Board porting up using Processor SDK RTOS for AM335x/AMIC110

ABSTRACT

This document describes the procedure to customize and bring up Processor SDK RTOS board package on custom AM335x/AMIC110 target hardware. This application walks user through key steps and components which users need to update and modify when bringing the software and application on the custom hardware.

Tutorial Environment:

Code Composer Studio™ (CCS)

Processor SDK RTOS AM335x x.x.x.x

TI supported AM335x evaluation hardware (AM335x GP EVM, AM335x ICEv2, AMIC110 ICE, etc)

Suitable emulator for the evaluation hardware

Pre-requisites:

The document assumes that user have already setup the Processor SDK RTOS on AM335x using the Getting started guide and are familiar with EVM setup and with software examples available.

Intended audience:

This document is primarily targeted for AM335x, AMIC110 and AM437x developers while porting their applications to custom hardware using Processor SDK RTOS. This document is also useful to Linux developers who want perform basic sanity check of underlying hardware using diagnostics software provide in Processor SDK RTOS.

Typical flow to bring up application on custom board

In this section we define a typical flow for customers to use to bring up custom hardware. Typically the bring up involves three stages:

1. **Checking out basic HW functionality:**
   This step involves both HW and SW steps to check the custom HW electrically and to ensure full address space is enabled for the end application.

- **HW steps :**
  - Check power sequencing and basic JTAG connectivity
  - Confirm Boot pin settings based on MAIN DEVSTAT and MCU DEVSTAT settings.
  - Create pinmux project, EMIF and PHY config for interfacing with DDR
- **SW steps:**
  - Update clocking and DDR settings in Gel file in CCS.
  - Load and run simple hello world example code out of OCRAM on MCU domain, MSMC on main domain and DDR.

2. **Updating board library in Processor SDK RTOS:**
   - Update clocking, pinmux and DDR settings and debug UART settings in board library
   - Confirm DDR stability using memtest and debug UART functionality using diagnostics.
   - Test as many peripherals using diagnostics/ bare-metal tests before bringing up drivers.
   - Test SBL functionality using SD/MMC for debug and then primary boot mode.

3. **Application porting**
   - Modify basic LLD driver examples to test functionality and update driver SOC configurations to match usage in final application.
   - Port prototype application on custom board and add additional functionality not available on evaluation platform.
   - Configure power management to setup SOC in optimal thermal and power conditions
   - Create production flashing setup to flash application boot on custom board.
   - Test and validate for any system issues.

In this document, we primarily focus on Step2 which relates to porting of the board library which is key step to port the Processor SDK RTOS from TI evaluation platform to custom HW.

Board Library Updates in Processor SDK RTOS:

Processor SDK RTOS component known as board library consolidates all the board-specific information so that all the modifications made when moving to a new custom platform using the SOC can be made in the source of this library.

There are three different components in PRSDK that help in porting and bring up of a custom board:

- Board library updates
    a. PLL Clocking and PRCM
    b. Pin mux Update
    c. DDR Configuration
    d. Peripheral instances updates
- Diagnostics tests
- Boot loader updates

PLL Clocking

There are two places where the device PLL configurations are performed when using Processor SDK RTOS and CCS.

Debug environment

Debug environment refers to development setup where code is debugged using JTAG emulator on the SOC. The PRSDK software relies on the GEL file that is part of the target configuration to setup the clocks and the DDR for the device. The CCS GEL file for AM335x platforms is located in the CCS package at the location ccsv7\ccs_base\emulation\boards\<boardName>

For example for beagle bone black, the files can be found at ccsv7\ccs_base\emulation\boards\beaglebone\gel

The GEL is the first piece of software that should be brought up on a custom board.

Production environment

Production environment refers to the setup when the base application is booted from a boot media like a flash memory or host interface. In this environment, the bootloader sets performs all the SOC and board initialization and copies the application from flash memory to the device memory.

