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 and u-boot.

LABS:

Sep 2012
Agenda

• Board Port Overview
• Porting U-Boot to an AM335x Target
• U-Boot Board Port Labs
Presentation Overview

• Goal is to gain an understanding of the components of a board port for U-Boot

• 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
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
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

[Diagram showing the flow from TI-SDK-AM335x-05.04.01.00 to a new target with Mem, Serial, Eth Phy, and Power boxes]
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
- ADC (8 channel)
- 12-bit SAR
- JTAG/ETB
- Crystal Oscillator x2
- Memory interface
  - mDDR (LPDDR) / DDR2 / DDR3
    - (16-bit, 200 / 266 / 303 MHz)
  - NAND/NOR (16-bit ECC)

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
- EMAC (2-port) 10M/100M/1G
- IEEE1588, and switch
  - (MII, RMII, RGMII)

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-05.05.00.00 directory

```plaintext
- bin
- board-support
- docs
- example-applications
- filesystem
- host-tools
- linux-devkit
- Makefile
- Rules,make
- sdcard
- setup.sh
- extradriver
- linux-3.2-psp04.06.00.07.sdk
- prebuilt-images
- u-boot-2011.09-psp04.06.00.07

- api
- arch
- board
- common
- disk
- doc
- drivers
- examples
- fs
- include
- init
- ipc
- kernel
- lib
- mm
- net
- patches
- samples
- scripts
- security
- sound
- tools
- usr
- virt```

- 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
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)
  – 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
  #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 preprocessor 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?
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
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`
- `ddr_def.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 */
#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 */
    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_init0();
#endif

    gpmc_init();

    return 0;
}
```

`board/ti/am335x/evm.c`
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

static struct module_pin_mux mii1_pin_mux[] = {
        {OFFSET(mii1_rxerr), MODE(0) | RXACTIVE},       /* MII1_RXERR */
        {OFFSET(mii1_t xen), 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_txd0), MODE(0)},                /* MII1_TXD0 */
        {OFFSET(mii1_rxclk), MODE(0) | RXACTIVE},     /* MII1_RXCLK */
        {OFFSET(mii1_rxd), MODE(0) | RXACTIVE},       /* MII1_RXD */
        {OFFSET(mii1_txd3), MODE(0) | RXACTIVE},      /* MII1_RXD */
        {OFFSET(mii1_txd2), MODE(0) | RXACTIVE},      /* MII1_RXD */
        {OFFSET(mii1_txd1), MODE(0) | RXACTIVE},      /* MII1_RXD */
        {OFFSET(mii1_txd0), MODE(0) | RXACTIVE},      /* MII1_RXD */
        {OFFSET(mdio_data), MODE(0) | RXACTIVE | PULLUP_EN}, /* MDIO_DATA */
        {OFFSET(mdio_clk), MODE(0) | PULLUP_EN},       /* MDIO_CLK */
    };
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"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")));

#define CONFIG_GENERIC_MMC

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

board/ti/am335x/evm.c

#define CONFIG_MMC

#define in"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")));

#define CONFIG_GENERIC_MMC

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

board/ti/am335x/evm.c

#define CONFIG_MMC

#define in"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")));

#define CONFIG_GENERIC_MMC

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

board/ti/am335x/evm.c

#define CONFIG_MMC

#define in"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")));

#define CONFIG_GENERIC_MMC

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

board/ti/am335x/evm.c

#define CONFIG_MMC

#define in"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")));

#define CONFIG_GENERIC_MMC

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

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

```c
/*
 * 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")));
```

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

int board_eth_init(bd_t *bis)
{
    oth_getenv_enetaddr(,)
    __raw_writel(MII_MODE_ENABLE, MAC_MII_SEL);
    return cpsw_register(&cpsw_data);
}
```
THANK YOU