara-linuxsdk-sdg-05.04.01.00.pdf	Software Developers Guide	9276K
	Wiki version of Software Developers Guide	Link to the online Software Developers Guide which has the latest content	
	Software Manifest	Software Manifest of Components Inside the SDK	640K
	am335x-evm-sdk-src-05.04.01.00.tar.gz	AM335x SDK PSP Source Code	110840K
	am335x-evm-sdk-bin-05.04.01.00.tar.gz	AM335x SDK prebuilt PSP binaries and root filesystem	236796K
	Download Pinmuxtool	Sitara Pin Mux Configuration Utility	
AM335x SDK Checksums			
	md5sum.txt	MD5 Checksums	4K

- The list of available Sitara Linux SDKs can be found at: http://www.ti.com/tool/linuxezsdk-sitara

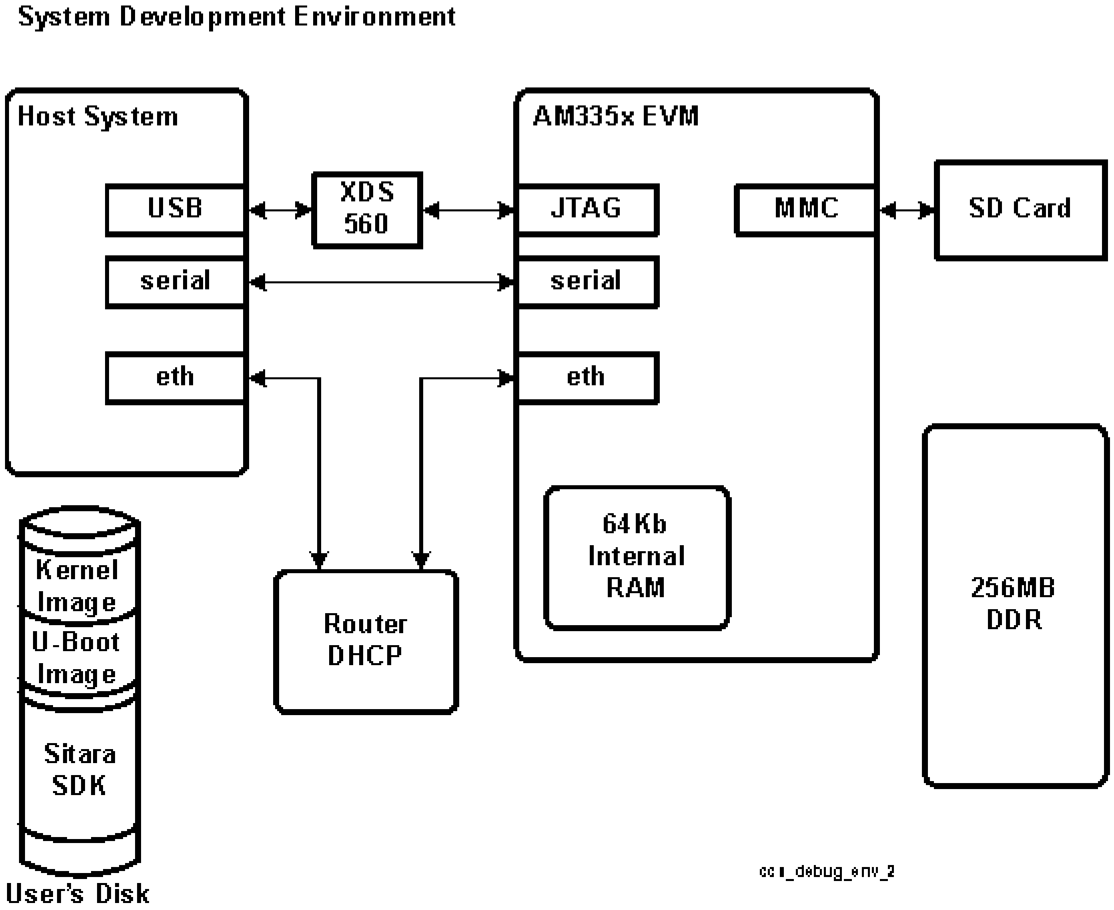
CCS Installation – Key things to know..

