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:

July 2012
Creative Commons Attribution-ShareAlike 3.0 (CC BY-SA 3.0)

You are free:
- to Share – to copy, distribute and transmit the work
- to Remix – to adapt the work
- to make commercial use of the work

Under the following conditions:
- Attribution – You must give the original author(s) credit
- Share Alike - If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

With the understanding that:
- Waiver — Any of the above conditions can be waived if you get permission from the copyright holder.
- Public Domain — Where the work or any of its elements is in the public domain under applicable law, that status is in no way affected by the license.
- Other Rights — In no way are any of the following rights affected by the license:
- Notice — For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page.

CC BY-SA 3.0 License: http://creativecommons.org/licenses/by-sa/3.0/us/legalcode
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
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
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)
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. “
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

Architecture Specific
- ARM Cortex-A8
- 275/500/600/720 MHz
- 32K/32K L1 w/SED
- 256K L2 w/ECC
- 176K ROM 64K RAM

SOC Specific
- Display
  - 24-bit LCD controller (WXGA)
  - Touch screen controller
- PRU subsystem
  - PRU x2 200 MHz
  - 8K/8K w/SED
- System
  - eDMA
  - Timers x8
  - WDT
  - RTC
  - eHRPWM x3
  - eQEP x3
  - PRCM
- Memory interface
  - mDDR (LPDDR) / DDR2 / DDR3 (16-bit, 200 / 266 / 303 MHz)
  - NAND/NOR (16-bit ECC)

SOC Specific
- Serial
  - UART x6
  - SPI x2
  - I²C x3
  - McASP x2 (4 channel)
  - CAN x2 (Ver. 2 A and B)
  - USB 2.0 HS OTG + PHY x2
- Parallel
  - MMC/SD/SDIO x3
  - GPIO

Board Specific
- System Power
  - TPS
- DDR2
  - 1x16 256MB
- 10/100 Ethernet Phy
- USB Host
- uSD

Texas Instruments
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

  ti-sdk-am335x-05.05.00.00/

  - bin
  - board-support
  - docs
  - example-applications
  - filesystem
  - host-tools
  - linux-devkit
  - Makefile
  - Rules.make
  - sdcard
  - setup.sh

  ti-sdk-am335x-05.05.00.00/board-support/

  - extra-drivers
  - linux-3.2-psp04.06.00.07.sdk
  - prebuilt-images
  - u-boot-2011.09-psp04.06.00.07

  ti-sdk-am335x-05.05.00.00/board-support/u-boot-2011.09-psp04.06.00.08/

  ti-sdk-am335x-05.05.00.00/board-support/linux-3.2-psp04.06.00.08.sdk/

• 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
• 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

<table>
<thead>
<tr>
<th>Pad Config</th>
<th>Bot/Top Bal</th>
<th>IO Power</th>
<th>Mode 0</th>
<th>Mode 1</th>
<th>Mode 2</th>
<th>Mode 3</th>
<th>Mode 4</th>
<th>Mode 5</th>
<th>Mode 6</th>
<th>Mode 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>IO IEN OFF</td>
<td>E15 / -</td>
<td>VDDSHV6=3.3V</td>
<td>UART0 RXD</td>
<td>SPI1 CS0 M...</td>
<td>DCAN0 TX ...</td>
<td>I2C2 SDA M...</td>
<td>ECAP1 IN P...</td>
<td>PR1 PRU1 P...</td>
<td>PR1 PRU1 P...</td>
<td>GPIO1[10]</td>
</tr>
<tr>
<td>IO IEN OFF</td>
<td>E16 / -</td>
<td>VDDSHV6=3.3V</td>
<td>UART0 TXD</td>
<td>SPI1 CS1 M...</td>
<td>DCAN0 RX ...</td>
<td>I2C2 SCL M...</td>
<td>ECAP1 IN P...</td>
<td>PR1 PRU1 P...</td>
<td>PR1 PRU1 P...</td>
<td>GPIO1[11]</td>
</tr>
</tbody>
</table>

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

• Utility helps find conflicts, two pins are simultaneously selected

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

Pin Mux Utility User Guide

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

• Link to the U-Boot Template Source tree (clone this tree)

• PSP U-boot Repo
  – [http://arago-project.org/git/projects/?p=u-boot-am33x.git;a=summary](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:
  
  ```c
  #ifdef CONFIG_SPL_BUILD
      ...  // Code for SPL
  #endif

  #ifndef CONFIG_SPL_BUILD
      ...  // Code for U-Boot
  #endif
  ```
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

Lab Git Tree tag progression
tags are SDK <version>.-<modification>

- Base
  board-evm.c

- Create Template Board file
  05.04.01.00-template

- MMC enable
  05.04.01.00-mmc

- Ethernet enable
  05.04.01.00-ethernet
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

