Introduction to Processor SDK RTOS Part 1
Agenda

• Processor RTOS SDK Overview
• TI-RTOS Kernel
• Inter-Processor Communication (IPC)
• Network Development Kit (NDK)
• Diagnostic Software
• Algorithm Libraries
• Drivers (Covered in Processor SDK RTOS Overview P2)
Processor RTOS SDK Overview

Introduction to Processor SDK RTOS
Processor SDK RTOS Development Ecosystem

Eclipse IDE

- Code Composer Studio™ (CCS)
- Third Party Plug-Ins

For CCS Training:
processors.wiki.ti.com/index.php/Category:CCS_Training

Target Board

Host Computer

Emulator
Processor SDK RTOS Install

• Each TI part has its own install page. For example:
  — AM335x: http://www.ti.com/tool/processor-sdk-am335x
  — AM57x: http://www.ti.com/tool/processor-sdk-am57x

• Click on the Get Software link and it will take you to the install page (like the one shown on the next slide).

• The Getting Started Guide and the Developer Guide show how to start developing Processor SDK RTOS-based applications.
### Processor SDK RTOS Install Page

**PART OF PROCESSOR SDK RTOS INSTALL PAGE**

<table>
<thead>
<tr>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AM57xx RTOS SDK Essentials</strong></td>
<td></td>
</tr>
<tr>
<td><code>ti-processor-sdk-RTOS-am57xx-evm-02.00.00.00-Windows-x86-Install.exe</code></td>
<td>AM57xx RTOS SDK installer for Windows Host</td>
</tr>
<tr>
<td><code>ti-processor-sdk-RTOS-am57xx-evm-02.00.00.00-Linux-x86-Install.bin</code></td>
<td>AM57xx RTOS SDK installer for Linux Host</td>
</tr>
<tr>
<td><strong>Download Code Composer Studio 6.1.1</strong></td>
<td>Code Composer Studio IDE (includes compiler)</td>
</tr>
<tr>
<td><strong>AM57xx RTOS SDK Optional Addons</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Download Pin Mux Tool</strong></td>
<td>AM572x Pin Mux Configuration Utility</td>
</tr>
<tr>
<td><strong>Download Clock Tree Tool</strong></td>
<td>AM572x Clock Tree Configuration Utility</td>
</tr>
<tr>
<td><strong>AM57xx RTOS SDK SD Card Creation</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Windows SD Card Creation Wiki</strong></td>
<td>Instructions for creating an SD Card with Windows Host</td>
</tr>
<tr>
<td><strong>Linux SD Card Creation Wiki</strong></td>
<td>Instructions for creating an SD Card with Linux Host</td>
</tr>
<tr>
<td><strong>AM57xx RTOS SDK Documentation</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Processor SDK RTOS Release Notes</strong></td>
<td>Link to Release Notes for Processor SDK RTOS</td>
</tr>
<tr>
<td><strong>Processor SDK Getting Started Guide</strong></td>
<td>Link to Getting Started Guide for Processor SDK RTOS</td>
</tr>
<tr>
<td><strong>Processor SDK RTOS Developer Guide</strong></td>
<td>Link to Developer Guide for Processor SDK RTOS</td>
</tr>
<tr>
<td><strong>Software Manifest</strong></td>
<td>Software Manifest of Components Inside the SDK</td>
</tr>
<tr>
<td><strong>AM57xx EVM Documentation</strong></td>
<td></td>
</tr>
<tr>
<td><strong>AM572x EVM Quick Start Guide</strong></td>
<td>Quick Start Guide that was included in the EVM kit</td>
</tr>
<tr>
<td><strong>AM57xx RTOS SDK Checksums</strong></td>
<td></td>
</tr>
<tr>
<td><code>md5sum.txt</code></td>
<td>MDS Checksums</td>
</tr>
</tbody>
</table>
Processor SDK RTOS: Overview

The RTOS (Real Time Operating System) perspective of the TI Processor SDK (Software Development Kit):

