Introduction to Processor SDK RTOS Part 1

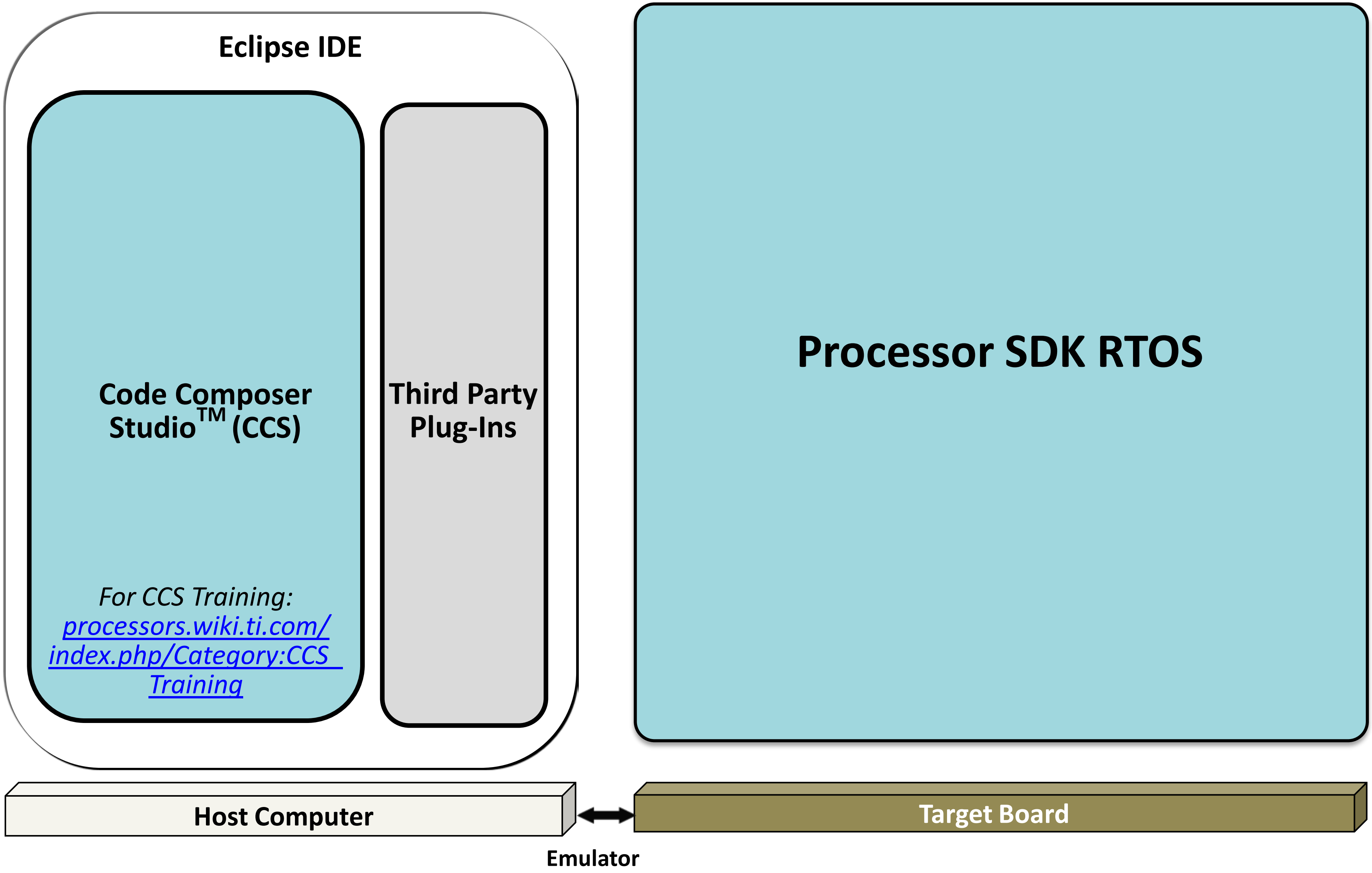
Agenda

- 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)

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

Third Party Plug-Ins

Processor SDK RTOS

Host Computer

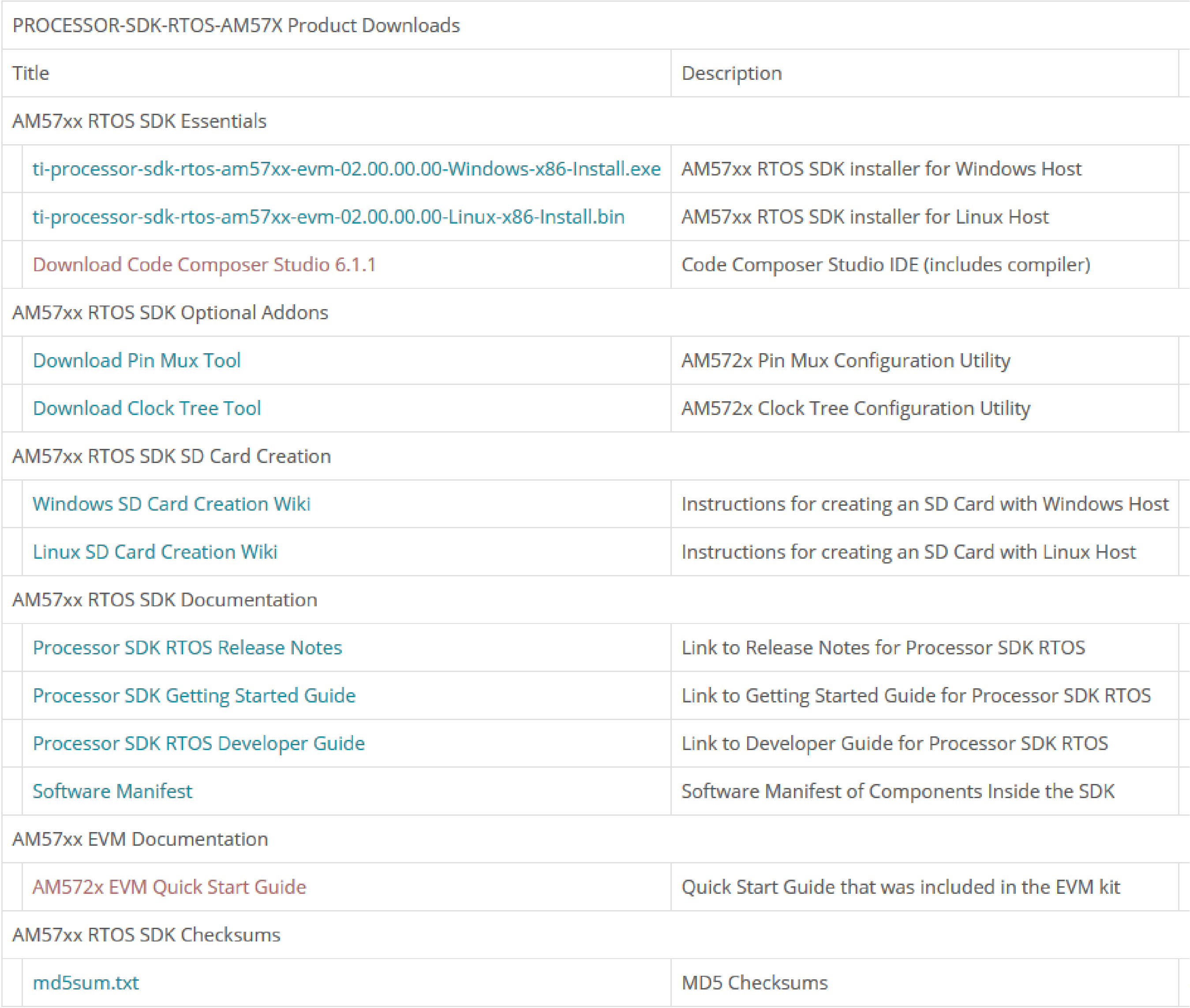
Emulator

Target Board

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.