The DDR is set up within the SPL context
- enable_ddr_clocks in pll.c,
- ddrdefs.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
The SPL entry function

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

```c
/*
 *   early system init of muxing and clocks.
 */
void s_init(void)
{
    /* u-boot context */
#endif 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 */
    init_timer();
    preloader_console_init();
    config_am335x_ddr();
#endif
}
```

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.

```c
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);
}
```

- 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

```c
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

    gpme_init();

    return 0;
}
```

board/ti/am335x/evm.c
DO LAB 1........
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
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

```c
static struct module_pin_mux mii1_pin_mux[] = {
    {OFFSET(mii1_rxerr), MODE(0) | RXACTIVE},       /* MII1_RXERR */
    {OFFSET(mii1_txen), MODE(0) | RXACTIVE},        /* 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_txclk), MODE(0) | RXACTIVE},      /* MII1_TXCLK */
    {OFFSET(mii1_rxd), MODE(0) | RXACTIVE},        /* MII1_RXD */
    {OFFSET(mdio_data), MODE(0) | RXACTIVE | PULLUP_EN}, /* MDIO_DATA */
    {-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

```c
#define CONFIG_MMC

drivers/mmc/mmc.c

int board_mmc_init(bd_t *bis)
{
    omap_mmc_init(0);
    omap_mmc_init(1);
    return 0;
}

#define CONFIG_GENERIC_MMC

board/ti/am335x/evm.c
```

/* 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

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

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

```c
#include <linux/init.h>

static int __def_eth_init(bd_t *bis) {
    return 1;
}

int board_eth_init(bd_t *bis) __attribute__((weak, alias("__def_eth_init"))) {
    if (getenv("enetaddr"))
        return cpsw_register(cpsw_data);
    return 0;
}
```

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

```c
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
- “git diff” this is used to isolate code between git commits.

```
schuyler@morphus:~/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
+ifndef 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@morphus:~/bp_uboot/sitara-board-port-uboot$
```
**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
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.
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.

```c
+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?

```c
+ .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_IDO_LO);
+     mac_hi = __raw_readl(MAC_IDO_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\n");
+     }
+   }
+   
+   __raw_write(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

• 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
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.
OMAP2+ Machine Shared Common Code

- There are several board files in the mach-omap2 directory. These board files typically 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
- devices: Init calls, platform registration for most peripherals
- common: Init calls to define global address range for select interfaces
- clocks: Define clock domain mgmt functions
- control: OMAP2 control registers
- display: Display init calls, handles the differences between OMAP2,3 and 4
- gpio: Initialization function
- i2c: Reset and Mux functions
- mux: Defines a Pin Mux abstraction with supporting functions
- hsmmc: Init functions, hw and platform data
- sdrc: Init function for SDRC and SMS
- voltage: Voltage domain support functions
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.
LINUX BOARD PORT LABS
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
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.
The am33xx_init_early is a function within the OMAP2+ Shared common code.

This is called directly from the common code without modification.
• All three of these functions defined come from the OMAP2+ Shared Common Code, none of these needed to be modified.
MACHINE_START(AM335XEVM, "am335xevm")
/* Maintainer: Texas Instruments */
.atag_offset = 0x100,
.map_io = am335x_evm_map_io,
.init_early = am335x_init_early,
.init_irq = ti81xx_init_irq,
.handle_irq = omap3_intc_handle_irq,
.timer = &omap3_am33xx_timer,
.init_machine = am335x_evm_init,

MACHINE_END

static int am335x_rtc_init(void)
{  
  /* Initials RTC registers and registers with the kernel */
}

static void __init clkout2_enable(void)
{  
  /* Sets up pin mux, registers with kernel, enables clock*/
}

static void __init am335x_evm_init(void)
{
  am33xx_cpuidle_init();
  am33xx_mux_init(NULL);
  omap_serial_init();
  am335x_rtc_init();
  clkout2_enable();
}

arch/arm/mach-omap2/board-am335xevm.c

• 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
DO LAB 1........
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

