Application Development Using Processor SDK RTOS
Processor SDK RTOS: Software Stack

Let's start with basic, bare metal code to talk to peripherals. This code is OS Agnostic, great for OS porting, board diagnostics, etc. From this solid base, Platform or EVM software can be built, serving as functional examples of what customers will need to do and exercising lower-level code examples. Add in some cool DSP enablement.

Application OOB Demos

Industrial
HPC
Video
Audio
Industrial
Infotainment
Vision

Software Framework Components

Inter-Processor Communication
Framework Components
OS Abstraction Layer (OSAL)
NIMU

Network

NDK or 3rd Party Stack
TI RTOS Kernel

Algorithm Libraries

DSPLIB
IMGLIB
MATHLIB

TI RTOS/Bare Metal Drivers

EDMA3
ICSS-EMAC
PCIe
PRUSS
I2C
EMAC
USB
McSPI/QSPI
GPIO
UART
MMCSSE
.....

Platform/EVM Software

Secondary Bootloader
FATFS
Board Library
Diagnostics

Chip Support Library

Hardware

OS-Independent Software

OS-Independent Software
Processor SDK RTOS: Maximize Software Reuse

TI Demo Application on TI Evaluation Platform

- Demo Application
- Tools (UIA)
- EDMA, Etc.
- LLD
- IPC
- CSL
- TI Platform
- Network Dev Kit

Platform Migration

TI Demo Application on Customer Platform

- Demo Application
- Tools (UIA)
- EDMA, Etc.
- LLD
- IPC
- CSL
- Custom Platform
- Network Dev Kit

Application Migration

Customer Application on Customer Platform

- Custom Application
- Tools (UIA)
- EDMA, Etc.
- LLD
- IPC
- CSL
- Custom Platform
- Network Dev Kit

Future Proof

Custom App on Next Generation TI SOC Platform

- Customer Application
- Tools (UIA)
- EDMA, Etc.
- LLD
- IPC
- CSL
- Next Gen TI Platform
- Network Dev Kit

Software may be different. But API remains the same (CSL, LLD, etc.)
Processor SDK RTOS: Typical Development Flow

- **Setup**
  - EVM Kit, Processor SDK RTOS Package

- **Start**
  - Boot RTOS O/S, Start UART, Network, USB

- **Run**
  - Run SDK demo applications

- **Develop**
  - Develop application

- **Port**
  - Custom hardware bring up

- **Customize**
  - Customize application software
Processor SDK RTOS: Setup

Application Development Using Processor SDK RTOS
Processor SDK RTOS: AM572x GP EVM

- LCD Module
- Camera Module
- Processor Module
Removing the Processor Module from the LCD

NOTE: This is mandatory to connect an external emulator to the AM572x GP EVM.
Processor SDK RTOS: AM572x GP EVM Setup

- Connect emulator (only for debugging)
- Configure boot jumpers
- Connect Ethernet cable
- Insert SD card (only for SD boot & mass storage)
- Plug in FTDI cable for UART console out

Optional peripheral connections

CAUTION: EVM Power Up/Down Sequence (AM572x EVM Only)

Safe power up/ power down sequence:
Refer to wiki article for safe power up/down sequence:
AM572x_General_Purpose_EVM_HW_User_Guide

PMIC shutdown in 7 seconds:
- PMIC on the TMDXEVM5728 turns off the board in 7 seconds due to a hardware errata.
- Software needs to write to PMIC register to keep it on.
- GEL files and board library provide board configuration.

Errata: http://www.ti.com/product/AM5728
Processor SDK RTOS: Software Setup

Recommended host setup supported:
• Windows: Windows 7 on 64-bit machine
• Linux: Ubuntu 14.04 or 12.04 on 64-bit machine

Setting up the development environment:
  – Processor-SDK RTOS installer
  – Code Composer Studio v6.1.1 or later

NOTE: Mandatory CCS restart is required for product discovery to take effect.

- **Gel files for A15, C66x, M4 are auto-populated.**
- **Test connection option available.**
- **Advanced options allows customization.**

**CCS Edit View:** File ->New->Target Configuration

**CCS Debug View:** Launch target configuration

- Connect to CortexA15_0
- GEL initializes SoC clocks, DDR, PMIC
- All slave cores are in reset and need wake up
Processor SDK RTOS: Start

Application Development Using Processor SDK RTOS
Processor SDK RTOS: **Start**

1. **Init O/S, Interrupts, Timers**
2. **Start UART**
3. **Start Ethernet Driver**
4. **Start USB**
5. **Start RTOS Tasks**

**Boot RTOS O/S, Start UART, Network, USB**
Processor SDK RTOS: Bare Metal Hello World Example

• CCS “Hello World” template available.
• Template provided for all cores on SoCs.