- Provides a set of software building blocks that facilitate development of (real-time) applications
- Consists of SoC (device) and platform dependent modules, Core dependent software, TI-RTOS kernel and utilities and application examples
- Includes source code and prebuilt libraries
- Embedded OS: TI-RTOS kernel for DSP, ARM, and M4
- Development OS: Windows and Linux PC support
- Available as a free download with all components in one installer
## Processor SDK Elements

<table>
<thead>
<tr>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implemented on top of the operating system and may be architecture dependent.</td>
</tr>
</tbody>
</table>

### Operating System Dependent Components

- TI-RTOS kernel, Tools, Utilities, Drivers

### Core-Specific / OS-Independent Components

- Optimized Libraries

### SoC -Dependent / OS-Independent Components

- device and platform drivers
Processor SDK RTOS

Single product supports multiple SoCs
## Processor SDK RTOS Release (AM335x)

<table>
<thead>
<tr>
<th>Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>processor_sdk_rtos_335_version</td>
</tr>
<tr>
<td>bios_version</td>
</tr>
<tr>
<td>cg_xml</td>
</tr>
<tr>
<td>edma3_lld_version</td>
</tr>
<tr>
<td>ndk_version</td>
</tr>
<tr>
<td>pdk_am335x_version</td>
</tr>
<tr>
<td>processor_sdk_rtos_am335x_version</td>
</tr>
<tr>
<td>xdctools_version_core</td>
</tr>
</tbody>
</table>
### Processor SDK RTOS Release (AM437x)

<table>
<thead>
<tr>
<th>Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>processor_sdk_rtos_437_version</td>
</tr>
<tr>
<td>bios_version</td>
</tr>
<tr>
<td>cg_xml</td>
</tr>
<tr>
<td>edma3_lld_version</td>
</tr>
<tr>
<td>ndk_version</td>
</tr>
<tr>
<td>pdk_am437x_version</td>
</tr>
<tr>
<td>processor_sdk_rtos_am437x_version</td>
</tr>
<tr>
<td>xdcctools_version_core</td>
</tr>
</tbody>
</table>
The AM57x release is a superset of Processor SDK RTOS features.
The **pdk** folder contains the platform development kit, which is a collection of CSL and low-level drivers that configure, manage the hardware, and providing I/O capabilities.

The **edma** folder includes multiple EDMA controllers, management drivers, and the resource manager.

The **framework components** folder includes a set of utilities to manage the target board hardware, memories, interfaces, etc.
The following folders contain optimized libraries for DSP core applications:

- **dsplib**: FFT, Filters, etc.
- **imglib**: Image processing
- **mathlib**: Standard math functions (sin, cosin, sqrt)

**NOTE**: Many more libraries are available as source code outside of the release.

The **xdais** folder includes a set of standard DSP interfaces that enable easy integration of XDAIS-compatible algorithms (voice and video codecs) into applications.
Processor SDK RTOS Release (AM57x)

bios_version

cg_xml

xdctools_version

ipc_version

processor_sdk_rtos_am57xx_version

ndk_version

The bios folder includes
the RTOS operating
system kernel (scheduler
and utilities).

The processor_sdk_rtos
folder contains collateral,
documentation, scripts,
makefiles, and examples.

The cg_xml and xdctools
folders contain sets of
utilities used to build and
configure OS modules
using a GUI interface or
ASCII configuration file.

The ipc folder contains a
set of utilities used to
facilitate inter-processor
communications internal
and external the device.

The ndk folder includes
the TCP/IP stack.
The **ctools** folder is a collection of libraries that control real-time debug and collect debug information (instrumentation).

The **uiar** (universal instrumentation architecture) folder contains utilities, which are used to process, analyze, and display debug data from the hardware (visualization).
TI-RTOS Kernel

Introduction to Processor SDK RTOS
TI-RTOS: Generic Real-Time Operating System

- TI-RTOS is a scalable OS that is currently available for multiple cores:
  - Tiva-C (M4)
  - Concerto (M3+C28x)
  - C28x
  - MSP430
  - C6000
  - Sitara

- TI-RTOS kernel is embedded within Processor SDK RTOS, along with associated tools, utilities, and drivers.