<table>
<thead>
<tr>
<th>Mode 0</th>
<th>Mode 1</th>
<th>Mode 2</th>
<th>Mode 3</th>
<th>Mode 4</th>
<th>Mode 5</th>
<th>Mode 6</th>
<th>Mode 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMC0_DAT3</td>
<td>GPMC_A20</td>
<td>UART4_CTSN</td>
<td>TIMERS5_MUX0</td>
<td>UART1_DCD</td>
<td>PR1_PRU0</td>
<td>PR1_PRU0</td>
<td>GPI02[26]</td>
</tr>
<tr>
<td>MMC0_DAT2</td>
<td>GPMC_A2</td>
<td>UART4_RTSN</td>
<td>TIMERS6_MUX0</td>
<td>UART1_DSR</td>
<td>PR1_PRU0</td>
<td>PR1_PRU0</td>
<td>GPI02[27]</td>
</tr>
<tr>
<td>MMC0_DAT1</td>
<td>GPMC_A22</td>
<td>UART5_CTSN</td>
<td>UART3_RXD</td>
<td>UART1_DTR</td>
<td>PR1_PRU0</td>
<td>PR1_PRU0</td>
<td>GPI02[28]</td>
</tr>
<tr>
<td>MMC0_DAT0</td>
<td>GPMC_A23</td>
<td>UART5_RTSN</td>
<td>UART3_TXD</td>
<td>UART1_RI</td>
<td>PR1_PRU0</td>
<td>PR1_PRU0</td>
<td>GPI02[29]</td>
</tr>
<tr>
<td>MMC0_CLK</td>
<td>GPMC_A24</td>
<td>UART3_CTSN</td>
<td>UART2_RXD</td>
<td>DCA11_TX</td>
<td>PR1_PRU0</td>
<td>PR1_PRU0</td>
<td>GPI02[30]</td>
</tr>
<tr>
<td>MMC0_CMD</td>
<td>GPMC_A25</td>
<td>UART3_RTSN</td>
<td>UART2_TXD</td>
<td>DCA11_RX</td>
<td>PR1_PRU0</td>
<td>PR1_PRU0</td>
<td>GPI02[31]</td>
</tr>
<tr>
<td>SPI0_CS1</td>
<td>UART3_RXO</td>
<td>ECAP1_IN</td>
<td>MMC0_POW</td>
<td>XDMA_EVE</td>
<td>MMC0_SDC</td>
<td>EMU4_MUX1</td>
<td>GPI00[16]</td>
</tr>
<tr>
<td>MCASPO_AC</td>
<td>EQEPOA_IN</td>
<td>MCASPO_AX</td>
<td>MCAS1_P</td>
<td>MMC0_SDW</td>
<td>PR1_PRU0</td>
<td>PR1_PRU0</td>
<td>GPI03[18]</td>
</tr>
</tbody>
</table>

Have simplified the pin mux tool to show the pins necessary for the mmc0 interface
MMCC0 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
Lab 2 Board File Additions – Pin Mux Initialization Data

**Capture from the Pin Mux tool, AM3358 ZCZ package**

**Use existing pinmux_config struct to create pin mux initialization data for mmc0**

**Number of pins has to match**

```c
/* 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},
    {
        "mcaspo_aclk.mmc0_stdp", OMAP_MUX_MODE7 | AM33XX_PIN_INPUT_PULLUP},
    {
        "spi0_cs1.mmc0_sdc", OMAP_MUX_MODE7 | AM33XX_PIN_INPUT_PULLUP},
    {NULL, 0},
};
```

**Pin Mux definition for MMC0**

- 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

76
Lab 2 Board File Additions – MMC Device Initialization Data

Init Data for MMC0

```c
/* Convert SPIO 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, 10),
    .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

```c
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
Initialization Function Call Sequence for MMC Enabling

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

```c
MACHINE_START(AM335XEVM, "am335xevm")
/* Maintainer: Texas Instruments */
.slot_offset = 0x100,
.map_io = am335x_evm_map_io,
.init_early = am33xx_init_early,
.init_irq = ti81xx_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();
    clko ut2_enable();
    omap_sdr_c_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
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
[1.095784] mmcblk0: mmc0:1234 SA046 3.63 GbR
[1.102752] mmcblk0: p1 p2
[4.159171] kjournal: 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 ...
```

```
lnwxrwxrwx  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
```

```
-rw-r--r--  1 root root 4096 Dec 28 09:52 modalias
```

```
drwxr-xr-x  2 ruul ruul 0 Dec 28 09:52 power
```

```
lnwxrwxrwx  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 curiosity 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)
```

```
/media/mmcblk0p2 on /media/mmcblk0p2 type ext3 (rw,relatime,errors=continue,bARRIER=1,data=ordered)
/dev/mmcblk0p1 on /media/mmcblk0p1 type vfat (rw,relatime,fsmask=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
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.
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
DO LAB 2 ........
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
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

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

```c
/* Module pin mux for miii */
static struct pinmux_config miii_pin_mux[] = {
    "miil_txen.miil_txen", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLDOWN,
    "miil_rxdv.miil_rxdv", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLDOWN,
    "miil_txd3.miil_txd3", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT,
    "miil_txd2.miil_txd2", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT,
    "miil_txd1.miil_txd1", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT,
    "miil_txd0.miil_txd0", OMAP_MUX_MODE0 | AM33XX_PIN_OUTPUT,
    "miil_txclock.miil_txclock", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLDOWN,
    "miil_rxclock.miil_rxclock", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLDOWN,
    "miil_rxd3.miil_rxd3", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLDOWN,
    "miil_rxd2.miil_rxd2", OMAP_MUX_MODE0 | AM33XX_PIN_INPUT_PULLDOWN,
    "miil_rxd1.miil_rxd1", 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},
};
```
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