For more details:
Processor SDK Bare Metal Examples
Processor SDK RTOS: SYSBIOS Hello World Example

Wiki Link: http://processors.wiki.ti.com/index.php/Processor_SDK_RTOS_Examples
Set Up Build Environment to Build PDK Components

Build instructions:

• Navigate to processor_sdk_rtos_<soc>_2_xx_xx_xx>
• Set environment variables:
  – SDK_INSTALL_PATH is SDK and CCS installation path.
  – Default sets it to “C:\TI” (Windows) & “/home/[user]/ti” (Linux).
• Run the script setup.bat (Windows) and source setupenv.sh (Linux)

Build all components:
make clean
make all

For other build target options:
http://processors.wiki.ti.com/index.php/Processor_SDK RTOS_Building_The_SDK

Custom installation options:
http://processors.wiki.ti.com/index.php/Processor_SDK RTOS_Install_In_Custom_Path
Script to Create Unit Tests for Device Drivers

pdkProjectCreate.bat [soc] [board] [endian] [module] [processor] [pdkDir]

(Windows)

pdkProjectCreate.sh [soc] [board] [endian] [module] [processor] [pdkDir]

(Linux)

File location: {PDK_INSTALL_DIR}\packages

Description:
soc – eg. am335x
board – refer ${PDK_INSTALL_DIR}\package\ti\board\lib
endian - little
module - all – eg uart
processor – eg arm, dsp
pdkDir - THIS FILE LOCATION

Example:
pdkProjectCreate.bat am572x evmAM572x little uart arm

Refer to PDK Example and Test Project Creation in the RTOS Software Developer Guide:
http://processors.wiki.ti.com/index.php/Rebuilding_The_PDK
Processor SDK RTOS: Set Up GPIO LED Example

GPIO example location:
pdk_1_x_x/packages/exampleProjects/GPIO_LedBlink_<soc>_evm_armExampleProject

- Import the project in CCSv6 and build the project.
- Connect the serial cable on host to view console.
- Host setup for serial console software:

  *Baud rate: 115,200
  *Data bits: 8
  *Parity: None
  *Stop bits: 1
  *Flow control: None

 UART console output

User LED blink output

GPIO LLD and example documentation:
http://processors.wiki.ti.com/index.php/Processor_SDK_RTOS_GPIO
Processor SDK RTOS: Set Up UART

Locate UART example:
pdk_1_0_0/packages/exampleProjects/UART_BasicExample_<SOC>_armTestproject

• Import the project in CCSv6 and build the project.
• Connect UART using FTDI or microUSB cable.
• Configure the serial terminal on host to view console.
• Host setup for Teraterm:
  *Baud rate: 115,200
  *Data bits: 8
  *Parity: None
  *Stop bits: 1
  *Flow control: None

Example output

UART LLD and example documentation:
http://processors.wiki.ti.com/index.php/Processor_SDK_RTOS_UART
**Processor SDK RTOS: Set Up USB Device**

- USB device instance will behave like a USB thumb drive.
- EVM DDR memory acts as storage to external host.
- Compile and run project under `pdk/packages/exampleProjects/usb_dev_msc_<BoardName>_arm_project`
- Connect USB cable to USB device port on EVM and to USB port on the PC.
- Hook up UART cable to PC to view console logs.
- PC detects the EVM hardware as USB mass storage and prompts user to format disk before using the device.

**Wiki Link:**
**Processor SDK RTOS: Set Up USB HOST (MSC)**

- USB instance acts as USB host communicating with a USB mass storage class device.

- Compile and run the following project under pdk/packages/exampleProjects
  usb_host_msc_<BoardName>_arm_project

- Plug in USB flash driver (FAT formatted) in the USB host port (USB0/1 on AM437x EVM).

- Connect UART cable to view example console prompt. Screenshot of example console is shown.

- Example demonstrates mass storage class functionality of the USB driver.