- The RTOS kernel is a real-time multi-tasks scheduler.
Real Time Multi-Tasks Scheduler

By definition, real-time is a controlled response time to (multiple) external events.

• Able to accept multiple interrupts
• Controls the maximum latency in responding to interrupt

NOTE: Deterministic latency is hard to achieve

• Provides a strong priority scheme for tasks
TI-RTOS Real Time Multi-Tasks Scheduler

- Event-driven operating system
  NOTE: Event can be clock, but usually not.
- Very small adaptive footprint
- Very efficient context switching

```c
Audio_ISR()
{
    read_sample();
    Swi_post(Swi_2);
}
```
More Information About TI-RTOS

• Comprehensive TI RTOS online training:
  – 10 video presentations cover TI-RTOS and CCS in great detail.
  – All slides are available for download.

• Other sources for RTOS training include:
  http://processors.wiki.ti.com/index.php/SYS/BIOS_Online_Training and

• The back-up slides at the end of the PDF of this presentation (See Resources, upper right) provide a brief description of all TI-RTOS thread types.
Inter-Processor Communication (IPC)

Introduction to Processor SDK RTOS
IPC Principles

IPC provides standard APIs to communicate between threads:

• The same APIs for all SoCs
• The same APIs regardless of what CPU is the sender and what CPU is the receiver
• The same APIs regardless of the operating system
• The same APIs regardless of the transport mechanism
IPC Challenges

• Cooperation between multiple cores requires a smart way to exchange data and messages.
• IPC must support any number of cores within a single SOC with the ability to connect multiple devices.
• An efficient scheme is required to avoid high cost in terms of CPU cycles.
• Implementations depend on the hardware, transport layer, and operating system.
• There are the usual trade-offs: performance (speed, flexibility) versus cost (complexity, more resources).
SoC Architecture Support for IPC

• Depends on the SoC
  — Memories that can be shared between cores
  — Mailboxes or interrupt registers
  — Multicore Navigator or other DMA mechanism

• Future support is planned for peripheral communication between cores on different SoCs.
IPC Module

- Current IPC implementation may use multiple transports:
  - Core ➔ Core
  - Device ➔ Device (SoC peripheral interface)
- Chosen at configuration; Same code regardless of thread location.

- IPC Manager initializes IPC and synchronization

API summary:
- `Ipc_start` reserves memory, create default gate and heap
- `Ipc_stop` releases all resources
- `Ipc_attach` sets up transport between two processors
- `Ipc_detach` finalizes transport

[Diagram showing IPC connections between ARM and C66x cores for Device 1 and Device 2 with TI-RTOS and SoC Peripheral.]
IPC Services

- The IPC package is a set of standard APIs. MessageQ is the highest layer,
- The implementation is device- and OS-dependent.
SINGLE reader, multiple WRITERS model (READER owns queue/mailbox)

- MessageQ transactions **begin** with **READER** creating a MessageQ.
- **READER’s** attempt to get a message results in a block (unless timeout was specified), since no messages are in the queue yet.
Using MessageQ (2/3)

Core 1 - WRITER

- WRITER begins by opening MessageQ created by READER.
- WRITER gets a message block from a heap and fills it, as desired.
- WRITER puts the message into the MessageQ.

