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

- **Application OOB Demos**
  - Inter-processor communication (IPC)
  - Framework Components
  - OS Abstraction Layer (OSAL)
  - NIMU
- **RTOS - Network**
  - TCP/IP Networking (NDK)
  - TI RTOS kernel
- **Platform/EVM Software**
  - Secondary bootloader
  - FATFS
  - Board Library
  - Diagnostics

- **Algorithm Libraries**
  - DSPLIB
  - IMGLIB
  - MATHLIB

- **TI RTOS/Bare Metal Drivers**
  - EDMA3
  - ICSS-EMAC
  - PCIe
  - PRUSS
  - I2C
  - EMAC
  - USB
  - McSPI/QSPI
  - GPIO
  - UART
  - MMCSD

- **Chip Support Library**

- **Hardware**
Processor SDK RTOS: Maximize Software Reuse

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

TI Demo Application on Customer Platform
- Demo Application
- Network Dev Kit
- LLD
- IPC
- Custom Platform
- CSL

Customer Application on Customer Platform
- Custom Application
- Network Dev Kit
- LLD
- IPC
- Custom Platform
- CSL

Custom App on Next Generation TI SOC Platform
- Customer Application
- Network Dev Kit
- LLD
- IPC
- Next Gen TI Platform
- CSL

Platform Migration
Application Migration
Future Proof

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

1. **Customize**
   - Customize application software

2. **Port**
   - Custom hardware bring up

3. **Develop**
   - Develop application

4. **Run**
   - Run SDK demo applications

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

6. **Setup**
   - Setup EVM
     - Connect UART, Network Cable, USB, Power

   EVM Kit, Processor SDK RTOS Package
Processor SDK RTOS: Setup
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 external emulator to AM572x GP EVM
Processor SDK RTOS – AM572x GP EVM Setup

- Connect Emulator (Only for Debugging)
- Configure Boot Jumpers
- Insert SD card (only for SD boot & Mass Storage)
- Connect Ethernet cable
- Plug In FTDI cable for UART Console out

Optional Peripheral connections

For EVM specific instructions here:
CAUTION: EVM Power Up/Down Sequence (AM572x EVM Only)

Safe power up/ power down sequence:
Refer 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 Development environment:
– Processor-SDK RTOS installer
– Code Composer Studio v6.1.1 or later

<table>
<thead>
<tr>
<th>CPU</th>
<th>Tool</th>
<th>TI-Software Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>A15</td>
<td>Linaro GCC Toolchain</td>
<td>CCS</td>
</tr>
<tr>
<td>A8</td>
<td>Linaro GCC Toolchain</td>
<td>CCS</td>
</tr>
<tr>
<td>A9</td>
<td>Linaro GCC Toolchain</td>
<td>CCS</td>
</tr>
<tr>
<td>M4</td>
<td>TI ARM Toolchain</td>
<td>CCS</td>
</tr>
<tr>
<td>C66x</td>
<td>TI CGT Toolchain</td>
<td>CCS</td>
</tr>
</tbody>
</table>

Software setup instructions:
Processor SDK RTOS: CCSv6 Product Discovery

Note: Mandatory CCS Restart is required for product discovery to take effect
Processor SDK RTOS: CCSv6 Target Configuration

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

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

**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

**EVM-specific instructions:**
Processor SDK RTOS: Start
Processor SDK RTOS - Start

- Init O/S, Interrupts, Timers
- Start UART
- Start Ethernet Driver
- Start USB
- 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

Refer: http://processors.wiki.ti.com/index.php/Processor_SDK_RRTOS_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** - 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_RTCOS_Building_The_SDK
Custom installation options:
http://processors.wiki.ti.com/index.php/Processor_SDK_RTCOS_Install_In_Custom_Path
Script to Create Unit Tests for Device Drivers:

```bash
pdkProjectCreate.bat [soc] [board] [endian] [module] [processor] [pdkDir]  
```
(Windows)

```bash
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`
Processor SDK RTOS – Setup 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_SDKRTOS_GPIO
**Processor SDK RTOS – Setup 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

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

**Example Output:**

![UART Example Output](image)

**UART LLD and Example Documentation:**

Wiki Link:

Processor SDK RTOS – Setup 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:
http://processors.wiki.ti.com/index.php/Processor_SDK_RTOS_USB
Processor SDK RTOS – Setup 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 described.
- 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 – Setup 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
    Eg. 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:
CSL examples:

- Chip Support Library provides 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>

Wiki Link:
http://processors.wiki.ti.com/index.php/Processor_SDK_RTOS_CSL
Processor SDK RTOS: Run
Creating SD Card To Boot SDK Demos:

Script location in Processor SDK:

<SDK INSTALL DIR>/bin/create-sdcards.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
Adding filesystem support to the application

Booting an application

IPC code to enable slave cores.

Boot RTOS O/S, Start UART, Network, USB
Processor SDK RTOS – Enabling UART

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

Sample Source code
```
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

