# AM335X Hands-on Training

Texas Instruments

SITARA™ ARM® PROCESSORS

Boot camp

Texas Instruments

## Giving Linux the Boot

This session covers all facets of the Linux boot process from power up to running a user application beginning with ROM boot loader progressing through secondary program loader, u-boot, kernel and finishing with user-level initialization.

2012

## Agenda

- What you will learn
- Overview of the Boot Process
- Boot Modes
- SPL
- U-Boot
- Kernel
- User Level
- Further Reading

Texas Instruments

## What You Will Learn

- The elements involved in booting into a Linux Kernel
- RBL Stage (ROM Boot Loader)
- Boot ROM Stage (boot mode determination)
- SD, NAND, NOR, USB, Ethernet, Serial .....
- SPL Stage (Secondary Program Loader)
- Reads in U-Boot
- U-Boot Stage

- Processor initialization
- Read in Linux Kernel
- Memory Map of boot process
- Kernel Stage - Kernel Initialization
- Init process
- Memory Map after Linux boots
- User login

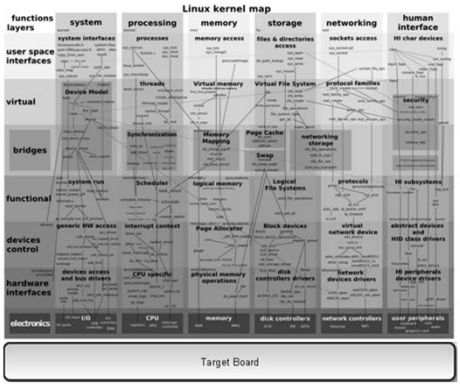
Texas Instruments

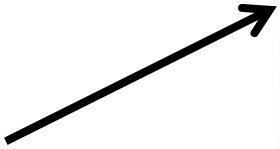
## Go from nothing to Linux OS Environment

•To this full OS

•The different boot stages create the Linux OS “Stack”

•Boot process enables the Linux environment of concurrent user applications, management of HW peripherals, common services.



Target Board

- Starting with bare silicon

•Why are there stages to the boot process????

Texas Instruments

## Overview of Any Bootstrap Process

- Basic components to any processor boot –
- Most processors boot through a chained loading method, each step gains functionality.
- Processor POR jumps to reset vector in ROM after reset line is released
- ROM Code initializes and reads persistent external storage source for bootstrap code into memory (maybe internal or external depends on ROM)
- Processor jumps to load point of read in code, from here it's the role of the bootstrap code to continue to perhaps read in more code to stage the boot process. Or perhaps this bootstrap code is all that is needed to start intended operation.



## Why are there Boot Stages?

- At POR the internal ROM code in the processor knows nothing about the system it is in. Therefore the processor uses pre-defined methods on where to find the boot code that can be accessed with a minimal standard configuration of external interfaces.
- The internal RAM is limited in size and due to that only a portion of the boot process can be read into it. Subsequent stages are enabled from this partial boot from Internal RAM.
- Biggest reason why is due to system configurations that can only be defined during the application design process such as memory DDR types and settings.

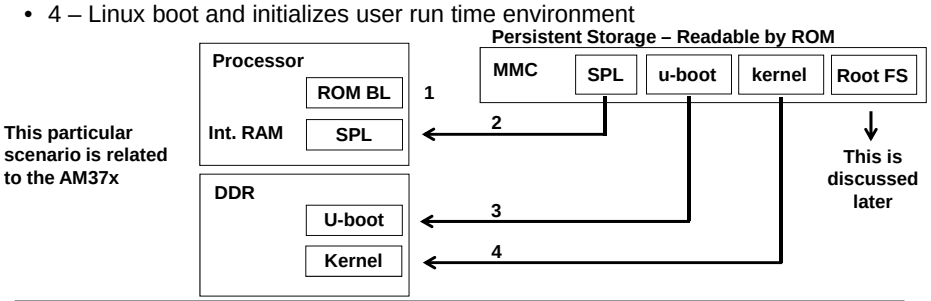
# Linux Boot Process Components

- Components of the Linux Boot Process (Stages)
- RBL – ROM Boot Loader, Contained in the ROM of the Part, minimal capability to initialize the processor and read in from off chip into internal RAM the SPL.
- SPL – Secondary Program Loader, called many different names depending on processor (UBL,Xloader) but is code of a minimal configuration specific to the target board that has the capability to setup the processor to be able to read in the next stage which is U-Boot.
- U-boot – Enables most of the specific processor functionality for the target board and end application to configure the part for booting Linux and to load the kernel image from persistent storage.
- Kernal image – Final stage of the boot process. Kernel initilization, MMU enable, Device Initilization, User Init process and finally user level applications.



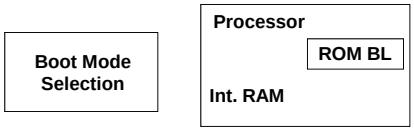
# 4 Stages of the Linux Boot Process

- 4 Stages to Boot Process – ROM, SPL, u-boot, Kernel
- 1 – ROM BL Reads from initialized persistent storage (selected by boot mode) SPL into internal Ram
- 2 – SPL does additional setup and reads from persistent storage the next stage u-boot into DDR
- 3 – u-boot continues the processor setup and reads the kernel into DDR
- 4 – Linux boot and initializes user run time environment



# 4 Stages of the Linux Boot Process - Cont

- (stage 1) ROM
- (stage 2) UBL/X-loader/U-boot SPL
- (stage 3) U-boot
- (stage 4) Kernel



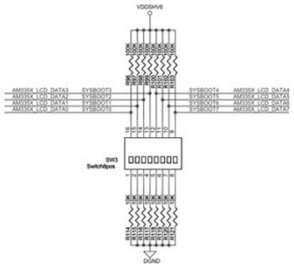

-**Selects which source of SPL**
-**Only used by the ROM BL**

- Each stage adds functionality.
- Boot mode indicates to the ROM BL the order of boot modes to try.
- ROM BL may indicate to SPL where it came from.
- While SPL should use the same method to find U-Boot, U-Boot is free to do as the user configures. For simplicity here, we assume it also uses that mode.



# Overview – Processor Boot Modes

- Boot Modes in TI perspective is strictly related to hardware and about where to read the next stage in the boot process. This is unrelated to traditional UNIX or Linux runlevels.
- Boot Modes supported vary by device so please consult the data sheet. Typical boot modes are NAND, I2C, MMC, SPI, Serial, Ethernet.
- Boot mode pins are read and latched once, typically the pins are muxed with another interface such as the video display lines, input on start and then output after that.
- Each Processor has an app note on this topic



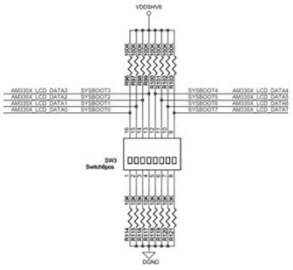

AM335x SYS_BOOT Pin Configuration Selection

Table 26-7. SYSBOOT Configuration Pins[5]

| SYSBOOT[15:14] | SYSBOOT[13:12] | SYSBOOT[11:10] | SYSBOOT[9] | SYSBOOT[8] | SYSBOOT[7:6] | SYSBOOT[5] | SYSBOOT[4:0] | Boot Sequence | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **For all boot modes**: Crystal Frequency | **For all boot modes**: Set to 00b for normal operation | **For XIP boot**: Muxed or non-muxed device. **For NAND boot**: must be 00b | **For NAND and NAND2C boot**: NAND ECC. **For Fast External Boot**: WAIT enable[2] | **For XIP and NAND boot**: Bus width[1] | **For EMAC boot**: PHY mode | **For all boot modes**: CLKOUT1 output enabled/disabled on XDMA_EVENT_INTR0 | | | | | |
| CONTROL STATUS[23:22] | CONTROL STATUS[21:20] | CONTROL STATUS[19:18] | CONTROL STATUS[17] | CONTROL STATUS[16] | CONTROL STATUS[7:6] | CONTROL STATUS[5] | CONTROL STATUS[4:0] | 1st | 2nd | 3rd | 4th |
| | | | | | | | 00000b | Reserved | | | |
| 00b = 19.2MHz<br>01b = 24MHz<br>10b = 25MHz<br>11b = 26MHz | 00b (all other values reserved) | **For XIP boot**: 00b = non-muxed device<br>10b = muxed device<br>x1b = reserved | Don't care for ROM code[5] | 0 = 8-bit device<br>1 = 16-bit device | Don't care for ROM code | 0 = CLKOUT1 disabled<br>1 = CLKOUT1 enabled | 00001b | UART0 | XIP w/ WAIT[2] (MUX2)[3] | MMC0 | SPI0 |
| 00b = 19.2MHz<br>01b = 24MHz<br>10b = 25MHz<br>11b = 26MHz | 00b (all other values reserved) | **For NAND boot**: must be 00b | 0 = ECC done by ROM<br>1 = ECC handled by NAND | 0 = 8-bit device<br>1 = 16-bit device | Don't care for ROM code | 0 = CLKOUT1 disabled<br>1 = CLKOUT1 enabled | 00010b | UART0 | SPI0 | NAND | NAND2C |
| 00b = 19.2MHz<br>01b = 24MHz<br>10b = 25MHz<br>11b = 26MHz | 00b (all other values reserved) | **For XIP boot**: 00b = non-muxed device<br>10b = muxed device<br>x1b = reserved | Don't care for ROM code | 0 = 8-bit device<br>1 = 16-bit device | Don't care for ROM code | 0 = CLKOUT1 disabled<br>1 = CLKOUT1 enabled | 00011b | UART0 | SPI0 | XIP (MUX2)[3] | MMC0 |
| 00b = 19.2MHz<br>01b = 24MHz<br>10b = 25MHz<br>11b = 26MHz | 00b (all other values reserved) | **For XIP boot**: 00b = non-muxed device<br>10b = muxed device<br>x1b = reserved<br>**For NAND boot**: must be 00b | 0 = ECC done by ROM<br>1 = ECC handled by NAND | 0 = 8-bit device<br>1 = 16-bit device | Don't care for ROM code | 0 = CLKOUT1 disabled<br>1 = CLKOUT1 enabled | 00100b | UART0 | XIP w/ WAIT[2] (MUX1)[3] | MMC0 | NAND |

# Overview – Boot Modes

- Some processors support Boot Sequences based on the Boot Mode. This allows the ROM code to handle possible failure modes in case the primary selected persistent storage is not available. Please refer to the appropriate Data Sheet and Technical Reference Manual (TRM) for the part.
  - AM335x/AM37x/AM35x/AM387x/AM389x (Sequencing supported, good for sys dev)



AM335x SYS_BOOT Pin Configuration Selection

Table 26-7. SYSBOOT Configuration Pins[5]

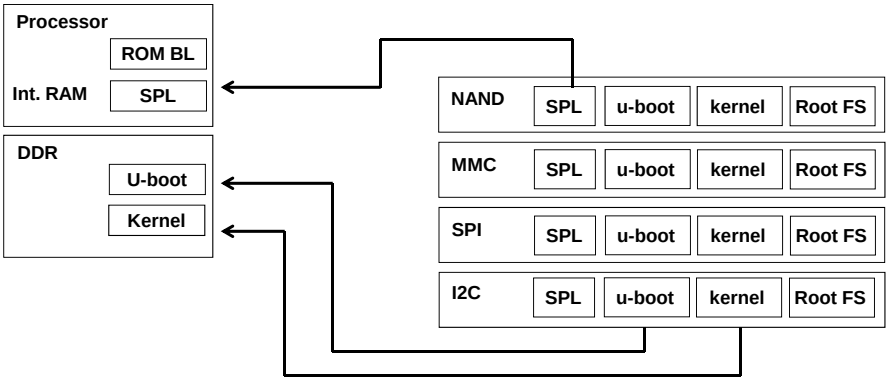
| SYSBOOT[15:14] | SYSBOOT[13:12] | SYSBOOT[11:10] | SYSBOOT[9] | SYSBOOT[8] | SYSBOOT[7:6] | SYSBOOT[5] | SYSBOOT[4:0] | Boot Sequence | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **For all boot modes**: Crystal Frequency | **For all boot modes**: Set to 00b for normal operation | **For XIP boot**: Muxed or non-muxed device<br>**For NAND boot**: must be 00b | **For NAND and NAND2C boot**: NAND ECC<br>**For Fast External Boot**: WAIT enable[2] | **For XIP and NAND boot**: Bus width[1] | **For EMAC boot**: PHY mode | **For all boot modes**: CLKOUT1 output enabled/disabled on XDMA_EVENT_INTR0 | | | | | |
| CONTROL_STATUS[23:22] | CONTROL_STATUS[21:20] | CONTROL_STATUS[19:18] | CONTROL_STATUS[17] | CONTROL_STATUS[16] | CONTROL_STATUS[7:6] | CONTROL_STATUS[5] | CONTROL_STATUS[4:0] | 1st | 2nd | 3rd | 4th |
| | | | | | | | 00000b | Reserved | | | |
| 00b = 19.2MHz<br>01b = 24MHz<br>10b = 25MHz<br>11b = 26MHz | 00b<br>(all other values reserved) | **For XIP boot**:<br>00b = non-muxed device<br>10b = muxed device<br>x1b = reserved | Don't care for ROM code[5] | 0 = 8-bit device<br>1 = 16-bit device | Don't care for ROM code | 0 = CLKOUT1 disabled<br>1 = CLKOUT1 enabled | 00001b | UART0 | XIP w/ WAIT[2] (MUX2)[3] | MMC0 | SPI0 |
| 00b = 19.2MHz<br>01b = 24MHz<br>10b = 25MHz<br>11b = 26MHz | 00b<br>(all other values reserved) | **For NAND boot**: must be 00b | 0 = ECC done by ROM<br>1 = ECC handled by NAND | 0 = 8-bit device<br>1 = 16-bit device | Don't care for ROM code | 0 = CLKOUT1 disabled<br>1 = CLKOUT1 enabled | 00010b | UART0 | SPI0 | NAND | NAND2C |
| 00b = 19.2MHz<br>01b = 24MHz<br>10b = 25MHz<br>11b = 26MHz | 00b<br>(all other values reserved) | **For XIP boot**:<br>00b = non-muxed device<br>10b = muxed device<br>x1b = reserved | Don't care for ROM code | 0 = 8-bit device<br>1 = 16-bit device | Don't care for ROM code | 0 = CLKOUT1 disabled<br>1 = CLKOUT1 enabled | 00011b | UART0 | SPI0 | XIP (MUX2)[3] | MMC0 |
| 00b = 19.2MHz<br>01b = 24MHz<br>10b = 25MHz<br>11b = 26MHz | 00b<br>(all other values reserved) | **For XIP boot**:<br>00b = non-muxed device<br>10b = muxed device<br>x1b = reserved<br>**For NAND boot**: must be 00b | 0 = ECC done by ROM<br>1 = ECC handled by NAND | 0 = 8-bit device<br>1 = 16-bit device | Don't care for ROM code | 0 = CLKOUT1 disabled<br>1 = CLKOUT1 enabled | 00100b | UART0 | XIP w/ WAIT[2] (MUX1) | MMC0 | NAND |



# Persistent Storage based Boot Modes

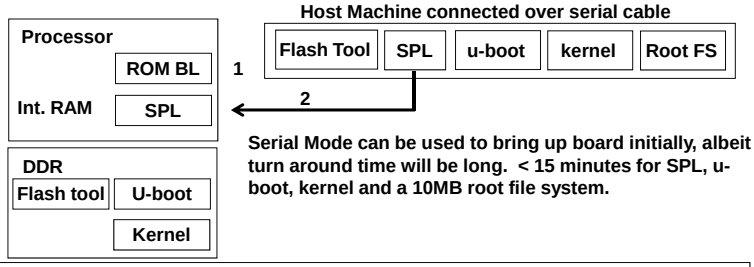
- **Booting from persistent storage**
- This is the most typical workflow.
- Exact location and format of each chunk is media and processor specific
  - ROM BL sets requirements for how SPL "looks" so it can tell if it's found a programmed device or not.

**Persistent Storage – Readable by ROM**



# Connectivity Based Boot Modes

- Booting over Serial Interface
  - Typically used for initial programming of persistent storage such as NAND or SPI when other interfaces aren't available or practical.
- After loading an SPL that sets up DDR a flash tool could be loaded for programming NAND, SPI, etc.
  - In this case the ROM BL will try and initiate a serial download (often X or Y modem based) and if that succeeds, execute what was downloaded.
    - SPL continues to read in next stage (u-boot) over the serial interface
    - u-boot may also continue to load the kernel and root filesystem over serial




**Serial Mode can be used to bring up board initially, albeit turn around time will be long. < 15 minutes for SPL, u-boot, kernel and a 10MB root file system.**



# But where in memory did it go?

- Processors define a memory map that says where both internal memory and DDR are mapped to
- SPL, U-Boot and the Linux Kernel are all statically linked to start running at specific locations within this map.
  - Once running they may relocate themselves elsewhere within DDR (once initialized).
  - Different boot modes may require different link addresses.
  - For more details about the map for a given processor please refer to the TRM

# U-Boot Overview

- Monolithic code image
- Runs processor in physical or a single address space
- Enables clocking, sets up some of the pin mux settings
- Reads in Kernel image (uImage)
- Jumps to load address pointed to in uImage header
- What are environment variables and how are they used
  - Default environment variables and how they are used
- Passes Kernel Command Line to Kernel
  - ATAGs (Memory, Command Line, etc)
  - Flattened Device Trees (FTDs) in the future
- Debugging capabilities (just mentioning, not used during boot process)



# U-Boot Environment Variables

**U-boot example default u-boot environment print out (printenv)**

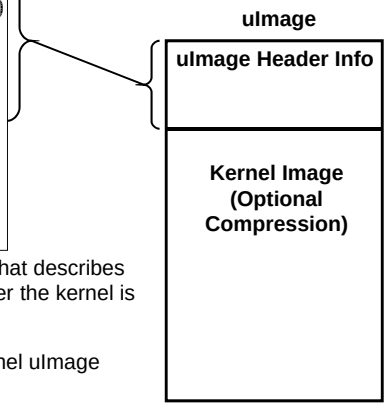
```
U-Boot > printenv
baudrate=115200
bootargs_defaults=setenv bootargs console=${console} ${optargs}
bootcmd=if mmc rescan; then echo SD/MMC found on device ${mmc_dev};if run
loadbootenv; then echo Loaded environment from ${bootenv};run importbootenv;fi;if test -n
$uenvcmd; then echo Running uenvcmd ...;run uenvcmd;fi;if run mmc_load_uimage; then run
mmc_args;bootm ${kloadaddr};fi;fi;run nand_boot;
bootdelay=3
bootenv=uEnv.txt
bootfile=uImage
console=ttyO0,115200n8
kloadaddr=0x80007fc0
loadaddr=0x82000000
```

- Example printout shows boot arguments and boot command.
- Other u-boot environment definitions telling u-boot what persistent storage to read for the kernel and what parameters that need to be passed to the kernel and where to load in memory. Advance knowledge is that the user has to know the kernel load address and where u-boot is loaded, can use the iminfo command after the uimage is loaded somewhere in memory, besides on top of u-boot

# Kernel uImage – What is it?

**U-boot partial console output**

```
## Booting kernel from Legacy Image at 80007fc0 ...
   Image Name:   Linux-3.2.0
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    3163488 Bytes = 3 MiB
   Load Address: 80008000
   Entry Point:  80008000
   Verifying Checksum ... OK
   XIP Kernel Image ... OK
OK

Starting kernel ...

Uncompressing Linux... done, booting the kernel.
```

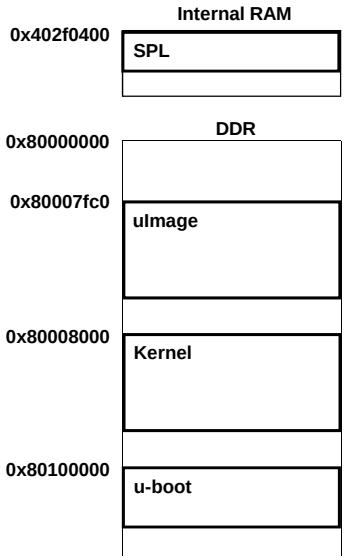



- uImage is a kernel image "wrapped" with a header that describes among other things load point entry point and whether the kernel is compressed of not.
- mkimage utility from u-boot is used to make the kernel uImage during kernel build process
- After the "boot the kernel" message appears on the console u-boot is no longer used.



# Memory Map of Boot Process (AM335x)

- SPL is loaded into internal RAM
- U-Boot is loaded first into DDR
- U-Boot then loads the uImage from where directed
- U-Boot performs a checksum and then relocates (if required) the Kernel found in the uImage to the load address
- U-Boot then jumps to the entry address specified in the uImage header, linux boot starts at this point
- Please note the addresses used here are for reference only and do not apply to all devices



# Kernel Command Line

- Need to define a few required items such as console port and where the root filesystem is located. Please note the kernel command line in the box below. The command line is printed out in the first few lines as the kernel boots.

```
Linux version 3.2.0 (root@ubuntu) (gcc version 4.5.3 20110311 (prerelease) (GCC) ) #1 Tue Aug 28
18:43:59 PDT 2012
CPU: ARMv7 Processor [413fc082] revision 2 (ARMv7), cr=10c53c7d
CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
Machine: am335xevm
Memory policy: ECC disabled, Data cache writeback
AM335X ES1.0 (sgx neon )
Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 65024
Kernel command line: console=ttyO0,115200n8 root=/dev/mmcblk0p2 rw rootfstype=ext3 rootwait
ip=none
```

•Optional Command Line

–Specific peripheral initialization



# Kernel Command Line NFS example

- NFS boot command line (from below)

–console=ttyO0,115200n8

•Where to send console output and take user input, external serial terminal hooks up to this port

–root=/dev/nfs

•What device interface is the root file system mounted on

–nfsroot=192.168.1.166:/home/user/ti-sdk-am335x-evm-05.04.01.00/targetNFS

•Tells the kernel what external host has the Root FS, requires setup on external host

–ip=dhcp

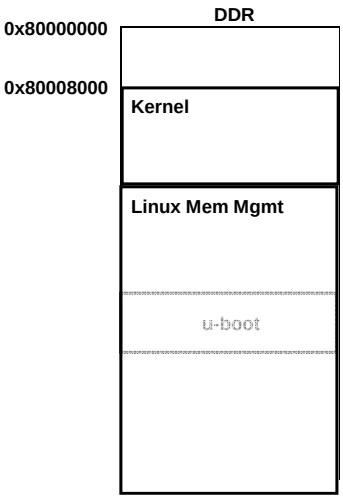
•tells kernel method to configure IP addresses and set up the IP routing table. Here the kernel is being to use DHCP

```
Linux version 3.2.0 (root@stevenliu-laptop) (gcc version 4.5.3 20110311 (prerelease) (GCC) ) #1 Mon May 28 13:38:48 CST
CPU: ARMv7 Processor [413fc082] revision 2 (ARMv7), cr=10c53c7d
CPU: PIPT / VIPT nonaliasing data cache, VIPT aliasing instruction cache
Machine: am335xevm
Memory policy: ECC disabled, Data cache writeback
AM335X ES1.0 (sgx neon )
Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 65024
Kernel command line: console=ttyO0,115200n8 root=/dev/nfs nfsroot=192.168.1.166:/home/userti-sdk-am335x-evm-
05.04.01.00/targetNFS,nolock rw ip=dhcp=/dev/nfs rw ip=dhcp
```

**More Info on configuration for NFS can be found here at kernel.org**
**http://www.kernel.org/doc/Documentation/filesystems/nfs/nfsroot.txt**

## Overview – Kernel Memory Usage

- Kernel boot process begins
- Kernel initializes MMU and takes over DDR. Overwriting anything previously there such as u-boot.
- Look at virtual memory layout printed out by kernel



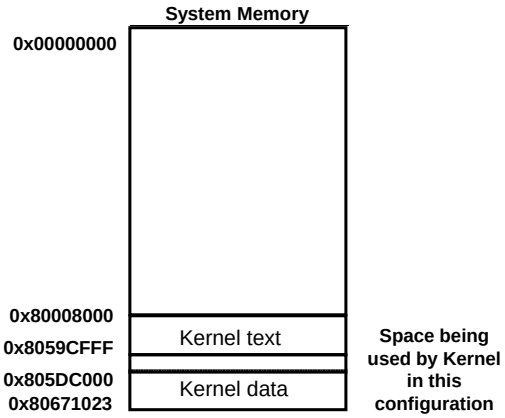



## Linux Kernel Start

- Performs low level system setup
- Initializes the virtual memory used the kernel and applications
- Perform Device Initialization, interrupt handler setup, Idle Task and the scheduler
- Calls the init task which sets up user space context

## Memory Map from User Application Perspective

- Linux creates a virtual memory space for each user application as it is loaded. Each User Application loaded sees a map based from a start address of 0x00000000.
- User Apps cannot write to physical addresses listed in the data sheet such IO ports, EMIF, or DDR without causing a segmentation fault.
- To see how much memory is available in a system type free, this is total space of allowed for executable programs, provided swap space is disabled.




```
root@am335x-evm:~# free
             total       used       free     shared    buffers
Mem:        253504     103932     149572          0       2968
Swap:            0          0          0
Total:      253504     103932     149572
```

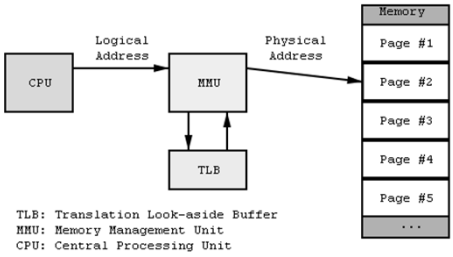
TEXAS INSTRUMENTS

## Virtual Memory Map

- These are components that make up a virtual memory map system

–CPU/MMU/Memory

- All logical address go through an MMU to be mapped to a physical address
- This allows run time link and load of an application
- Once the processor is running in a virtual mode, the memory map can no longer be considered contiguous




•Want to re-iterate the point of not writing/reading physical addresses in the data sheet of the part, this will terminate the application with a segmentation fault.

**Please Note this picture was taken from Wikipedia discussion on virtual memory**
**http://en.wikipedia.org/wiki/File:MMU_principle.png**

TEXAS INSTRUMENTS

# Linux Virtual Memory (.....Review....)

- Linux memory map
- Total address range supported by the part is no longer relevant, MMU creates its own memory map
- The actual memory on the board is hidden and managed by the OS virtual memory manager.
  - You can use the /dev/mem interface with the devmem2 application to read and write physical memory addresses.
  - Some drivers may offer an mmap functionality for their buffers and registers as well.
- All apps are compiled to the same starting address
- Linux sets up a process for an app so they all load to same address, this is possible with an MMU that sets up a virtual memory map
- Every Application sees the same memory map
- Once virtual, the memory can no longer be considered contiguous meaning a physical addressed DMA cannot be used.



# Linux Memory Info Table Dump

- Mem Info Table Dump
- Board has 256MB on it
- Note that the mem total is less than 256M, this difference is where the kernel is stored.
- Only usage described here and not location in the virtual memory map

```
root@am335x-evm:~# cat /proc/meminfo
MemTotal:         253504 kB
MemFree:          149772 kB
Buffers:            2984 kB
Cached:            74148 kB
SwapCached:            0 kB
Active:            21084 kB
Inactive:          68756 kB
Active(anon):      13192 kB
Inactive(anon):      300 kB
Active(file):       7892 kB
Inactive(file):    68456 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:             0 kB
SwapFree:              0 kB
Dirty:                 4 kB
Writeback:             0 kB
AnonPages:         12736 kB
Mapped:            21272 kB
Shmem:               784 kB
Slab:               6488 kB
SReclaimable:       2976 kB
SUnreclaim:         3512 kB
KernelStack:        1104 kB
PageTables:          660 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:      126752 kB
Committed_AS:      60560 kB
VmallocTotal:     761856 kB
VmallocUsed:       25676 kB
VmallocChunk:     634876 kB
```

•Link that describes the components of meminfo

–http://www.linuxweblog.com/meminfo

•Google “meminfo explained”

# System Physical Memory Map

- The physical memory map that has been registered on a platform.
- Look at the data sheet for the part and see how these addresses match up.