Part of Processor SDK RTOS Install Page



PROCESSOR-SDK-RTOS-AM57X Product Downloads

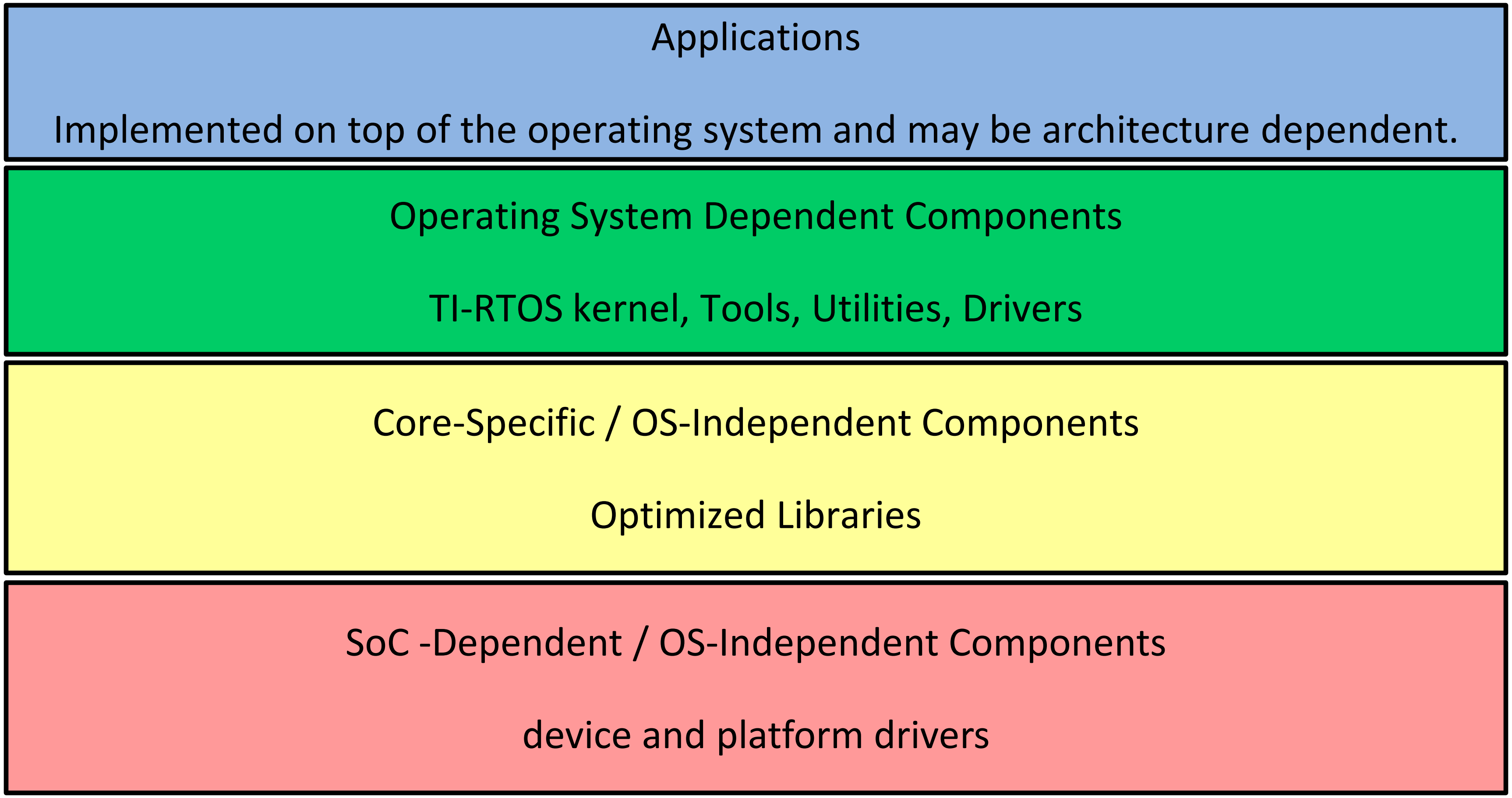
Title	Description
AM57xx RTOS SDK Essentials	
ti-processor-sdk-rtos-am57xx-evm-02.00.00.00-Windows-x86-Install.exe	AM57xx RTOS SDK installer for Windows Host
ti-processor-sdk-rtos-am57xx-evm-02.00.00.00-Linux-x86-Install.bin	AM57xx RTOS SDK installer for Linux Host
Download Code Composer Studio 6.1.1	Code Composer Studio IDE (includes compiler)
AM57xx RTOS SDK Optional Addons	
Download Pin Mux Tool	AM572x Pin Mux Configuration Utility
Download Clock Tree Tool	AM572x Clock Tree Configuration Utility
AM57xx RTOS SDK SD Card Creation	
Windows SD Card Creation Wiki	Instructions for creating an SD Card with Windows Host
Linux SD Card Creation Wiki	Instructions for creating an SD Card with Linux Host
AM57xx RTOS SDK Documentation	
Processor SDK RTOS Release Notes	Link to Release Notes for Processor SDK RTOS
Processor SDK Getting Started Guide	Link to Getting Started Guide for Processor SDK RTOS
Processor SDK RTOS Developer Guide	Link to Developer Guide for Processor SDK RTOS
Software Manifest	Software Manifest of Components Inside the SDK
AM57xx EVM Documentation	
AM572x EVM Quick Start Guide	Quick Start Guide that was included in the EVM kit
AM57xx RTOS SDK Checksums	
md5sum.txt	MD5 Checksums

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



Applications

Implemented on top of the operating system and may be architecture dependent.