Wiki Link:
http://processors.wiki.ti.com/index.php/Processor_SDK_RTOS_USB
Processor SDK RTOS: Set Up Networking

Example Application:
NIMU_BasicExample_<SOC>_Evm_armExampleproject

- Import project into CCSv6 and build unit test.
- Load unit test via CCS using emulator.
- Example configures IP address 192.168.1.2 on the target.
- Before running:
  - Create interface on PC with static address 192.168.1.x
  - Hook up Ethernet cable from PC to Ethernet port on EVM.
    e.g., ETH0 interface. (top Ethernet port) on AM572x GP EVM
- To verify, ping 192.168.1.2 IP address (EVM board) from your host.

Wiki Link: http://processors.wiki.ti.com/index.php/Processor_SDK_RTOS_NDK
CSL Examples

• Chip Support Library (CSL):
  – Provides a set of well-defined APIs
  – Abstracts low-level interface details of underlying SoC
  – Allow users to configure, control (start/stop, etc.) and read/write from peripherals

• User can use the CSL layer to create examples and custom drivers.

• Example location: (TI_PDK_INSTALL_DIR)\packages\ti\csl\test

<table>
<thead>
<tr>
<th>Example Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WDT (Watchdog timer)</td>
<td>The application resets the A15 CPU0 core.</td>
</tr>
<tr>
<td>RTC (Real Time Clock)</td>
<td>The application prints date and time on UART console.</td>
</tr>
<tr>
<td>GMAC(External PHY)</td>
<td>The application prints on console the configuration of PHY.</td>
</tr>
</tbody>
</table>

Processor SDK RTOS: Run

Application Development Using Processor SDK RTOS
Creating SD Card to Boot SDK Demos

Script location in Processor SDK:

<SDK INSTALL DIR>/bin/create-sdcard.sh (Linux host only)

Notes:

• Linux script formats, partitions and loads the boot images to the SD card.

• Windows requires formatting, partitioning and copying of boot image using Win32 Disk Imager.

Location of prebuilt binaries for OOB demo images and sd-card image:

<SDK INSTALL DIR>/demos/oob/<SOC_EVM>/sd_card_img

Reference: Processor_SDK_RTOS_Creating_a_SD_Card_with_Windows

Processor_SDK_RTOS_create_SD_card_script_for_Linux
Processor SDK Demonstration: Image Processing Demo

- TI RTOS kernel based OOB demo demonstrates:
  - Booting from SD card using SBL,
  - UART, SD/MMC drivers
  - IPC messaging between ARM and DSP
  - IMGLIB functionality

- Application flow:
  - ARM reads the input image from SD card.
  - ARM partitions image across DSP cores.
  - ARM sends messages to DSP cores via IPC MessageQ.
  - DSP cores process partitioned images concurrently using IMGLIB edge detection functions.
  - DSP stores resulting image in DDR and notifies ARM cores.
  - ARM writes the resulting image into the SD card.

- Demo supports UART console logs and user input.
Processor SDK RTOS: Develop

Application Development Using Processor SDK RTOS
Boot RTOS O/S, Start UART, Network, USB

Link to UART LLD source to enable console output

Link to USB LLD location in package

Link to EMAC LLD, NIMU and NDK source location in package

Adding filesystem support to the application

Booting an application

IPC code to enable slave cores.
Processor SDK RTOS: Enabling UART

API Header Files:
- ti/drv/uart/UART_stdio.h
- board.h
- board_cfg.h

Sample Source Code:
```c
main()
{
    Board_initCfg boardCfg;
    boardCfg = BOARD_INIT_UART_STDIO;
    Board_init(boardCfg);
    UART_printf(" Text to output ");
}
```

Libraries to Link:
- ti.board.aXX
- ti.drv.uart.aXX

Where XX indicates target CPU

Wiki Link:
http://processors.wiki.ti.com/index.php/Processor_SDK_RTOS_UART
Processor SDK RTOS: Enabling USB Device

Sequence of APIs used to enable USB device

1. Populate tUSBDMSCDevice with vendor / device info and media access functions
2. Setup usb_params, usbClassData = tUSBDMSCDevice
3. USB_open()
4. Register USB interrupt handlers
5. USB_irqConfig()
6. Do other application tasks