–AM 335x Example

•Has a serial interface (UART 0) starting at 0x44e09000

•Has DDR starting at 0x80000000

•These are peripherals specific to this particular part.

```
root@am335x-evm:~# cat /proc/iomem
00000000-00000000 : omap2-nand.0
08000000-08000003 : omap2-nand
44e05000-44e053ff : omap_timer.0
  44e05000-44e053ff : omap_timer
44e07000-44e07fff : omap_gpio.0
44e09000-44e0afff : omap_uart.0
  44e09000-44e0afff : omap_uart
44e0b000-44e0bfff : omap_i2c.1
  44e0b000-44e0bfff : omap_i2c
44e0d000-44e0efff : tsc
  44e0d000-44e0efff : tsc
44e31000-44e313ff : omap_timer.1
  44e31000-44e313ff : omap_timer
44e35000-44e35fff : omap_wdt
  44e35000-44e35fff : omap_wdt
44e37000-44e37fff : smartreflex0
44e39000-44e39fff : smartreflex1
44e3e000-44e3efff : omap_rtc
  44e3e000-44e3efff : omap_rtc
47400000-4740ffff : usbss
47401000-474017ff : musb0
  47401000-474017ff : musb0
47401800-47401fff : musb1
  47401800-47401fff : musb1
47810100-478200ff : omap_hsmmc.2
  47810100-478200ff : omap_hsmmc
48022000-48023fff : omap_uart.1
  48022000-48023fff : omap_uart
48024000-48025fff : omap_uart.2
  48024000-48025fff : omap_uart
4802a000-4802afff : omap_i2c.2
  4802a000-4802afff : omap_i2c
48030100-480304ff : omap2_mcspi.1
  48030100-480304ff : omap2_mcspi.1
```

•Read this article about how to use this memory interface.

– http://lwn.net/Articles/102232/

•Google “iomem linux explained”



# Kernel Root File System

- Kernel Must Have a Root File System defined in the kernel command line
- Must be of a certain format in terms of directories contained
- Can be sourced from various device interfaces, examples are:

–RAM Disk : /dev/ram0

–MMC : /dev/mmcblk0p2

–HDA : /dev/hda1

–NFS : /dev/nfs

- Acronyms Explained

–HDA – The master device on the first ATA channel,

–NFS – Network File System

–MMC – This is the MMC/SD interface or Multi-Media Card/Secure Digital

- Link to Overview of Linux Root File Systems

–http://www.linux-arm.org/LinuxFilesystem/WebHome

## Linux Init Process

- The last step of the Linux Kernel boot process is to call the user space initilization function “init”

–This is one call of several made by the kernel looking for the user space init function - run_init_process("/sbin/init"); (in init/main.c)

- Is the first User Space application to run that setups the user environment and allows user interaction such as login. Below is a ps (process status) dump from a console, init typically has a PID of 1, first process started.

```
root@am335x-evm:~# ps
  PID USER       VSZ STAT COMMAND
    1 root      1628 S    init [5]
```

•/etc/inittab is the script that contains instructions for the init process that sets up and calls other scripts as part of the user space initialization

–Sets the run level, this in turn sets which init scripts based on run level are run

–*Identifier:RunLevel:Action:Command*

–Based on sysv5init scheme

Texas Instruments

# Q & A

Texas Instruments

# Thank you!

SITARA™ ARM® PROCESSORS

Boot camp

Texas Instruments

## ARM based multimedia using GStreamer & FFmpeg

In this session we will discuss open-source multimedia codecs for ARM processors, the capability of the NEON coprocessor to accelerate multimedia. We will also introduce GStreamer, an open-source pipeline-based multimedia framework, and the FFmpeg codec libs.

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

July 2012

## Agenda

- Overview
  - Multimedia on Cortex-A8
  - NEON support in opensource community
- Example Applications
  - SDK codec portfolio
- SDK multimedia framework
  - Gstreamer – FFmpeg/Libav
  - NEON ecosystem
  - Performance and Benchmark
- Software components & dependencies
- References
- Support
- Lab

Texas Instruments

## Pre-work check list

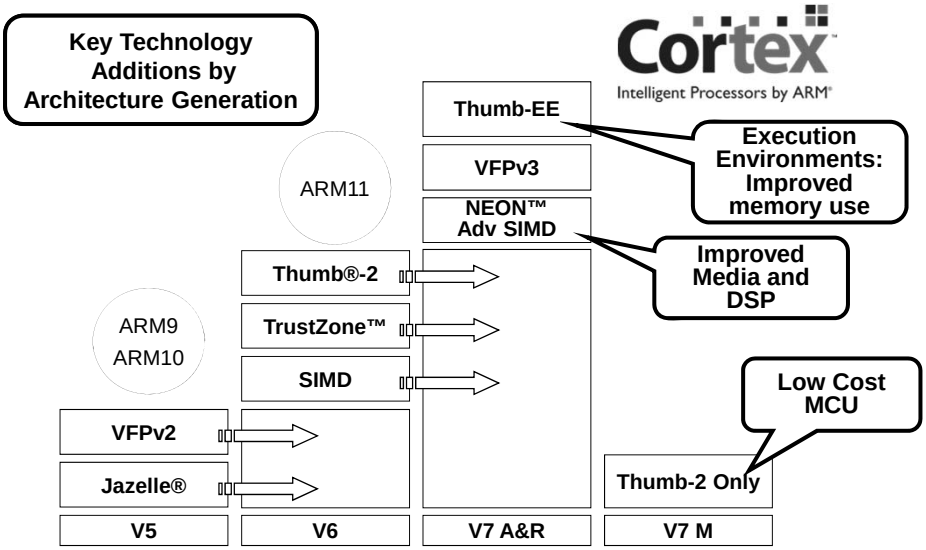
- ❑ Installed and configured VMWare Player v4 or later
- ❑ Installed Ubuntu 10.04
- ❑ Installed the latest Sitara Linux SDK and CCSv5
- ❑ Within the Sitara Linux SDK, ran the setup.sh (to install required host packages)
- ❑ Using a Sitara EVM, followed the QSG to connect ethernet, serial cables, SD card 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)



## What you will learn

- Features of Cortex-A8 architecture
  - Advantages of using NEON co-processor in Multimedia applications
  - NEON benchmarks
- ARM Multimedia software stack
  - GStreamer
    - Plug-ins to source, parse and sink audio/video data
  - Codecs
    - FFmpeg/Libav opensource codecs
    - NEON optimization in codecs
- Labs
  - Understand GStreamer pipelines
  - Enable decoding and Parsing elements pipelines

# ARM Cortex-A8






# Multimedia on Cortex-A8

**Cortex-A8 Features and Benefits**

- Dual-issue, in-order, superscalar architecture delivering high performance
  - First implementation of the ARMv7 instruction-set architecture, including the advanced SIMD media Instructions (NEON™)
  - Advanced dynamic Branch prediction
- Integrated, 256 KB unified L2 cache
  - Dedicated, low-latency, high-BW interface to L1 cache
- NEON™ : 64/128-bit Hybrid SIMD Engine for Multimedia
  - Supports both Integer and Floating Point SIMD
- Enhanced VFPv3 – doubles number of double-precision registers and new instructions to convert between fixed and floating point
- Efficient Run Time Compilation Target
  - Jazelle-RCT: Target for Java. Memory footprint reduced up to 3x
  - Can also target languages such as Microsoft .NET MSIL, Perl, Python

# Multimedia on Cortex-A8

**Neon Features and Benefits**

- Independent HW block to support advanced SIMD instructions
- Comprehensive instruction set with support of 8, 16 & 32-bit signed & unsigned data types
- 256 byte register file (dual 32x64/16x128 view) with hybrid 32/64/128 bit modes
- Large register files enables efficient data handling and minimizes access to memory, thus enhancing data throughput
- Processor can sleep sooner which leads to an overall dynamic power saving
- Independent 10-stage pipeline
- Dual-issue of limited instruction pairs
- Significant code size reduction

Texas Instruments

# Multimedia on Cortex-A8

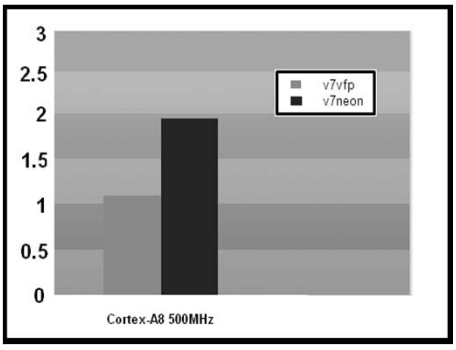
**Neon Multimedia benchmark**

Test Parameters:

- Sep 21 2009 snapshot of gst-ffmpeg.org

| | |
|---|---|
| **Resolution** | 480x270 |
| **Frame Rate** | 30fps |
| **Audio** | 44.1KHz |
| **Video Codec** | H.264 |
| **Audio Codec** | AAC |

- Real silicon measurements on Omap3 Beagleboard
- Benchmarks released by ARM demonstrating an overall performance improvement of ~2x




Texas Instruments

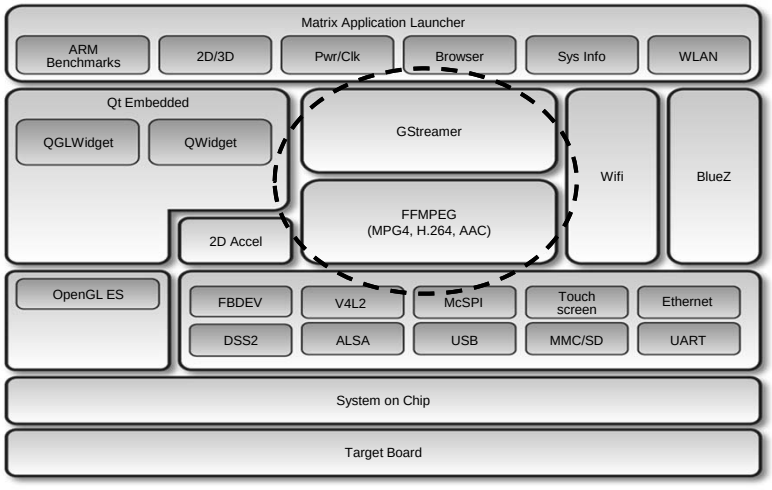
# NEON support on opensource community

NEON is currently supported in the following Open Source projects

- ffmpeg/libav
  - NEON Video: MPEG-2, MPEG-4 ASP, H.264 (AVC), VC-1, VP3, Theora
  - NEON Audio: AAC, Vorbis, WMA
- x264 –Google Summer Of Code 2009
  - GPL H.264 encoder –e.g. for video conferencing
- Bluez –official Linux Bluetooth protocol stack
  - NEON sbc audio encoder
- Pixman (part of cairo 2D graphics library)
  - Compositing/alpha blending
  - X.Org, Mozilla Firefox, fennec, & Webkit browsers
  - e.g. fbCompositeSolidMask_nx8x0565neon 8xfaster using NEON
- Ubuntu 09.04 & 09.10 –fully supports NEON
  - NEON versions of critical shared-libraries
- Android –NEON optimizations
  - Skia library, S32A_D565_Opaque 5xfaster using NEON
  - Available in Google Skia tree since 03-Aug-2009

Texas Instruments

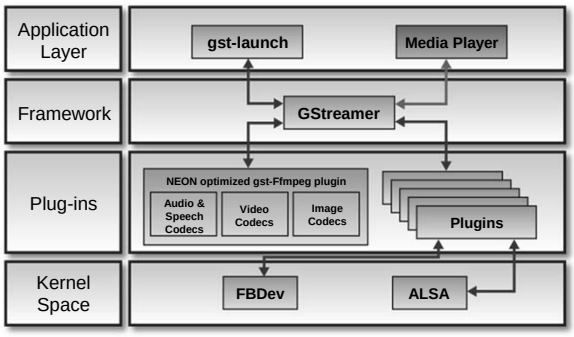
# SDK: ARM multimedia framework




Texas Instruments

# ARM multimedia framework

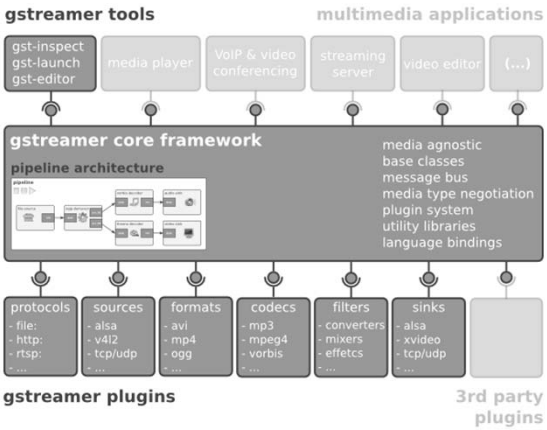
- **Gstreamer**
  - Multimedia processing library
  - Provides uniform framework across platforms
  - Includes parsing & A/V sync support
  - Modular with flexibility to add new functionality via plugins
  - Easy bindings to other frameworks
- **FFmpeg/Libav**
  - Free audio and video decoder/encoder code licensed under LGPL (GPL licensed codecs can be build seperately)
  - A comprehensive suite of standard compliant and robust multimedia codecs
    - Audio, Video, Image, Speech
  - Codec software package
    - Codec libraries with standard C based API
    - Audio/Video parsers that support popular multimedia content
  - Use of SIMD/NEON instructions
    - Neon will give 1.6x-2.5x performance on complex video codecs




Texas Instruments

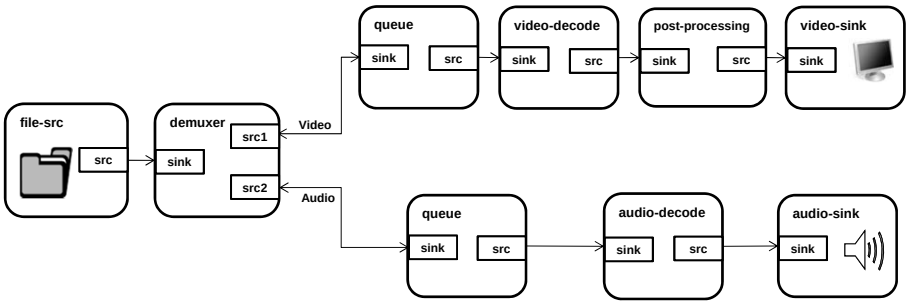
# GStreamer software stack

- Over 150 plugins available
- Plugin
  - Collection of elements
- Elements
  - Sources, filters, sinks
- Bins and Pipelines
  - Bin is a container for collection of elements
  - Pipeline is a top-level bin that allows scheduling and running of all of the elements
- Pads
  - Element source / sink connection points
- Caps
  - Capabilities organized by stream type with a set of properties
- Bus
  - Message interface that allows asynchronous interaction with an active pipeline




Texas Instruments

# GStreamer pipeline architecture




- Each elements are connected through src/sink pads
- Data is queued until maximum specified buffer limit is reached
  - Element queue will create a new thread to decouple src/sink processing
- Post-processing element
  - Eg: color conversion may be required to support various display panels
- In AMSDK, AV decoders call into opensource libavcodecs via gst-ffmpeg plug-ins
- Parsers can be used to cut streams into buffers, they do not modify the data otherwise

Texas Instruments

# FFmpeg/Libav codecs

- libavcodec is the code library developed as part of the FFmpeg/Libav project
- Supports around 200 audio/video formats
- Used by many free and open source media players and encoders
- To enable NEON optimization extra compiler flags should be enabled
  - cflag ‘mfpu’ should be set to ‘neon’
  - Setting cflag ‘mfloat-abi’ to ‘softfp’ enables generation of code using hardware floating-point instructions
- License
  - FFmpeg libraries include LGPL, GPLv2, GPLv3 and other license based codecs, enabling GPLv3 codecs subjects the entire framework to GPLv3 license
  - Sitara SDK enables GPLv2+ codecs
  - Additional details of legal and license of these codecs can be found on FFmpeg/libav webpage.

Texas Instruments

# NEON ecosystem

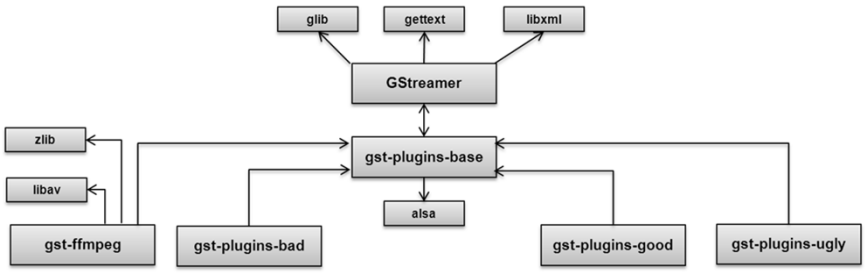
Several third parties provide NEON optimized codec solutions

| Company | Application |
|---|---|
| ingenient technologies | H.264, VC1, MPEG-4 |
| on2 technologies | VP6/7, MPEG-4, VC1, H.264, video stabilization |
| Ittiam | MPEG-4, MPEG-2, H.263, H.264, WMV9, VC1 |
| ARICENT | MPEG-4, H.263, H.264, WMV9, audio |
| TATA ELXSI LIMITED | H.264, VC1 |
| SPIRIT DSP | TEAMSpirit voice and video |
| VisualOn | H.264, MPEG-4, H.263, WMV |
| ACTIMAGINE | MobiClip |
| Fraunhofer IIS | Video and audio codecs |
| DOLBY | Multichannel audio processing |
| Algorithm Specialist TMC | MPEG-4 |
| ESPICO | Audio and consulting |

* For complete list of supported codecs please contact the respective 3P

Texas Instruments

# GStreamer components & build dependencies




- gstreamer: The core package
- gst-plugins-base: An essential exemplary set of elements
- gst-plugins-good: A set of good-quality plug-ins under LGPL
- gst-plugins-ugly: A set of good-quality plug-ins that might have distribution problems
- gst-plugins-bad: A set of plug-ins that need more quality
- gst-ffmpeg: Plug-in with a set of elements which use libav codec libraries

Texas Instruments

# GStreamer: Installed programs

- **gst-feedback-0.10**
  - generates debug info for GStreamer bug reports
- **gst-inspect-0.10**
  - prints information about a GStreamer plugin or element
- **gst-launch-0.10**
  - is a tool that builds and runs basic GStreamer pipelines
- **gst-typefind-0.10**
  - uses the GStreamer type finding system to determine the relevant GStreamer plugin to parse or decode a file
- **gst-xmlinspect-0.10**
  - prints information about a GStreamer plugin or element in XML document format
- **gst-xmllaunch-0.10**
  - is used to build and run a basic GStreamer pipeline, loading it from an XML description

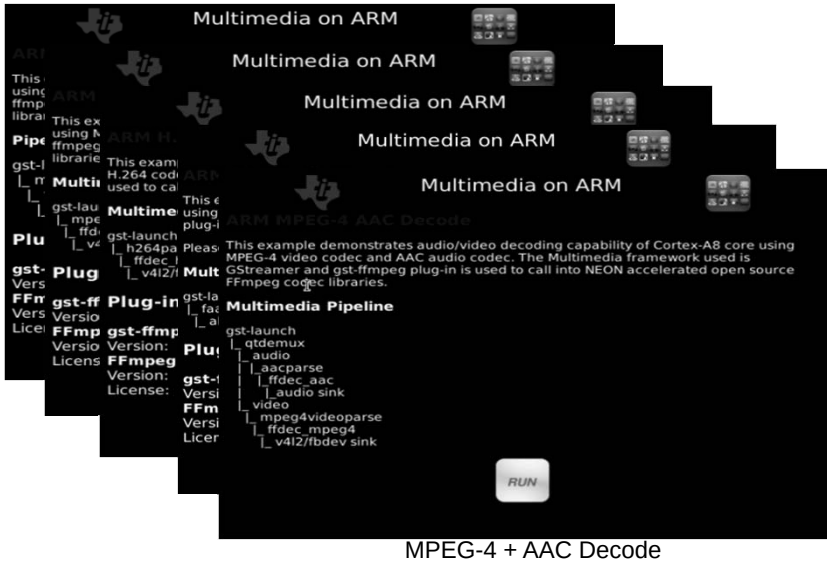


# SDK example application

**SDK Codec Portfolio**

- gst-launch is used to construct multimedia pipelines to demonstrate ARM based audio/video decoding examples
- Video
  - MPEG-4
  - MPEG-2
  - H.264
- Audio
  - AAC
- Video clips are displayed in default LCD resolution or in 480p when DVI out is enabled
- GStreamer elements such as qtdemux are used for demuxing AV content

# Example applications




MPEG-4 + AAC Decode



# Mpeg4 + AAC decode pipeline

**Pipeline:**
**gst-launch-0.10 filesrc location=$filename ! qtdemux name=demux demux.audio_00 ! faad ! alsasink sync=false demux.video_00 ! queue ! ffdec_mpeg4 ! ffmpegcolorspace ! fbdevsink device=/dev/fb0**

- Src pad of each element links to the sink pad on the other element
- Buffers flow between pads of the elements
- Each element has a list of pad structures for each of their input (sink) or output (src)
- Process of caps negotiation is used to configure each element to stream a particular media format over their pads
- Requirements for media format negotiation differs in each element

**Source Element: filesrc**

- No sink pads that generates content for the next element
- Reads from file and presents data on its source pad

**Demuxer: Qtdemux**

- Demuxer element used to timestamp raw, unparsed data into elementary audio and video streams: AAC header for audio and mpeg4 header for video
- Creates output pad for the elementary stream
- Set caps for audio/video stream
- Has fixed caps since data type is embedded in the data stream
- Supports push and pull-based scheduling, depending on the capabilities of the upstream elements

# Mpeg4 + AAC decode pipeline

**Queue**
- Creates a new thread on the source pad to decouple the processing on sink and source pad.

**Decoder: Faad/ffdec_mpeg4**
- Decodes header and data coming in through the sink pad
- Typically each decoder can output data in different formats
  - List of supported formats can be viewed using ‘gst-inspect’
- Downstream elements are notified of new caps only when data passes through their pad
- Negotiation
  - Fixed caps
    - Having fixed caps on source pad restricts re-negotiation
    - While demuxers typically have fixed caps some decoders could also have fixed caps on a pad
    - Fixed cap is a set-up property of a pad, called when creating a pad
  - Non-fixed caps
    - Involves downstream negotiation, format is set on a source pad to configure output format
    - Allows re-negotiation since format is configured on the sinkpad caps or multiple formats are supported



# Mpeg4 + AAC decode pipeline

**Filters: ffmpegcolorspace**
- Handles state changes
- Inspects buffer data, by default sets same format on source and sink
- Capsfilter could be used to restrict the data format

**Sink Element: alsasink/fbdevsink/v4l2sink**
- Critical element which handles preroll- manages state change from pause to play

# Performance and benchmark

| Audio/Video Codec | CPU Frequency | %MEM | %MEM WQVGA %CPU | %MEM 480p %CPU | %MEM WVGA %CPU | VGA 480p %MEM | VGA 480p WQVGA %CPU | VGA 480p 480p %CPU | VGA 480p WVGA %CPU |
|---|---|---|---|---|---|---|---|---|---|
| **MPEG4 + AAC** | 720M | | 44 | 86 | 91 | | 41 | 73 | 80 |
| **WQVGA Clip**: HistoryOfTIAV-WQVGA.mp4 | 600M | | 52 | 98 | 97 | | 49 | 88 | 87 |
| **480p Clip**: HistoryOfTIAV-480p.mp4 | 500M | 22 | 58 | 97 | 97 | 22 | 61 | 97 | 96 |
| **WVGA Clip**: HistoryOfTIAV-WVGA.mp4 | 275M | | 95 | NA | NA | | 96 | NA | NA |
| **MPEG4** | 720M | | 42 | 88 | 71 | | 42 | 76 | 60 |
| **WQVGA Clip**: HistoryOfTI-WQVGA.m4v | 600M | | 45 | 98 | 78 | | 46 | 88 | 69 |
| **480p Clip**: HistoryOfTI-480p.m4v | 500M | 16 | 55 | 97 | 88 | 15 | 53 | 97 | 79 |
| **WVGA Clip**: HistoryOfTI-WVGA.m4v | 275M | | 88 | 96 | 95 | | 87 | 96 | 96 |
| **MPEG2** | 720M | | 43 | 77 | 93 | | 41 | 66 | 80 |
| **WQVGA Clip**: HistoryOfTI-WQVGA.m2v | 600M | | 46 | 84 | 98 | | 47 | 75 | 89 |
| **480p Clip**: HistoryOfTI-480p.m2v | 500M | 16 | 54 | 95 | 97 | 15 | 54 | 86 | 97 |
| **WVGA Clip**: HistoryOfTI-WVGA.m2v | 275M | | 85 | 96 | 96 | | 84 | 96 | 96 |
| **H.264** | 720M | | 62 | 98 | 98 | | 63 | 98 | 98 |
| **WQVGA Clip**: HistoryOfTI-WQVGA.264 | 600M | | 73 | 98 | 97 | | 70 | 98 | 98 |
| **480p Clip**: HistoryOfTI-480p.264 | 500M | 16 | 81 | 97 | 97 | 16 | 79 | 98 | 97 |
| **WVGA Clip**: HistoryOfTI-WVGA.264 | 275M | | 95 | 96 | 92 | | 96 | 98 | 93 |
| | 720M | | 9 | | | | 2 | | |
| **AAC** | 600M | 10 | 2 | Same as VGA | Same as VGA | 10 | 4 | Same as VGA | Same as VGA |
| | 500M | | 3 | | | | 2 | | |
| | 275M | | 26 | | | | 25 | | |

# Power benchmark

- Total processor power is measured for the following peripherals
  - MPU set to OPP 300MHz, Core, on-chip SRAM, LDO, DPLL, DDR & Flash (POP)

| Power measurement set-up | Total power [mW] |
|---|---|
| Default power consumption with Dynamic power switching (DPS) enabled<br>• With sleep_while_idle and enable_off_mode features enabled<br>• With Matrix GUI enabled | 252.87 |
| •With sleep_while_idle and enable_off_mode features enabled<br>• Matrix GUI enabled<br>• MPEG-4 decode running | 329.22 |

• Dynamic voltage frequency scaling (DVFS) can be enabled to scale power values at run-time depending on system-level requirements.
  • scaling_governor is set to ondemand
• Power consumption can be further optimized disabling clocks of unused modules. Additional details of power optimization can be obtained from power management guide and PSP user guide for 2.6.37 kernel

Texas Instruments

# Profiling

- Oprofile, a common Linux profiling tool is used
- Uses hardware performance counters of CPU for profiling
  - hardware and software interrupt handlers
  - kernel modules
  - Kernel
  - shared libraries
  - Applications
- Table depicts profiling results for MPEG4 decode at 300MHz and 1GHz using video pipe for display

| 300MHz | | | 1GHz | | |
|---|---|---|---|---|---|
| samples\| | %\| | app name | samples\| | %\| | app name |
| 2294 | 61.1082 | vmlinux-2.6.37 | 4968 | 89.2562 | vmlinux-2.6.37 |
| 894 | 23.8146 | libgstffmpeg.so | 311 | 5.5875 | libgstffmpeg.so |
| 215 | 5.7272 | libc-2.9.so | 120 | 2.1559 | libgstffmpegcolorspace.so |
| 164 | 4.3687 | libgstffmpegcolorspace.so | 95 | 1.7068 | libc-2.9.so |
| 45 | 1.1987 | libgstreamer-0.10.so.0.26.0 | 19 | 0.3414 | libgobject-2.0.so.0.2400.1 |
| 34 | 0.9057 | libglib-2.0.so.0.2400.1 | 18 | 0.3234 | libgstreamer-0.10.so.0.26.0 |
| 33 | 0.8791 | libgobject-2.0.so.0.2400.1 | 14 | 0.2515 | libglib-2.0.so.0.2400.1 |
| 28 | 0.7459 | libgstmpeg4videoparse.so | 7 | 0.1258 | libgstbase-0.10.so.0.26.0 |
| 16 | 0.4262 | libpthread-2.9.so | 5 | 0.0898 | libgstmpeg4videoparse.so |
| 13 | 0.3463 | libgstbase-0.10.so.0.26.0 | 4 | 0.0719 | ld-2.9.so |
| 9 | 0.2397 | ld-2.9.so | 2 | 0.0359 | busybox |
| 6 | 0.1598 | busybox | 2 | 0.0359 | libpthread-2.9.so |
| 1 | 0.0266 | libm-2.9.so | 1 | 0.0180 | libgthread-2.0.so.0.2400.1 |
| 1 | 0.0266 | libgstcoreelements.so | | | |
| 1 | 0.0266 | libgstvideo4linux2.so | | | |



# Support

- GStreamer
- http://gstreamer.freedesktop.org/
- FFmpeg/libav

SITARA™ ARM® PROCESSORS

Boot camp

Texas Instruments

## Linux Power Management Overview

In this session you will learn how to improve product power performance by minimizing power consumption and guaranteeing system performance. In addition, power management techniques enabled via the Linux SDK will be discussed.