Operating System Dependent Components

TI-RTOS kernel, Tools, Utilities, Drivers

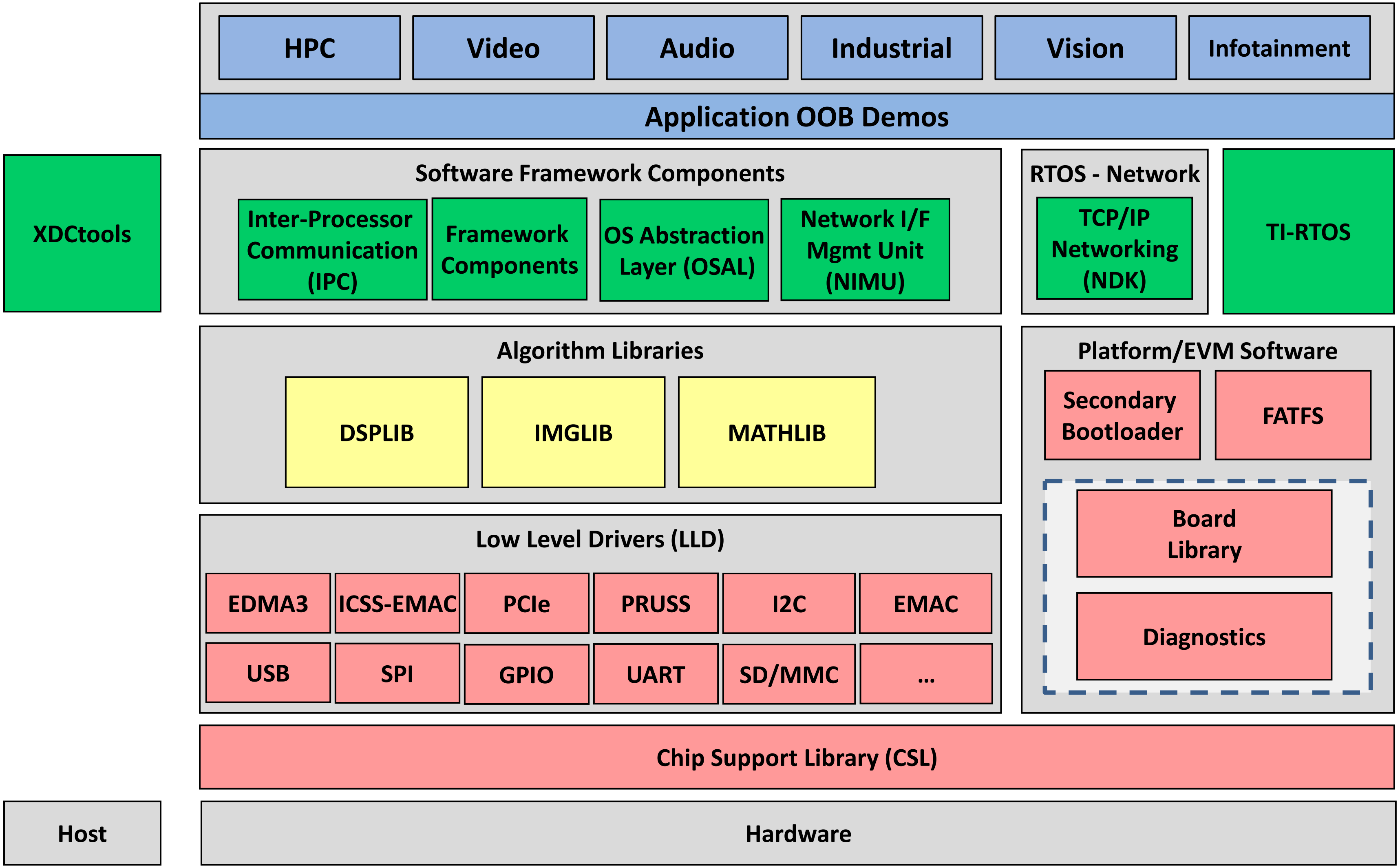
Core-Specific / OS-Independent Components

Optimized Libraries

SoC -Dependent / OS-Independent Components

device and platform drivers

Processor SDK RTOS Software: AM57x Superset



HPC
Video
Audio
Industrial
Vision
Infotainment

Application OOB Demos

XDCtools

Software Framework Components

Inter-Processor Communication (IPC)
Framework Components
OS Abstraction Layer (OSAL)
Network I/F Mgmt Unit (NIMU)

RTOS - Network

TCP/IP Networking (NDK)

TI-RTOS

Algorithm Libraries

DSPLIB
IMGLIB
MATHLIB

Platform/EVM Software

Secondary Bootloader
FATFS
Board Library
Diagnostics

Low Level Drivers (LLD)

EDMA3
ICSS-EMAC
PCIe
PRUSS
I2C
EMAC
USB
SPI
GPIO
UART
SD/MMC
…

Chip Support Library (CSL)