- JTAG use requires a License
  - The XDS100v2 can be used without a license
  - You can use a free 90 day evaluation license for all other emulators
- To get JTAG support the CCS installer needs to be run in root mode using “sudo”

EXAMPLE DEVELOPMENT ENVIRONMENT

Example AM335x EVM Debug Environment

- Ubuntu 10.04 LTS
- Sitara Linux SDK
- CCSv5.1
- XDS560v2 USB JTAG
- Serial Console
- Ethernet
- SD Card
  - MLO/U-Boot Pre-builts
  - Root File System installed

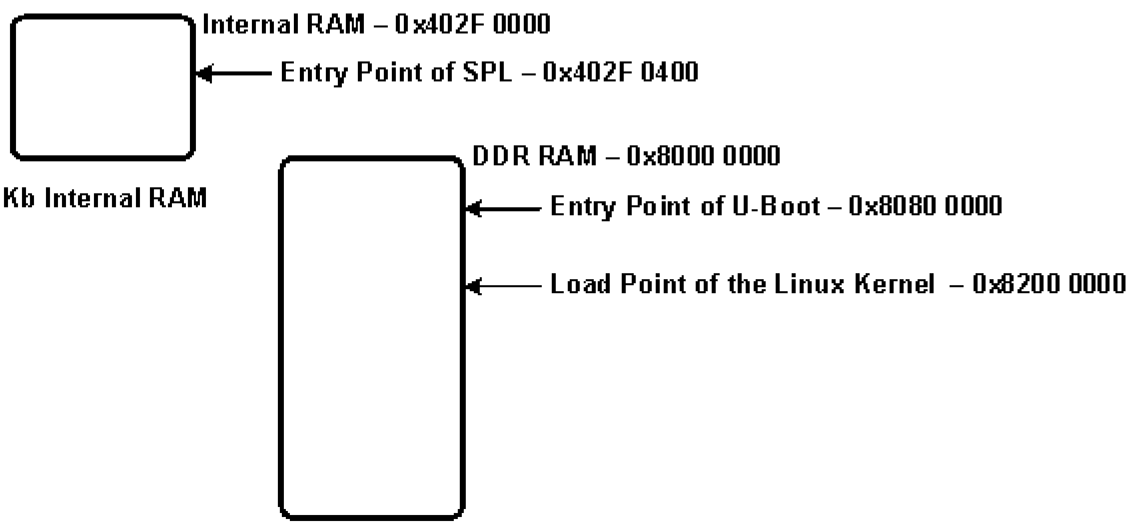


AM335x Image Load Addresses

- These addresses are related to the AM335x, other processors will have different addresses, please refer to respective TRMs. The Addresses here are pulled from the AM335x TRM.

Starting Address of Internal RAM

Reserved	0x402f 0000	0x4020 03FF	64KB	Reserved
SRAM internal	0x402F_0400	0x402F_FFFF		32-bit Ex/R/W[1]



EMIF0 SDRAM	0x8000_0000	0xBFFF_FFFF	1GB	8 /16 bit External Memory (Ex/R/W)[2]

U-BOOT DEBUG OVERVIEW

U-Boot Debug Overview

- Familiarize yourself with the u-boot load address. This can be found in the configuration file (i.e. include/configs/am335x_evm.h) and look for the following variables:
  - For SPL - CONFIG_SPL_TEXT_BASE
  - For U-boot - CONFIG_SYS_TEXT_BASE
- Define a CCS project and point to the source tree within the SDK
  - This will take a couple minutes since CCS will index the u-boot source tree
- Create a target configuration (can specify a gel file)
- Power on the EVM with no SD card installed
- Launch the target configuration
- From CCS connect to the Target, this suspends the target

U-Boot Debug Overview - Cont.

- Switch from THUMB2 to ARM mode
- Load the SPL image
  - Load the binary (bin) image if you are not debugging the SPL on the persistent storage
  - If using persistent storage you do not need to load anything
- Load the U-Boot information
  - Load the ELF image if you are not debugging the u-boot on the persistent storage (i.e. SD card or NAND)
  - Load the symbols only if you are using the u-boot from the persistent storage
- Navigate Source Code and set desired HW Break Point

U-Boot Debug Overview - Cont.

- Depending on how code was loaded:
  - Start target execution from the U-boot load address if your loaded the ELF image
  - Perform system reset from CCS if you loaded the symbold for u-boot in the persistent storage
- These steps will be performed in the Debug Lab and will emphasize that how U-Boot is loaded matters (i.e. whether in SPL context or not)

U-BOOT DEBUG LAB

U-Boot Debug Lab Overview

- Have to build U-Boot to get an binary SPL and ELF U-Boot image to work with
- Load the SPL which has some knowledge that is needed beforehand
- Halt the SPL to allow loading U-Boot
- Load the U-Boot ELF image
- Open the source browser and set a HW break point
- Run the U-Boot binary and you will hit the HW break point

U-Boot Build (Refresher)

- To get an image with Debug Information in it you have to build U-Boot, which is already configured to build with Debug information in it.