Sep 2012

## Agenda

- What You Will Learn
- Motivation for PM
- Overview of Power Management Techniques
- Understanding PM Features in the AMSDK
- All of the above focused on AM335x.

Texas Instruments

# 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)

Texas Instruments

# What you will learn

- GOAL: Improve product power performance
  - Minimize power consumption
  - Guarantee system performance
- Motivation for power management techniques
- Understand power management techniques of AM335x system
  - DVFS: Dynamic Voltage and Frequency Scaling
  - SmartReflex (aka AVS: Adaptive Voltage Scaling)
  - Dynamic Power Switching (DPS)
    - Implemented by Linux's Runtime PM Framework
  - Static Leakage Management (SLM)
    - Device Idle and Standby modes
  - Basic PM hardware architecture:
    - Voltage domains, Power domains, Clock domains

Texas Instruments

## Why power management?

| | |
|---|---|
| **TI Customers** | ✓ **Competitive differentiator**<br>✓ **Less heat dissipation**<br>✓ **Smaller, sleeker products with smaller battery**<br>✓ **Longer battery life**<br>✓ **Better Features + Increased Usage = Increased Revenue**<br>✓ **Quicker time-to-marked with advanced features** |
| **End Users** | ✓ **Smaller, sleeker products**<br>✓ **Cooler device**<br>✓ **More exciting, power hungry features like multimedia, streaming video, etc.**<br>✓ **Improved experience (longer battery life)**<br>✓ **Lower cost** |

Texas Instruments

## Power Management Principles

1) Power consumed is proportional to frequency and square of voltage.
   - Small reductions in voltage can be very significant
   - Applicable PM Techniques:
     - Dynamic Voltage and Frequency Scaling (DVFS)
     - SmartReflex (aka Adaptive Voltage Scaling or AVS)

2) Keeping devices powered on consumes a lot of power. Cut or reduce power to entire device or idle device portions.
   - Applicable PM Techniques:
     - Dynamic Power Switching (DPS)
       - Turn off what you're not using!
       - At a granular, per-module level
       - Called "Runtime PM" in Linux
     - Static Leakage Management (SLM)
       - System level idle and/or suspend

Texas Instruments

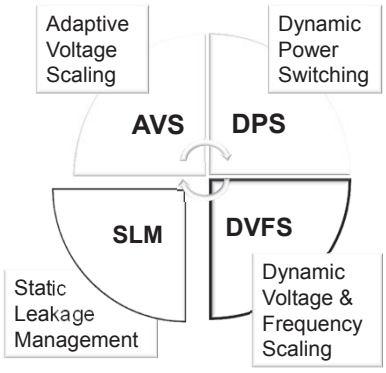
# Power Management Principles (cont.)

3) For proper operation, you should run at one of the recommended operating voltage and frequency combinations (OPPs)
   - Too low voltages can result in data propagation errors and system failure
   - Too high voltages result in excessive power consumption
   - Hardware timing closure results in a few PROVEN operating performance points (OPPs) which guarantee the system to work properly.



# Power Management Techniques: Basics

PM Techniques can be categorized as “Active” or “Idle”:




➢ Active PM Technique

Performed while device is on and in use. Involves all systems components: HW modules, device drivers, OS & apps. Techniques: AVS (SmartReflex), DPS & DVFS.

Idle PM Technique

Entire device is idled. Various idle states selected based on trade-off of power consumption and wakeup latency.

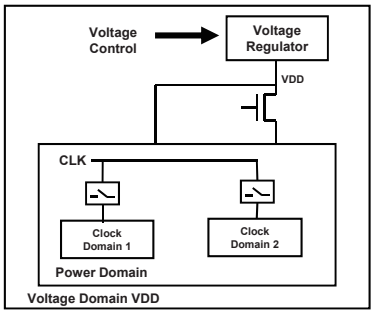
Techniques: SLM

**➔ Power Management is centralized and is handled by the PRCM (Power, Reset and Clock Management) module.**

# PM Architecture Basics

Devices are partitioned into the following architectural blocks:

- Clock Domain:
  - One or more modules fed with the same clock
  - Clock has independent gating control
- Power Domain:
  - One or more modules fed with the same power rail
  - Power rail controlled by independent power switches
- Voltage Domain
  - Group of modules supplied by the same voltage regulator (embedded or external)
  - "VDDn" sometimes referred to in documentation as a "power rail"
- Domains hierarchy is generally:
  - Voltage domains contain one or more Power Domain
  - Power domains contain one or more Clock Domain
- Control of this infrastructure is done by the Power Reset and Clock Manager (PRCM), which is split in two entities:
  - PRM: Handles voltage, power, reset, wake-up management, system clock source control and clock generation
  - CM: Handles the clock generation, distribution and management




Texas Instruments

# AM335x Power Domains Overview

Refer to Ch. 8 of the TRM for more information:

| Power Supply | Power Domain | Modules |
|---|---|---|
| VDD_CORE | PD_WKUP | PRCM, Control Module, GPIO0, DMTIMER0, DMTIMER1, UART0, I2C0, TSC, WDT1, SmartReflex, L4_WKUP, DDR_PHY |
| VDD_CORE | PD_PER | EMIF4, EDMA, GPMC, OCMC, PRUSS, LCD controller, CPSW, USB, MMC0..2, DMTIMER2..7, Uart1..5, SPI0..1, I2C1..2, DCAN0..1, McASP0..1,ePWM0..2, eCAP0..2, eQeP0..1,GPIO1..3,ELM |
| VDD_CORE | PD_GFX | SGX530 |
| VDD_MPU | PD_MPU | CPU, L1, L2 of MPU |
| VDD_RTC | PD_RTC | RTC |

Texas Instruments

# Dynamic Voltage and Frequency Scaling (DVFS)

Active PM Technique

- Technique used to scale operating frequency and voltage of hardware
- Well-characterized Operating Performance Points (OPPs) defined for each device
  - OPP specified as pair: (MPU FREQUENCY, VOLTAGE (for weak silicon))
    - Indicates minimum voltage at which ALL devices can meet that frequency requirement
  - For each OPP, software sends control signals to external regulators (DC/DC converters) to set the minimum voltage required.
  - The OS monitors the workload and can **dynamically** adapt voltage/frequency when using certain governor policies ("ondemand").
- DVFS applicability:
  - AM335x: VDD_MPU and VDD_CORE
- Software Support:
  - OPP can be statically selected by the user via Matrix, or at the Linux command line.
    - Only if governor is "userspace"!
  - The CPUFreq driver in Linux supports multiple "governors".
    - Governors implement different policies for dynamically managing OPP selection.
      - "powersave" governor: selects the lowest possible frequency
      - "performance" governor: selects the highest possible frequency.

Texas Instruments

# Dynamic Voltage and Frequency Scaling (DVFS)

These charts show the defined OPP's for current devices:

| | OPP | ARM MHz | VDD_MPU |
|---|---|---|---|
| AM335x | SRTurbo | 720 | 1.26 |
| | 120 | 600 | 1.2 |
| | 100 | 500 | 1.1 |
| | 50 | 275 | 0.95 |

| OPP | L3/L4 MHz | VDD_CORE |
|---|---|---|
| 100 | 200/100 | 1.1 |
| 50 | 100/50 | 0.95 |

| | OPP | ARM MHz | Vdd1 |
|---|---|---|---|
| AM37x | 1G | 1000 | 1.33 |
| | 130 | 800 | 1.27 |
| | 100 | 600 | 1.14 |
| | 50 | 300 | 0.97 |

| OPP | L3 MHz | Vdd2 |
|---|---|---|
| 100 | 200 | 1.14 |
| 50 | 100 | 0.95 |

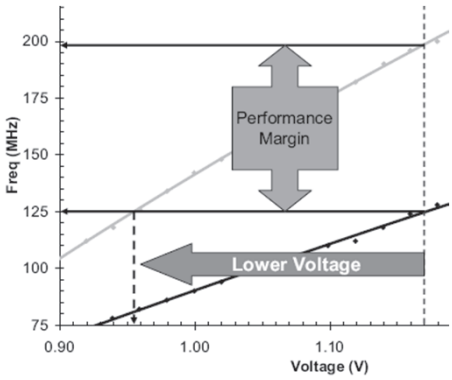
Active PM Technique

Texas Instruments

# Adaptive Voltage Scaling

Active PM Technique

- **SmartReflex** (trademark of Texas Instruments)
  - Silicon manufacturing process yields a distribution of performance capability
  - For a given frequency requirement:
    - Hot/strong/fast devices can meet this at a lower voltage
    - Cold/weak/slow devices needs higher voltage
  - Simple system will set the higher voltage for operating all devices
  - **Smarter system will adapt operating voltage per device!**
  - Green line: "hot" device
  - Blue line: "cold" device




Texas Instruments

# SmartReflex Classes

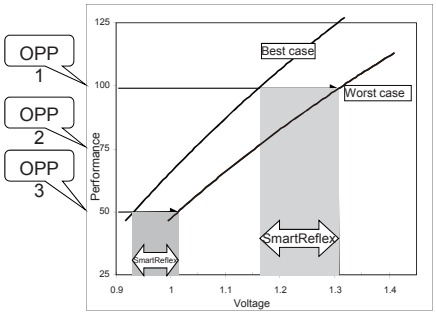
- **Class-0: Manufacturing Test Calibration**
  - At manufacturing test, the device-optimized operating point voltages are permanently fused into each die. A one time optimization to account for process variations.
- **Class-1: Boot-Time Software Calibration**
  - At boot-up time, device-optimized operating point voltages of the die are determined during calibration. Optimization also accounts for process variations.
- **Class-2: Continuous Software Calibration (AM335x uses Class-2B)**
  - SmartReflex sub-chip does real-time voltage optimization via software loop
  - Optimizes for process, temperature and silicon degradation effects
  - Variants:
    - **Class-2A**: Timer interrupt or other system event (e.g. frequency change) used to initiate interrogation of SmartReflex sub-chip
    - **Class-2B**: SmartReflex sub-chip generates a host CPU interrupt when frequency is outside acceptable range
  - **Key: Software intervention required**
- **Class-3 (AM37x): Continuous Hardware Calibration**
  - SmartReflex sub-chip has a dedicated hardware loop to dynamically optimize voltage for process, temperature and silicon degradation effects
  - MPU intervention not required – SmartReflex sub-chip communicates any required voltage change directly to the PMIC via a hardware interface (e.g. i2c)
  - Optimizes for process, temperature and silicon degradation effects
  - **Key: No Software intervention required**
- **Class-4: Fully Integrated Solution**
  - Class 3 hardware control loop plus voltage regulator integrated on single die

Texas Instruments

# DVFS vs AVS

DVFS:

- Select OPP from available OPPs based on MIPS requirement in scenario
- Lower frequency requirement -> lower voltage
- All die treated the same for a particular application scenario




| | OPP | ARM MHz | Vdd1 |
|---|---|---|---|
| AM335x | OPPTurbo | 720 | 1.26 |
| | 120 | 600 | 1.2 |
| | 100 | 500 | 1.1 |
| | 50 | 275 | 0.95 |

| OPP | L3 MHz | Vdd2 |
|---|---|---|
| 100 | 200 | 1.1 |
| 50 | 100 | 0.95 |

AVS (SmartReflex)

- **After** selection of OPP, scale voltage based on device-specific properties (process capability, temperature)
- Faster (also stronger/warmer) process -> lower voltage
- Each die treated differently



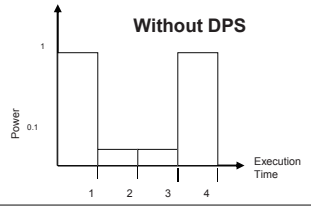
# Dynamic Power Switching

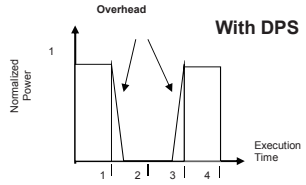
Active PM Technique

Allows certain modules to idle when not being used.

- Example use cases: during MP3 playback, we could idle CPSW (among others)

- These modules operate in a high performance state to complete its tasks as fast as possible, then dynamically switch to a low-power state ("retention" or "off state")
  - Context save/restore may be necessary if memory is lost → additional overhead
  - Acceptable wakeup latencies on the order of microseconds
- Software support: In Linux, handled by the "Runtime PM" framework. This is driver-local suspend/resume, managed entirely by individual device drivers operation independently. Drivers can relinquish clocks, resulting in portions of the device in a "clock stop" state.



## Static Leakage Management

Idle PM Technique

- Device switches into low-power system modes automatically or under user requests when no application is running
  - Example: Media player shuts off display and enters standby after 10-seconds of on-time with no processing and no user input
  - Typically whole device is in some sort of standby mode
  - Acceptable latencies on the order of milliseconds. Deeper sleep state == longer wakeup latency.
- Devices have multiple sleep states (or C-states)
  - AM335x supports two C-states
    - C1 – MPU WFI
    - C2 – MPU WFI + DDR Self Refresh (lowest power)
- Software support:
  - CPUIdle algorithm automatically chooses from available sleep states based upon the inactivity period and current HW state. Useful for power savings during shorter periods of inactivity.
  - Suspend/Resume support available for long inactivity periods (Deep sleep state)

Texas Instruments

## DPS vs. SLM Review

| DPS (Active PM Technique) | SLM (Idle PM Technique) |
|---|---|
| Section of the device in low power mode | Entire device in low power mode (except WKUP domain) |
| Some parts of system stay active | Full system is inactive |
| Smaller transition latencies (us) | Larger transition latencies (ms) |
| Use case :<br>Audio/video Playback - Some domains are going into an idle mode when not needed | Use case:<br>OS idle: Drop into lower-power C-states<br>Suspend-to-RAM: lowest power case |

Texas Instruments

SITARA™ ARM® PROCESSORS

Boot camp

Texas Instruments

# U-Boot & Linux Kernel Board Port

In this session we will cover fundamentals necessary to port a TI Linux-based EVM platform to a custom target platform. We will introduce the necessary steps needed to port the following components: secondary program loader, u-boot and Linux kernel.

LABS:

- http://processors.wiki.ti.com/index.php/Sitara_Linux_Training:_UBoot_Board_Port
- http://processors.wiki.ti.com/index.php/Sitara_Linux_Training:_Linux_Board_Port

July 2012

# 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

- Board Port Overview
- Porting U-Boot to an AM335x Target
- U-Boot Board Port Labs
- Porting the Linux Kernel to a AM335x Target
- Linux Kernel Board Port Labs

# BOARD PORT OVERVIEW

Texas Instruments

## Presentation Overview

- Goal is to gain an understanding of the components of a board port for both U-Boot and Linux
- The board or target portion is the last part of a three step method (Architecture/SOC/Target Board)
- Explain how the SDK will support board ports going forward

Texas Instruments

## Linux Board Background Assumptions

- Already Familiar with :
  - SPL/U-Boot/Linux (☺)
  - SPL/U-Boot/Linux boot sequence
  - U-Boot/Linux build process (kernel configuration)
  - Minicom setup
  - Root File Systems
- Very limited time,
  - Really only have time to show the tip of the iceberg, not going to all inclusive or discuss every facet of board porting, this is a starting place
  - we'll have to take extended question/answer after the class in the foyer or later over email. (or in the bar…. You buy ☺)
- This information is good for today only……. always in flux…..
- What's presented here today may not be the only way of implementation
- Standard disclaimer of "You can and should use what others have done as a method on what to do to move forward"



## Things not covered today..

- Not covering all of the board port steps
  - Limited time today, so we will just be focusing on the code portion of the port
  - Directory setup
  - Machine ID discussion
  - Makefile modifications
  - Git Setup
  - Other Processors

## Linux Board Port Workshop Agenda

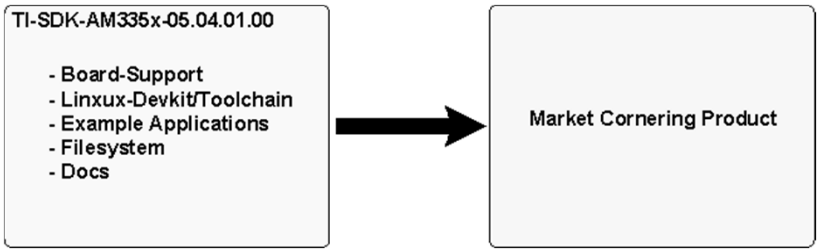
- The Mission: “So…what’s a board port?”
- Look at the System Block Diagram of the target board being used
- Stages of a port
- Pin Mux Utility Tool Overview
- U-Boot Port
  - source tree
  - introduce the target board file
  - Perform two labs that use an already ported example (the code added by with each lab will be discussed )
- Linux Kernel Port
  - source tree
  - introduce the target board file
  - Perform four labs that use an already ported example (the source additions for each lab will be discussed)

Texas Instruments

## The Mission

“Good Morning … the AM335x has been chosen as the processor for your new exciting market cornering product. Your job (no choice but to accept it ☺) is to get U-Boot and the Linux kernel running on this new platform as soon as possible.

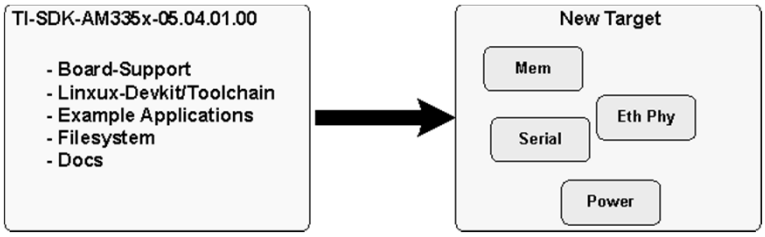
To accomplish this you will take the board design from your HW team and use the AM335x EVM and accompanying Sitara Linux SDK and port U-Boot and the Linux kernel to your new Hardware. “




Texas Instruments

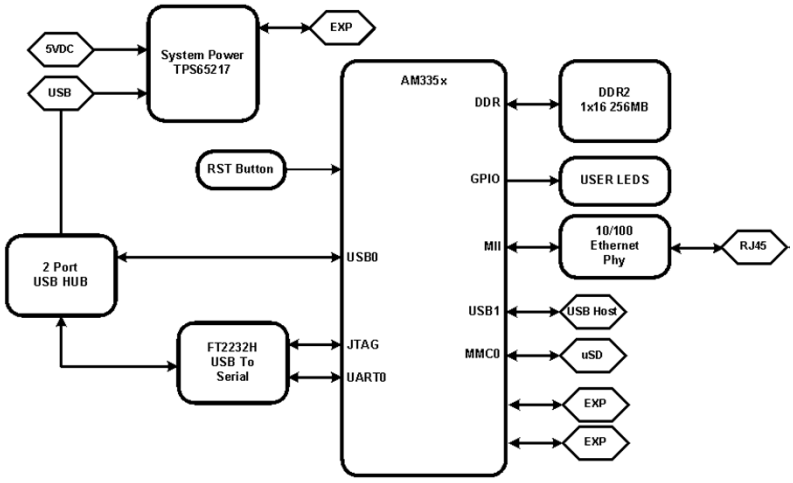
## So….What's a board port?

- It is taking the Sitara Linux SDK that is working on a known platform and moving it to a new target platform that is based on the same TI AM335x processor

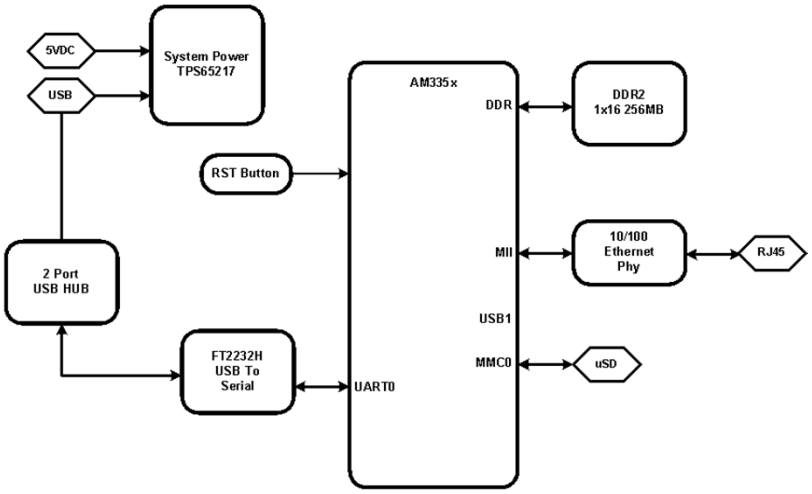





## Target Board for this Exercise…. Beagle Bone

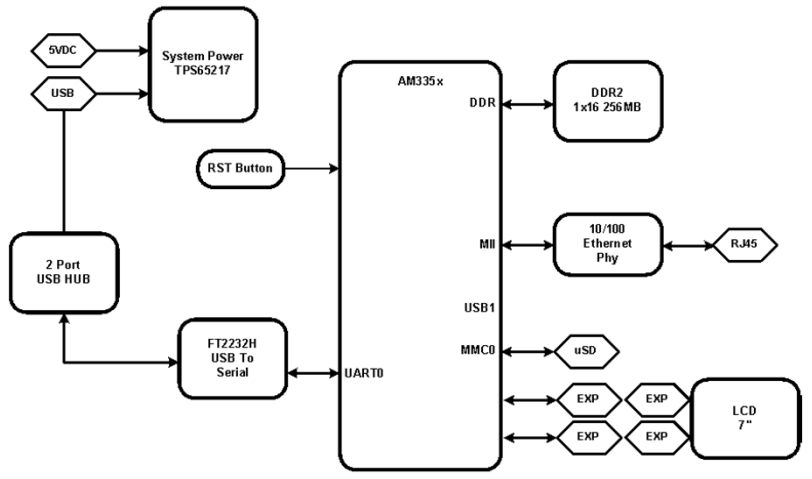


## Target Board Port Configuration Example

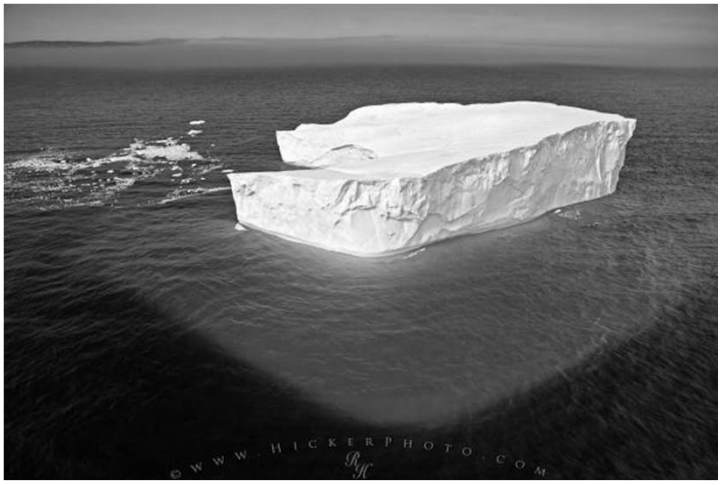





## Will be adding an LCD to the system…..



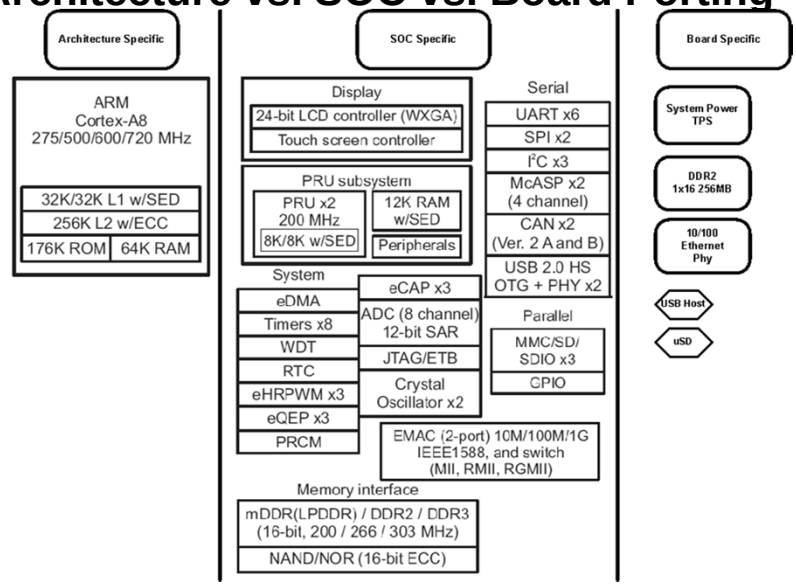
## Board Port…. Tip of the iceberg



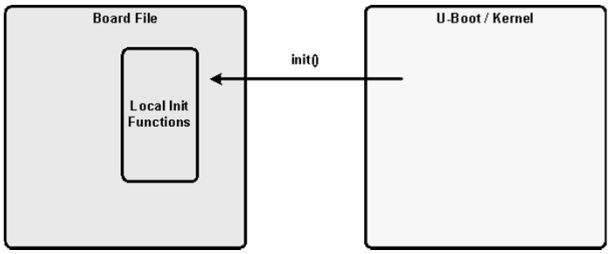

**Used to show the balance of work necessary**



## Architecture vs. SOC vs. Board Porting



# A Tale of Two Board Files

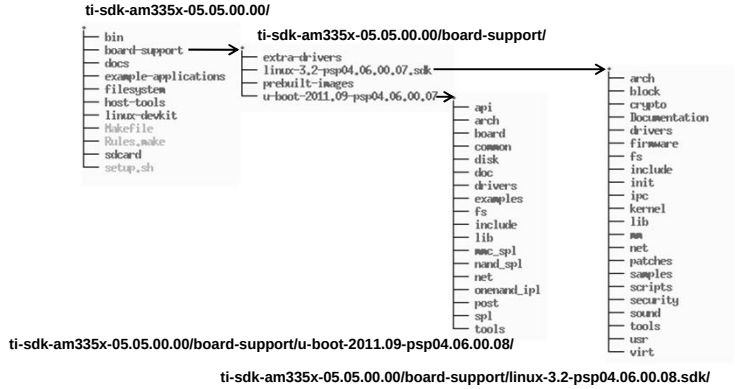



- Both U-boot and Linux follow a similar board file abstraction approach
- The Core Architecture is ported first
- The SOC supporting functions are ported next
- The last part to tie U-Boot/Kernel to the target is the Board file that defines “well known” initialization or entry functions that U-Boot and the Linux Kernel will call to handle “a priori” type board knowledge



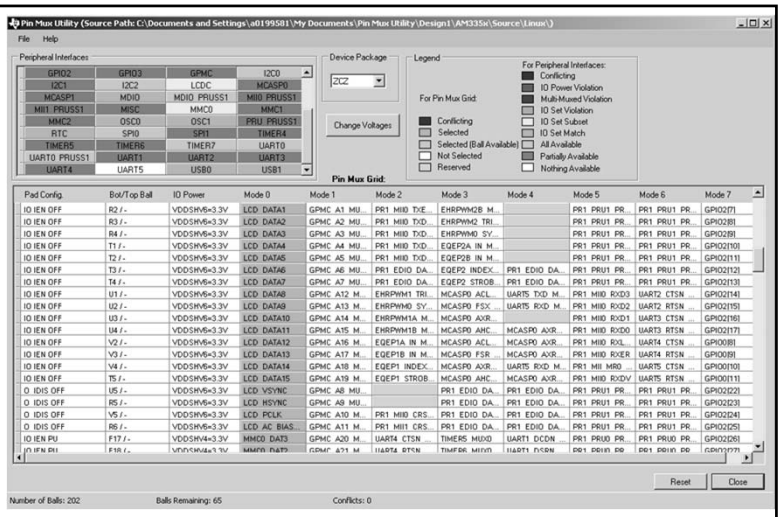
# Where the U-boot and Kernel Sources are after TI-SDK-AM335x-05.05.00.00 installation

- Both the U-Boot and the Linux Kernel Sources are found in the installed TI-SDK-AM335x-05.04.01.00 directory




- Later in the presentation you will see references to just the specific sub-tree that has the respective source such as U-Boot or Linux

## Pin Mux Utility




- GPIO Signals are “muxed” with peripheral interfaces. These can be configured into one of several modes either supporting the peripheral or remaining in a GPIO mode.