Host

Hardware

🟥 Device/platform-dependent

🟩 OS-dependent

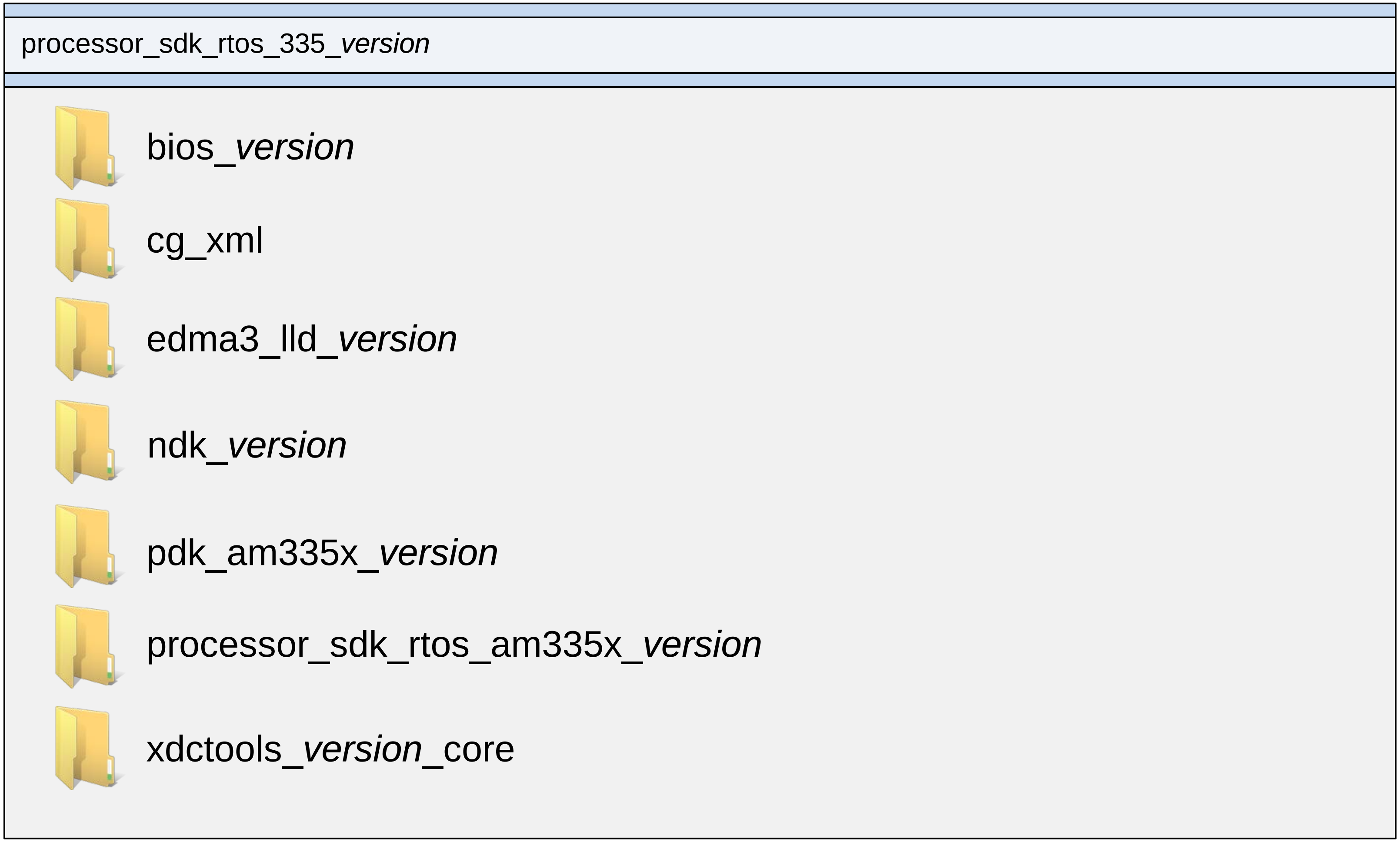
🟨 Core-specific optimized library

🟦 Applications

Processor SDK RTOS

Single product supports multiple SoCs

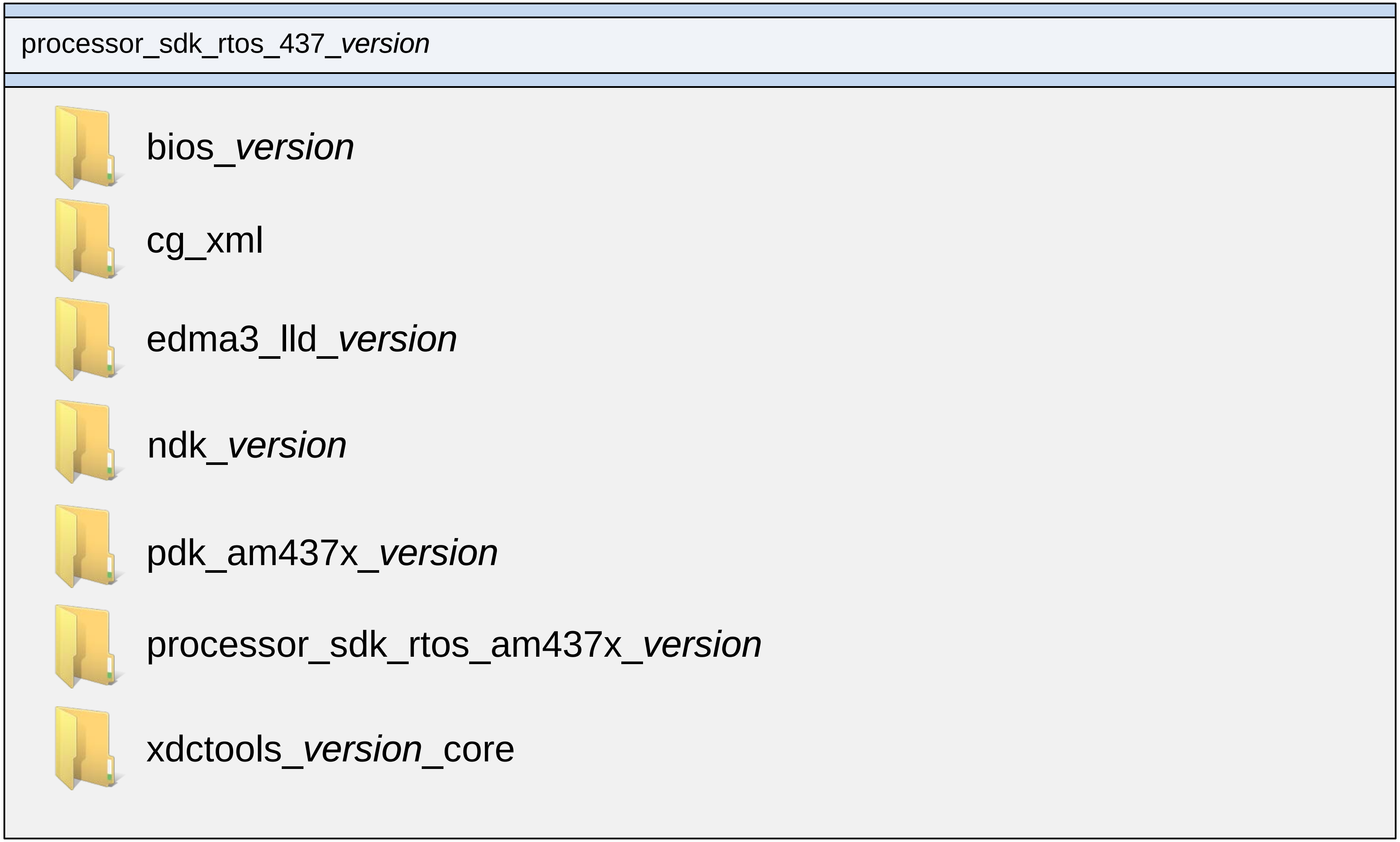
Processor SDK RTOS Release (AM335x)



processor_sdk_rtos_335_*version*

- bios_*version*
- cg_xml
- edma3_lld_*version*
- ndk_*version*
- pdk_am335x_*version*
- processor_sdk_rtos_am335x_*version*
- xdctools_*version*_core