USB device implementation in PDK

API header file:
- usb_drv.h
- usbdmsc.h

Libraries to link:
- ti.board.aXX
- ti.drv.usb.aXX

Where XX indicates target CPU

Wiki Link:
http://processors.wiki.ti.com/index.php/Processor_SDK_RTOS_USB
Processor SDK RTOS: Enabling USB Host

Sequence of APIs used to enable USB host

Start

1. Setup `usb_params`
2. `USB_open()`
3. Register USB interrupt handlers
4. `USB_irqConfig()`
5. `USBHMSCDriveOpen()` - This function registers a MSC call back function which then notify application of USB events
6. `USBHCDMain()`
7. Do MSC disc access (called by FATFS in example)
8. Do other application tasks

USB Host Mode example implementation in PDK

- `fs shell app`
- `FATFS LLD`
- `UART LLD`
- `MSC Disk Access Functions`
- `fatfs_port_usbmisc.c`
- `USB LLD`

API header file:
- `usb_drv.h`
- `usbhmsc.h`

Libraries to link:
- `ti.board.aXX`
- `ti.drv.usb.aXX`

Wiki Link:

Where XX indicates target CPU
Processor SDK RTOS: Enabling Networking

NIMU/EMAC header files:
ti/transport/ndk/nimu/nimu_eth.h

NDK header files:
ti/ndk/inc/netmain.h
ti/ndk/inc/stkmain.h

Libraries to link:
ti.transport.ndk.nimu.aXX
ti.ndk.config.<NDKModule>

Where XX indicates target CPU

Wiki Link: http://processors.wiki.ti.com/index.php/Processor_SDK_RTOS_NDK
Network Development Kit (NDK)

- NDK is a set of libraries + example code that initialize/configure/operate the hardware (EMAC) & perform all of the TCP/IP functionality through a set of “socket” programming APIs (e.g. socket, bind, send, recv, etc.)
- Provides a seamless interface to the physical layer (EMAC/PHY)

TCP/IP Model

- App
  - Transport
  - Network
  - Data
  - Physical

NDK Model

- App
  - NDK/EMAC
    - Sockets Programming Services
    - Internal stack functions
    - Configures stack/services and configures the EMAC
  - Physical
    - HTTP
    - TFTP
    - Telnet
    - DHCP
    - PPP
    - DNS
    - PPPoE
    - many others

What does the user touch?
- Configuration

Do you know all of the details of what is going on underneath?
- No

Would you like an example to play with?
- NIMU example in PDK...
Network Stack (NDK) System Overview

BIOS configuration file for NDK example:

Global Initializations
var Global = xdc.useModule('ti.ndk.config.Global');

Network layer modules:
var Ip = xdc.useModule('ti.ndk.config.Ip');

Transport layer modules:
var Tcp = xdc.useModule('ti.ndk.config.Tcp');
var Udp = xdc.useModule('ti.ndk.config.Udp');

Application layer modules:
var Telnet = xdc.useModule('ti.ndk.config.Telnet');

NDK Transport device driver(specific to device)
var Nimu = xdc.loadPackage('ti.transport.ndk.nimu');

Wiki Link: http://processors.wiki.ti.com/index.php/Processor_SDK_RTOS_NDK
Processor SDK RTOS: FATFS Filesystem Support

FATFS module driver enables device interface with FAT file system compatible device via the MMCSD, USB, etc.

**Header files:**
- ti/drv/FATFS/FATFS.h
- ti/drv/FATFS/ff.h

**Libraries to link:**
- ti.fs.fatfs.aXX

XX indicates the target CPU

**Examples:**
- \$(PDK_INSTALL_PATH)/packages/exampleProjects/FATFS_Console_<SOC>_Evm_armExampleProject

**Wiki Link:**
http://processors.wiki.ti.com/index.php/Processor_SDK_RTOS_FATFS

- **Configure application specific parameters. Enable clock and pinmuxing of peripheral.**
- **Application has to configure drive instances of FATFS for the driver specific functional configuration in FATFS_config.**
- **FATFS_Init(). This will create the handle for all instances of drives.**
- **FATFS_params_init()** This will initialize the parameters structure with default values. If other than default values then the parameters have to be overwritten.
- **FATFS_open(index, fatfsParams)** This will perform the configuration of driver controller for the specific instance based on the parameters and will return the handle corresponding to that instance.
- **FAT file system API like f_open, f_write, f_read, etc. can be used to perform file operations.**
- **FATFS_Close().**
Processor SDK RTOS: **Bootloader**