The clock setup in the bootloader code can be located at pdk_am335x_x_x_x\packages\ti\starterware\bootloader\src\am335x

Users can choose to use the platform clocking similar to one of TI reference platforms or can modify them as per their application requirements. By default the PLL settings are setup for OPP_NOM settings (MPU= 600 MHz.)

TI provides **Clock Tree tool** to allow users to simulate the clocking on the SOC. For quick reference of the multiplier and divider settings to change the PLL setting is provided in the spreadsheet AM335x_DPLL_CALCv3.xlsx.

After modifying the clocking in the bootloader, users need to rebuild the bootloader using instructions provided in **Processor_SDK_RTOS_BOOT_AM335x/AM437x**

PRCM Modules Enable

PRCM Module Enable is required to turn on the power domain and the clocking to each of the modules on the SOC. The PRCM Enable calls to enable each module are made from the function Board_moduleClockInit which is found in the location.

pdk_am335x_x_x_x\packages\ti\board\src\bbbAM335x\bbbAM335x.c

Check every instance and peripheral required in the application platform and enable the module in the board library.

For example to use three UARTs 0, 1 and 4, ensure that you have the following code as part of the board library setup:

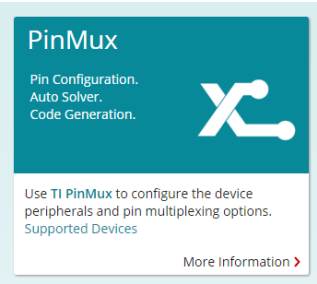
/* UART */

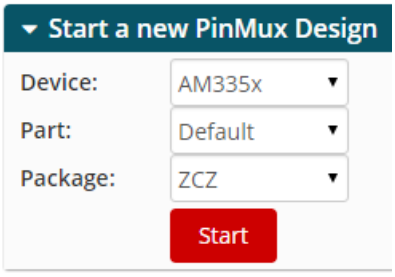
status = PRCMModuleEnable(CHIPDB_MOD_ID_UART, 0U, 0U);
status = PRCMModuleEnable(CHIPDB_MOD_ID_UART, 1U, 0U);
status = PRCMModuleEnable(CHIPDB_MOD_ID_UART, 4U, 0U);

Note: PRCMEnable function is defined in pdk_am335x_x_x_x\packages\ti\starterware\soc\am335x