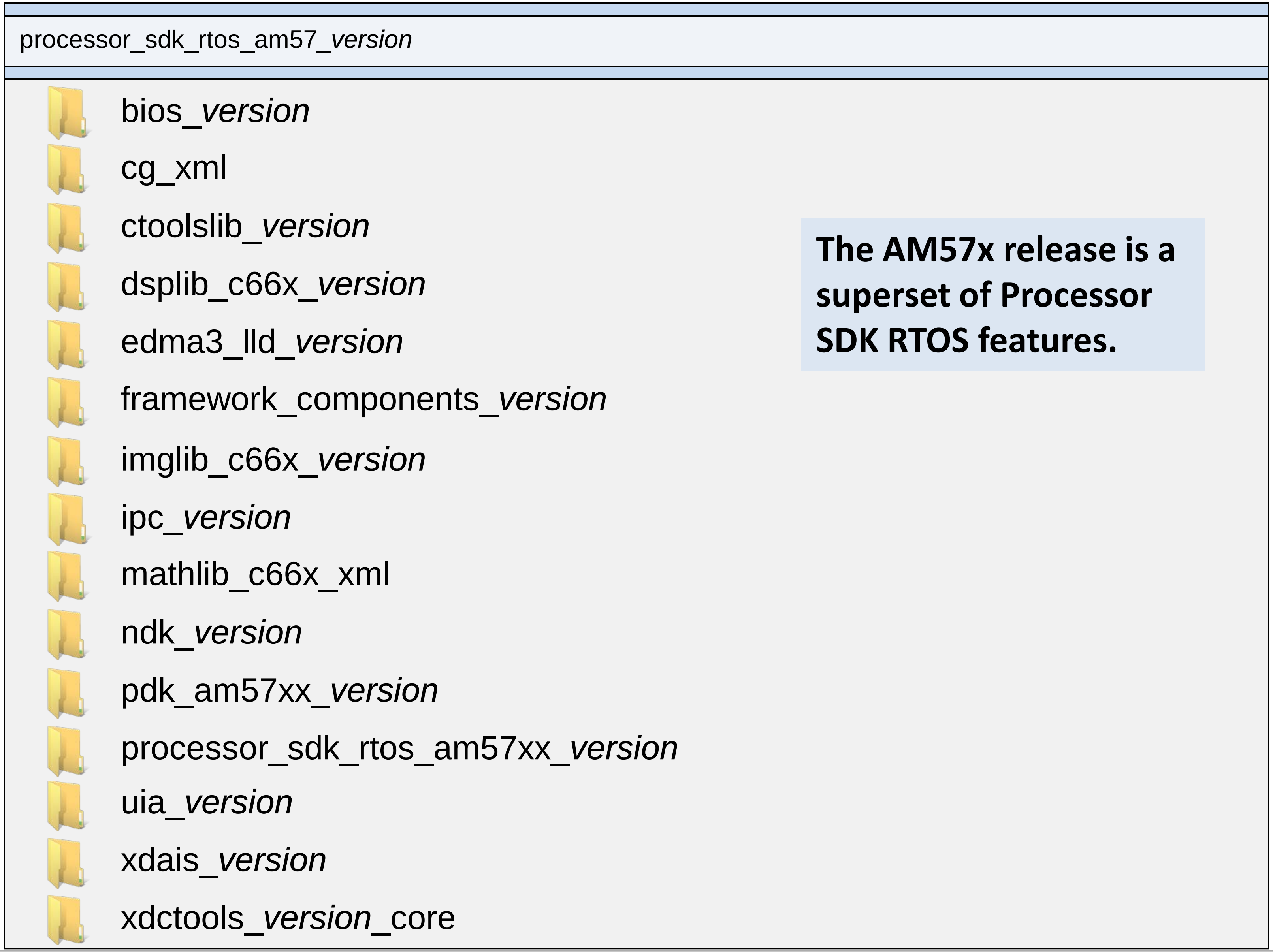
Processor SDK RTOS Release (AM437x)



processor_sdk_rtos_437_*version*

- bios_*version*
- cg_xml
- edma3_lld_*version*
- ndk_*version*
- pdk_am437x_*version*
- processor_sdk_rtos_am437x_*version*
- xdctools_*version*_core

Processor SDK RTOS Release (AM57x)



processor_sdk_rtos_am57_*version*

- bios_*version*
- cg_xml
- ctoolslib_*version*
- dsplib_c66x_*version*
- edma3_lld_*version*
- framework_components_*version*
- imglib_c66x_*version*
- ipc_*version*
- mathlib_c66x_xml
- ndk_*version*
- pdk_am57xx_*version*
- processor_sdk_rtos_am57xx_*version*
- uia_*version*
- xdais_*version*
- xdctools_*version*_core

The AM57x release is a superset of Processor SDK RTOS features.

Processor SDK RTOS Release (AM57x)

pdk_am57xx_*version*

edma3_lld_*version*

framework_components_*version*

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.

Processor SDK RTOS Release (AM57x)

dsplib_c66x_*version*

imglib_c66x_*version*

mathlib_c66x_*version*

xdais_*version*

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.

Processor SDK RTOS Release (AM57x)

ctoolslib_*version*

uia_*version*

The **ctools** folder is a collection of libraries that control real-time debug and collect debug information (instrumentation) .

The **uia** (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.

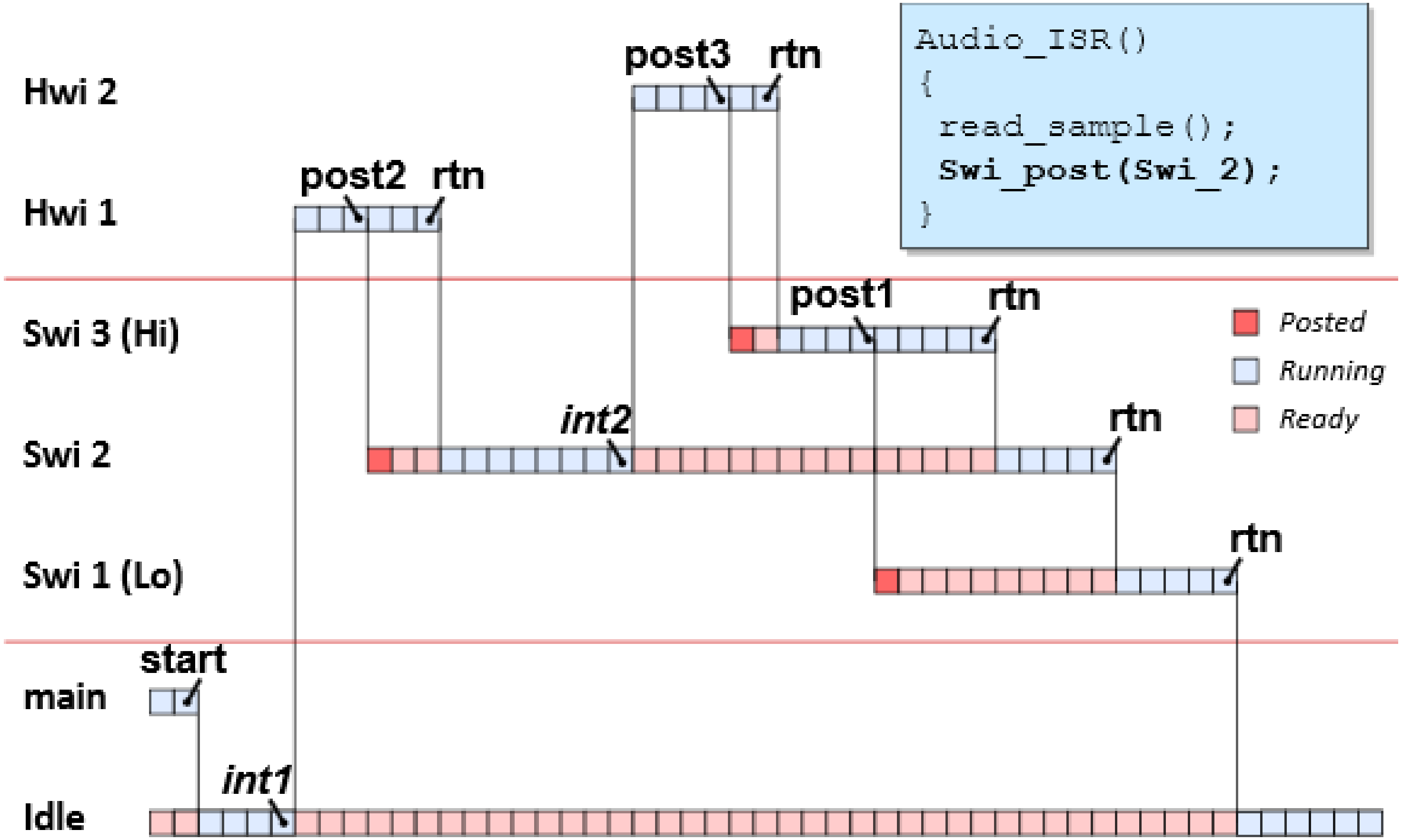
- Able to accept multiple interrupts
- Controls the maximum latency in responding to interrupt

  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



```
Audio_ISR()
{
 read_sample();
 Swi_post(Swi_2);
}
```

More Information About TI-RTOS

- Comprehensive TI RTOS online training: **http://processors.wiki.ti.com/index.php/Introduction_to_the_TI-RTOS_Kernel_Workshop**
  - 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 http://processors.wiki.ti.com/index.php/Hands-On_Training_for_TI_Embedded_Processors
- 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).