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

Start

1. Populate tUSBDMSCDevice with vendor / device info and media access functions
2. Setup usb_params, usbcClassData = 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

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

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

### USB Host Mode Example implementation in PDK

1. **fs shell app**
2. **FATFS LLD**
3. **MSC Disk Access Functions**
   - `fatfs_port_usbmisc.c`
4. **UART LLD**
5. **USB LLD**

### API Header file:

- `"usb_drv.h"
- `"usbhmsc.h"

### Libraries to link:

- `ti.board.aXX`
- `ti.driv.usb.aXX`

**XX**: indicates target CPU

**Wiki Link:**

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>
XX: Indicates the target cpu

Wiki Link:
http://processors.wiki.ti.com/index.php/Processor_SDK_RRTOS_Network
**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**
- **Physical**

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

---

**What does the user touch?**
- Configuration

**Do you know all of the details of what is going on underneath?**
- Nope

**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:
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 to include:
- ti/drv/FATFS/FATFS.h
- ti/drv/FATFS/ff.h

Libraries to link:
- ti.fs.fatfs.aXX
  - XX : indicates the target CPU

Libraries to link:
- $(PDK_INSTALL_PATH)/packages/exampleProjects/FATFS_Console_<SOC>_Ev<SOC>armExampleProject

Wiki documentation:

Examples:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configure</td>
<td>application specific parameters. Enable clock and pinmuxing of peripheral</td>
</tr>
<tr>
<td>Application</td>
<td>has to configure drive instances of FATFS for the driver specific functional configuration in FATFS_config.</td>
</tr>
<tr>
<td>FATFS_Init()</td>
<td>This will create the handle for all instances of drives.</td>
</tr>
<tr>
<td>FATFS_params_init()</td>
<td>This will initialize the parameters structure with default values. If other than default values, then the parameters have to be overwritten.</td>
</tr>
<tr>
<td>FATFS_open(index, fparams)</td>
<td>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.</td>
</tr>
<tr>
<td>FAT file system API like fopen, fwrite, f_read, etc. can be used to perform file operations</td>
<td></td>
</tr>
<tr>
<td>FATFS_Close()</td>
<td></td>
</tr>
</tbody>
</table>
**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 documentation:**
http://processors.wiki.ti.com/index.php /Processor_SDK_RTOS_Boot
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
Processor SDK RTOS - Port

TI Application on TI Evaluation Platform

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

TI Application on Customer Platform

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

Platform Migration

- No modifications required
- May be used “as is” or customer can implement value-add modifications
- Needs to be modified or replaced with customer version
Processor SDK RTOS: Functional View

- Components that will definitely need modification
- Components that may need modification
Board Library: Configuration Options

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_UART_STDIO;
ret = Board_init(boardCfg);
```

API Header file:

```
“ti/board/board.h”
```

Library to link:

```
ti.board.aXX
```

Application Integration of Board library:

http://processors.wiki.ti.com/index.php/Processor_SDK_RTOS_Board_Support#Application_Integration

Creating Custom Board library:

http://processors.wiki.ti.com/index.php/Processor_SDK_RTOS_Board_Support#Creating_a_Custom_Board_Library
Board Library: Modifying Source For Custom Platform

- PinMux
- Clocking
- DDR configuration
- IO configuration
- External components
- Board initialization

<table>
<thead>
<tr>
<th>Name</th>
<th>Date modified</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>evmAM572x.c</td>
<td></td>
<td>C/C++ Header</td>
</tr>
<tr>
<td>evmAM572x_clock.c</td>
<td></td>
<td>C/C++ Header</td>
</tr>
<tr>
<td>evmAM572x_pll.c</td>
<td></td>
<td>C/C++ Header</td>
</tr>
<tr>
<td>board_padding.h</td>
<td>9/11/2015 2:07 PM</td>
<td>C/C++ Header</td>
</tr>
<tr>
<td>board_paddingDevice.c</td>
<td>9/11/2015 2:07 PM</td>
<td>C Source</td>
</tr>
<tr>
<td>board_paddingInit.c</td>
<td>9/11/2015 2:07 PM</td>
<td>C Source</td>
</tr>
<tr>
<td>board_paddingTune.h</td>
<td>9/11/2015 2:07 PM</td>
<td>C/C++ Header</td>
</tr>
</tbody>
</table>

External components:
Processor SDK RTOS: Modifying Board PinMux Settings.


Board Library: Clock Tree Tool To Simulate SoC Clocks

- Interactive clock tree configuration tool
- 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
  - Registers 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
**DDR Configuration Tools**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CS1 populated</strong></td>
<td>Yes</td>
</tr>
<tr>
<td>Chip-select interleaving</td>
<td>Yes</td>
</tr>
<tr>
<td>Turnaround time between chip-selects, in DDR clock cycles</td>
<td>3</td>
</tr>
<tr>
<td><strong>SRAM Data Bus Width</strong></td>
<td>512bits</td>
</tr>
<tr>
<td><strong>SRAM Low-power mode</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Clock Stop Timer</strong></td>
<td>Immediate</td>
</tr>
<tr>
<td><strong>Power-down Timer</strong></td>
<td>Immediate</td>
</tr>
<tr>
<td>Self-refresh Timer</td>
<td>Immediate</td>
</tr>
<tr>
<td>Deep power-down enable</td>
<td>No</td>
</tr>
<tr>
<td><strong>Max Number of LL Transactions in the Command FIFO</strong></td>
<td>0</td>
</tr>
<tr>
<td><strong>Max Number of MPU Transactions in the Command FIFO</strong></td>
<td>10</td>
</tr>
<tr>
<td><strong>Max Number of SYS Transactions in the Command FIFO</strong></td>
<td>10</td>
</tr>
<tr>
<td><strong>SRAM Clock Frequency</strong></td>
<td>133MHz</td>
</tr>
<tr>
<td>Operating Performance Point</td>
<td>OPP/MOM</td>
</tr>
<tr>
<td>Time between short ZQ calibration command</td>
<td>50ms</td>
</tr>
<tr>
<td>Perform a long ZQ calibration when exiting self-refresh</td>
<td>Yes</td>
</tr>
<tr>
<td>Perform a long ZQ calibration when exiting power down</td>
<td>No</td>
</tr>
<tr>
<td><strong>Enable write leveling</strong></td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Enable read gate training</strong></td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Enable data eye training</strong></td>
<td>Yes</td>
</tr>
<tr>
<td>Incremental write leveling interval</td>
<td>0Us</td>
</tr>
<tr>
<td>Incremental read DQS training interval</td>
<td>0Us</td>
</tr>
<tr>
<td>Incremental read data eye training interval</td>
<td>0Us</td>
</tr>
<tr>
<td>DQS gate training interval samples</td>
<td>8</td>
</tr>
<tr>
<td>Write leveling number of samples</td>
<td>7</td>
</tr>
<tr>
<td>Hardware Leveling</td>
<td>48</td>
</tr>
</tbody>
</table>
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

Commons Tests:

- **UART** – Testing UART standard IO by sending/receiving characters at 115.2k baud
- **GPIO LEDs** – Flashes the LEDs connected to GPIO on board.
- **I2C LEDs** – Flashes 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:

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

**Steps for building 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/cslr_device.h>

/* SD/MMC and FAT FS Header files */
#include "MMCSD_log.h"
#include <fs/fatfs/diskio.h>
#include <fs/fatfs/FATFS.h>
#include <drv/mmcsm/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_STIO;
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

```javascript
// Add modules
xdc.useModule('ti.sdo.ipc.Eipc');
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');
```

SoC Platform and Board Libraries to link

```javascript
// Load Board package
var Board = xdc.loadPackage('ti.board');
Board.Settings.boardName = "idkAM572x";
```

Add other IPC modules here

OSAL Libraries for TI RTOS

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

Driver Libraries to link

```javascript
// Load the MMCSD package
var Mmcst = xdc.loadPackage('ti.drv.mmcst');
var Fails = xdc.loadPackage('ti.drv.fails');
var UART = xdc.loadPackage('ti.drv.uart');
```

Change default SYSBIOS settings here

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 alternatively dynamically during runtime.

**For Static configuration:**

```c
#define CSL_GPIO_PER_CNT 8U

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 *)((portNum - 1U) & 0x1);
hwAttrs->lineIntNum = 62;
```

**Note:** Example refer to ARM application
Define Application Memory Map

SoC Memory requires to be partitioned to allow all cores to have their own memory space and also setup shared memory regions for cores.

Eg 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 your application will link with.

- 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.customCCOpt = BIOS.customCCOpt.replace("-o3", "-o0"); // change optimization level
  BIOS.customCCOpt = BIOS.customCCOpt.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:

Download:
- ProcSDK_RTNOS_AM335x
- ProcSDK_RTNOS_AM437x
- ProcSDK_RTNOS_AM572x

Software Documentation
- Processor_SDK_RTNOS_Software_Developer_Guide

Hardware Wikis
- AM335x EVM
- AM437x EVM
- AM572x 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