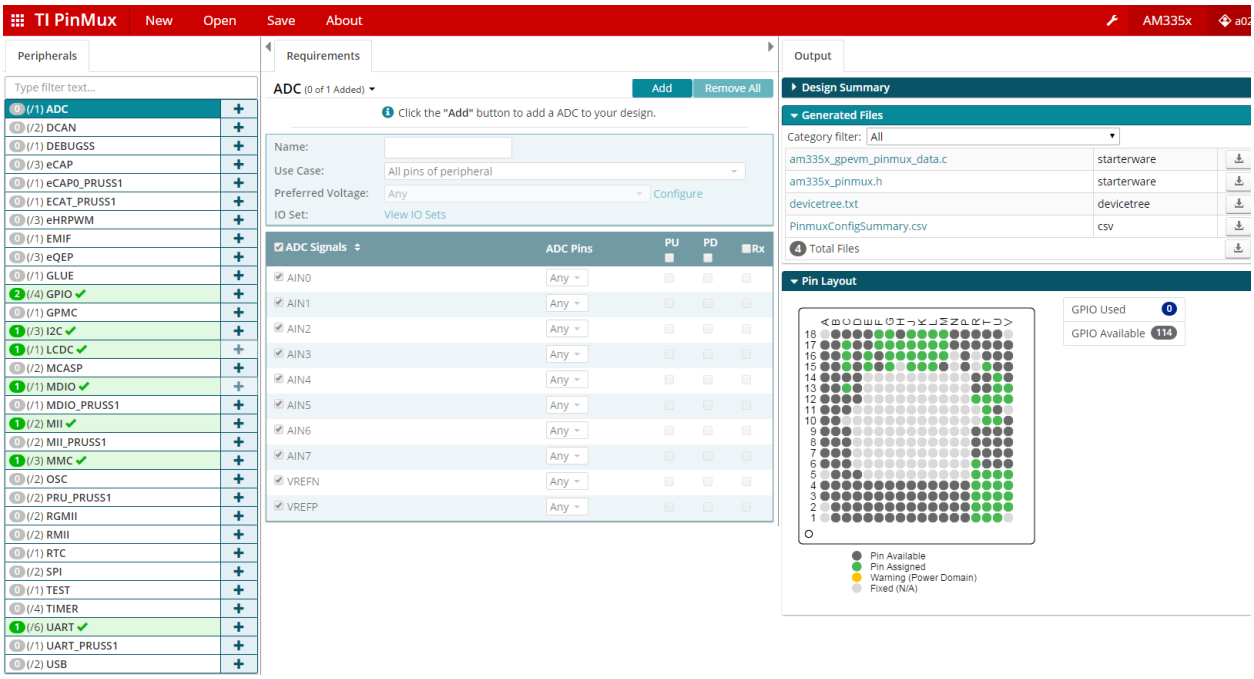
Pinmux updates in the Board library

Generating a New PinMux Configuration Using the PinMux Utility: This procedure uses the cloud-based pinmux utility





Navigate to ${PDK_INSTALL_DIR}\packages\ti\starterware\tools\pinmux_config\am335x and Load beaglebone_black_config



Add and remove peripheral instances and select the appropriate use cases required for development based on the application platform requirements and resolve all conflicts.

Refer [Pin_Mux_Utility_for_ARM_MPU_Processors]

Post Processing steps:

1. Change the Category filter to *starterware* and download the pinmux files am335x_pimnmux.h and am335x_pinmux_data.c
2. At the bottom of am335x_pinmux.h change extern pinmuxBoardCfg_t gAM335xPinmuxData[]; to extern pinmuxBoardCfg_t gBbbPinmuxData[];
3. Change am335x_pinmux_data.c to am335x_beagleboneblack_pinmux_data.c.
4. Change gAM335xPinmuxData to gBbbPinmuxData at the end of the file in file am335x_beagleboneblack_pinmux_data.c.
5. The last step is to invoke the PinMuxModuleConfig in the file <BoardName>_pinmux.c that is found at <PDK_INSTALL_PATH>\packages\ti\board\src\<BoardName>. For Example to add three instances of UART in the pinmux setup, users can add :

   \/ UART *\/*

   status = PINMUXModuleConfig(CHIPDB_MOD_ID_UART, 0U, NULL);

   status = PINMUXModuleConfig(CHIPDB_MOD_ID_UART, 1U, NULL);

   status = PINMUXModuleConfig(CHIPDB_MOD_ID_UART, 4U, NULL);

Replace the existing files with the new files and rebuild the board library using the instructions in the section [Rebuilding board Library in Processor SDK RTOS]

Updating DDR settings

Similar to clock and PLL settings, DDR initialization is configured in the Debug environment through GEL files and in production environment using bootloader source files.

TI provides AM335x_EMIF_Configuration_tips which contains a spreadsheet to enter the timing from the DDR datasheet to compute the EMIF timing number required to initialize DDR.

We strongly recommend changing the value and testing using GEL files before using them in the bootloader software. For Sanity test, you can perform read/write tests using CCS Memory Browser or run the diagnostic memory read/write test that we provide in diagnostics package here:

PDK_INSTALL_PATH\packages\ti\board\diag\mem

Once the DDR timings have been confirmed, you can use the settings in the file:

PDK_INSTALL_PATH \packages\ti\starterware\bootloader\src\am335x\sbl_am335x_platform_ddr.c

Peripheral initialization

The board library is responsible for most of the SOC initialization but it also setup some board level components such as ethernet PHY and debug UART and I2C for reading board ID from EEPROM. All of the other peripheral instances and initialization needs to be done from the application level.