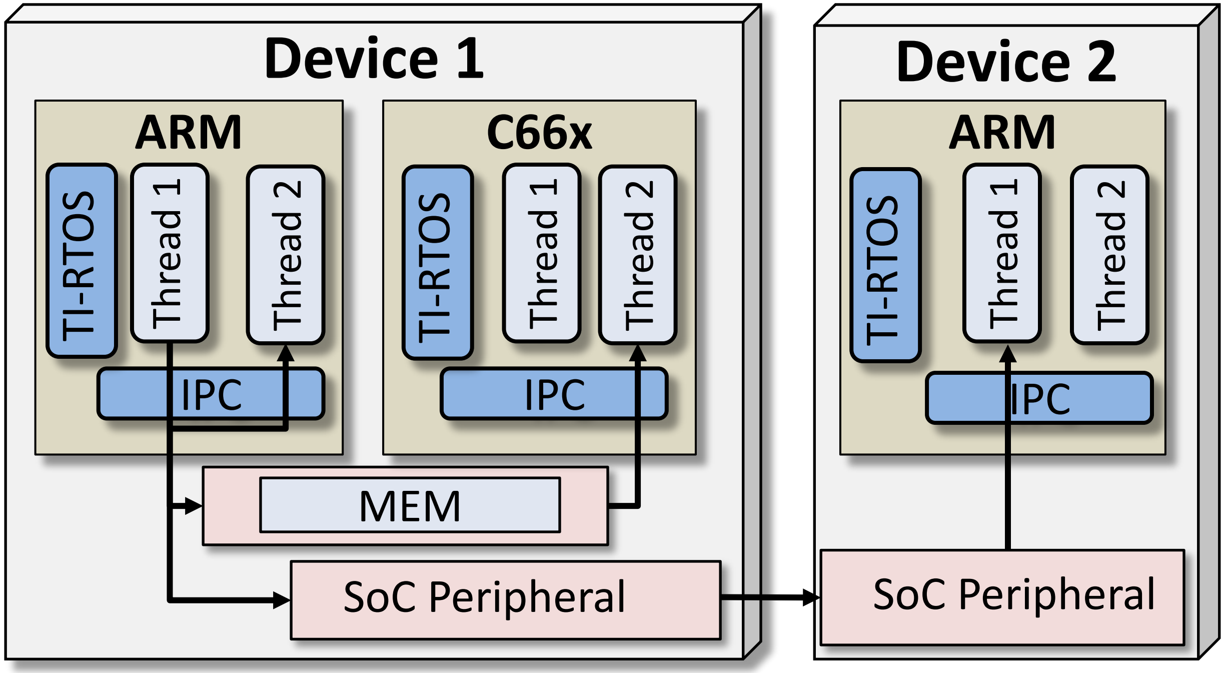
SoC Architecture Support for IPC

- 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

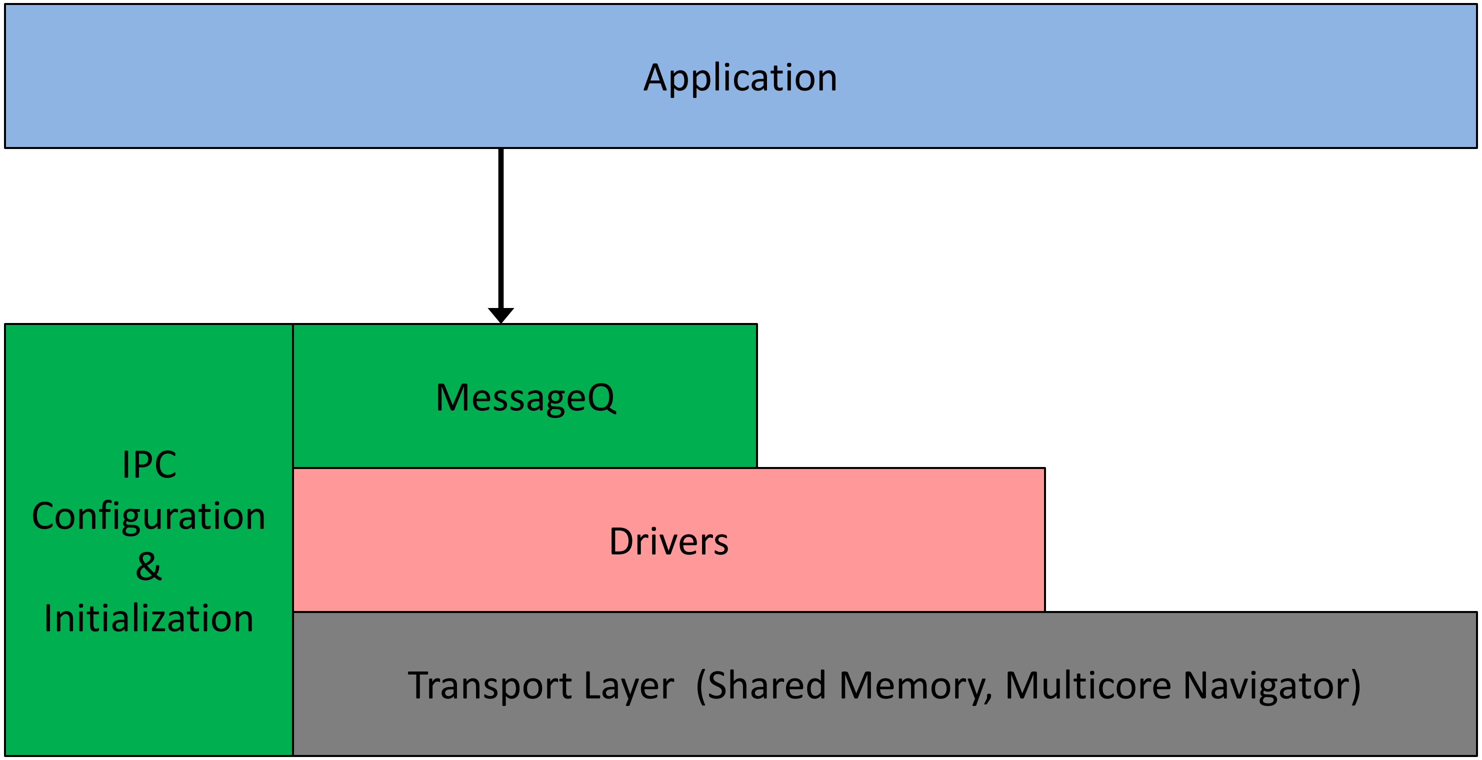
- Current IPC implementation may use multiple transports:
  - Core → Core
  - Device → Device (SoC peripheral interface)
- Chosen at configuration; *Same code* regardless of thread location.
- IPC Manager initializes IPC and synchronization

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



IPC Services

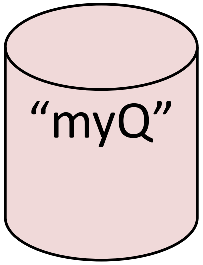
- The IPC package is a set of standard APIs. MessageQ is the highest layer,
- The implementation is device- and OS-dependent.