- Start with clean configuration

```
make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi- distclean
```

- Configure for EVM

```
make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi- am335x_evm_config
```

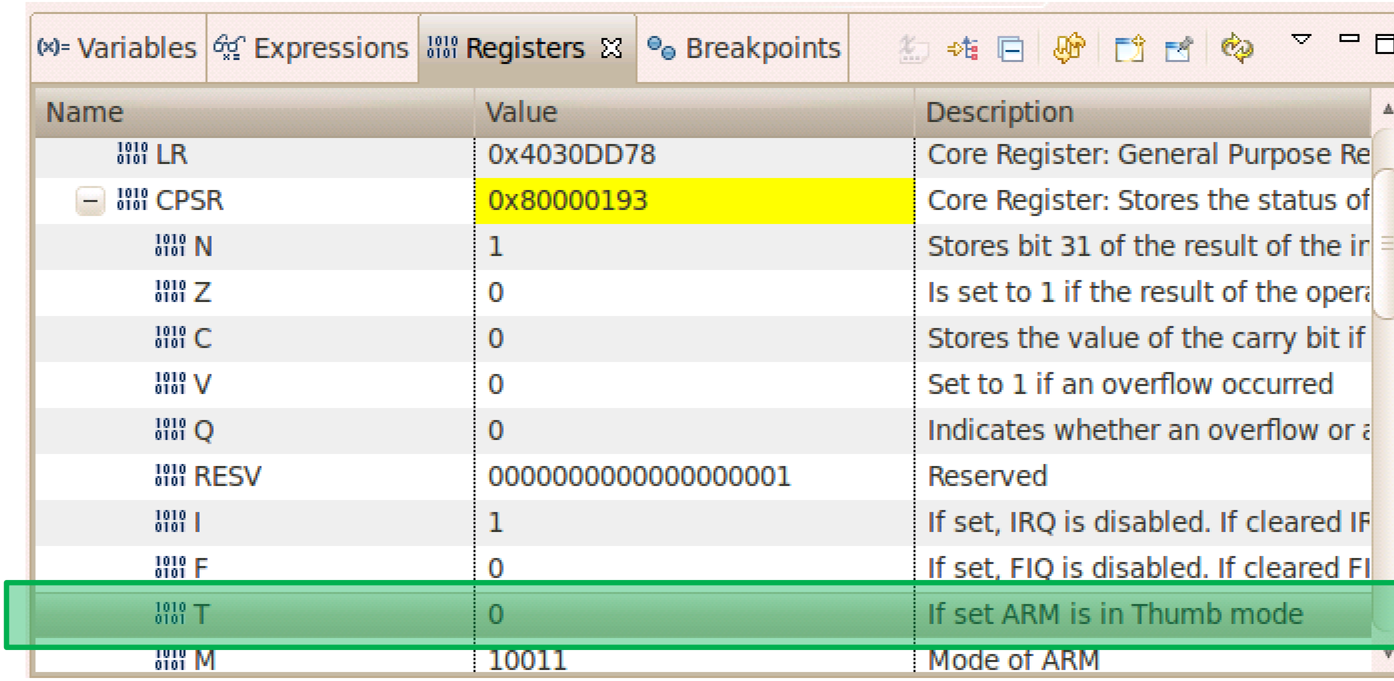
- Build for EVM

```
make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi-
```

- U-Boot ELF Image

```
    17 2012-03-21 12:05 snapshot.commit
  4096 2012-07-06 16:56 spl
 32622 2012-07-06 16:56 System.map
  4096 2012-07-06 16:55 tools
971339 2012-07-06 16:56 u-boot
231324 2012-07-06 16:56 u-boot.bin
231388 2012-07-06 16:56 u-boot.img
   846 2012-07-06 16:56 u-boot.lds
 89562 2012-07-06 16:56 u-boot.map
694052 2012-07-06 16:56 u-boot.srec
```

Changing from Thumb2 to ARM Mode



- ROM Code runs in Thumb2 mode, after loading the SPL binary image and the PC is set to the entry point of SPL, the processor must be set to ARM mode. The reason this has to be done is the ROM code is not being used to jump from ROM to SPL and therefore the BX instruction that the ROM uses to jump to SPL and change the execution mode is not being used.