Core 2 - READER

- MessageQ_create("myQ", ...);
- MessageQ_get("myQ", &msg...);

Heap

- WRITER begins by opening MessageQ created by READER.
- WRITER gets a message block from a heap and fills it, as desired.
- WRITER puts the message into the MessageQ.
Using MessageQ (3/3)

Core 1 - WRITER

```
MessageQ_open("myQ", ...);
msg = MessageQ_alloc(heap, size,...);
MessageQ_put("myQ", msg, ...);
MessageQ_close("myQ", ...);
```

Core 2 - READER

```
MessageQ_create("myQ", ...);
MessageQ_get("myQ", &msg...);
*** PROCESS MSG ***
MessageQ_free("myQ", ...);
MessageQ_delete("myQ", ...);
```

- Once **WRITER** puts `msg` in `MessageQ`, **READER** is unblocked.
- **READER** can now read/process the received message.
- **READER** frees message back to `Heap`.
- **READER** can optionally delete the created `MessageQ`, if desired.
Network Developer’s Kit (NDK)

Introduction to Processor SDK RTOS
NDK Services

The Network Developer’s Kit (NDK) serves as a rapid prototype platform for the development of network and packet-processing applications. NDK includes the following:

- IPv6 and IPv4 compliant TCP/IP stack
- Layer 3 & 4 network protocols
- High-level network applications including HTTP server and DHCP

NOTE: NDK was developed as a prototype code example. It is not aimed for high-throughput networking.
NDK Parts (NIMU, UIU)

• The NDK is divided into two parts:
  – NIMU (Network Interface Management Unit)
  – UIU (User Interface Unit)
• For more information, refer to the NDK User’s Guide.
Algorithm Libraries

Introduction to Processor SDK RTOS
Optimized Algorithm Libraries

- The Processor SDK release contains three algorithm libraries.
- Each release directory has a C66 DSP-optimized code as well as a standard ANSI C implementation of all the functions.
- The standard ANSI C implementation is used to validate the results of the optimized library functions.
- Compiling the ANSI C source code using another core (like M4 or A15) compiler provides (non-optimized) libraries for non-DSP core.
DSP Algorithm Libraries

- Optimized algorithm libraries contain C66x C-callable, C with intrinsic functions for specific usage.
- Few legacy functions are written using assembly code.
- The following three libraries are part of the Processor SDK release:
  - Fundamental math & signal processing libraries:
    - DSPLIB: Signal-processing math and vector functions
    - MathLIB: Floating-point math functions
  - IMGLIB: Image/video processing functions
- A complete set of libraries that are available as source code can be found here:
Diagnostics Software

Introduction to Processor SDK RTOS
CCS Diagnostic Elements

• CCS-based Debug
  – break points
  – watch points
  – step/step into
  – resume

• CCS-based Trace (Instrumentation)
  – Configure trace logic
  – Getting trace information back to host

• CCS-based data processing (Visualization)
Run-Time Diagnostics in Processor SDK

CToolsLib (Chip Tools Library) has multiple libraries that provide run-time debug capabilities.

NOTE: Not all features are available for all devices. Usage is dependent on core and device hardware.

More details at http://processors.wiki.ti.com/index.php/CToolsLib
Run-Time Diagnostic Elements (1/3)

- **AET** (Advanced Event Trigger Library) configures state machines that control tracing.

- **DSPTraceLib** and **ETBLib** (Embedded Trace Buffer Library) provide a set of functions to control the DSP trace buffer operation and trace data transport.

Run-Time Diagnostic Elements (2/3)

- **ETMLib** (Embedded Trace Macrocell Library) controls the ARM macrocell trace facilities.

- **PMICMLib** (Power and Clock Management Instrumentation library) provides programming and control APIs for the PMI/CMI units, which provide power and clock state profiling.

Run-Time Diagnostic Elements (3/3)

• **SCILib** (Statistic Collectors Library) collects statistical data from hardware counters (core dependent).

• **STMLib** (System Trace Library) provides a set of utilities to collect real-time, non-intrusive system trace messages during run-time.

For More Information

• Processor SDK RTOS Getting Started Guide
• Processor SDK Training Series
• Additional training:
  – TI-RTOS Kernel Workshop
  – Processor SDK RTOS Overview P2
• For questions regarding topics covered in this training, visit the Sitara Processor support forum at the TI E2E Community website.