**SBL functions:**
- Sets up the PLL clock, pinmux
- Powers on the I/O Peripherals, initializes the DDR
- Loads the application image from memory device into DDR
- Brings the slave cores out of reset

*Wiki Link:*  
Bootloader: Multicore Application Image Creation

AM57xImageGen script for creation of bootable multi-core:

**Location**: $(PDK_INSTALL_DIR)/packages/ti/boot/sbl/tools/scripts

**Step 1**: Set BIN_PATH variable in environment for output.

**Step 2**: Set path to ARM, DSP and M4 binaries:
- **App_MPU_CPU0**: Path to location of A15 MPU application .out
- **App_IPU1_CPU0**: Path to location of M4 core 1 application .out
- **App_DSP1**: Path to location of DSP core 1 application .out

**Step 3**: Run the script to create app.out

**Tools used for image generation:**

- Convert ELF Images of application binary to rprc format.
  
  `out2rprc.exe <App_In_name(elf or coff)> <App_out_name>`

- Multi-core image generator:

  `MulticoreImageGen.exe <ENDIAN> <Dev Id> <App out file> <Core Id 1> <RPRC in file for Core Id 1> [<Core Id n> <RPRC in file for Core Id n> ...]`
Bootloader: Boot Media-Specific Details

SD/MMC boot:
1. Create a primary FAT partition on MMC/SD card (FAT32 format with sector size 512).
2. Rename the SBL image as MLO (RBL requirement) and copy to the SD card.
3. Rename the Application multicore image file as “app” and copy to the SD card.
4. Copy the MLO and application to the bootable SD card.
NOTE: SD card formatting tool is not included in SDK.

For other boot media-specific details:

http://processors.wiki.ti.com/index.php/Processor_SDK_RTOS_Boot
Processor SDK RTOS: IPC Examples

SOC IPC examples path:
IPC_DIR\examples\<SOC>_bios_elf

List of Examples:

MessageQ: Send round-trip message from client to server and back

Ping: Send a message between all cores in the system

NotifyPeer: Use notify to communicate to a peer processor

Hello Example: Send one-way messages from writer to reader


Processor SDK RTOS: Port

Application Development Using Processor SDK RTOS
Processor SDK RTOS: Port

May be used "as is" or customer can implement value-add modifications

Needs to be modified or replaced with customer version

No modifications required

Platform Migration

Texas Instruments
Processor SDK RTOS: Functional View

- Components that will definitely need modification
- Components that may need modification
Example code:

```c
// Setting up for pinmux and uart
Board_STATUS ret;
Board_initCfg boardCfg;
boardCfg = BOARD_INIT_MODULE_CLOCK |
          BOARD_INIT_PINMUX_CONFIG |
          BOARD_INIT_UARTSnackbar;
ret = Board_init(boardCfg);
```

API Header file:
“ti/board/board.h”

Library to link:
ti.board.aXX

See Application Integration for AM5x in the RTOS Software Developer Guide:
http://processors.wiki.ti.com/index.php/Processor_SDK_RTOS_Board_Support

See Custom Board Addition in the RTOS Software Developer Guide:
http://processors.wiki.ti.com/index.php/Processor_SDK_RTOS_Board_Support
Board Library: Modifying Source for Custom Platform

- PinMux
- Clocking
- DDR configuration
- IO configuration
- External components
- Board initialization
Processor SDK RTOS: Modifying Board PinMux Settings


AM57xx Sitara IO Configuration Requirements: [http://www.ti.com/lit/an/sprac44/sprac44.pdf](http://www.ti.com/lit/an/sprac44/sprac44.pdf)
Board Library: Clock Tree Tool to Simulate SoC Clocks

- Interactive Clock Tree Tool (CTT) for configuration:
  - Helps with visualization of the device clock tree
  - Allows users to customize clock tree as per specific use-case

- The CTT GUI is composed of 5 sub-views:
  - Main View
  - Thumbnail View
  - Controller View
  - Register View
  - Trace View

- Allows users to save register settings that can then be used to configure the software.