## Selecting a mode using Pin Mux Utility

- Each Pin has a mode selection, using UART0 as an example here
- UART0 RXD Mode 0 is selected and GPIO 1.9 is de-selected

| Pad Config. | Bot/Top Ball | IO Power | Mode 0 | Mode 1 | Mode 2 | Mode 3 | Mode 4 | Mode 5 | Mode 6 | Mode 7 |
|---|---|---|---|---|---|---|---|---|---|---|
| IO IEN OFF | E15 / - | VDDSHV6=3.3V | UART0_RXD | SPI1_CS0_M... | DCAN0_TX_... | I2C2_SDA_M... | ECAP2_IN_P... | PR1_PRU1_P... | PR1_PRU1_P... | GPIO1[10] |
| IO IEN OFF | E16 / - | VDDSHV6=3.3V | UART0_TXD | SPI1_CS1_M... | DCAN0_RX_... | I2C2_SCL_M... | ECAP1_IN_P... | PR1_PRU1_P... | PR1_PRU1_P... | GPIO1[11] |

- UART0 RXD Mode 0 is selected and GPIO 1.9 is de-selected, notice Pad config changed too.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| IO IEN OFF | E17 / - | VDDSHV6=3.3V | UART0_RTSN | UART4_TXD_... | DCAN1_RX_M... | I2C1_SCL_MU... | SPI1_D1_MUXD | SPI1_CS0_MU... | PR1_EDC_SY... | GPIO1[9] |
| I_IEN OFF | E15 / - | VDDSHV6=3.3V | UART0_RXD | SPI1_CS0_MU... | DCAN0_TX_M... | I2C2_SDA_MU... | ECAP2_IN_PW... | PR1_PRU1_P... | PR1_PRU1_P... | GPIO1[10] |
| IO IEN OFF | E16 / - | VDDSHV6=3.3V | UART0_TXD | SPI1_CS1_MU... | DCAN0_RX_M... | I2C2_SCL_MU... | ECAP1_IN_PW... | PR1_PRU1_P... | PR1_PRU1_P... | GPIO1[11] |

- Utlitity helps find conflicts, two pins are simultaneously selected

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| IO IEN OFF | E17 / - | VDDSHV6=3.3V | UART0_RTSN | UART4_TXD_... | DCAN1_RX_M... | I2C1_SCL_MU... | SPI1_D1_MUXD | SPI1_CS0_MU... | PR1_EDC_SY... | GPIO1[9] |
| I_IEN OFF | E15 / - | VDDSHV6=3.3V | UART0_RXD | SPI1_CS0_MU... | DCAN0_TX_M... | I2C2_SDA_MU... | ECAP2_IN_PW... | PR1_PRU1_P... | PR1_PRU1_P... | GPIO1[10] |
| Conflict | E16 / - | VDDSHV6=3.3V | UART0_TXD | SPI1_CS1_MU... | DCAN0_RX_M... | I2C2_SCL_MU... | ECAP1_IN_PW... | PR1_PRU1_P... | PR1_PRU1_P... | GPIO1[11] |

- Each Pin has a mode selection, using UART0 as an example here

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| IO IEN OFF | E17 / - | VDDSHV6=3.3V | UART0_RTSN | UART4_TXD_... | DCAN1_RX_M... | I2C1_SCL_MU... | SPI1_D1_MUXD | SPI1_CS0_MU... | PR1_EDC_SY... | GPIO1[9] |
| I_IEN OFF | E15 / - | VDDSHV6=3.3V | UART0_RXD | SPI1_CS0_MU... | DCAN0_TX_M... | I2C2_SDA_MU... | ECAP2_IN_PW... | PR1_PRU1_P... | PR1_PRU1_P... | GPIO1[10] |
| O_IDIS OFF | E16 / - | VDDSHV6=3.3V | UART0_TXD | SPI1_CS1_MU... | DCAN0_RX_M... | I2C2_SCL_MU... | ECAP1_IN_PW... | PR1_PRU1_P... | PR1_PRU1_P... | GPIO1[11] |

- Pin Mux Utility User Guide
  **http://processors.wiki.ti.com/index.php/Pin_Mux_Utility_for_ARM_MPU_Processors**

# PORTING U-BOOT TO AN AM335X TARGET



## U-Boot Port Agenda

- Introduce the board file, where it fits in the Port Picture, where it is in the source tree
- What is the anatomy of the board file
- Introduce the Board File Template that can be used to port u-boot
- Labs Introduction

## U-Boot Board Port Exercises and Source Links

- Link to the U-Boot Labs
  - http://processors.wiki.ti.com/index.php/Sitara_Linux_Training:_UBoot_Board_Port
- Link to the U-Boot Template Source tree (clone this tree)
  - git://gitorious.org/sitara-board-port/sitara-board-port-uboot.git
- PSP U-boot Repo
  - http://arago-project.org/git/projects/?p=u-boot-am33x.git;a=summary



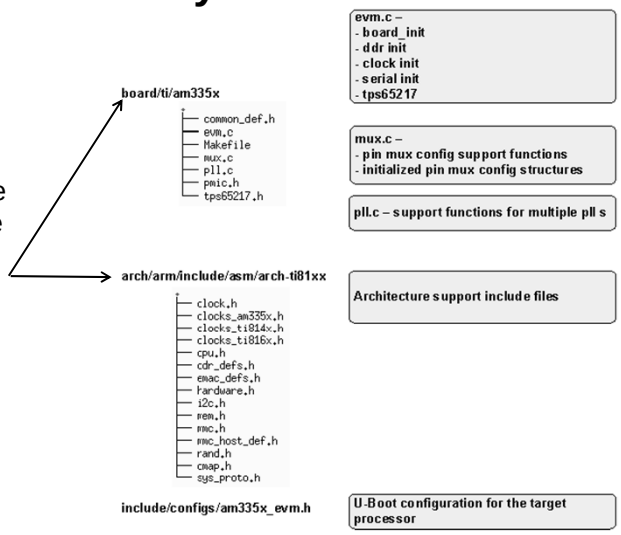
## SPL and U-Boot Builds

- "Dude……. Where's my X-Loader?"
  - It has left the building…. Been replaced by SPL
- The same code base is used to build U-Boot (u-boot.img) and the SPL (still called MLO). Since the same code base is used pre-processor flags are used to isolate the code between the two builds. For example, you do not want the DDR and MPU clock init code in both builds. Also of merit is that one build yields both images.
- Below are examples of the pre-processor flags used:

```
#ifdef CONFIG_SPL_BUILD

#ifndef CONFIG_SPL_BUILD
```

# U-Boot Source Directory

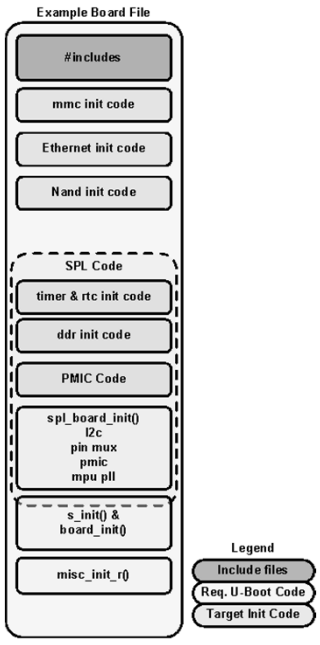
- Using the existing am335x source directory
- The developer will be concentrating on one source directory and for the most part one include directory





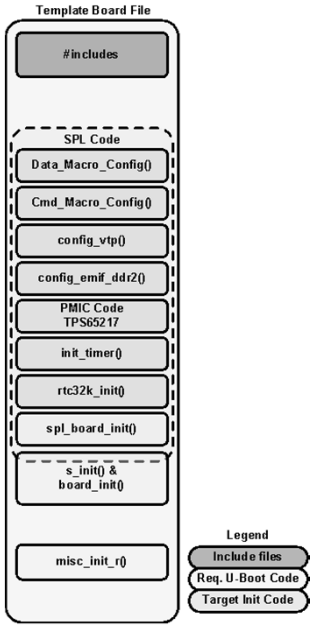

# U-Boot Anatomy of a board File

- Defines Required interface functions for SPL and U-Boot
- One source file contains the code for both SPL and U-Boot and are separated by pre-processor flags
- SPL handles the initialization of clocks, DDR, Serial Port and PMIC
- Some functions are defined twice in both an SPL context and then again in a U-Boot context (s_init & board_init)
- The board file is where the developer will spend most of their effort for a port



## U-Boot/SPL Board Template File

- The board file (evm.c) used here today is different from the one provided in the SDK
- Contains the code for both SPL and U-Boot
- This Board Template only enables MPU Clock, DDR and the Serial Port
- It's up to developer to decide how much functionality they choose to put into the board file and hence the u-boot.img. If the target board supports more peripherals but only one or two is needed to boot into the kernel why add that code?






# U-BOOT BOARD PORT LABS

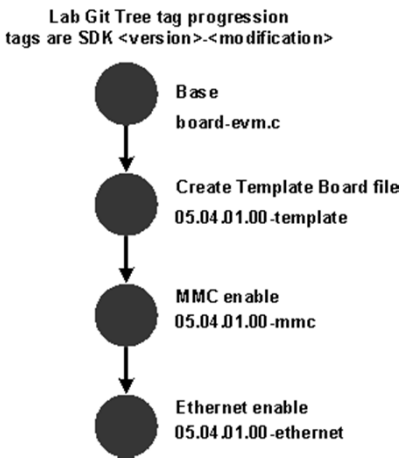
## Board Port Labs

- Lab 1
  - Introduce the template board file and how SPL and u-boot.img are built
- Lab 2
  - Build on the template file demonstrating how to add the MMC and Ethernet peripherals



## Board Port Source Tree being used

- Currently Source is derived from AM-SDK-05.05.00.00, the Port Tree will follow or track each SDK release
- A git tree has been setup for these labs on the host machines
- Using existing board file name and build methods
- Using the default U-Boot configuration supplied with the SDK



## U-Boot Board Port Exercise 1 - Overview

- Goal : Introduce workshop attendees to a board template file that can be used later for a U-Boot Board port
- How this is Demonstrated
  - Build both an SPL and u-boot.img using provided AM335x board template file, which has:
    - Base processor configuration for u-boot, ddr, clocks and a serial console are initialized
- What is being done:
  - Examine the board file to see what is being initialized
- Perform the Lab

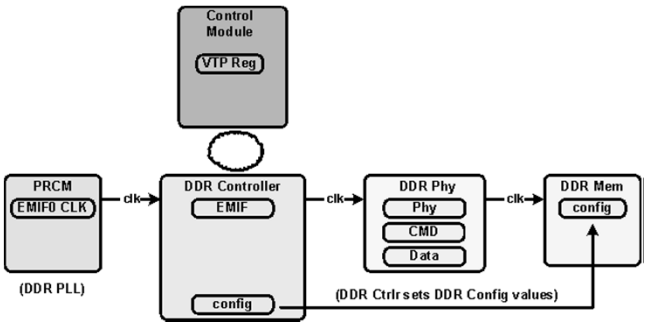


First Burning Question:

## SO… WHERE ARE THE DDR TIMINGS AND THE CLOCK SET?

# First Burning Question: So… where are the DDR timings and the clock set? DDR First

- DDR Setup requires portions of 4 functional blocks to be setup. (Block Diagram)
- EMIF , CMD, DATA and EMIF0 CLK are dependent on Memory selected






# First Burning Question: So… where are the clock and DDR timings set? DDR First

```
Data_Macro_Config()
{
   raw reg write to DDR Phy control registers
}
```

```
CMD_Macro_Config()
{
   raw reg write to DDR Phy control registers
}
```

```
config_vtp()
{
   raw reg write to Control Module registers
}
```

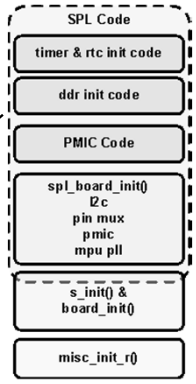
```
config_emif_ddr2()
{
   raw reg write to EMIF Timing control registers
}
```

```
static void config_am335x_ddr(void)
{
      enable_ddr_clocks();
      config_vtp();
      Cmd_Macro_Config();
      Data_Macro_Config(data_macro_0);
      Data_Macro_Config(data_macro_1);

      /* Several Raw Reg writes to DDR IO control */

      config_emif_ddr2();
}
```

board/ti/am335x/evm.c

All Register values are found in:
(change here to support mem changes)
arch/arm/include/asm/arch-ti81xx/ddr_defs.h

All Register offsets are found in:
arch/arm/include/asm/arch-ti81xx/cpu.h




- The DDR is set up within the SPL context
- enable_ddr_clocks in pll.c,
- ddr_defs.h and cpu.h

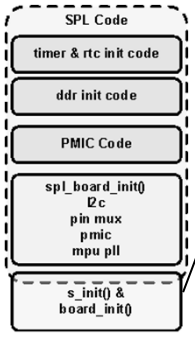
## Here is link to a Tool that can be used to generate necessary values to configure DDR

- Spread Sheet Tool can be found here
  - http://processors.wiki.ti.com/index.php/AM335x_EMIF_Configuration_tips



## The SPL entry function

- s_init is called from lowlevel_init.S to setup system PLL, RTC, UART, timer and finally configures DDR



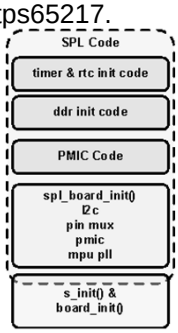

```
/*
 * early system init of muxing and clocks.
 */
void s_init(void)
{
  /* u-boot context */

#ifdef CONFIG_SPL_BUILD
      /* Setup the PLLs and the clocks for the peripherals */
      pll_init();

      /* Enable RTC32K clock */
      rtc32k_enable();

      /* UART softreset */

      enable_uart0_pin_mux();

      /* Disable smart idle */

      /* Initialize the Timer */
      init_timer();

      preloader_console_init();

      config_am335x_ddr();
#endif
}
```

board/ti/am335x/evm.c

This function has both SPL and u-boot contexts

# And now to Set the MPU Clock Rate….

- SPL Context Function
- Before setting the MPU PLL the voltage and current are increased using I2C commands to the tps65217.




```
void spl_board_init(void)
{
   enable_i2c0_pin_mux();
   i2c_init(,);

   /* BeagleBone PMIC Code */
   i2c_probe(TPS65217_CHIP_PM)

   /* Increase USB current limit to 1300mA */
   tps65217_reg_write(, ,USB_INPUT_CUR_LIMIT_1300MA,
                      USB_INPUT_CUR_LIMIT_MASK)
   /* Set DCDC2 (MPU) voltage to 1.275V */
   tps65217_voltage_update(,DCDC_VOLT_SEL_1275MV)

   /* Set LDO3, LDO4 output voltage to 3.3V */
   tps65217_reg_write(,,LDO_VOLTAGE_OUT_3_3,)
   tps65217_reg_write(,,LDO_VOLTAGE_OUT_3_3, LDO_MASK)

   /* Set MPU Frequency to 720MHz */
   mpu_pll_config(MPUPLL_M_720);
}
(Representative code, simplified for the point of discussion)
```

- Called from arch/arm/cpu/armv7/start.S
- If you have a different PMIC you will most likely need a different code base than what is shown here



# Board File Template for u-boot.img

- Within the u-boot context this is the entry function
- Same source file as used for SPL
- Pin Mux config is setup for i2c, uart (already done in SPL) and

```
int board_init(void)
{
    /* Configure the i2c0 pin mux */
    enable_i2c0_pin_mux();

    i2c_init(CONFIG_SYS_I2C_SPEED, CONFIG_SYS_I2C_SLAVE);

    board_id = BONE_BOARD;

    configure_evm_pin_mux(board_id);

#ifndef CONFIG_SPL_BUILD
    board_evm_init();
#endif

    gpmc_init();

    return 0;
}
```

board/ti/am335x/evm.c

# DO LAB 1…….

Texas Instruments

## U-Boot Board Port Exercise 2 - Overview

- Goal : Take the board template file (evm.c) and add both MMC and Ethernet support
- How this is Demonstrated
  - Using the supplied git tree checkout a Ethernet tagged branch, this has both the MMC and Ethernet support code. Build the kernel.
    - This adds Pin Mux support for both Ethernet and MMC
    - Adds the init functions for Ethernet and MMC.
- What is being done:
  - Examine the code changes necessary to implement Ethernet and MMC
- Perform the Lab

Texas Instruments

# Steps to adding MMC and Ethernet to the target board file

- Review system info to see how peripheral is attached
- Pin Mux
  - Use the Pin Mux Utility to configure Pin Init data
- Create Device Init function
  - If device is supported in U-Boot, set the desired include in include/configs
- Add Device Init Function to board file



# Pin Mux Utility

- Pin Mux tool capture for MII interface
- While the tool shows GMII this is the MII interface, doc bug in tool

```
static struct module_pin_mux mii1_pin_mux[] = {
    {OFFSET(mii1_rxerr), MODE(0) | RXACTIVE},      /* MII1_RXERR */
    {OFFSET(mii1_txen), MODE(0)},                  /* MII1_TXEN */
    {OFFSET(mii1_rxdv), MODE(0) | RXACTIVE},       /* MII1_RXDV */
    {OFFSET(mii1_txd3), MODE(0)},                  /* MII1_TXD3 */
    {OFFSET(mii1_txd2), MODE(0)},                  /* MII1_TXD2 */
    {OFFSET(mii1_txd1), MODE(0)},                  /* MII1_TXD1 */
    {OFFSET(mii1_txd0), MODE(0)},                  /* MII1_TXD0 */
    {OFFSET(mii1_txclk), MODE(0) | RXACTIVE},      /* MII1_TXCLK */
    {OFFSET(mii1_rxclk), MODE(0) | RXACTIVE},      /* MII1_RXCLK */
    {OFFSET(mii1_rxd3), MODE(0) | RXACTIVE},       /* MII1_RXD3 */
    {OFFSET(mii1_rxd2), MODE(0) | RXACTIVE},       /* MII1_RXD2 */
    {OFFSET(mii1_rxd1), MODE(0) | RXACTIVE},       /* MII1_RXD1 */
    {OFFSET(mii1_rxd0), MODE(0) | RXACTIVE},       /* MII1_RXD0 */
    {OFFSET(mdio_data), MODE(0) | RXACTIVE | PULLUP_EN}, /* MDIO_DATA */
    {OFFSET(mdio_clk), MODE(0) | PULLUP_EN},       /* MDIO_CLK */
    {-1},
};
```

| Pad Config. | Bot/Top Ball | IO Power | Mode 0 | Mode 1 | Mode 2 | Mode 3 | Mode 4 | Mode 5 | Mode 6 | Mode 7 |
|---|---|---|---|---|---|---|---|---|---|---|
| I IEN PD | J15 / - | VDDSHV5=3.3V | GMII1 RXER | RMII1 RXER | SPI1 D1 M... | I2C1 SCL M... | MCASP1 FS... | UART5 RTSN... | UART2 TXD ... | GPIO3[2] |
| O IDIS PD | J16 / - | VDDSHV5=3.3V | GMII1 TXEN | RMII1 TXEN | RGMII1 TCTL | TIMER4 MUX0 | MCASP1 AX... | EQEP0 IND... | MMC2 CMD... | GPIO3[3] |
| I IEN PD | J17 / - | VDDSHV5=3.3V | GMII1 RXDV | LCD MEMO... | RGMII1 RCTL | UART5 TXD ... | MCASP1 AC... | MMC2 DAT0... | MCASP0 AC... | GPIO3[4] |
| O IDIS PD | J18 / - | VDDSHV5=3.3V | GMII1 TXD3 | DCAN0 TX ... | RGMII1 TD3 | UART4 RXD... | MCASP1 FS... | MMC2 DAT1... | MCASP0 FS... | GPIO0[16] |
| O IDIS PD | K15 / - | VDDSHV5=3.3V | GMII1 TXD2 | DCAN0 RX ... | RGMII1 TD2 | UART4 TXD ... | MCASP1 AX... | MMC2 DAT2... | MCASP0 AH... | GPIO0[17] |
| O IDIS PD | K16 / - | VDDSHV5=3.3V | GMII1 TXD1 | RMII1 TXD1 | RGMII1 TD1 | MCASP1 FS... | MCASP1 AX... | EQEP0A IN ... | MMC1 CMD... | GPIO0[21] |
| O IDIS PD | K17 / - | VDDSHV5=3.3V | GMII1 TXD0 | RMII1 TXD0 | RGMII1 TD0 | MCASP1 AX... | MCASP1 AC... | EQEP0B IN ... | MMC1 CLK ... | GPIO0[28] |
| I IEN PD | K18 / - | VDDSHV5=3.3V | GMII1 TXCLK | UART2 RXD... | RGMII1 TCLK | MMC0 DAT7 | MMC1 DAT0... | UART1 DCD... | MCASP0 AC... | GPIO3[9] |
| I IEN PD | L18 / - | VDDSHV5=3.3V | GMII1 RXCLK | UART2 TXD ... | RGMII1 RCLK | MMC0 DAT6 | MMC1 DAT1... | UART1 DSR... | MCASP0 FS... | GPIO3[10] |
| I IEN PD | L17 / - | VDDSHV5=3.3V | GMII1 RXD3 | UART3 RXD... | RGMII1 RD3 | MMC0 DAT5 | MMC1 DAT2... | UART1 DTR... | MCASP0 AX... | GPIO2[18] |
| I IEN PD | L16 / - | VDDSHV5=3.3V | GMII1 RXD2 | UART3 TXD ... | RGMII1 RD2 | MMC0 DAT4 | MMC1 DAT3... | UART1 RIN ... | MCASP0 AX... | GPIO2[19] |
| I IEN PD | L15 / - | VDDSHV5=3.3V | GMII1 RXD1 | RMII1 RXD1 | RGMII1 RD1 | MCASP1 AX... | MCASP1 FS... | EQEP0 STR... | MMC2 CLK ... | GPIO2[20] |
| I IEN PD | M16 / - | VDDSHV5=3.3V | GMII1 RXD0 | RMII1 RXD0 | RGMII1 RD0 | MCASP1 AH... | MCASP1 AH... | MCASP1 AC... | MCASP0 AX... | GPIO2[21] |
| IO IEN PD | H18 / - | VDDSHV5=3.3V | RMII1 REFC... | XDMA EVE... | SPI1 CS0 M... | UART5 TXD ... | MCASP1 AX... | MMC0 POW... | MCASP1 AH... | GPIO0[29] |
| IO IEN PU | M17 / - | VDDSHV5=3.3V | MDIO DATA | TIMER6 MUX2 | UART5 RXD... | UART3 CTSN... | MMC0 SDC... | MMC1 CMD... | MMC2 CMD... | GPIO0[0] |
| O IDIS PU | M18 / - | VDDSHV5=3.3V | MDIO CLK | TIMER5 MUX2 | UART5 TXD ... | UART3 RTSN... | MMC0 SDW... | MMC1 CLK ... | MMC2 CLK ... | GPIO0[1] |

# Adding MMC to the U-Boot Board file

- Find the pre-processor flags in the am335x_evm.h config file that control inclusion of MMC
- Use the name found for a weak alias to define in the board file
- Create the init function in the board file

```
/* HSMMC support */
#ifdef CONFIG_MMC
#define CONFIG_GENERIC_MMC
#define CONFIG_OMAP_HSMMC
#define CONFIG_CMD_MMC
#define CONFIG_DOS_PARTITION
#define CONFIG_CMD_FAT
#define CONFIG_CMD_EXT2
#endif
```

`#define CONFIG_MMC` — Define in the config file include/configs/am335x_evm.h

drivers/mmc/mmc.c — In the driver file look for a weak alias definition, the name defined here is the one to name the init function in the board file

```
int board_mmc_init(bd_t *bis) __attribute__((weak, alias("__def_mmc_init")));
```

```
#ifdef CONFIG_GENERIC_MMC
int board_mmc_init(bd_t *bis)
{
        omap_mmc_init(0);
        omap_mmc_init(1);
        return 0;
}
#endif
```

Define in the board file and U-boot will call to initialize

board/ti/am335x/evm.c



# Adding Ethernet to the U-Boot Board File

- Use the name found for a weak alias to define in the board file, in net/eth.c
- Create the init functions in the board file
  - 2 functions are created one to init the phy (local) and the board_eth_init definition for u-boot network driver to call
- There are additional supporting structures define in the board file

net/eth.c — In the driver file look for a weak alias definition, the name defined here is the one to name the init function in the board file

```
/*
 * CPU and board-specific Ethernet initializations.  Aliased function
 * signals caller to move on
 */
static int __def_eth_init(bd_t *bis)
{
        return -1;
}

int board_eth_init(bd_t *bis) __attribute__((weak, alias("__def_eth_init")));
```

```
static void evm_phy_init(char *name, int addr)
{
/* Large function.... */

}
```

board/ti/am335x/evm.c

```
int board_eth_init(bd_t *bis)
{
        eth_getenv_enetaddr( , )

        __raw_writel(MII_MODE_ENABLE, MAC_MII_SEL);

        return cpsw_register(&cpsw_data);
}
```

board/ti/am335x/evm.c

# git diff – Code Difference between template and mmc commit

- "git tag" is used to list tags on the git tree

```
05.04.01.00-base
05.04.01.00-ethernet
05.04.01.00-mmc
05.04.01.00-template
```

- "git diff" this is used to isolate code between git commits.

```
schuyler@morpheus:~/bp_uboot/sitara-board-port-uboot$ git diff 05.04.01.00-template..05.04.01.00-mmc
diff --git a/board/ti/am335x/evm.c b/board/ti/am335x/evm.c
index 1635871..b4d7e55 100644
--- a/board/ti/am335x/evm.c
+++ b/board/ti/am335x/evm.c
@@ -487,3 +487,15 @@ int board_late_init(void)
        return 0;
 }
 #endif
+
+#ifndef CONFIG_SPL_BUILD
+#ifdef CONFIG_GENERIC_MMC
+int board_mmc_init(bd_t *bis)
+{
+       omap_mmc_init(0);
+       omap_mmc_init(1);
+       return 0;
+}
+#endif
+#endif /* CONFIG_SPL_BUILD */
schuyler@morpheus:~/bp_uboot/sitara-board-port-uboot$
```

- Do not be concerned about knowing git at this point, here we are using this for illustration purposes.