Application
MessageQ
IPC Configuration & Initialization
Drivers
Transport Layer (Shared Memory, Multicore Navigator)

MessageQ Highest Layer API (1/3)

SINGLE reader, multiple WRITERS model (READER owns queue/mailbox)



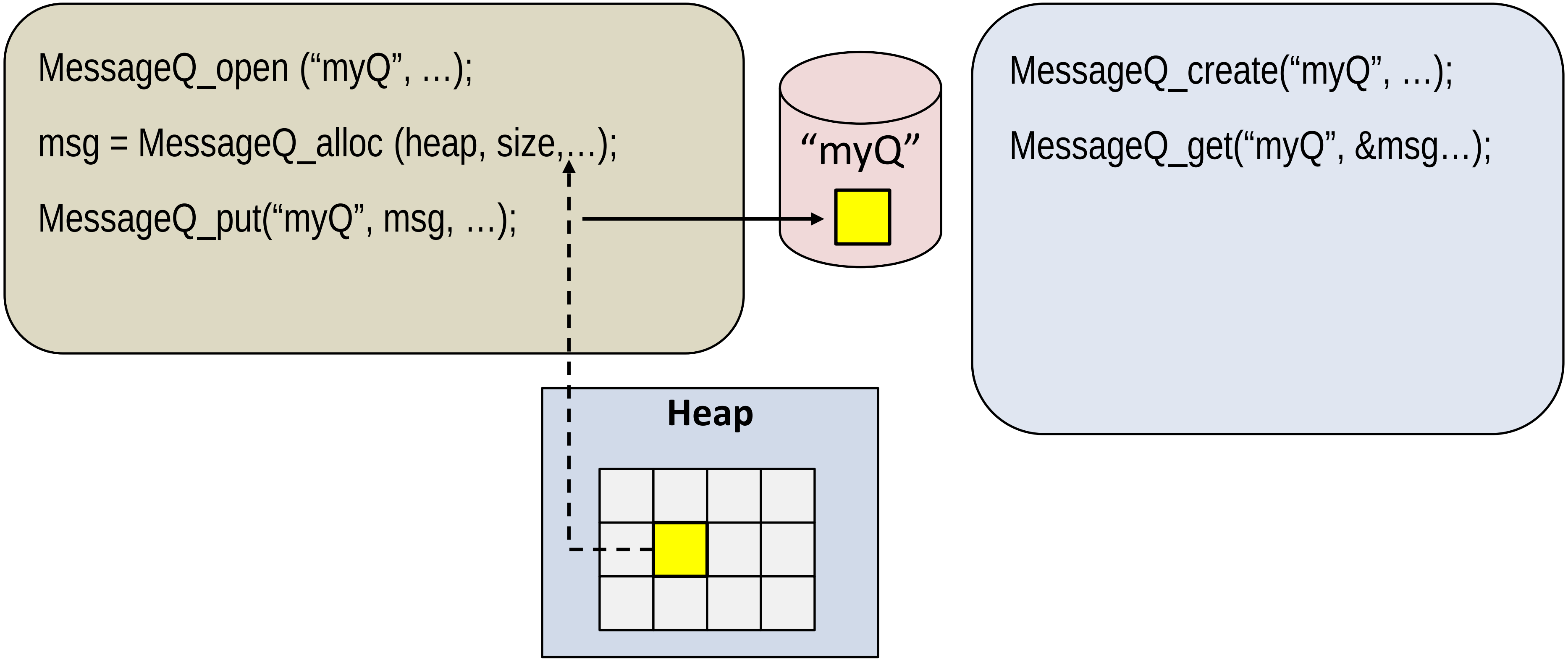
Core 2 - READER

```
MessageQ_create(“myQ”, *synchronizer);

MessageQ_get(“myQ”, &msg, timeout);
```

- 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

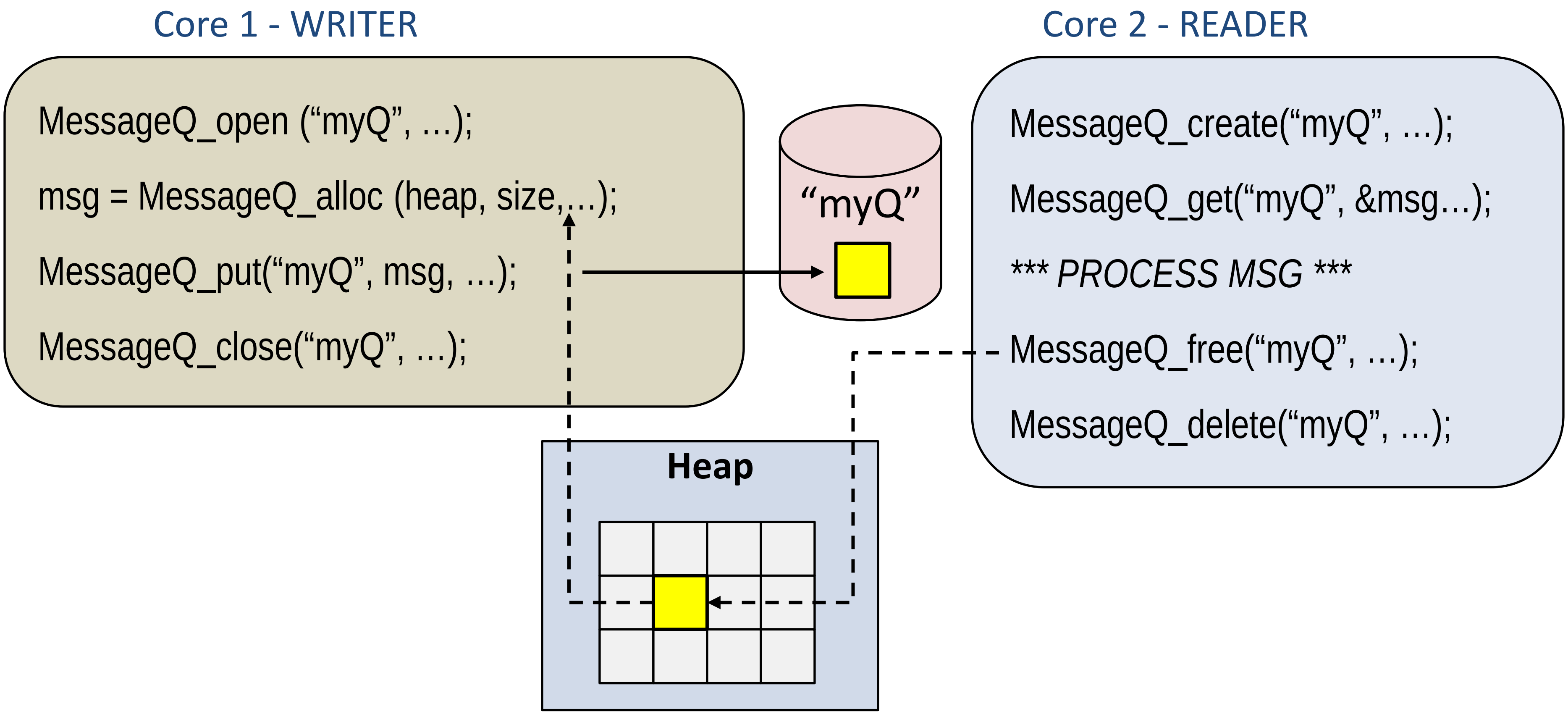
```
MessageQ_open (“myQ”, …);
msg = MessageQ_alloc (heap, size,…);
MessageQ_put(“myQ”, msg, …);
```