Clock Tree Tool Download: [http://www.ti.com/tool/CLOCKTREETOOL](http://www.ti.com/tool/CLOCKTREETOOL)
DDR Configuration Tools

Diagnostics: Tests to Bring up Custom Hardware

- Software to verify the functionality of on-board peripherals and external interfaces of each board.
- Constitute of ARM based bare metal (non-OS) code designed to validate TI EVM hardware
- Tests can be adapted to test new boards and/or peripherals.
- Validation suite utilizes:
  - board library for hardware configuration
  - UART drivers for standard output
  - relevant peripheral drivers for which the test are designed.
- Tests can be manually executed over an emulator or can be run off a SD card.

Diagnostics: Tests in the Board Package

Common tests:

- **UART**: Testing UART standard IO by sending/receiving characters at 115.2k baud
- **GPIO LEDs**: Flash the LEDs connected to GPIO on board
- **I2C LEDs**: Flash the LEDs connected to I2C on board
- **EEPROM**: Read/write to eeprom connected to I2C
- **DDR read/write**: Writes and reads back bits in the DDR memory
- **MCSPI**: Similar to QSPI, multichannel SPI also reads/writes to connected memory

For complete list of diagnostics for your SoC, refer to:

http://processors.wiki.ti.com/index.php/Processor_SDK_RTOS_DIAG
Processor SDK RTOS: Customize

Application Development Using Processor SDK RTOS
Processor SDK RTOS: Application Customization

Start with the example template of Image Processing demo

Add ARM or DSP algorithms, processing, tasks code

Customize and Run

Develop and run custom application
Example Application Template: Image Processing Demo

• Typical RTOS Application development starts from an existing template.
• CCS provide SYS BIOS application template with typical or minimal configurations.

Example application template for training:
processor_sdk_rtos_am57xx_2_xx_xx_xx_xx\demos\image_processing

Steps for building a custom application:
  – Include header files for all drivers and OS dependencies
  – Configure the BIOS configuration file to link to required driver libraries.
  – Creation of task for adding application functionality.
  – Porting and optimizing IPC configuration for communication with slave cores.
  – Add algorithm for processing.
Application Development: Includes and Initialization

Include required header files:

```c
/* TI CSL Header files */
#include <ti/csl/cslt_device.h>

/* SD/MMC and FAT FS Header files */
#include "MMCSD_log.h"
#include <ti/fs/fatfs/diskio.h>
#include <ti/fs/fatfs/FATFS.h>
#include <ti/drv/mmcscd/MMCSD.h>

/* UART Console IO header files */
#include <ti/drv/uart/UART.h>
#include <ti/drv/uart/UART_osal.h>
#include <ti/drv/uart/UART_stdio.h>

#include <ti/board/board.h>
```

Add headers for other drivers here.

Board initialization:

```c
Board_initCfg boardCfg;

boardCfg = BOARD_INIT_PINMUX_CONFIG | BOARD_INIT_MODULE_CLOCK | BOARD_INIT_UART_STDIO;
Board_init(boardCfg);
```

Create application tasks and custom algorithms here.

```c
/* Start BIOS */
BIOS_start();
return (0);
```
Add function gpio_test to the application source.
Application Development: Modifying Configuration Script

IPC libraries:

```c
xdc.useModule('ti.sdo.ipc.Tpc');
xdc.useModule('ti.sdo.ipc.MessageQ');
xdc.useModule('ti.sdo.ipc.SharedRegion');
xdc.useModule('ti.sdo.utils.MultiProc');
var HeapBufMP = xdc.useModule('ti.sdo.ipc.heaps.HeapBufMP');
```

Add other IPC modules here.

OSAL libraries for TI RTOS:

```c
/* Load the OSAL package */
var osType = "tirtos"
var Osal = xdc.useModule('ti.osal.Settings');
Osal.osType = osType;
```

Change default SYSBIOS settings here.

SoC platform and board libraries to link:

```c
/* Load the Board package and set the board name */
var Board = xdc.loadPackage('ti.board');
/* Board.Settings.boardName = "idkAM572x"; */
Board.Settings.boardName = "evmAM572x";
```

Driver libraries to link:

```c
/* Load the MMC3D package */
var Mmc3d = xdc.loadPackage('ti.drv.mmcsd');
var Fatfs = xdc.loadPackage('ti.fs.fatfs');
var UART = xdc.loadPackage('ti.drv.uart');
```

Add other drivers to link here.

Application Development: **Customize And Run**

- Driver instance and interrupt configuration
- Memory configuration
- Debugging
Application Development: **Customize Driver Instance**

<Module>_soc.c binds driver with Default Driver Attributes on the board.