# git diff – Code Difference between mmc and ethernet commit

- "git diff" commands goes across several screens
- Type "q" to quit command at any point
- Note the plus sign on the edge of the diagram, code addition

```
diff --git a/board/ti/am335x/evm.c b/board/ti/am335x/evm.c
index b4d7e55..863e4cb 100644
--- a/board/ti/am335x/evm.c
+++ b/board/ti/am335x/evm.c
@@ -499,3 +499,133 @@ int board_mmc_init(bd_t *bis)
 #endif
 #endif /* CONFIG_SPL_BUILD */

+#ifdef CONFIG_DRIVER_TI_CPSW
+/* TODO : Check for the board specific PHY */
+static void evm_phy_init(char *name, int addr)
+{
+       unsigned short val;
+       unsigned int cntr = 0;
+       unsigned short phyid1, phyid2;
+
+
+       /* Enable Autonegotiation */
+       if (miiphy_read(name, addr, MII_BMCR, &val) != 0) {
+               printf("Failed to read bmcr\n");
+               return;
+       }
+       val |= BMCR_FULLDPLX | BMCR_ANENABLE | BMCR_SPEED100;
+
+       if (miiphy_write(name, addr, MII_BMCR, val) != 0) {
+               printf("Failed to write bmcr\n");
+               return;
+       }
+       miiphy_read(name, addr, MII_BMCR, &val);
+
+       /* Setup general advertisement */
+       if (miiphy_read(name, addr, MII_ADVERTISE, &val) != 0) {
+               printf("Failed to read anar\n");
+               return;
+       }
:
```

# git diff – Code Difference between mmc and ethernet commit (cont)

- Code continuation for Ethernet PHY setup
- This code was extracted from Beagle Bone specific code from the SDK release.

```
+       val |= (LPA_10HALF | LPA_10FULL | LPA_100HALF | LPA_100FULL);
+
+       if (miiphy_write(name, addr, MII_ADVERTISE, val) != 0) {
+               printf("Failed to write anar\n");
+               return;
+       }
+       miiphy_read(name, addr, MII_ADVERTISE, &val);
+
+       /* Restart auto negotiation*/
+       miiphy_read(name, addr, MII_BMCR, &val);
+       val |= BMCR_ANRESTART;
+       miiphy_write(name, addr, MII_BMCR, val);
+
+       /*check AutoNegotiate complete - it can take upto 3 secs*/
+       do {
+               udelay(40000);
+               cntr++;
+               if (!miiphy_read(name, addr, MII_BMSR, &val)) {
+                       if (val & BMSR_ANEGCOMPLETE)
+                               break;
+               }
+       } while (cntr < 250);
+
+       if (cntr >= 250)
+               printf("Auto negotitation failed\n");
+
+       return;
+}
+
+static void cpsw_control(int enabled)
+{
+       /* nothing for now */
+       /* TODO : VTP was here before */
+       return;
+}
+
:
```



# git diff – Code Difference between mmc and ethernet commit (cont)

- Code continuation for Ethernet setup
- This code was extracted from Beagle Bone specific code from the SDK release.

```
+static struct cpsw_slave_data cpsw_slaves[] = {
+       {
+               .slave_reg_ofs  = 0x208,
+               .sliver_reg_ofs = 0xd80,
+               .phy_id         = 0,
+       },
+       {
+               .slave_reg_ofs  = 0x308,
+               .sliver_reg_ofs = 0xdc0,
+               .phy_id         = 1,
+       },
+};
+
+static struct cpsw_platform_data cpsw_data = {
+       .mdio_base              = AM335X_CPSW_MDIO_BASE,
+       .cpsw_base              = AM335X_CPSW_BASE,
+       .mdio_div               = 0xff,
+       .channels               = 8,
+       .cpdma_reg_ofs          = 0x800,
+       .slaves                 = 2,
+       .slave_data             = cpsw_slaves,
+       .ale_reg_ofs            = 0xd00,
+       .ale_entries            = 1024,
+       .host_port_reg_ofs      = 0x108,
+       .hw_stats_reg_ofs       = 0x900,
+       .mac_control            = (1 << 5) /* MIIEN */,
+       .control                = cpsw_control,
+       .phy_init               = evm_phy_init,
+       .gigabit_en             = 1,
+       .host_port_num          = 0,
+       .version                = CPSW_CTRL_VERSION_2,
+};
+
+int board_eth_init(bd_t *bis)
+{
+       uint8_t mac_addr[6];
:
```

# git diff – Code Difference between mmc and ethernet commit (cont)

- Code continuation for Ethernet setup
- This code was extracted from Beagle Bone specific code from the SDK release.
- How is board_eth_init(..) called?

```
+       .version                = CPSW_CTRL_VERSION_2,
+};
+
+int board_eth_init(bd_t *bis)
+{
+       uint8_t mac_addr[6];
+       uint32_t mac_hi, mac_lo;
+       u_int32_t i;
+
+       if (!eth_getenv_enetaddr("ethaddr", mac_addr)) {
+               debug("<ethaddr> not set. Reading from E-fuse\n");
+               /* try reading mac address from efuse */
+               mac_lo = __raw_readl(MAC_ID0_LO);
+               mac_hi = __raw_readl(MAC_ID0_HI);
+               mac_addr[0] = mac_hi & 0xFF;
+               mac_addr[1] = (mac_hi & 0xFF00) >> 8;
+               mac_addr[2] = (mac_hi & 0xFF0000) >> 16;
+               mac_addr[3] = (mac_hi & 0xFF000000) >> 24;
+               mac_addr[4] = mac_lo & 0xFF;
+               mac_addr[5] = (mac_lo & 0xFF00) >> 8;
+
+               if (is_valid_ether_addr(mac_addr))
+                       eth_setenv_enetaddr("ethaddr", mac_addr);
+               else {
+                       printf("Caution: Using hardcoded mac address. "
+                              "Set <ethaddr> variable to overcome this.\n");
+               }
+       }
+
+       __raw_writel(MII_MODE_ENABLE, MAC_MII_SEL);
+                       /* No gigabit */
+       cpsw_data.gigabit_en = 0;
+
+       return cpsw_register(&cpsw_data);
+}
+#endif
(END)
```



# DO LAB 2…….

## U-Boot Board Port Summary

- Introduced a board port template file with a minimal feature set. Discussed the components in this file. This file could be used for actual board ports.
- Performed two labs demonstrating the template file in action.



# PORTING THE LINUX KERNEL TO AN AM335X TARGET

## Linux Port Agenda

- What are the different stages of a Port
- Introduce the board file, where it fits in the Port Picture, where it is in the source tree
- Discuss the OMAP2+ Machine Shared Common Code
- Labs Introduction



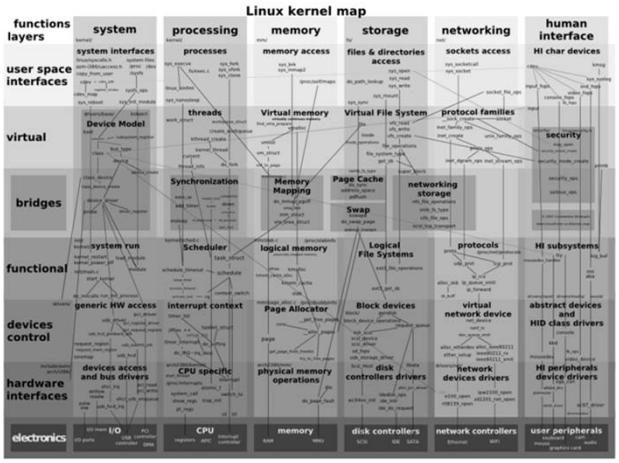
## Linux Board Port Exercises and Source Links

- Link to the U-Boot Labs
  - http://processors.wiki.ti.com/index.php/Sitara_Linux_Training:_Linux_Board_Port
- Link to the Linux Template Source tree (clone this tree)
  - git://gitorious.org/sitara-board-port/sitara-board-port-linux.git
- PSP Linux Kernel Repo –
  - http://arago-project.org/git/projects/?p=linux-am33x.git;a=summary

# Linux Kernel Overview
## (AHHHHH…. The Kernel…)

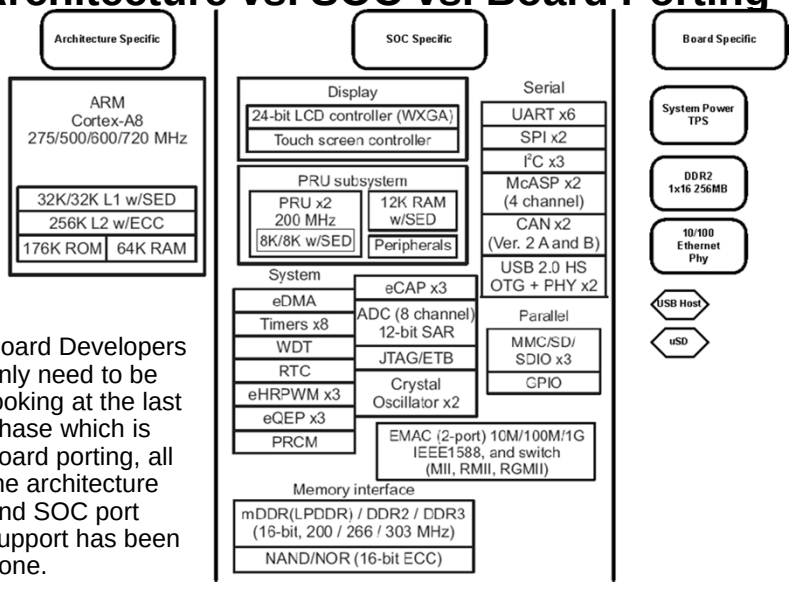
- A very complex and overwhelming kernel block diagram, this is just to make you aware of what's below the waterline…..
- With a target port the architecture and SOC port has already been done. Therefore, the majority of this block diagram has been taken care of for the target port developer.

Source is:
http://en.wikipedia.org/wiki/File:Linux_kernel_map.png





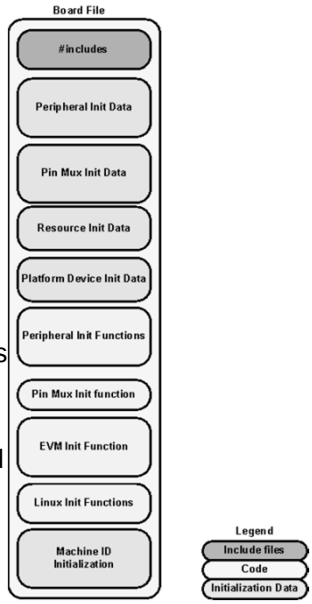

# Architecture vs. SOC vs. Board Porting




- Board Developers only need to be looking at the last phase which is board porting, all the architecture and SOC port support has been done.

# The Target Port Starts with a Board File

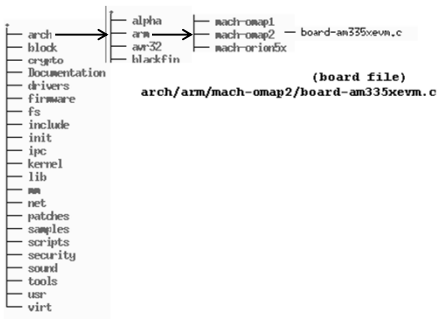
- Defines the Machine Name
- Declares Initialization Data for Peripherals being used
- Declare Pin Mux initialization Data
- Defines Initialization functions
- Provides required Machine Initialization functions
- Calls Common Initialization functions
- Summary is that this file defines several required elements required to boot a Linux kernel, one of several bricks in the wall so to speak.





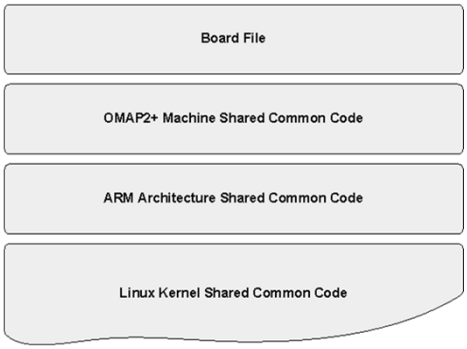

# Linux Kernel Source Tree Overview (Where is the Board file.)

- The board file is located in a source directory called arch/arm/mach-omap2/ where all other board files are located of the same machine type.




# How the Board File fits in the stack

- Board Developer will spend most of their time in the Board file.
- The Board file makes use of the machine shared common code
- The underlying port to the ARM Architecture Shared common code is already done and does not need to be looked at
- Finally everything rests on the Linux Kernel Shared Common Code.
- The lower in the stack you go the less direct interaction the board developer will or need to have.




Texas Instruments

# OMAP2+ Machine Shared Common Code

- There are several board files in the mach-omap2 directory. These board files typical use the support functions defined within this directory. Below is a sampling of some of the supporting common code, not all are mentioned here.

OMAP2 Machine Shared Common Code – arch/arm/mach-omap2
Not a complete listing of the interfaces, just a few are highlighted and to explain how they are used, review this directory to see additional interfaces

| | | | |
|---|---|---|---|
| serial | Sets up UARTs including pin mux | gpio | Initilization function |
| devices | Init calls, platform registration for most peripherals | i2c | Reset and Mux functions |
| common | Init calls to define global address range for select interfaces | mux | Defines a Pin Mux abstraction with supporting functions |
| clocks | Define clock domain mgmt functions | hsmmc | Init functions, hw and platform data |
| control | OMAP2 control registers | sdrc | Init function for SDRC and SMS |
| display | Display init calls, handles the differences between OMAP2,3 and 4 | voltage | Voltage domain support functions |

Texas Instruments

## OMAP2+ Machine Shared Common Code

- Provided as means to provide a common interface to the SOC peripherals to reduce the time necessary to implement a board port
- This interface is not always a clear dividing between maintainers and board developers.
- This is not a documented interface and due to the changing nature of the Linux kernel will almost always be in flux. Maintainers in the end have the authority to accept reject code for their particular tree.

Texas Instruments

## LINUX BOARD PORT LABS

Texas Instruments

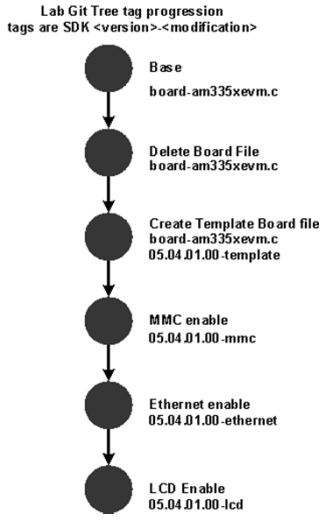
# Board Port Labs

- Lab 1
  - Introduce the template board file
- Lab 2
  - Build on the template file demonstrating how to add the MMC peripheral to provide a Root file system
- Lab 3
  - Build onto template file again this time adding Ethernet for network connectivity
- Lab 4
  - Demonstrate how to add an LCD panel to the board file



# Board Port Source Tree being used

- Currently Source is derived from AM-SDK-05.04.01.00, the Port Tree will follow or track each SDK release
- A git tree has been setup for these labs on the host machines
- Using existing board file name and build methods
- Using the default kernel configuration supplied with the SDK



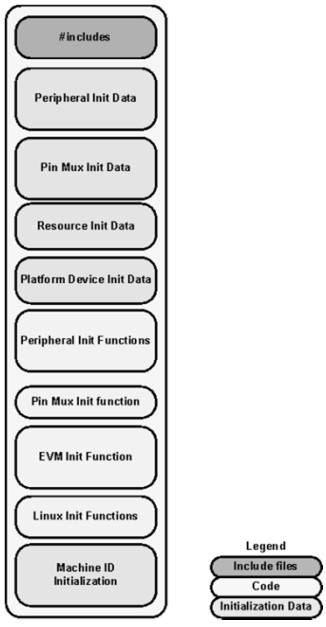
# Linux Board Port Exercise 1 - Overview

- Goal : Introduce workshop attendees to a board template file that can be used later for a Linux board port
- How this is Demonstrated
  - Build a kernel using provided AM335x board template file, which has:
    - Base processor configuration for Linux, just serial console peripheral is initialized
    - This board will not completely boot… no peripheral is defined for a Root File System
- What is being done:
  - Examine the board file to see what is being initialized
- Perform the Lab

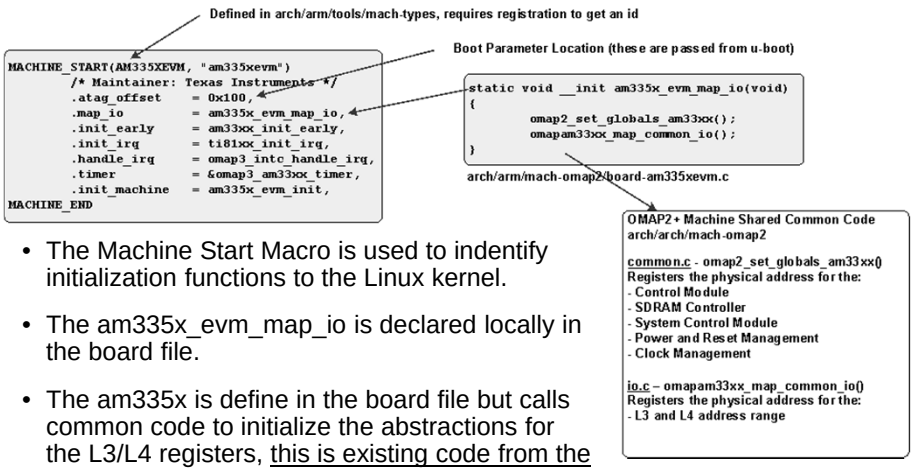


# Template Board File Anatomy

- Binds Linux to a particular target
- Interfaces with the OMAP2+ Machine Shared Common Code.
- Defines pin mux configuration
- The file contains device initialization functions and data.
- Defines the Machine ID and identifies to the Linux Kernel initialization functions



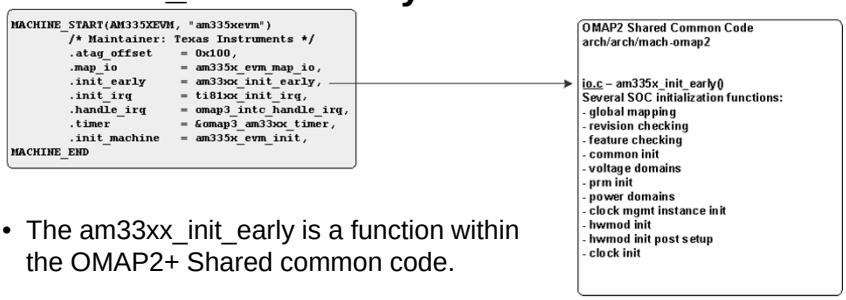
# Template Board File Elements
## MACHINE_START – Key Interface To Kernel




- The Machine Start Macro is used to indentify initialization functions to the Linux kernel.
- The am335x_evm_map_io is declared locally in the board file.
- The am335x is define in the board file but calls common code to initialize the abstractions for the L3/L4 registers, this is existing code from the OMAP2+ Shared Common Code, no need to modify.

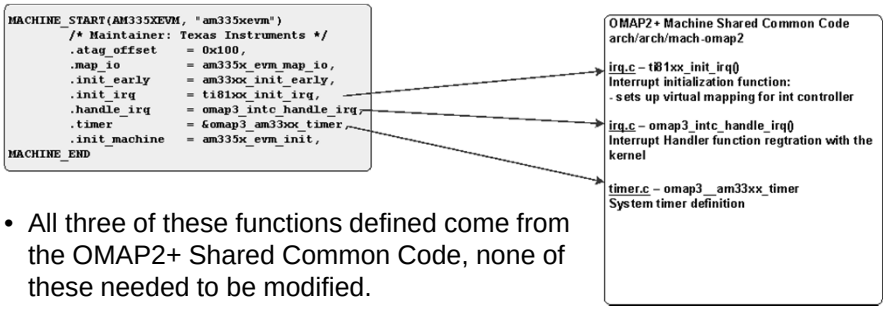


# Template Board File Elements – (cont.)
## MACHINE_START – Key Interface To Kernel




- The am33xx_init_early is a function within the OMAP2+ Shared common code.
- This is called directly from the common code without modification

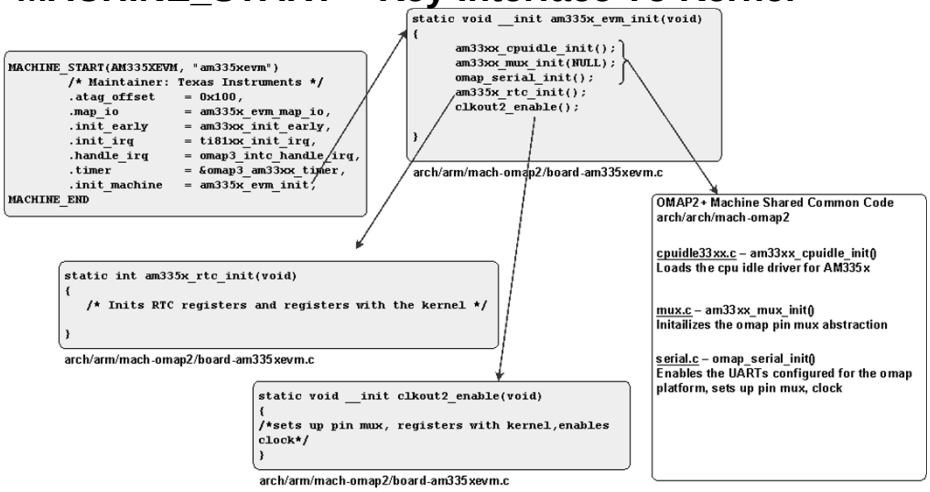
# Board Template File Elements – (cont)
## MACHINE_START – Key Interface To Kernel




- All three of these functions defined come from the OMAP2+ Shared Common Code, none of these needed to be modified.



# Template Board File Elements – (cont)
## MACHINE_START – Key Interface To Kernel




- The am335x_evm_init() is defined by the developer, but uses several functions from the OMAP2 Common Code without modification.

## Question

Within the kernel source, where is the am335xevm board file located?

arch/arm/mach-omap2

Texas Instruments

## DO LAB 1…….

Texas Instruments

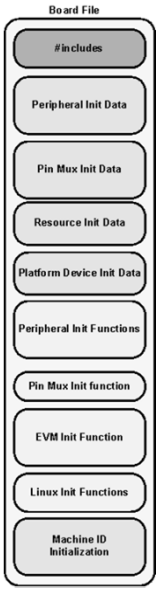
# Linux Board Port Exercise 2 - Overview

- Goal: Build on the template file demonstrating how to add the MMC peripheral to provide a Root file system
- How this is demonstrated:
  - Using the provided lab git tree branch that has the code additions necessary to enable MMC
  - With MMC enabled the root file system can now be mounted
- What is being done:
  - Explaining the code addition components
- Perform the Lab



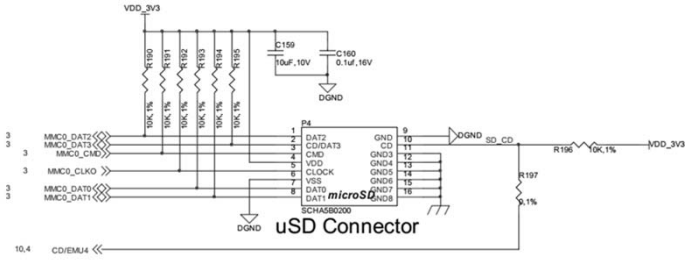
# Steps to adding an MMC interface to target board file

- Review system info to see how peripheral is attached
- Pin Mux
  - Use the Pin Mux Utility to configure Pin Init data
- Device/Platform Initialization data
  - Some peripherals may not require init data
- Create Device Init function
- Add Device Init function to EVM Init Function



# How is the peripheral attached? – Schematic to Pin Mux Utility

- Beagle Bone Schematic




- Pin Mux Tool Capture
- Beagle Bone does not use the WP pin

**Using the pin mux tool to isolate the pin necessary for mmc0**

| Mode 0 | Mode 1 | Mode 2 | Mode 3 | Mode 4 | Mode 5 | Mode 6 | Mode 7 |
|---|---|---|---|---|---|---|---|
| MMC0_DAT3 | GPMC_A20 ... | UART4_CTSN... | TIMER5_MUX0 | UART1_DCD... | PR1_PRU0 ... | PR1_PRU0 ... | GPIO2[26] |
| MMC0_DAT2 | GPMC_A2' ... | UART4_RTSN... | TIMER6_MUX0 | UART1_DSR... | PR1_PRU0 ... | PR1_PRU0 ... | GPIO2[27] |
| MMC0_DAT1 | GPMC_A22 ... | UART5_CTSN... | UART3_RXD... | UART1_DTR... | PR1_PRU0 ... | PR1_PRU0 ... | GPIO2[28] |
| MMC0_DAT0 | GPMC_A23 ... | UART5_RTSN... | UART3_TXD ... | UART1_RIN ... | PR1_PRU0 ... | PR1_PRU0 ... | GPIO2[29] |
| MMC0_CLK | GPMC_A24 ... | UART3_CTSN... | UART2_RXD... | DCAN1_TX ... | PR1_PRU0 ... | PR1_PRU0 ... | GPIO2[30] |
| MMC0_CMD | GPMC_A25 ... | UART3_RTSN... | UART2_TXD ... | DCAN1_RX ... | PR1_PRU0 ... | PR1_PRU0 ... | GPIO2[31] |
| SPI0_CS1 | UART3_RXD... | ECAP1_IN_P... | MMC0_POW... | XDMA_EVE... | MMC0_SDC... | EMU4_MUX1 | GPIO0[6] |
| MCASP0_AC... | EQEP0A_IN ... | MCASP0_AX... | MCASP1_AC... | MMC0_SDW... | PR1_PRU0 ... | PR1_PRU0 ... | GPIO3[18] |

**Have simplified the pin mux tool to show the pins necessary for the mmc0 interface**
**MMC0 pins for data, clk, cmd are being used from mode 0**
**GPIO 0.6 and GPIO 3.18 are being used for card detect and write protect respectively mode 7**

TEXAS INSTRUMENTS

# Lab 2 Board File Additions – Pin Mux Initialization Data

- Capture from the Pin Mux tool, AM3358 ZCZ package

**Using the pin mux tool to isolate the pin necessary for mmc0**

| Mode 0 | Mode 1 | Mode 2 | Mode 3 | Mode 4 | Mode 5 | Mode 6 | Mode 7 |
|---|---|---|---|---|---|---|---|
| MMC0_DAT3 | GPMC_A20 ... | UART4_CTSN... | TIMER5_MUX0 | UART1_DCD... | PR1_PRU0 ... | PR1_PRU0 ... | GPIO2[26] |
| MMC0_DAT2 | GPMC_A2' ... | UART4_RTSN... | TIMER6_MUX0 | UART1_DSR... | PR1_PRU0 ... | PR1_PRU0 ... | GPIO2[27] |
| MMC0_DAT1 | GPMC_A22 ... | UART5_CTSN... | UART3_RXD... | UART1_DTR... | PR1_PRU0 ... | PR1_PRU0 ... | GPIO2[28] |
| MMC0_DAT0 | GPMC_A23 ... | UART5_RTSN... | UART3_TXD ... | UART1_RIN ... | PR1_PRU0 ... | PR1_PRU0 ... | GPIO2[29] |
| MMC0_CLK | GPMC_A24 ... | UART3_CTSN... | UART2_RXD... | DCAN1_TX ... | PR1_PRU0 ... | PR1_PRU0 ... | GPIO2[30] |
| MMC0_CMD | GPMC_A25 ... | UART3_RTSN... | UART2_TXD ... | DCAN1_RX ... | PR1_PRU0 ... | PR1_PRU0 ... | GPIO2[31] |
| SPI0_CS1 | UART3_RXD... | ECAP1_IN_P... | MMC0_POW... | XDMA_EVE... | MMC0_SDC... | EMU4_MUX1 | GPIO0[6] |
| MCASP0_AC... | EQEP0A_IN ... | MCASP0_AX... | MCASP1_AC... | MMC0_SDW... | PR1_PRU0 ... | PR1_PRU0 ... | GPIO3[18] |