For example for beagleboneblack, the peripheral initialization are performed from the source file pdk_am335x_x_x_x\packages\ti\board\src\bbbAM335x\bbbAM335x_lld_init.c

The debug UART instance, I2C Addresses are set using the file board_cfg.h found under:

pdk_am335x_x_x_x\packages\ti\board\src\bbbAM335x\include

Default UART instance is set to 0 in the board library. The Board initialization will configure the UART instance 0 to send binary log data to serial console using the Board_UARTInit function. If you wish to use more UART instances then we recommend linking in the UART driver in the application and using UART_open() and UART_stdioInit API calls from the application.

Each peripheral driver in the Processor SDK RTOS has a SOC configuration that provides the interrupt numbers, base address, EDMA channels which can be updated using the file <peripheral>_soc.c file. This is used as default setup for initializing the driver instance. It can be overridden from the application using peripheral_getSOCInitCfg() and peripheral_setSOCInitCfg()

For Example: All instances of UART for AM335x have been mapped in the file

pdk_am335x_x_x_x\packages\ti\drv\uart\soc\am335x\UART_soc.c

System integrators need to ensure that no interrupt numbers and EDMA resource conflicts exist in the SOC configuration for all drivers used in the system.

To exercise three UARTs in the system, users can use the following code:

```
//Setup Debug UART
boardCfg = BOARD_INIT_PINMUX_CONFIG |
           BOARD_INIT_MODULE_CLOCK  |
           BOARD_INIT_UART_STDIO;

Board_init(boardCfg);

// Open Additional UART Instances:
/* UART SoC init configuration */
   UART_initConfig(false);
   /* Initialize the default configuration params. */
   UART_Params_init(&uartParams);

// Open UART Instance 1
  uartTestInstance =1;
  uart1 = UART_open(uartTestInstance, &uartParams);

//Open UART Instance 4
 uartTestInstance = 4;
 uart4 = UART_open(uartTestInstance, &uartParams);
```

BoardID Detect

TI supports multiple evaluation and reference platforms for AM335x hence the hardware platforms are populated with an EEPROM which contains information that identifies the hardware and its revision. The board library and software components read the boardID and initialize the platform based on the boardID. The BoardID_detect function can be found in the source in the file bbbAM335x_info.c in the board library and board_am335x.c in the bootloader source at:

<PDK_INSTALL_PATH>\packages\ti\starterware\board\am335x

Rebuilding board Library in Processor SDK RTOS

While Creating a new folder for the custom board is an option users can explore, TI recommends that users make there changes in existing board package using either bbbAM335x, evmAM335x or iceAM335x folder to avoid spending additional effort to modify the build files for including the customBord.

Once all the update to the board library are completed, the board library can be updated using the following instructions.

Instructions to rebuild board library:

Setup Processor SDK build environment before following steps provided below.

cd pdk_am335x_x_x_x\packages

gmake board_lib

For a specific board users are required to provide the LIMIT_BOARDS argument.

LIMIT_BOARDS : evmAM335x icev2AM335x iceAMIC110 bbbAM335x skAM335x

For Example for beagleboneblack, users can use the following build option:

gmake board_lib LIMIT_BOARDS=bbbAM335x

Diagnostics

After the board library is built, we highly recommend that you create a diagnostics package similar to one provided in board library to test different interfaces functionally during board bring up.

The diagnostics package can be located at pdk_am335x_x_x_x\packages\ti\board\diag. These are simple bare-metal tests that use peripheral drivers to help functionally validate the pins and interfaces.

Documentation for all available diagnostic tests is provided here:

Processor_SDK_RTOS_DIAG

Bootloader in Processor SDK RTOS

As part of the production flow, users are required to develop/port flashing and booting utilities so the application can be launched on the custom board with JTAG. TI provides a bootloader mechanism where the ROM bootloader loads a secondary bootloader on the onchip memory that initializes the SOC and DDR and then copies the application into DDR memory.

The boot process and flashing tools have been described in detail in the following article that is part of processor SDK RTOS Software developer`s guide:

Processor_SDK_RTOS_BOOT_AM335x/AM437x#Building_the_Bootloader

Revision History