Hardware attributes includes base address, interrupt number, etc.
Module behavior can be configured statically ... or dynamically during runtime.

**For Static configuration:**

```c
/* Number of GPIO ports */
#define CSL_GPIO_PER_CNT 8U

/* GPIO Driver hardware attributes */
GPIO_v1_HwAttrs GPIO_v1_hwAttrs[CSL_GPIO_PER_CNT] = {
    {
        CSL_MPU_GPIO1_REGS,
        #ifdef _TMS320C6X
        15,
        #else
        61,
        #endif
        0,
        55,
        0
    },
```

**Dynamic Runtime Configuration**

```c
GPIO_v1_HwAttrs *hwAttrs = NULL;
uint32_t portNum = 1;
hwAttrs = (GPIO_v1_HwAttrs *)&GPIO_v1_hwAttrs[(portNum - 1U)];
hwAttrs->line1IntNum = 62;
```

**NOTE:** The example shown refers to an ARM application.
Define Application Memory Map

SoC memory requires partitioning to allow all cores to have their own memory space and also to set up shared memory regions for cores.

Example: Application Memory Map

<table>
<thead>
<tr>
<th>Memory Segment</th>
<th>Start Address</th>
<th>Length</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCMC_SBL</td>
<td>0x40300000</td>
<td>112KB</td>
<td>SBL reserved L3</td>
</tr>
<tr>
<td>OCMC_0</td>
<td>0x4031C000</td>
<td>400KB</td>
<td>Shared L3 section 1</td>
</tr>
<tr>
<td>OCMC_1</td>
<td>0x40400000</td>
<td>1MB</td>
<td>Shared L3 section 2</td>
</tr>
<tr>
<td>OCMC_2</td>
<td>0x40500000</td>
<td>1MB</td>
<td>Shared L3 section 3</td>
</tr>
<tr>
<td>DDR3_Shared1</td>
<td>0x80000000</td>
<td>50MB</td>
<td>Shared DDR region</td>
</tr>
<tr>
<td>DDR3_MPU</td>
<td>0x83200000</td>
<td>50MB</td>
<td>ARM code/data</td>
</tr>
<tr>
<td>DDR3_DSP</td>
<td>0x86400000</td>
<td>50MB</td>
<td>DSP code/data</td>
</tr>
<tr>
<td>DDR3_M4</td>
<td>0x89600000</td>
<td>50MB</td>
<td>M4 code/data</td>
</tr>
</tbody>
</table>
Creating Custom RTSC Platform For BIOS Applications

Platform Definition in BIOS: $BIOS_INSTALL_DIR/packages/ti/platforms/<PlatformName>
Debugging SYSBIOS Applications

• SYSBIOS and IPC generate a highly optimized, minimally debug-able custom SYS/BIOS library that will link to your application.

• Building Debug-able SYSBIOS library in configuration file for your application:
  ```javascript
  var BIOS = xdc.useModule('ti.sysbios.BIOS');
  BIOS.libType = BIOS.LibType_Debug; // build custom BIOS library.
  BIOS.customCCOpts = BIOS.customCCOpts.replace("-o3", "-o0"); //change optimization level
  BIOS.customCCOpts = BIOS.customCCOpts.replace("--opt_for_speed=2", "); // For ARM only
  ```

• All PDK prebuilt libraries are built to support single-stepping into drivers and board libraries.

• In addition to single-stepping, [ROV tools](#), [RTOS analyzer](#) and [System Analyzer](#) tools in CCS can be used to view logs, task execution logs, and benchmark applications.
For More Information

Processor SDK Downloads:
- AM335x
- AM437x
- AM572x
- C667x
- C665x
- 66AK2Ex
- 66AK2Gx
- 66AK2Hx
- 66AK2Lx

Software Documentation:
- Processor_SDK_RTOS_Software_Developer_Guide

Hardware Wikis:
- AM335x EVM
- AM437x EVM
- AM572x EVM
- C6678 EVM
- C6657 EVM
- 66AK2Ex EVM
- 66AK2Gx EVM
- 66AK2Hx EVM
- 66AK2Lx EVM

Tools and Utilities:
- PINMUX Utility
- Clocking Tree Utility
- DDR Timing & Hardware Leveling
- PRU_ICSS

TI RTOS Trainings:
- TI RTOS Workshop
- Processor_SDK RTOS Overview