KERNEL DEBUG OVERVIEW

Linux Kernel Debug Steps using CCS

- Define a CCS project and point to the source tree within the SDK
  - This will take several minutes since CCS will index the kernel source tree
- Create a target configuration
- Power on the EVM with an SD card installed that contains MLO and U-Boot, press any key to stop the auto-boot into Linux
- Launch the target configuration
- From CCS connect to the Target, this suspends the target
- Build the Linux kernel to generate the images needed (vmlinux and uImage)
- Copy the arch/arm/boot/uImage image to the tftpboot directory

Linux Kernel Debug Steps using CCS (cont)

- Load the vmlinux symbols (no code), and now navigate kernel source and set desired HW Break Point
- From CCS resume target execution
- From the serial console type “boot” to tftp the kernel image and start execution.
- The breakpoint will now be hit if it was set on code that will be executed
- This will be performed in the Lab accompanying this lecture.

What expected in a Linux Kernel Debug

- The board developer typically wants to:
  - Look at code they modified such as the board file
  - Halt the system and examine system registers
    - Will need to be aware of Virtual to Physical addressing requirements
- The board developer will not typically be looking at core kernel functions such as the scheduler
  - Kernel Optimization will make single stepping hard to follow, kernel is built with O2.
  - Many of the core functions make heavy use of inline functions which are not handled well by CCS

KERNEL DEBUG LAB

Linux Debug Lab Overview

- The Kernel has to be configured and then built to generate the debug images
- Hardware Breakpoints will be used to stop the kernel during the boot
- While stopped a couple of locations will be examined using the memory browser
- Single stepping will also be performed so the optimization affects can be witnessed.

Kernel Build

- Under General Setup in the Kernel Config
- Select Prompt for development and/or incomplete code/drivers

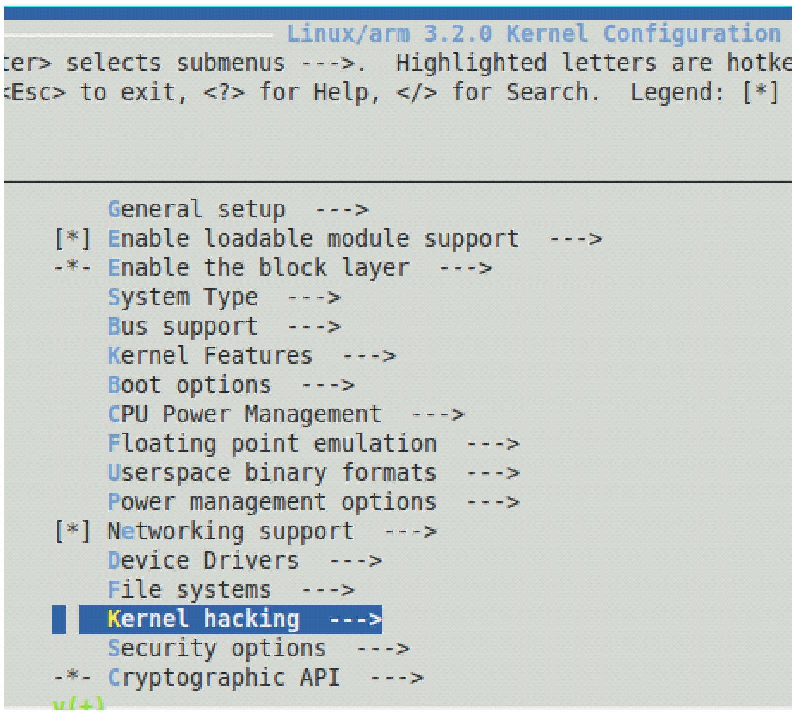
```
                    General setup
> selects submenus --->.  Highlighted letters are hotkeys.  Pressing
;c> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ]

 [*] Prompt for development and/or incomplete code/drivers
 ()  Cross-compiler tool prefix
 ()  Local version - append to kernel release
 [*] Automatically append version information to the version string
     Kernel compression mode (Gzip)  --->
 ((none)) Default hostname
 [*] Support for paging of anonymous memory (swap)
 [*] System V IPC
 [*] POSIX Message Queues
 [*] BSD Process Accounting
 [ ]   BSD Process Accounting version 3 file format
 [ ] open by fhandle syscalls
 [ ] Export task/process statistics through netlink (EXPERIMENTAL)
 [ ] Auditing support
     IRQ subsystem  --->
     RCU Subsystem  --->
 <*> Kernel .config support
 v(+)
```

Kernel Debug Configuration

- Start a Kernel configuration session, here starting with a default configuration

```
make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi- distclean
make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi- tisdk_am335x-evm_defconfig
make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi- menuconfig
```



- Locate the Kernel Hacking entry in the configuration menu

- Note: Use the cursor keys to navigate and the space bar to select the desired kernel option.