```c
void am33xx_cpsw_init_generic(unsigned int phy_type, unsigned int gigen) {
    u32 mac_lo, mac_hi;
    
    mac_lo = omap_ctrl1_readl(TI81XX_CONTROL_MAC_ID0_L0);
    mac_hi = omap_ctrl1_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_ctrl1_readl(TI81XX_CONTROL_MAC_ID1_L0);
    mac_hi = omap_ctrl1_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.

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

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

```
/* Called as part of board initialization, defined in MACHINE_START */
static void __init am335x_evm_init(void)
{
    
    mii1_init();
    am335x_cpsw_init_generic(MII_MODE_ENABLE,gigabit_enable);
}
```
**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 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...
[ 15.288b/4] PHY: 0:00 - Link is Up - 100/-u11
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..........
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

- 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

<table>
<thead>
<tr>
<th>Pad Config</th>
<th>Bot/Top Ball</th>
<th>IO Power</th>
<th>Mode 0</th>
<th>Mode 1</th>
<th>Mode 2</th>
<th>Mode 3</th>
<th>Mode 4</th>
<th>Mode 5</th>
<th>Mode 6</th>
<th>Mode 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEN OFF</td>
<td>A7 / -</td>
<td>VDDA ADC=</td>
<td>AIN3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IEN OFF</td>
<td>B7 / -</td>
<td>VDDA ADC=</td>
<td>AIN2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IEN OFF</td>
<td>C7 / -</td>
<td>VDDA ADC=</td>
<td>AIN1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IEN OFF</td>
<td>B6 / -</td>
<td>VDDA ADC=</td>
<td>AIN0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IEN OFF</td>
<td>B9 / -</td>
<td>VDDA ADC=</td>
<td>VREFP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IEN OFF</td>
<td>A9 / -</td>
<td>VDDA ADC=</td>
<td>VREFN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

/* 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

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

<table>
<thead>
<tr>
<th>Pad Config</th>
<th>Bot/Top 8 all</th>
<th>I/O Power</th>
<th>Mode 0</th>
<th>Mode 1</th>
<th>Mode 2</th>
<th>Mode 3</th>
<th>Mode 4</th>
<th>Mode 5</th>
<th>Mode 6</th>
<th>Mode 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1E1S PD</td>
<td>U14 /-</td>
<td>YEDS1V3=3.3V</td>
<td>GPMC A2 MUX3</td>
<td>GMI2 TxD3</td>
<td>RSMI2 TxD3</td>
<td>MMC2 DAT1 M...</td>
<td>GPMC A18 MUX3</td>
<td>PR1 MII1 TxD2</td>
<td>EHRPWM1A MUX...</td>
<td>GFC11181</td>
</tr>
</tbody>
</table>
Add LCD Panel – Data Initialization

- 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

```c
/* Define display configuration */
static struct lcd_ctrl_config bbcap7_cfg = {
  .ac_bias = 255,
  .ac_bias_inttrpt = 0,
  .dma_burst_sz = 16,
  .bpp = 16,
  .fdd = 0x00,
  .tft_alt_mode = 0,
  .stn_565_mode = 0,
  .mono_8bit_mode = 0,
  .invert_line_clock = 1,
  .invert_frame_clock = 1,
  .sync_edge = 0,
  .sync_ctrl = 1,
  .raster_order = 0,
};

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_inttrpt;
  /* 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_frame_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

- LCD Panel interfacing numbers have to be added in the da8xx-fb.c if they are not already defined.

- These numbers are derived from the datasheet for the panel (to name a few)
  - Screen resolution
  - Timings
  - Pixel Clock and Polarity

```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 */
};
```

Backlight Initialization Data

- PWM is used to control the LCD Panel Brightness

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

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

```c
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

```c
/* 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_clk');
    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;
}
```

```c
/* 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_lcd(&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

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

```c
/* 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 = am335x_register_tsc(&am335x_touchscreen_data);
    if (err)
        pr_err("failed to register touchscreen device\n");
}
```
LCD Init Sequence in the EVM Init function

• Calling three functions, initialization of
  – Backlight
  – LCD
  – touchscreen

```c
/* 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.........
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
So…. does it work yet?  Works Enough!
THANK YOU
ADDITIONAL INFORMATION SOURCES FOR POST WORKSHOP REVIEW