**Have simplified the pin mux tool to show the pins necessary for the mmc0 interface**
**MMC0 pins for data, clk, cmd are being used from mode 0**
**GPIO 0.6 and GPIO 3.18 are being used for card detect and write protect respectively mode 7**

- Use existing pinmux_config struct to create pin mux initialization data for mmc0
- Number of pins has to match

**Pin Mux definition for MMC 0**

```
/* Module pin mux for mmc0 */
static struct pinmux_config mmc0_pin_mux[] = {
        {"mmc0_dat3.mmc0_dat3", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
        {"mmc0_dat2.mmc0_dat2", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
        {"mmc0_dat1.mmc0_dat1", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
        {"mmc0_dat0.mmc0_dat0", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
        {"mmc0_clk.mmc0_clk",   OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
        {"mmc0_cmd.mmc0_cmd",   OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
        {"mcasp0_aclkr.mmc0_sdwp", OMAP_MUX_MODE7 | AM33XX_PIN_INPUT_PULLUP},
        {"spi0_cs1.mmc0_sdcd",  OMAP_MUX_MODE7 | AM33XX_PIN_INPUT_PULLUP},
        {NULL, 0},
};
```

```
{"mmc0_dat3.mmc0_dat3", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP}
```

pin name – mmc0_dat3 <arch/arm/mach-omap2/mux33xx.c >

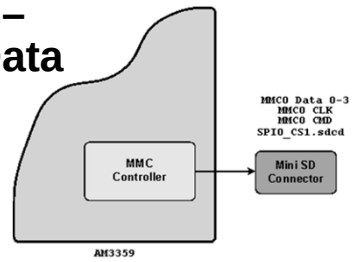
pin value and type < arch/arm/mach-omap2/mux.h >

TEXAS INSTRUMENTS

# Lab 2 Board File Additions – MMC Device Initialization Data

Init Data for MMC 0

```
/* Convert GPIO signal to GPIO pin number */
#define GPIO_TO_PIN(bank, gpio) (32 * (bank) + (gpio))

static struct omap2_hsmmc_info am335x_mmc[] __initdata = {
        {
                .mmc            = 1,
                .caps           = MMC_CAP_4_BIT_DATA,
                .gpio_cd        = GPIO_TO_PIN(0, 6),
                .gpio_wp        = GPIO_TO_PIN(3, 18),
                .ocr_mask       = MMC_VDD_32_33 | MMC_VDD_33_34, /* 3V3 */
        },
        {
                .mmc            = 0,    /* will be set at runtime */
        },
        {
                .mmc            = 0,    /* will be set at runtime */
        },
        {}      /* Terminator */
};
```




- MMC initialization structure to enable interface #1
- This init data is from EVM, BB does not use WP signal

- OMAP 2 mmc structure definition
- Only the elements used are shown, several more

```
struct omap2_hsmmc_info {
  u8      mmc;      /* controller 1/2/3 */
  u32     caps;     /* 4/8 wires and any additional host
                     * capabilities OR'd (ref. linux/mmc/host.h) */
  .
  .
  int     gpio_cd;  /* or -EINVAL */   <Card Detect>
  int     gpio_wp;  /* or -EINVAL */   <Write Protect>
  .
  int     ocr_mask; /* temporary HACK */ <voltage range for slot>
  .
};
```

arch/arm/mach-omap2/hsmmc.h – omap2_hsmmc_info

include/linux/mmc/host.h – capabilities definitions and voltage range definitions

TEXAS INSTRUMENTS

# Initialization Function Call Sequence for MMC Enabling

- This sequence of code is adding in the MMC initialization code to the template file.

```
MACHINE_START(AM335XEVM, "am335xevm")
    /* Maintainer: Texas Instruments */
    .atag_offset  = 0x100,
    .map_io       = am335x_evm_map_io,
    .init_early   = am33xx_init_early,
    .init_irq     = tl81xx_init_irq,
    .handle_irq   = omap3_intc_handle_irq,
    .timer        = &omap3_am33xx_timer,
    .init_machine = am335x_evm_init,
MACHINE_END
```

```
static void __init am335x_evm_init(void)
{
    am33xx_cpuidle_init();
    am33xx_mux_init(NULL);
    omap_serial_init();
    am335x_rtc_init();
    clkout2_enable();
    omap_sdrc_init(NULL, NULL);

    /* Beagle Bone has Micro-SD slot which doesn't have Write Protect pin */
    am335x_mmc[0].gpio_wp = -EINVAL;
    mmc0_init();

}
```

```
static void mmc0_init(void)
{
    setup_pin_mux(mmc0_pin_mux);

    omap2_hsmmc_init(am335x_mmc);
    return;
}
```

Registers MMC init data with Linux

TEXAS INSTRUMENTS

# mmc0 initialization – did it work?

**Did mmc0 messages show up in the console log or dmesg log?**

```
[    1.040191] Waiting for root device /dev/mmcblk0p2...
[    1.078430] mmc0: host does not support reading read-only switch, assuming write-enable.
[    1.089355] mmc0: new high speed SDHC card at address 1234
[    1.095764] mmcblk0: mmc0:1234 SA04G 3.63 GiB
[    1.102752]  mmcblk0: p1 p2
[    1.153137] kjournald starting.  Commit interval 5 seconds
```

**Did mmc0 show up in sysfs?**

```
root@am335x-evm:~# ls -la /sys/devices/platform/omap/omap_hsmmc.0/
drwxr-xr-x    4 root     root             0 Dec 28 09:46 .
drwxr-xr-x   29 root     root             0 Dec 28 09:46 ..
lrwxrwxrwx    1 root     root             0 Dec 28 09:46 driver -> ../../../../bus/platform/drivers/omap_hsmmc
drwxr-xr-x    3 root     root             0 Dec 28 09:46 mmc_host
-r--r--r--    1 root     root          4096 Dec 28 09:52 modalias
drwxr-xr-x    2 root     root             0 Dec 28 09:52 power
lrwxrwxrwx    1 root     root             0 Dec 28 09:46 subsystem -> ../../../../bus/platform
-rw-r--r--    1 root     root          4096 Dec 28 09:46 uevent
```

**Just for curiousity sake… did the root file system mount to mmc?**

```
root@am335x-evm:~# mount
rootfs on / type rootfs (rw)
/dev/root on / type ext3 (rw,relatime,errors=continue,barrier=1,data=ordered)
proc on /proc type proc (rw,relatime)
tmpfs on /mnt/.splash type tmpfs (rw,relatime,size=40k)
sysfs on /sys type sysfs (rw,relatime)
none on /dev type tmpfs (rw,relatime,size=1024k,nr_inodes=8192,mode=755)
/dev/mmcblk0p2 on /media/mmcblk0p2 type ext3 (rw,relatime,errors=continue,barrier=1,data=ordered)
/dev/mmcblk0p1 on /media/mmcblk0p1 type vfat (rw,relatime,fmask=0022,dmask=0022,codepage=cp437,iocharset=iso8859-1,shortname=mixed,errors=remount-ro)
devpts on /dev/pts type devpts (rw,relatime,gid=5,mode=620)
usbfs on /proc/bus/usb type usbfs (rw,relatime)
tmpfs on /var/volatile type tmpfs (rw,relatime,size=16384k)
tmpfs on /dev/shm type tmpfs (rw,relatime,mode=777)
tmpfs on /media/ram_type tmpfs (rw,relatime,size=16384k)
```



# git diff – Code Difference between template and mmc commit

- Code for MMC setup
- This code was extracted from Beagle Bone specific code from the SDK release.
- git tag result for linux board port tree
- git diff command for this commit

```
schuyler@morpheus:~/bp_linux/sitara-board-port-linux$ git tag
05.04.01.00-backlight
05.04.01.00-base
05.04.01.00-ethernet
05.04.01.00-lcd
05.04.01.00-mmc
05.04.01.00-remove-board-file
05.04.01.00-template
05.04.01.00-touchscreen
```

```
diff --git a/arch/arm/mach-omap2/board-am335xevm.c b/arch/arm/mach-omap2/board-am335xevm.c
index 22c2d71..c94e063 100644
--- a/arch/arm/mach-omap2/board-am335xevm.c
+++ b/arch/arm/mach-omap2/board-am335xevm.c
@@ -65,6 +65,25 @@
 /* Header for code common to all OMAP2+ machines. */
 #include "common.h"

+/* Convert GPIO signal to GPIO pin number */
+#define GPIO_TO_PIN(bank, gpio) (32 * (bank) + (gpio))
+
+static struct omap2_hsmmc_info am335x_mmc[] __initdata = {
+       {
+               .mmc            = 1,
+               .caps           = MMC_CAP_4_BIT_DATA,
+               .gpio_cd        = GPIO_TO_PIN(0, 6),
+               .gpio_wp        = GPIO_TO_PIN(3, 18),
+               .ocr_mask       = MMC_VDD_32_33 | MMC_VDD_33_34, /* 3V3 */
+       },
+       {
+               .mmc            = 0,    /* will be set at runtime */
+       },
+       {
+               .mmc            = 0,    /* will be set at runtime */
+       },
+       {}      /* Terminator */
+};

 /* module pin mux structure */
@@ -73,6 +92,19 @@ struct pinmux_config {
        int val; /* Options for the mux register value */
 };

+/* Module pin mux for mmc0 */
+static struct pinmux_config mmc0_pin_mux[] = {
```

```
git diff 05.04.01.00-template..05.04.01.00-mmc
```

# git diff – Code Difference between template and mmc commit (cont)

- Code for MMC setup
- Pin mux was started on previous page
- This code was extracted from Beagle Bone specific code from the SDK release.

```
+       {"mmc0_dat3.mmc0_dat3", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
+       {"mmc0_dat2.mmc0_dat2", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
+       {"mmc0_dat1.mmc0_dat1", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
+       {"mmc0_dat0.mmc0_dat0", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
+       {"mmc0_clk.mmc0_clk",   OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
+       {"mmc0_cmd.mmc0_cmd",   OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
+       {"mcasp0_aclkr.mmc0_sdwp", OMAP_MUX_MODE7 | AM33XX_PIN_INPUT_PULLUP},
+       {"spi0_cs1.mmc0_sdcd",  OMAP_MUX_MODE7 | AM33XX_PIN_INPUT_PULLUP},
+       {NULL, 0},
+};
+
 /*
 * @pin_mux - single module pin-mux structure which defines pin-mux
 *                       details for all its pins.
@@ -92,6 +124,14 @@ static struct pinmux_config clkout2_pin_mux[] = {
        {NULL, 0},
 };

+static void mmc0_init(void)
+{
+       setup_pin_mux(mmc0_pin_mux);
+
+       omap2_hsmmc_init(am335x_mmc);
+       return;
+}
+
 static void __init clkout2_enable(void)
 {
        struct clk *ck_32;
@@ -233,6 +273,10 @@ static void __init am335x_evm_init(void)
     clkout2_enable();
     omap_sdrc_init(NULL, NULL);

+   /* Beagle Bone has Micro-SD slot which doesn't have Write Protect pin */
+    am335x_mmc[0].gpio_wp = -EINVAL;
+       mmc0_init();
:
```



# git diff – Code Difference between template and mmc commit (cont)

- Code for MMC setup
- Note this looks like a repeat from previous page, only these lines are different…
- How is mmc0_init() called?
- This code was extracted from Beagle Bone specific code from the SDK release.
- use “q” to quit

```
+       {"mmc0_clk.mmc0_clk",   OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
+       {"mmc0_cmd.mmc0_cmd",   OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
+       {"mcasp0_aclkr.mmc0_sdwp", OMAP_MUX_MODE7 | AM33XX_PIN_INPUT_PULLUP},
+       {"spi0_cs1.mmc0_sdcd",  OMAP_MUX_MODE7 | AM33XX_PIN_INPUT_PULLUP},
+       {NULL, 0},
+};
+
 /*
 * @pin_mux - single module pin-mux structure which defines pin-mux
 *                       details for all its pins.
@@ -92,6 +124,14 @@ static struct pinmux_config clkout2_pin_mux[] = {
        {NULL, 0},
 };

+static void mmc0_init(void)
+{
+       setup_pin_mux(mmc0_pin_mux);
+
+       omap2_hsmmc_init(am335x_mmc);
+       return;
+}
+
 static void __init clkout2_enable(void)
 {
        struct clk *ck_32;
@@ -233,6 +273,10 @@ static void __init am335x_evm_init(void)
     clkout2_enable();
     omap_sdrc_init(NULL, NULL);

+   /* Beagle Bone has Micro-SD slot which doesn't have Write Protect pin */
+    am335x_mmc[0].gpio_wp = -EINVAL;
+       mmc0_init();
+
 }

 static void __init am335x_evm_map_io(void)
(END)
```

# DO LAB 2…….

Texas Instruments

## Lab 2 Summary

- Added code to the board port template file to handle pin mux, MMC controller initialization and evm initialization function.
- All changes happened within the board file

Texas Instruments

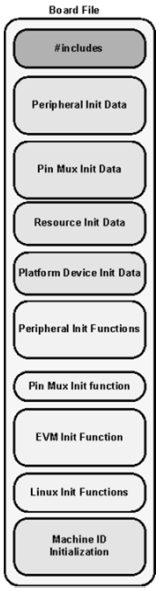
# Linux Board Port Exercise 3 - Overview

- Goal: Build onto the template file again adding Ethernet for Network connectivity
- How this is demonstrated:
  - Using the lab git tree branch with the code additions necessary to enable Ethernet
  - With Ethernet enabled Remote Matrix will be brought up on the browser on the Host machine
- What is being done:
  - Explaining the code addition components (in multiple files this time)
- Perform the Lab



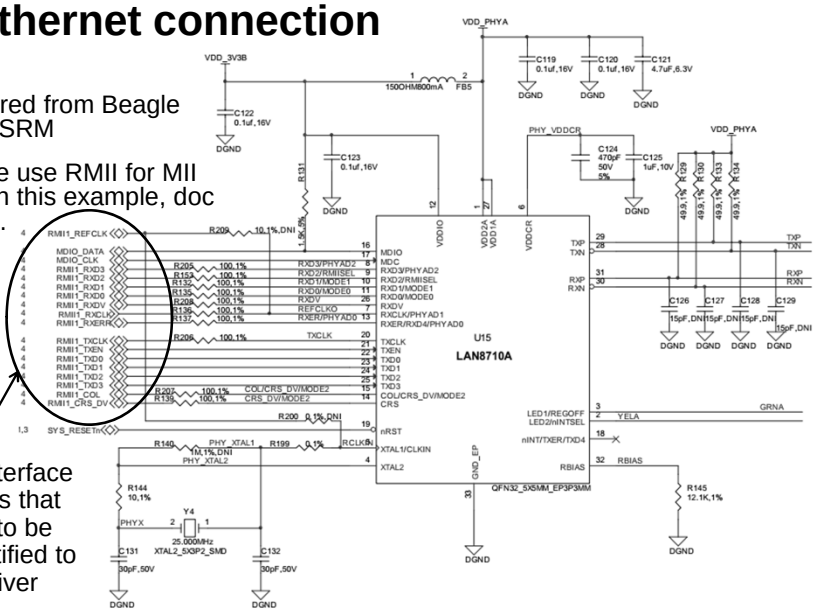
# Steps to adding Ethernet to target board file

- Review system info to see how peripheral is attached
- Pin Mux
  - Use the Pin Mux Utility to configure Pin Init data
- Device/Platform Initialization data
  - None required for this integration
- Create Device Init function
  - Additional Init code required outside the board file
- Add Device Init Function to EVM Init Function



# MII Ethernet connection

- Captured from Beagle Bone SRM
- Please use RMII for MII here in this example, doc bug….
- MII Interface signals that need to be indentified to the driver






# Pin Mux Utility and pinmux config struct

- Pin Mux tool capture for MII interface
- While the tool shows GMII this is the MII interface, doc bug in tool

| Pad Config. | Bot/Top Ball | IO Power | Mode 0 | Mode 1 |
|---|---|---|---|---|
| I IEN PD | H16 / - | VDDSHV5=3.3V | GMII1 COL | RMII2 REF |
| I IEN PD | H17 / - | VDDSHV5=3.3V | GMII1 CRS | RMII1 CRS |
| I IEN PD | J15 / - | VDDSHV5=3.3V | GMII1 RXER | RMII1 RXE |
| O IDIS PD | J16 / - | VDDSHV5=3.3V | GMII1 TXEN | RMII1 TXE |
| I IEN PD | J17 / - | VDDSHV5=3.3V | GMII1 RXDV | LCD MEM( |
| O IDIS PD | J18 / - | VDDSHV5=3.3V | GMII1 TXD3 | DCAN0 TX |
| O IDIS PD | K15 / - | VDDSHV5=3.3V | GMII1 TXD2 | DCAN0 RX |
| O IDIS PD | K16 / - | VDDSHV5=3.3V | GMII1 TXD1 | RMII1 TXD |
| O IDIS PD | K17 / - | VDDSHV5=3.3V | GMII1 TXD0 | RMII1 TXD( |
| I IEN PD | K18 / - | VDDSHV5=3.3V | GMII1 TXCLK | UART2 RX( |
| I IEN PD | L18 / - | VDDSHV5=3.3V | GMII1 RXCLK | UART2 TX( |
| I IEN PD | L17 / - | VDDSHV5=3.3V | GMII1 RXD3 | UART3 RX( |
| I IEN PD | L16 / - | VDDSHV5=3.3V | GMII1 RXD2 | UART3 TX( |
| I IEN PD | L15 / - | VDDSHV5=3.3V | GMII1 RXD1 | RMII1 RXD |
| I IEN PD | M16 / - | VDDSHV5=3.3V | GMII1 RXD0 | RMII1 RXD |
| IO IEN PD | H18 / - | VDDSHV5=3.3V | RMII1 REFCLK | XDMA EVE |
| IO IEN PU | M17 / - | VDDSHV5=3.3V | MDIO DATA | TIMER6 MU |
| O IDIS PU | M18 / - | VDDSHV5=3.3V | MDIO CLK | TIMER5 MU |

Pin Mux definition for Ethernet using the MII interface

```
/* Module pin mux for mii1 */
static struct pinmux_config mii1_pin_mux[] = {
        {"mii1_rxerr.mii1_rxerr", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLDOWN},
        {"mii1_txen.mii1_txen", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT},
        {"mii1_rxdv.mii1_rxdv", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLDOWN},
        {"mii1_txd3.mii1_txd3", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT},
        {"mii1_txd2.mii1_txd2", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT},
        {"mii1_txd1.mii1_txd1", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT},
        {"mii1_txd0.mii1_txd0", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT},
        {"mii1_txclk.mii1_txclk", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLDOWN},
        {"mii1_rxclk.mii1_rxclk", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLDOWN},
        {"mii1_rxd3.mii1_rxd3", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLDOWN},
        {"mii1_rxd2.mii1_rxd2", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLDOWN},
        {"mii1_rxd1.mii1_rxd1", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLDOWN},
        {"mii1_rxd0.mii1_rxd0", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLDOWN},
        {"mdio_data.mdio_data", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLUP},
        {"mdio_clk.mdio_clk", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT_PULLUP},
        {NULL, 0},
};
```

- This demonstrates how the Pin Mux utility can assist in filling out the pinmux_config structure

# devices.c - code addition outside of board file

- Reason - This is code added to devices.c to supplement existing am33x_cpsw_init, does not require eeprom support.
- Reads the MAC IDs
- Sets the PHY type
- Registers MDIO
- Register CPSW with Linux kernel

Added this function to devices.c to initialize ethernet peripheral, not a TI target board

```
void am33xx_cpsw_init_generic(unsigned int phy_type, unsigned int gigen)
{
      u32 mac_lo, mac_hi;

      mac_lo = omap_ctrl_readl(TI81XX_CONTROL_MAC_ID0_LO);
      mac_hi = omap_ctrl_readl(TI81XX_CONTROL_MAC_ID0_HI);
      am33xx_cpsw_slaves[0].mac_addr[0] = mac_hi & 0xFF;
      am33xx_cpsw_slaves[0].mac_addr[1] = (mac_hi & 0xFF00) >> 8;
      am33xx_cpsw_slaves[0].mac_addr[2] = (mac_hi & 0xFF0000) >> 16;
      am33xx_cpsw_slaves[0].mac_addr[3] = (mac_hi & 0xFF000000) >> 24;
      am33xx_cpsw_slaves[0].mac_addr[4] = mac_lo & 0xFF;
      am33xx_cpsw_slaves[0].mac_addr[5] = (mac_lo & 0xFF00) >> 8;

      mac_lo = omap_ctrl_readl(TI81XX_CONTROL_MAC_ID1_LO);
      mac_hi = omap_ctrl_readl(TI81XX_CONTROL_MAC_ID1_HI);
      am33xx_cpsw_slaves[1].mac_addr[0] = mac_hi & 0xFF;
      am33xx_cpsw_slaves[1].mac_addr[1] = (mac_hi & 0xFF00) >> 8;
      am33xx_cpsw_slaves[1].mac_addr[2] = (mac_hi & 0xFF0000) >> 16;
      am33xx_cpsw_slaves[1].mac_addr[3] = (mac_hi & 0xFF000000) >> 24;
      am33xx_cpsw_slaves[1].mac_addr[4] = mac_lo & 0xFF;
      am33xx_cpsw_slaves[1].mac_addr[5] = (mac_lo & 0xFF00) >> 8;

      __raw_writel(phy_type,
                  AM33XX_CTRL_REGADDR(MAC_MII_SEL));

      memcpy(am33xx_cpsw_pdata.mac_addr,
                  am33xx_cpsw_slaves[0].mac_addr, ETH_ALEN);
      platform_device_register(&am33xx_cpsw_mdiodevice);
      platform_device_register(&am33xx_cpsw_device);
      clk_add_alias(NULL, dev_name(&am33xx_cpsw_mdiodevice.dev),
                  NULL, &am33xx_cpsw_device.dev);
}
```



# Ethernet Device Init and EVM Init functions

- The MII init function – call pin mux setup.

Ethernet Initialization function

```
static void mii1_init(void)
{
        setup_pin_mux(mii1_pin_mux);
        return;
}
```

EVM Init function (simplified for discussion purposes, just the added part for Ethernet)

```
/* Called as part of board initialization, defined in MACHINE_START */
static void __init am335x_evm_init(void)
{
.
.
      mii1_init();
      am33xx_cpsw_init_generic(MII_MODE_ENABLE,gigabit_enable);

}
```

- The EVM init function – calls mii1_init and the cpsw init function.

# Ethernet Initialization – Did it work?

**Was an IP address obtained?**

```
root@am335x-evm:~# ifconfig -a
eth0      Link encap:Ethernet  HWaddr 40:5F:C2:76:86:1A
          inet addr:128.247.107.4  Bcast:0.0.0.0  Mask:255.255.254.0
          UP BROADCAST RUNNING ALLMULTI MULTICAST  MTU:1500  Metric:1
          RX packets:14495 errors:0 dropped:5377 overruns:0 frame:0
          TX packets:2 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1538756 (1.4 MiB)  TX bytes:1180 (1.1 KiB)
          Interrupt:40

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

**Did the PHY message show in the console or dmesg log?**

```
[   12.288146]
[   12.288177] CPSW phy found : id is : 0x7c0f1
[   12.294921] PHY 0:01 not found
eth0      no wireless extensions.

udhcpc (v1.13.2) started
Sending discover...
[   15.288576] PHY: 0:00 - Link is Up - 100/Full
Sending discover...
Sending select for 128.247.107.4...
Lease of 128.247.107.4 obtained, lease time 28800
adding dns 128.247.5.10
adding dns 157.170.147.7
```

**Can ping another machine on the network?**

```
root@am335x-evm:~# ping 128.247.106.201
PING 128.247.106.201 (128.247.106.201): 56 data bytes
64 bytes from 128.247.106.201: seq=0 ttl=64 time=0.580 ms
64 bytes from 128.247.106.201: seq=1 ttl=64 time=0.244 ms
64 bytes from 128.247.106.201: seq=2 ttl=64 time=0.214 ms
64 bytes from 128.247.106.201: seq=3 ttl=64 time=0.183 ms
64 bytes from 128.247.106.201: seq=4 ttl=64 time=0.275 ms
```

Texas Instruments

# DO LAB 3…….

Texas Instruments

## Lab 3 summary

- Followed the steps of system attach review, pin mux config, device init to evm init
- Had to add additional code outside the board file to support initializing the cpsw for a generic case

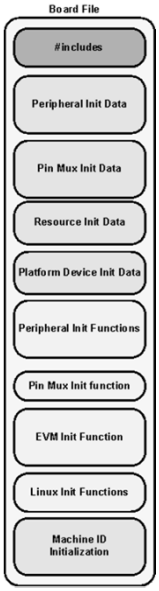


## Linux Board Port Exercise 4 - Overview

- Goal: Build onto the template file again adding support for an LCD panel
- How this is demonstrated:
  - Using the lab git tree tagged branch with code additions necessary to enable an LCD Panel
- What is being done:
  - Explaining the code addition components (multiple files this time)
- Perform the Lab

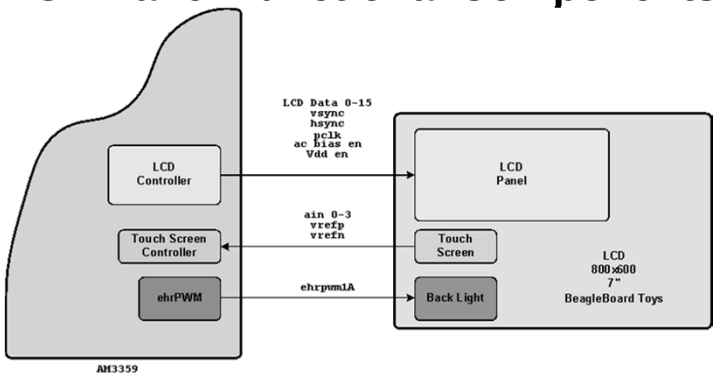
# Steps to adding an LCD Panel to target board file

- Review the system
  - 3 interfaces used: PWM (backlight), LCD, Touch Screen
- Pin Mux
  - Use the Pin Mux Utility to configure Pin Init data
- Device/Platform Initialization data?
  - Backlight , LCD and Touch screen all have initialization data
- Create Device Init function initializes all 3 components
- Add Device init to board_init






# LCD Panel Functional Components




- LCD is the same 7” panel currently found on the EVM
- The respective controllers require data initialization

# LCD Panel Pin Mux Initialization