Kernel Debug Configuration (cont)

- Under Kernel Hacking enable “Kernel Debugging”

```
Kernel hacking
 navigate the menu.  <Enter> selects submenus --->.  Highlighte
Y> includes, <N> excludes, <M> modularizes features.  Press <Es
for Search.  Legend: [*] built-in  [ ] excluded  <M> module  <

   [*] Show timing information on printks
   (4) Default message log level (1-7)
   [*] Enable __deprecated logic
   [*] Enable __must_check logic
   (1024) Warn for stack frames larger than (needs gcc 4.4)
   [*] Magic SysRq key
   [ ] Strip assembler-generated symbols during link
   [ ] Enable unused/obsolete exported symbols
   [*] Debug Filesystem
   [ ] Run 'make headers_check' when building vmlinux
   [ ] Enable full Section mismatch analysis
   [*] Kernel debugging
   [ ]   Debug shared IRQ handlers (NEW)
   [ ]   Detect Hard and Soft Lockups (NEW)
   [ ] Detect Hung Tasks (NEW)
   [*] Collect scheduler debugging info (NEW)
   [ ] Collect scheduler statistics (NEW)
   [ ] Collect kernel timers statistics (NEW)
   [ ] Debug object operations (NEW)
   [ ] Debug slab memory allocations (NEW)
   v(+)
```

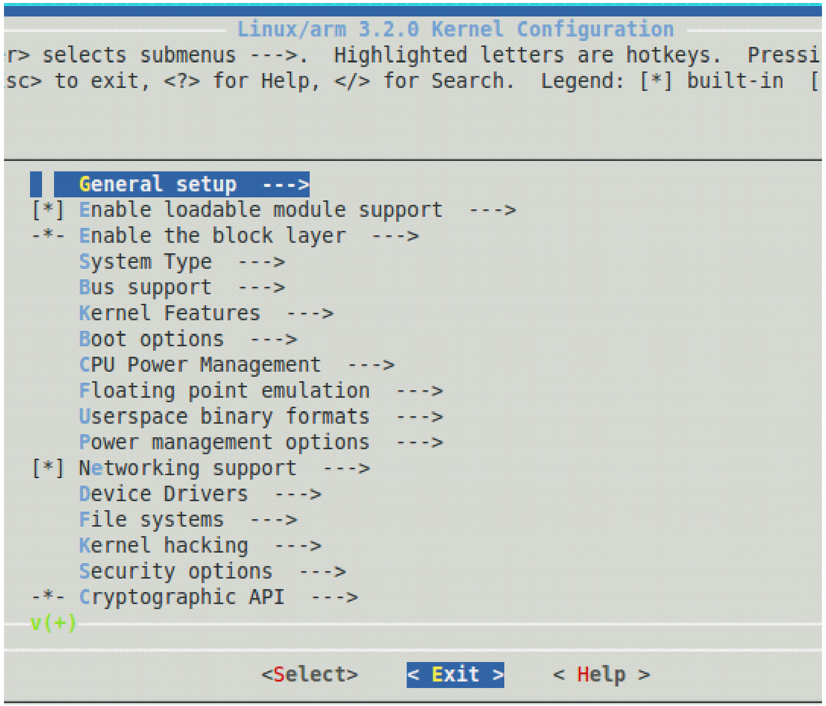
```
Kernel hacking
r> selects submenus --->.  Highlighted letters are hotkeys.
sc> to exit, <?> for Help, </> for Search.  Legend: [*] buil

 ^(-)
   [ ] Enable unused/obsolete exported symbols
   [*] Debug Filesystem
   [ ] Run 'make headers_check' when building vmlinux
   [ ] Enable full Section mismatch analysis
   [ ] Kernel debugging
   [ ] RCU debugging: sparse-based checks for pointer usage
   < > Linux Kernel Dump Test Tool Module
   [ ] Sysctl checks
   [ ] Tracers  --->
   [*] Enable dynamic printk() support
   [ ] Enable debugging of DMA-API usage
   [ ] Perform an atomic64_t self-test at boot
   [ ] Sample kernel code  --->
   < > Test kstrto*() family of functions at runtime
   [ ] Filter access to /dev/mem
   [*] Enable stack unwinding support (EXPERIMENTAL)
   [ ] Verbose user fault messages
   v(+)
```

- Also under Kernel Hacking enable “Enable stack unwinding support”

Kernel Debug Configuration (cont) and Build

- Be sure to save your configuration, you will be prompted to do so.



- Build the kernel with the following line:

```
make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi- uImage
```

- Since Debug information is being compiled this build will take considerable time.

SITARA™ ARM® PROCESSORS
Boot camp

For more Sitara Boot Camp sessions visit:
www.ti.com/sitarabootcamp

THANK YOU!