```
/* Module pin mux for Beagleboard 7" LCD cape */
static struct pinmux_config bbcape7_pin_mux[] = {
      {"lcd_data0.lcd_data0",       OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
            | AM33XX_PULL_DISA},
      {"lcd_data1.lcd_data1",       OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
            | AM33XX_PULL_DISA},
      {"lcd_data2.lcd_data2",       OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
            | AM33XX_PULL_DISA},
      {"lcd_data3.lcd_data3",       OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
            | AM33XX_PULL_DISA},
      {"lcd_data4.lcd_data4",       OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
            | AM33XX_PULL_DISA},
      {"lcd_data5.lcd_data5",       OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
            | AM33XX_PULL_DISA},
      {"lcd_data6.lcd_data6",       OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
            | AM33XX_PULL_DISA},
      {"lcd_data7.lcd_data7",       OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
            | AM33XX_PULL_DISA},
      {"lcd_data8.lcd_data8",       OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
            | AM33XX_PULL_DISA},
      {"lcd_data9.lcd_data9",       OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
            | AM33XX_PULL_DISA},
      {"lcd_data10.lcd_data10",     OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
            | AM33XX_PULL_DISA},
      {"lcd_data11.lcd_data11",     OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
            | AM33XX_PULL_DISA},
      {"lcd_data12.lcd_data12",     OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
            | AM33XX_PULL_DISA},
      {"lcd_data13.lcd_data13",     OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
            | AM33XX_PULL_DISA},
      {"lcd_data14.lcd_data14",     OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
            | AM33XX_PULL_DISA},
      {"lcd_data15.lcd_data15",     OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT
            | AM33XX_PULL_DISA},
      {"lcd_vsync.lcd_vsync",       OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT},
      {"lcd_hsync.lcd_hsync",       OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT},
      {"lcd_pclk.lcd_pclk",         OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT},
      {"lcd_ac_bias_en.lcd_ac_bias_en", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT},
      {"ecap0_in_pwm0_out.gpio0_7", OMAP_MUX_MODE7 | AM33XX_PIN_OUTPUT}, // AVDD_EN
      {NULL, 0},
};
```

| Pad Config. | Bot/Top Ball | IO Power | Mode 0 |
|---|---|---|---|
| IO IEN OFF | R1 / - | VDDSHV6=3.3V | LCD_DATA0 |
| IO IEN OFF | R2 / - | VDDSHV6=3.3V | LCD_DATA1 |
| IO IEN OFF | R3 / - | VDDSHV6=3.3V | LCD_DATA2 |
| IO IEN OFF | R4 / - | VDDSHV6=3.3V | LCD_DATA3 |
| IO IEN OFF | T1 / - | VDDSHV6=3.3V | LCD_DATA4 |
| IO IEN OFF | T2 / - | VDDSHV6=3.3V | LCD_DATA5 |
| IO IEN OFF | T3 / - | VDDSHV6=3.3V | LCD_DATA6 |
| IO IEN OFF | T4 / - | VDDSHV6=3.3V | LCD_DATA7 |
| IO IEN OFF | U1 / - | VDDSHV6=3.3V | LCD_DATA8 |
| IO IEN OFF | U2 / - | VDDSHV6=3.3V | LCD_DATA9 |
| IO IEN OFF | U3 / - | VDDSHV6=3.3V | LCD_DATA10 |
| IO IEN OFF | U4 / - | VDDSHV6=3.3V | LCD_DATA11 |
| IO IEN OFF | V2 / - | VDDSHV6=3.3V | LCD_DATA12 |
| IO IEN OFF | V3 / - | VDDSHV6=3.3V | LCD_DATA13 |
| IO IEN OFF | V4 / - | VDDSHV6=3.3V | LCD_DATA14 |
| IO IEN OFF | T5 / - | VDDSHV6=3.3V | LCD_DATA15 |
| O IDIS OFF | U5 / - | VDDSHV6=3.3V | LCD_VSYNC |
| O IDIS OFF | R5 / - | VDDSHV6=3.3V | LCD_HSYNC |
| O IDIS OFF | V5 / - | VDDSHV6=3.3V | LCD_PCLK |
| O IDIS OFF | R6 / - | VDDSHV6=3.3V | LCD_AC_BIAS_EN |

- Pin Mux Tool capture for the LCD Panel



# LCD Touch Screen Pin Mux Initialization

- Pin Mux Capture of Pins used for Touch Screen
- 4 Wire Resistive touch
- 2 Wire for Voltage reference
- Pin connections are determined by schematic reference

| Pad Config. | Bot/Top Ball | IO Power | Mode 0 | Mode 1 | Mode 2 | Mode 3 | Mode 4 | Mode 5 | Mode 6 | Mode 7 |
|---|---|---|---|---|---|---|---|---|---|---|
| I IEN OFF | A7 / - | VDDA_ADC= | AIN3 | | | | | | | |
| I IEN OFF | B7 / - | VDDA_ADC= | AIN2 | | | | | | | |
| I IEN OFF | C7 / - | VDDA_ADC= | AIN1 | | | | | | | |
| I IEN OFF | B6 / - | VDDA_ADC= | AIN0 | | | | | | | |
| I IEN OFF | B9 / - | VDDA_ADC= | VREFP | | | | | | | |
| I IEN OFF | A9 / - | VDDA_ADC= | VREFN | | | | | | | |

```
/* Module pin mux for touchscreen controller */
static struct pinmux_config tsc_pin_mux[] = {
      {"ain0.ain0",           OMAP_MUX_MODE0 | AM33XX_INPUT_EN},
      {"ain1.ain1",           OMAP_MUX_MODE0 | AM33XX_INPUT_EN},
      {"ain2.ain2",           OMAP_MUX_MODE0 | AM33XX_INPUT_EN},
      {"ain3.ain3",           OMAP_MUX_MODE0 | AM33XX_INPUT_EN},
      {"vrefp.vrefp",         OMAP_MUX_MODE0 | AM33XX_INPUT_EN},
      {"vrefn.vrefn",         OMAP_MUX_MODE0 | AM33XX_INPUT_EN},
      {NULL, 0},
};
```

# LCD Back Light Pin Mux Initialization

- Just a single pin used for the backlight.
- This is a pwm signal that is used to control brightness

```
/* Module pin mux for LCD backlight */
static struct pinmux_config ehrpwm_pin_mux[] = {
        {"gpmc_a2.ehrpwm1A", OMAP_MUX_MODE6 | AM33XX_PIN_OUTPUT},
        {NULL, 0},
};
```

| Pad Config. | Bot/Top Ball | IO Power | Mode 0 | Mode 1 | Mode 2 | Mode 3 | Mode 4 | Mode 5 | Mode 6 | Mode 7 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0_IDIS_PD | U14 / - | VDDSHV3=3.3V | GPMC_A2_MUX0 | GMII2_TXD3 | RGMII2_TD3 | MMC2_DAT1_M... | GPMC_A18_MUX0 | PR1_MII1_TXD2 | EHRPWM1A_MU... | GPIO1[18] |



# Add LCD Panel – Data Initialization

```
/* Define display configuration */
static struct lcd_ctrl_config bbcape7_cfg = {
      &bbcape7_panel,
      .ac_bias          = 255,
      .ac_bias_intrpt         = 0,
      .dma_burst_sz           = 16,
      .bpp              = 16,
      .fdd              = 0x80,
      .tft_alt_mode           = 0,
      .stn_565_mode           = 0,
      .mono_8bit_mode         = 0,
      .invert_line_clock      = 1,
      .invert_frm_clock = 1,
      .sync_edge        = 0,
      .sync_ctrl        = 1,
      .raster_order           = 0,
};
```

- This configures the registers in the LCD Controller.
- The datasheet for LCD will provide information (to name a few)
  - BPP
  - Clock polarity
  - Data Format
  - DMA

```
struct lcd_ctrl_config {
      const struct display_panel *p_disp_panel;

      /* AC Bias Pin Frequency */
      int ac_bias;
      /* AC Bias Pin Transitions per Interrupt */
      int ac_bias_intrpt;
      /* DMA burst size */
      int dma_burst_sz;
      /* Bits per pixel */
      int bpp;
      /* FIFO DMA Request Delay */
      int fdd;
      /* TFT Alternative Signal Mapping (Only for active) */
      unsigned char tft_alt_mode;
      /* 12 Bit Per Pixel (5-6-5) Mode (Only for passive) */
      unsigned char stn_565_mode;
      /* Mono 8-bit Mode: 1=D0-D7 or 0=D0-D3 */
      unsigned char mono_8bit_mode;
      /* Invert line clock */
      unsigned char invert_line_clock;
      /* Invert frame clock */
      unsigned char invert_frm_clock;
      /* Horizontal and Vertical Sync Edge: 0=rising 1=falling */
      unsigned char sync_edge;
      /* Horizontal and Vertical Sync: Control: 0=ignore */
      unsigned char sync_ctrl;
      /* Raster Data Order Select: 1=Most-to-least 0=Least-to-most */
      unsigned char raster_order;
      /* DMA FIFO threshold */
      int fifo_th;
};
```

# LCD Panel Initialization data used by the LCDC

```
/* ThreeFive S9700RTWV35TR */
    [2] = {
        .name = "TFC_S9700RTWV35TR_01B",
        .width = 800,
        .height = 480,
        .hfp = 39,
        .hbp = 39,
        .hsw = 47,
        .vfp = 13,
        .vbp = 29,
        .vsw = 2,
        .pxl_clk = 30000000,
        .invert_pxl_clk = 0,
    },
```

drivers/video/da8xx-fb.c

```
struct da8xx_panel {
        const char      name[25];       /* Full name <vendor>_<model> */
        unsigned short  width;
        unsigned short  height;
        int             hfp;            /* Horizontal front porch */
        int             hbp;            /* Horizontal back porch */
        int             hsw;            /* Horizontal Sync Pulse Width */
        int             vfp;            /* Vertical front porch */
        int             vbp;            /* Vertical back porch */
        int             vsw;            /* Vertical Sync Pulse Width */
        unsigned int    pxl_clk;        /* Pixel clock */
        unsigned char   invert_pxl_clk; /* Invert Pixel clock */
};
```

drivers/video/da8xx-fb.c

- LCD Panel interfacing numbers have to be added in the da8xx-fb.c if they are not alreay defined.
- These numbers are derived from the datasheet for the panel (to name a few)
  - Screen resolution
  - Timings
  - Pixel Clock and Polarity



# Backlight Initialization Data

- PWM is used to control the LCD Panel Brightness

```
/* LCD backlight platform Data */
#define AM335X_BACKLIGHT_MAX_BRIGHTNESS        100
#define AM335X_BACKLIGHT_DEFAULT_BRIGHTNESS    50
#define AM335X_PWM_PERIOD_NANO_SECONDS         (1000000 * 5)
```

```
/* Setup pwm-backlight for bbtoys7lcd */
static struct platform_device bbtoys7lcd_backlight = {
      .name           = "pwm-backlight",
      .id             = -1,
      .dev            = {
            .platform_data  = &bbcape7lcd_backlight_data,
      }
};
```

```
static struct platform_pwm_backlight_data bbcape7lcd_backlight_data = {
      .pwm_id         = BBCAPE7LCD_PWM_DEVICE_ID,
      .ch             = -1,
      .max_brightness = AM335X_BACKLIGHT_MAX_BRIGHTNESS,
      .dft_brightness = AM335X_BACKLIGHT_DEFAULT_BRIGHTNESS,
      .pwm_period_ns  = AM335X_PWM_PERIOD_NANO_SECONDS,
};
```

# LCD Init Function

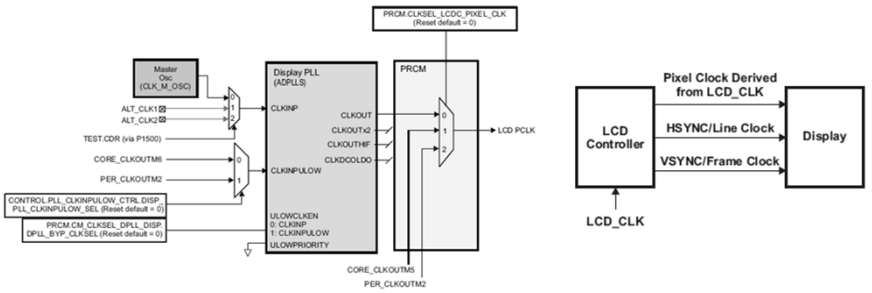
- The steps are:
  - Pin mux setup
  - Assign a GPIO to support VDD_en to the LCD
    - Refer to schematic on which to use
  - Define PLL value for the pixel clock
  - Register with the kernel

```
/* Configure display pll */
static int __init conf_disp_pll(int rate)
{
      struct clk *disp_pll;
      int ret = -EINVAL;

      disp_pll = clk_get(NULL, "dpll_disp_ck");
      if (IS_ERR(disp_pll)) {
            pr_err("Cannot clk_get disp_pll\n");
            goto out;
      }

      ret = clk_set_rate(disp_pll, rate);
      clk_put(disp_pll);
out:
      return ret;
}
```

```
/* Initialize and register lcdc device */
#define BEAGLEBONE_LCD_AVDD_EN GPIO_TO_PIN(0, 7)

static void bbcape7lcd_init(void)
{
      setup_pin_mux(bbcape7_pin_mux);
      gpio_request(BEAGLEBONE_LCD_AVDD_EN, "BONE_LCD_AVDD_EN");
      gpio_direction_output(BEAGLEBONE_LCD_AVDD_EN, 1);

      if (conf_disp_pll(300000000)) {
            pr_info("Failed to set pixclock to 300000000, not attempting to"
                        "register LCD cape\n");
            return;
      }

      if (am33xx_register_lcdc(&bbcape7_pdata))
            pr_info("Failed to register Beagleboard LCD cape device\n");

      return;
}
```



# LCD Clocking Layout




$$LCD\_PCLK = \frac{LCD\_CLK}{CLKDIV}$$

# Touch Screen and Backlight Init Functions

- These init functions call the pin mux config function with the earlier defined initialized structures

```
/* Initialize and register tsc device */
static void tsc_init(void)
{
      int err;

      am335x_touchscreen_data.analog_input = 1;
      setup_pin_mux(tsc_pin_mux);
      err = am33xx_register_tsc(&am335x_touchscreen_data);
      if (err)
            pr_err("failed to register touchscreen device\
n");
}
```

```
/* Enable ehrpwm for backlight control */
static void enable_ehrpwm1(void)
{
      ehrpwm_backlight_enable = true;
      setup_pin_mux(ehrpwm_pin_mux);
}
```



# LCD Init Sequence in the EVM Init function

- Calling three functions, initialization of
  - Backlight
  - LCD
  - touchscreen

```
/* Called as part of board initialization, defined in MACHINE_START */
static void __init am335x_evm_init(void)
{
.
.
.
      enable_ehrpwm1();
      bbcape7lcd_init();
      tsc_init();
.
.
}
```

# DO LAB 4…….

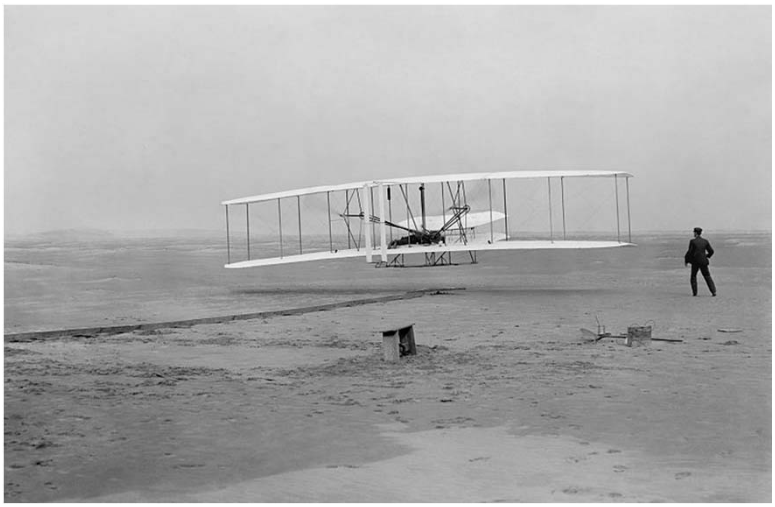
Texas Instruments

## Summary Lab 4

- LCD required 3 functions to be configured, Backlight, Touch Screen and LCDC
  - required device initialization data
  - required init functions
  - required pin mux configurations
- Made additions to the board file and the frame buffer support file

Texas Instruments

## So…. does it work yet?  Works Enough!



Texas Instruments

# THANK YOU

Texas Instruments

# ADDITIONAL INFORMATION SOURCES FOR POST WORKSHOP REVIEW

Texas Instruments

SITARA™ ARM® PROCESSORS

# Boot camp

Texas Instruments

## U-Boot 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 on an EVM platform.

July 2012

## 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

## Background for this Workshop

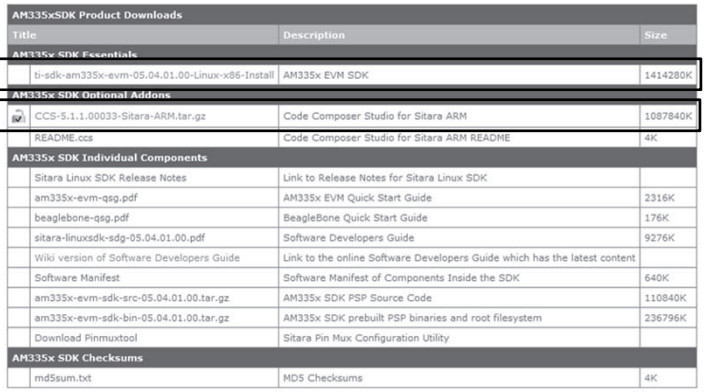
- Understand the SPL/U-Boot/Kernel boot process
- Knowledge of the Sitara Linux SDK and that it contains a cross compiler for the target device, CCSv5.1 and the source code for SPL/U-Boot/Kernel
- Have run the setup scripts in the Sitara Linux SDK to configure the target to boot the Linux kernel from 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



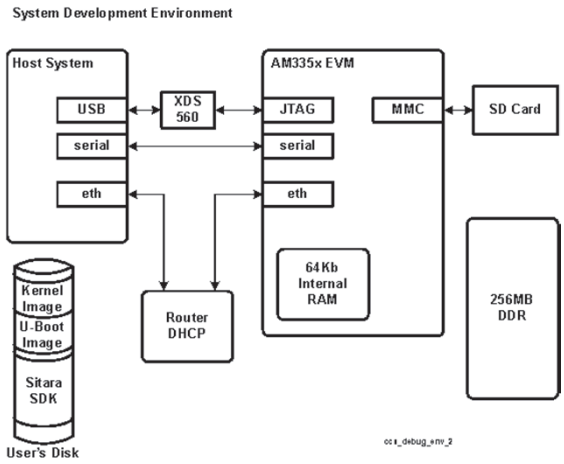
# CCS Installation – Key things to know..

- 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

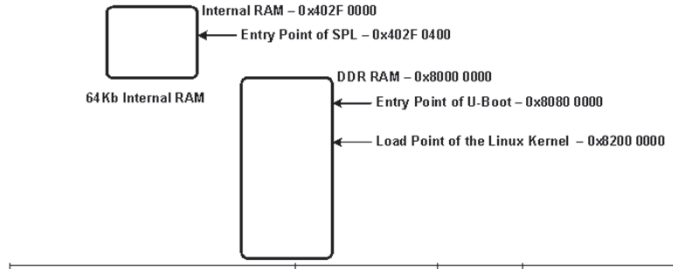


# AM335x Image Load Addresses

- 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 0x402F0400
  - For U-boot - CONFIG_SYS_TEXT_BASE 0x80100000
- 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)

Texas Instruments

## U-BOOT DEBUG LAB

Texas Instruments

# U-Boot/SPL building

- Have to build U-Boot to get an binary SPL and ELF U-Boot image to work with.
  - Add the debug information in the binary SPL and ELF U-Boot
    - Modify the ti-sdk-am335x-evm-05.05.00.00/board-support/u-boot-2011.09-psp04.06.00.08/config.mk:
      - » 278 ALL_AFLAGS = $(AFLAGS) $(AFLAGS_$(BCURDIR)/$(@F)) $(AFLAGS_$(BCURDIR)) **-g**
      - » 279 ALL_CFLAGS = $(CFLAGS) $(CFLAGS_$(BCURDIR)/$(@F)) $(CFLAGS_$(BCURDIR)) **-g**
  - Build the image
    - Follow the guide in http://processors.wiki.ti.com/index.php/AM335x_U-Boot_User%27s_Guide#Building_U-Boot.

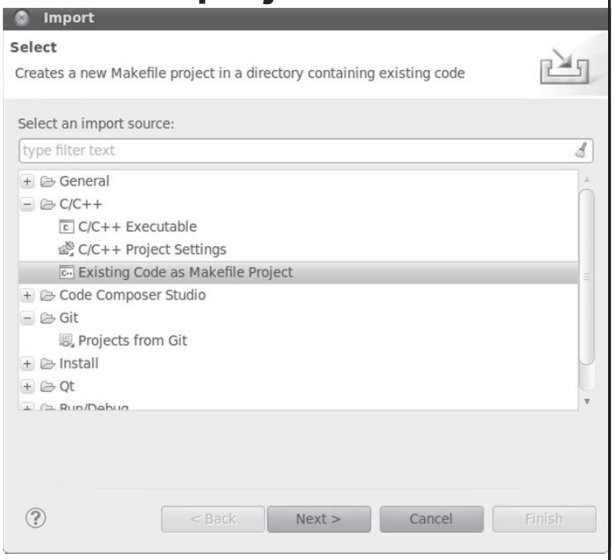


# U-Boot/SPL building Cont.

- Produced images
  - Locations: ti-sdk-am335x-evm-05.05.00.00/board-support/u-boot-2011.09-psp04.06.00.08/am335x
  - Images
    - MLO, u-boot.img are the SPL/U-Boot used for boot.
    - u-boot, U-Boot ELF image, also includes the symbol information.
    - u-boot.map, contains the memory map for each symbol
    - u-boot-spl.bin, the binary of SPL
    - u-boot-spl, SPL U-Boot ELF image, also includes the symbol information
    - u-boot-spl.map, contains the information for each symbol

# CCS – Import the U-Boot project

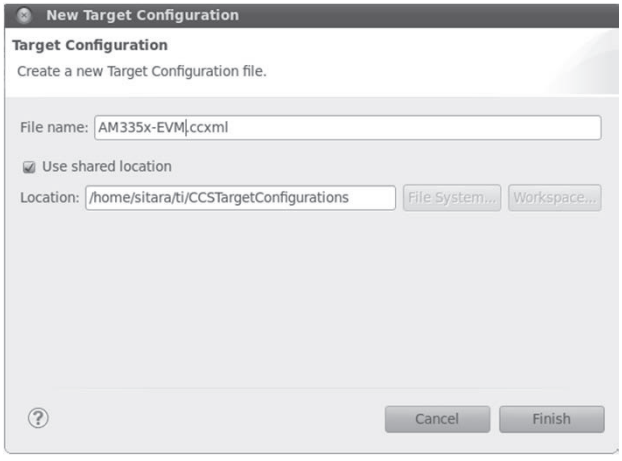
Menu File -> Import …



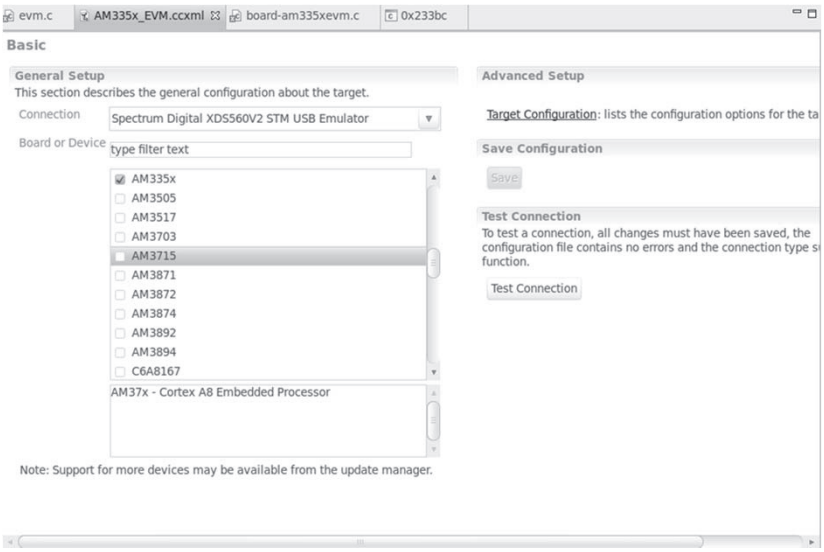



# CCS – Configure the target

View -> Target Configurations



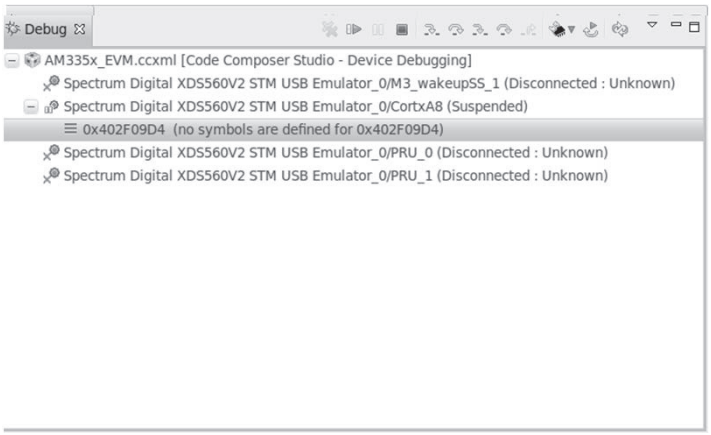
# CCS – Configure target – cont.





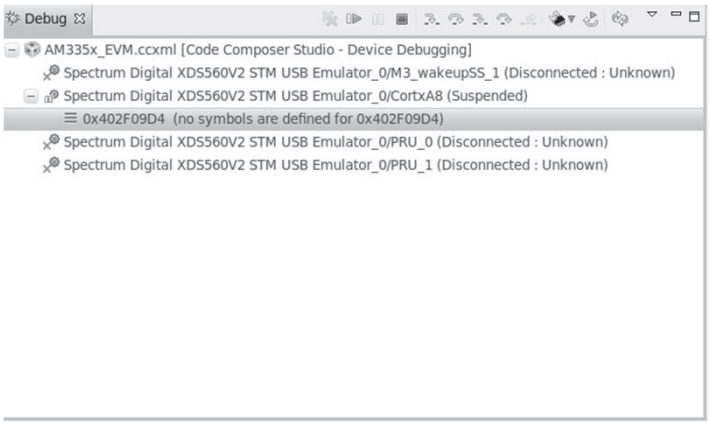

# CCS – Connect to the Cortex-A8 core

Right click on the target, click Launch Selected Configuration.
When it successes, it shows the following UI.
Now, the CCS successfully connect to the emulator.



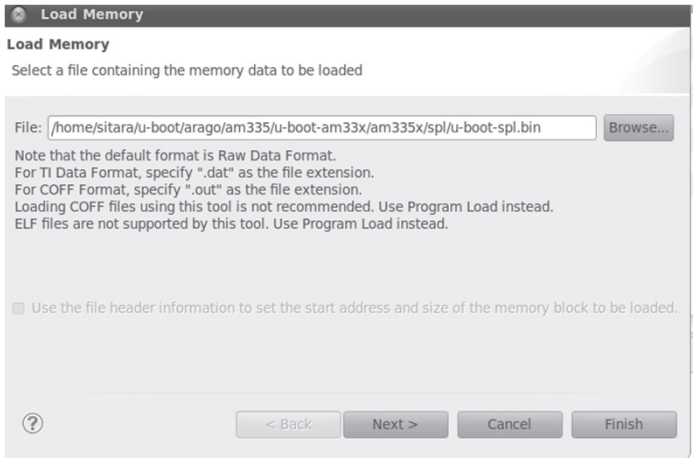
# CCS – Connect to the Cortex-A8 core, cont

Right click on the CortxA8 in last page, click Connect Target. Get the following picture When it is ready.
Now the CCS connect to the Coretex-A8 Core correctly.



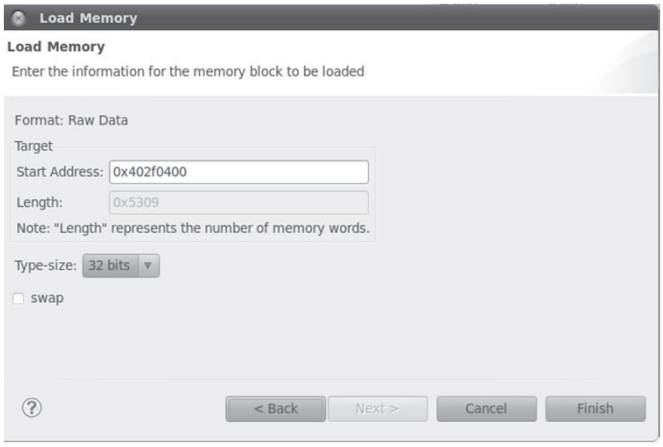

# SPL Debug – Load the image

Select the CortxA8 core of the target.
Menu Tools -> Load memory, select the u-boot-spl.bin, the binary of the SPL



# SPL Debug – Load the image cont.

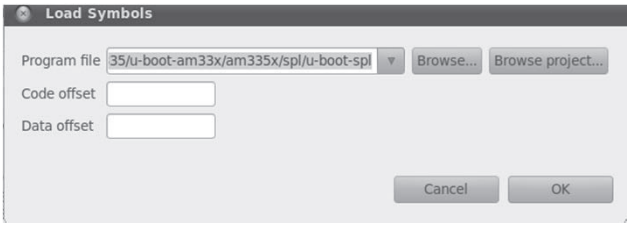
Start Address refers to 0x402f0400 mentioned as before.
Type-Size is 32 bit. This is because it is ARM code, not Thumb code.
The memory loading may fail for the first time, then trying it again will be OK.





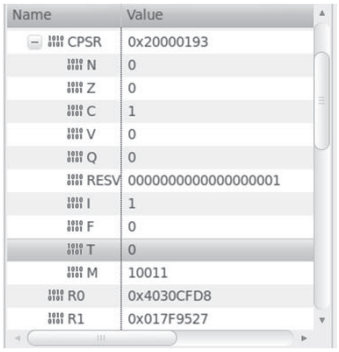

# SPL Debug – Load symbols

Menu Run -> Load -> Load Symbols
Choose the image with symbols' information, u-boot-spl



# SPL Debug – Change ARM core mode

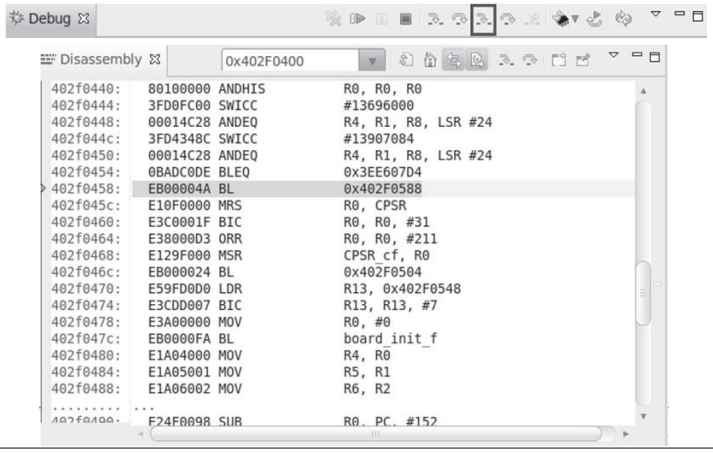
The initial status for the Cortex-A8 core here is Thumb, while, the system running On the SOC is in ARM mode.
Menu View -> Registers. Change T bit to 0.






# SPL Debug – Start to debug

- The SPL binary was loaded to 0x402F0400, and this is the start point for SPL. So the PC of the ARM core should be reset to this address. This operation is the same as to set the ARM mode in last slide.
- Click the Assembly Step Into button as in Red, and you can find the PC jumps to the code as below.



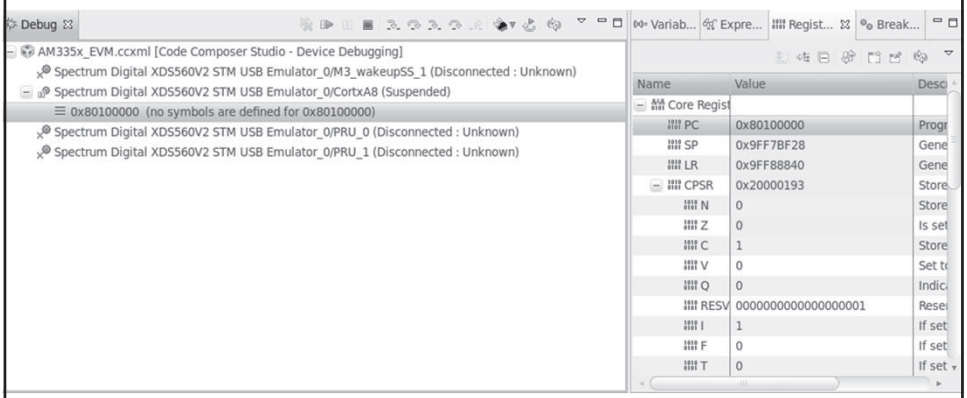
# Uboot Debug

- In general, the CCS debugging for SPL and Uboot has the same procedures in most steps, but there are still some differences. So only list the difference here
  - For U-boot, it can only run when the SPL finishes the DDR and other related low level initialization. So when debugging U-Boot, the SPL needs to be executed firstly.
  - U-Boot ELF image can be used directly for debugging.
  - There is the code relocation in the U-Boot, that is, part of the code will be relocated to higher memory. So the code memory map will be switched to an offset.



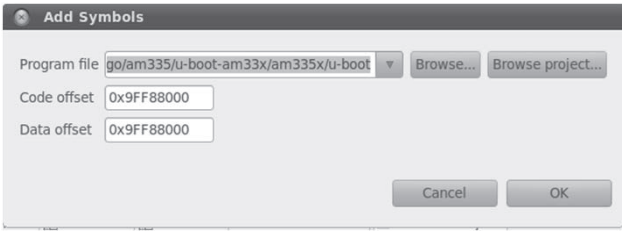
# Uboot Debug – Load image

- Here, the U-Boot ELF image can be loaded directly.
- Menu Run -> Load -> Load Program, choose u-boot.
- You can see that load address is the one mentioned previous, which is extract from ELF header.



# U-boot – Code relocation

- The code relocation is done in the function relocate_code(), which is called by board_init_f(). And the code offset here is 0x9FF88000. So if need to debug the code after relocate_code(), the symbol relocation is necessary before relocate_code().
- Menu Run -> Load ->  Add Symbols, the image is still u-boot,  and data/code offset is 0x9FF88000.






# SPL/U-Boot Debug -- Tips

- Although the debug information, the breakpoint can't be placed in the assembly code. So if want to place the breakpoint in the assembly code, it is necessary to find the entry address for the assembly function, which can be found in the memory map file, and read the code in disassembly window to find the correct place.
- Not every line of C code can place the breakpoint, so it can use the above approach as well.

# Q & A

Texas Instruments

# Thank you!

Texas Instruments

SITARA™ ARM® PROCESSORS

Boot camp

Texas Instruments

## Sitara Resource Introduction

This session means to make it clear for you to have an overview about Sitara resources . This presentation is a guide to help you to get the concept for the device / platform quickly.

Resource Guide:
http://processors.wiki.ti.com/index.php/AM335x_Resource_Guide(English/%E4%B8%AD%E6%96%87)

2012

## Agenda

- Hardware Design Resource
- Software Design Resource
- Tools & Utilities for AM355x
- Online support

Texas Instruments

## Hardware Reference Resource

- Symbol for AM335x
  - http://www.ti.com/product/am3359#symbols

  For EVM schematic and PCB Layout.
  - Beaglebone http://beagleboard.org/hardware/design/
  - GP EVM http://www.ti.com/tool/tmdxevm3358
  - StarterKit http://www.ti.com.cn/tool/cn/tmdssk3358

- BSDL/IBIS model for both ZCZ and ZCE package
  - http://processors.wiki.ti.com/index.php/Device:AM335x:Device_Evaluation#Important_Documentation



## Hardware Design Guide

- Schematic & PCB layout checklist
  - http://processors.wiki.ti.com/index.php/AM335x_Schematic_Checklist
  - http://processors.wiki.ti.com/index.php/AM335x_Layout_Checklist

- DDR2/3 design & layout guide
  - More detail in AM335x Datasheet.

- Multiple options to choose Power management Chips based on Cost Feature requirements.
  - http://processors.wiki.ti.com/index.php/Device:AM335x:Device_Evaluation#Power

## Power Solution for AM355x

- We have provided power-consumption summary
  - http://processors.wiki.ti.com/index.php/AM335x_Power_Consumption_Summary

- Power Estimation Tool
  - http://processors.wiki.ti.com/index.php/AM335x_Power_Estimation_Tool



## Hardware Migration Guide

- AM37x to AM335x Hardware Migration Guide
  - http://processors.wiki.ti.com/index.php/AM37x_To_AM335x_Hardware_Migration_Guide
- AM35x to AM335x Hardware Migration Guide
  - http://processors.wiki.ti.com/index.php/AM35x_To_AM335x_Hardware_Migration_Guide
- AM18x to AM335x Hardware Migration Guide
  - http://processors.wiki.ti.com/index.php/AM18x_To_AM335x_Hardware_Migration_Guide
- AM387x to AM335x Hardware Migration Guide
  - http://processors.wiki.ti.com/index.php/AM387x_To_AM335x_Hardware_Migration_Guide

## Sitara Linux SDK – OS Platform

- Based on a Arago Open project
  - http://www.arago-project.org/wiki/index.php/Main_Page
- Formal release
  - http://www.ti.com/tool/linuxezsdk-sitara
- The SDK User guide
  - In the installed folder.
  - http://processors.wiki.ti.com/index.php/Sitara_Linux_Software_Developer%E2%80%99s_Guide
  - More resources can be found in
  - http://processors.wiki.ti.com/index.php/Category:Linux



## AM335x Uboot Resource

- The Source Code
  - The formal release is inside the Ezsdk.
  - The OpenSource Project: http://arago-project.org/git/projects/?p=u-boot-am33x.git;a=summary
- The Guide for the AM335x u-boot

  http://processors.wiki.ti.com/index.php/AM335x_U-Boot_User%27s_Guide
- http://processors.wiki.ti.com/index.php/AM335x_board_bringup_tips

## AM335x Kernel Resource

- The User Guide. **This contains the guide for all the drivers**

  http://processors.wiki.ti.com/index.php/AM335x_PSP_User%27s_Guide

- The performance evaluation

  http://processors.wiki.ti.com/index.php/AM335x-PSP_04.06.00.07_Features_and_Performance_Guide

- The Source Code. (Now update to Linux Kernel 3.2)
  - The formal release is inside the Ezsdk.
  - The OpenSource Project: http://arago-project.org/git/projects/?p=linux-am33x.git;a=summary



## AM335x DDR Configuration

- AM335x support LPDDR/DDR2/DDR3

- AM335x EMIF Configuration Tips:
  - http://processors.wiki.ti.com/index.php/AM335x_EMIF_Configuration_tips

- AM335x DDR PHY register configuration for DDR3 :
  - http://processors.wiki.ti.com/index.php/AM335x_DDR_PHY_register_configuration_for_DDR3_using_Software_Leveling

## Android™ on AM335x – OS Platform

- **Rowboat (www.arowboat.org)**
  - **A community portal for Android on Sitara**
  - **Android 4.0 available on AM335x.**
  - **Mail Group, experts always on Line**


- **TI Android Development Kit**
  - **Stable periodic snapshots (approx. every 6 months)**
    http://www.ti.com/tool/androidsdk-sitara

Texas Instruments

## AM335x Android Documents

- **How to use the AM335x Android SDK**
  - http://processors.wiki.ti.com/index.php/TI-Android-ICS-4.0.3-DevKit-3.0.1_UserGuide
- **The Evaluation on the AM335x Android SDK**
  - http://processors.wiki.ti.com/index.php/TI-Android-ICS-4.0.3-DevKit-3.0.1_UserGuide
- **How to develop the AM335x Android SDK**
  - http://processors.wiki.ti.com/index.php/TI-Android-ICS-4.0.3-DevKit-3.0.1_DevelopersGuide
  - http://processors.wiki.ti.com/index.php/TI-Android-ICS-PortingGuide
- **More resources can be found in:**

http://processors.wiki.ti.com/index.php/Category:Sitara_Android

Texas Instruments

## Windows® Embedded – OS Platform

- **The WinCE package for AM335x**

  http://www.ti.com/tool/wincesdk-a8

- **The WinCE documents**

  - e2e support
    http://e2e.ti.com/support/embedded/wince/default.aspx



## RTOS on AM335x – OS Platform

- RTOS Overview

http://www.ti.com/lsds/ti/dsp/support/software/os_overview.page

- For the 3rd party RTOS, please contact the Vendor directly.

## Starterware on AM335x – Non-OS Platform

- **Where to download the AM335x starterware**
  - http://software-dl.ti.com/dsps/dsps_public_sw/am_bu/starterware/02_00_00_07/index_FDS.html
- **How to use the AM335x Starterware**
  - http://processors.wiki.ti.com/index.php/StarterWare_02.00.00.07_User_Guide
- **How to develop the AM335x Starterware**
  - http://processors.wiki.ti.com/index.php/AM335X_StarterWare_Booting_And_Flashing
  - http://processors.wiki.ti.com/index.php/AM335X_StarterWare_Environment_Setup

- **More resources can be found in** http://processors.wiki.ti.com/index.php/Category:StarterWare



## Development Environment

- Ubuntu 10.04 LTS downloading:

  http://www.ubuntu.com/download/desktop

- How to Build a Ubuntu Linux host under Vmware:
  http://processors.wiki.ti.com/index.php/How_to_Build_a_Ubuntu_Linux_host_under_VMware

# CCSv5

- The Sitara SDK also comes with CCSv5.1 available as:
  - Download for web updates
  - On the 3rd partition of the in-the-box SD card
- CCSv5 provides the following features
  - Eclipse based
  - Runs on Linux for debugging Linux
  - Multi-core debug
  - Run-Mode Linux debug support
    - Remote GDB debug
    - Debug Linux applications and kernel
  - Stop-Mode Linux support
    - Control target using JTAG
    - Enables examination of target and current Linux process
    - Requires installation of emulator package
- Download page :http://processors.wiki.ti.com/index.php/Category:CCS
- In addition to the standard CCSv5 package the Sitara SDK also adds the following plugins
  - Remote System Explorer





# Emulators for AM335x

- TI has different level emulators for AM335x
  - For the high end emulator
    - https://estore.ti.com/XDS560v2-System-Trace-P2124.aspx
  - For the low end emulator
    - http://www.ti.com/devnet/docs/catalog/embeddedsoftwarefulldetails.tsp?actionPerformed=productFolder&productId=10013
- The JTAG interface for Sitara, is not compatible with the ARM JTAG.
- The Document s for the emulators & JTAG
  - http://processors.wiki.ti.com/index.php/XDS560
  - http://processors.wiki.ti.com/index.php/XDS100
  - http://processors.wiki.ti.com/index.php/JTAG_Adapters

## Sitara online support

- The wiki page
  The hardware, software design guide are all on TI wiki as below
  – http://processors.wiki.ti.com/index.php/Sitara
  Resource Guide:
  – http://processors.wiki.ti.com/index.php/AM335x_Resource_Guide(English/%E4%B8%AD%E6%96%87)

- The online forum
  – E2E forum -- Global export will support you here
    - http://e2e.ti.com/support/dsp/sitara_arm174_microprocessors/default.aspx
    - http://e2e.ti.com/support/embedded/linux/default.aspx
    - http://e2e.ti.com/support/embedded/android/default.aspx
    - http://e2e.ti.com/support/embedded/starterware/default.aspx
    - http://e2e.ti.com/support/embedded/wince/default.aspx
  – DeyiSupport -- Local FAE will support you here
    - http://www.deyisupport.com/question_answer/f/25.aspx

Texas Instruments

# Q & A

Texas Instruments

# Thank you!

SITARA™ ARM® PROCESSORS

Boot camp

Texas Instruments

## Hands-on with the Sitara Linux SDK

This presentation provides a hands-on overview of the Sitara Linux SDK. It focuses on the software and tools found in the SDK and how to use these tools to develop for a Sitara device. This presentation is a guide to the actual hands-on demonstration.

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

2012

## Pre-work Check List

- ❑ Installed and configured VMWare Player v4 or later
- ❑ Installed Ubuntu 10.04 LTS
- ❑ Installed the latest Sitara Linux SDK
- ❑ Within the Sitara Linux SDK, ran the setup.sh (to install required host packages)
- ❑ Ready for the Sitara ARM Processors AM335x Starter Kit
  - ❑ Power supply with international adapter
  - ❑ 1 StarterKit Board
  - ❑ 2 Micro SD cards (Linux + Android)
  - ❑ 1 Micro-SD to SD card adapter
  - ❑ 1 USB 2.0 cable
  - ❑ AM335x Starter Kit Quick Start Guide

Texas Instruments

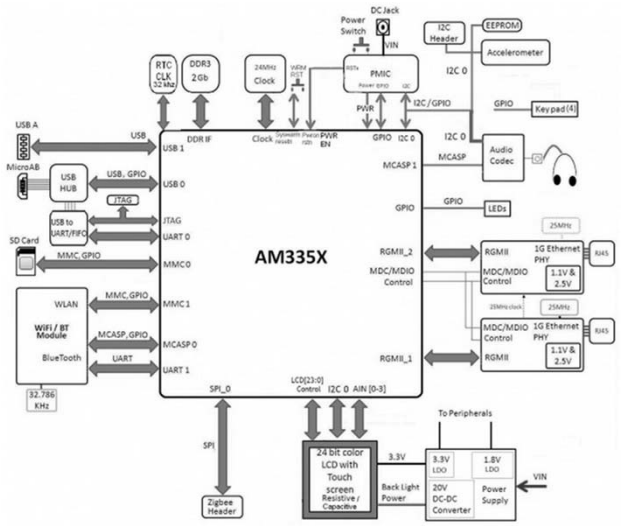
# Agenda

- Starter Kit Overview
- SDK Overview
- Obtaining the SDK
- Installing the SDK
- Sitara Linux SDK Directory Structure
- Building MLO and U-boot
- Building Kernel
- Creating an SD Card For EVM Booting



# Starter Kit Overview

http://processors.wiki.ti.com/index.php/AM335xStarterKitHardwareUsersGuide



## What is an SDK?

- Definitions and solutions vary, but we think an SDK should be….
- More than just a board support package (BSP) or a Demo
  - An SDK should also contain tools for developing on TI parts
  - Pre-built libraries that customer applications can link against without requiring building their own
  - Documentation
- Provide a known good starting point for product development
  - Start with the SDK reference distributions
  - Add and remove packages as needed

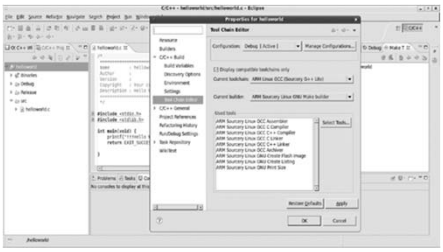


## Sitara Linux SDK Objectives

- The purpose of the Texas Instruments Sitara Linux SDK is to provide customers with a unique out-of-box experience and a quick path to their application development.
- The Sitara Linux SDK accomplishes this by providing
  - Example applications for key, high touch IP and peripherals
  - Tools for cross development and environment configuration
  - Host tools for device configuration
  - Documentation of SDK components
- Out-of-box in < 10 minutes and Development in < 1 hour
- A unified look and feel scaling across all Sitara devices from ARM9 to Cortex-A8 and beyond

# Sitara Linux SDK Overview

- Board Support Package (BSP)
  - Linux Kernel, Bootloaders & File System
- Documentation
  - Quick Start Guide & SW Developer's Guide
- Matrix Application Launcher
  - GUI based on PHP and HTML5
  - Touchscreen, Mouse & Keyboard, Web Browser
- Example Applications
  - ARM Benchmarks, Power, Multimedia, Camera, Cryptography, WLAN/Bluetooth, Profiling, 3D Graphics
- SDK Installer
  - Easy installation of all target and host software components and documentation
- CCSv5
  - Linux aware debug
  - Preconfigured projects for example applications
- Host Tools
  - Flash Tool, PinMux Utility

# Obtaining the SDK

- The Sitara SDK is provided on an SD card in the box with the EVM
  - NOTE: Some boards like the BeagleBoard and BeagleBone may come without the SDK SD card in the box
- Often you will want to download the latest SDK from ti.com to get updates.
- There is now a central location for finding all Sitara Linux SDKs at http://www.ti.com/tool/linuxezsdk-sitara

| Part Number | Texas Instruments | Status | Price (US$) | OS | Current Version | Version Date |
|---|---|---|---|---|---|---|
| **LINUXEZSDK-AM18X:** Linux EZ SDK for AM1808, AM1806, AM1802 | Get Software | ACTIVE | Free | Linux | v5.03.02 | 16 DEC 2011 |
| **LINUXEZSDK-AM335X:** Linux EZ SDK for AM3352, AM3354, AM3356, AM3357, AM3358, AM3359 | Get Software | ACTIVE | Free | Linux | v5.03.02 | 16 DEC 2011 |
| **LINUXEZSDK-AM35X:** Linux EZ SDK for AM3517, AM3505 | Get Software | ACTIVE | Free | Linux | v5.03.02 | 16 DEC 2011 |
| **LINUXEZSDK-AM37X:** Linux EZ SDK for AM3715, AM3703 | Get Software | ACTIVE | Free | Linux | v5.03.02 | 16 DEC 2011 |
| **LINUXEZSDK-AM389X-AM387X:** Linux EZ SDK for AM3871, AM3872, AM3874, AM3894, AM3892 - ALPHA | Get Software | ACTIVE | Free | Linux | v5.03.01 - ALPHA | 09 DEC 2011 |
| **LINUXEZSDK-BBXM:** Linux EZ SDK for BeagleBoard-xM | Get Software | ACTIVE | Free | Linux | v5.03.02 | 19 DEC 2011 |
| **LINUXEZSDK-BONE:** Linux EZSDK for BeagleBone | Get Software | ACTIVE | Free | Linux | v5.03.02 | 16 DEC 2011 |
| **LINUXSDK-AM17X:** Linux SDK for AM1707, AM1705 | Get Software | ACTIVE | Free | Linux | v1.10 - BETA | 12 MAY 2010 |
| **LINUXEZSDKAM1810:** Real-Time Linux Software Development Kit (SDK) for AM1810 | Get Software | ACTIVE | Free | Linux | v4.01 | 20 JUN 2011 |

# Obtaining the SDK – Cont.

- Each SDK page contains not only the single SDK installer, but also has many individual SDK components pulled out for smaller and quicker downloads

Single Installer

Individual Components

| AM335xSDK Product Downloads | | | |
|---|---|---|---|
| Title | | Description | Size |
| **AM335x SDK Essentials** | | | |
| | ti-sdk-am335x-evm-05.03.02.00-Linux-x86-Install | AM335x EVM SDK | 1205676K |
| **AM335x SDK Optional Addons** | | | |
| | CCS-5.1.0.08031-Sitara-ARM.tar.gz | Code Composer Studio for Sitara ARM | 1076748K |
| | README.ccs | Code Composer Studio for Sitara ARM README | 4K |
| **AM335x SDK Individual Components** | | | |
| | am335x-evm-qsg.pdf | AM335x EVM Quick Start Guide | 2316K |
| | beaglebone-qsg.pdf | BeagleBone Quick Start Guide | 176K |
| | sitara-linuxsdk-sdg-05.03.02.00.pdf | Software Developers Guide | 5284K |
| | Software Manifest | Software Manifest of Components Inside the SDK | 632K |
| | am335x-evm-sdk-src-05.03.02.00.tar.gz | AM335x SDK PSP Source Code | 109476K |
| | am335x-evm-sdk-bin-05.03.02.00.tar.gz | AM335x SDK prebuilt PSP binaries and root filesystem | 271480K |
| | Download Pinmuxtool | Sitara Pin Mux Configuration Utility | |
| **AM335x SDK Checksums** | | | |
| | md5sum.txt | MD5 Checksums | 4K |



# Installing the SDK

- The Sitara Linux SDK is delivered as a single installer
  - NOTE: The Sitara Linux SDK will also provide the option to install CCSv5 if the CCSv5 installer in found in the local directory. When installing from the SD card the CCSv5 installer is already placed in the local directory
- The Sitara Linux SDK installation has been streamlined to make installation quick and easy. The customer only needs to select:
  - The installation location
  - Whether or not to install CCSv5 if present
- The installer also notifies the user of important information such as
  - The default recommended environment
  - Notice of GPLv3 content within the SDK and information on how to remove it
  - The location of the setup scripts within the SDK for additional configuration
- The installer has been designed such that “root” permission is not required to install and evaluate the SDK.

# Sitara Linux SDK Directory Structure

- **bin** - Contains the helper scripts for configuring the host system and target device. Most of these scripts are used by the setup.sh script.
- **board-support** - Contains the SDK components that need to be modified when porting to a custom platform. This includes the kernel and boot loaders as well as any out of tree drivers.
- **docs** - Contains various SDK documentation such as the software manifest and additional user's guide. This is also the location where you can find the *training* directory with the device training materials.
- **example-applications** - Contains the sources for the TI provided example applications seen during the out-of-box demonstration.
- **filesystem** - Contains the reference file systems. These include the smaller base file system as well as the full-featured SDK file system.
- **host-tools** - Contains the host side tools such as pinmux and flash tool.
- **linux-devkit** - Contains the cross-compile toolchain and libraries to speed development for the target device.
- **Graphics_SDK_setuplinux_<version>.bin** - This is the installer for the graphics SDK. The graphics SDK components are used by the Sitara Linux SDK to provide additional demos as well as integrated with the pre-built Qt libraries to accelerate various Qt functions.
- **Makefile** - Provides build targets for many of the SDK components from the top-level of the SDK.
- **Rules.make** - Sets default values used by the top-level Makefile as well as sub-component Makefiles
- **setup.sh** - Configures the users host system as well as the target system for development



# Building MLO and U-boot

- Preparing to Build
  - export PATH="/usr/local/ti-sdk-am335x-evm/linux-devkit/bin:$PATH";
- Compiling MLO and u-boot
  - $ cd /usr/local/ti-sdk-am335x-evm/board-support/u-boot-2011.09-psp04.06.00.08
  - $ make O=am335xsk CROSS_COMPILE=arm-arago-linux-gnueabi- ARCH=arm am335x_evm
- You can find the MLO and u-boot.img ready in the current document.
- Reference:
  - 1. http://processors.wiki.ti.com/index.php/AM335x_U-Boot_User%27s_Guide
  - 2./usr/local/ti-sdk-am335x-evm/docs/sitara-linuxsdk-sdg-05.05.00.00.pdf

# Building Kernel

- Configuring the Kernel
  - make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi- am335x_evm_defconfig;

  or
  - make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi- menuconfig
- Compiling the Kernel
  - make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi- uImage
  - make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi- modules
- Installing the Kernel
  - make ARCH=arm CROSS_COMPILE=arm-arago-linux-gnueabi- INSTALL_MOD_PATH=/usr/local/ti-sdk-am335x-evm/targetNFS modules_install
- Reference:
  - 1.http://processors.wiki.ti.com/index.php/AMSDK_Linux_User%27s_Guide
  - 2. /usr/local/ti-sdk-am335x-evm/docs/sitara-linuxsdk-sdg-05.05.00.00.pdf



# Creating an SD Card For EVM Booting

- The create-sdcard.sh script is in /usr/local/ti-sdk-am335x-evm/bin.
- The create-sdcard.sh script must be run with root permissions.
  - host# sudo /usr/local/ti-sdk-am335x-evm/bin/create-sdcard.sh
- You will succeed in creating a 2 partition card by following the prompts.
- The 2 partition SD card need to be made as below:

| Partition No. | Format | Content |
|---|---|---|
| 1 | FAT32 | MLO, u-boot.img, uImage |
| 2 | Ext3 | The File System |

- Reference :
  - 1.http://processors.wiki.ti.com/index.php/Sitara_Linux_SDK_create_SD_card_script
  - 2./usr/local/ti-sdk-am335x-evm/docs/sitara-linuxsdk-sdg-05.05.00.00.pdf

# Q & A

Texas Instruments

# Thank you!

Texas Instruments

# TI Worldwide Technical Support

## Internet

TI Semiconductor Product Information Center

Home Page: support.ti.com

## TI Deyisupport Home Page

deyisupport.com

## Product Information Centers

**China**

Phone: 800-820-8682

Fax: +886-2-2378-6808

Email: tiasia@ti.com / ti-china@ti.com

Internet: support.ti.com/sc/pic/asia.htm