Core 2 - READER

```
MessageQ_create(“myQ”, …);
MessageQ_get(“myQ”, &msg…);
```

- 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

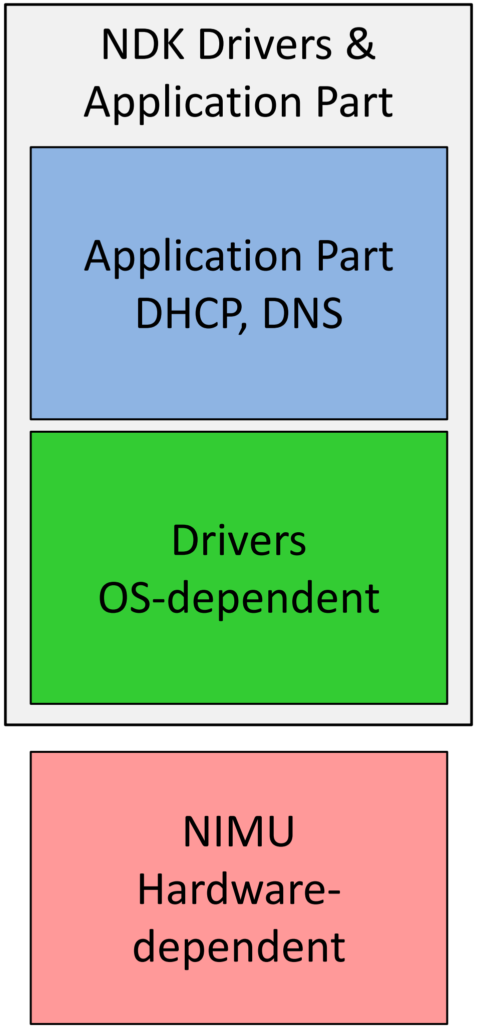
The Network Developer's Kit (NDK) serves as a rapid prototype platform for the development of network and packet-processing applications. NDK includes the following:

- 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: http://processors.wiki.ti.com/index.php/Software_libraries

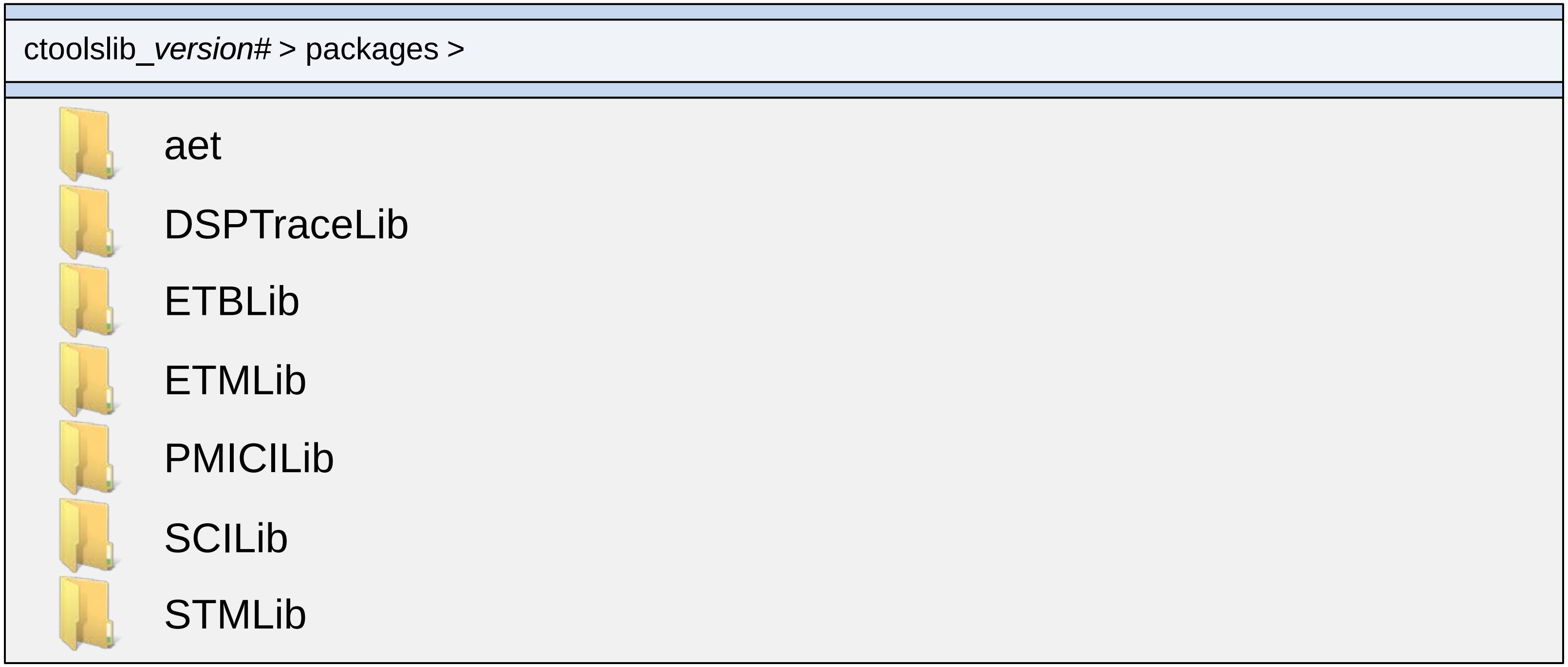
Diagnostics Software

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_*version#* > packages >

aet

DSPTraceLib

ETBLib

ETMLib

PMICILib

SCILib

STMLib

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

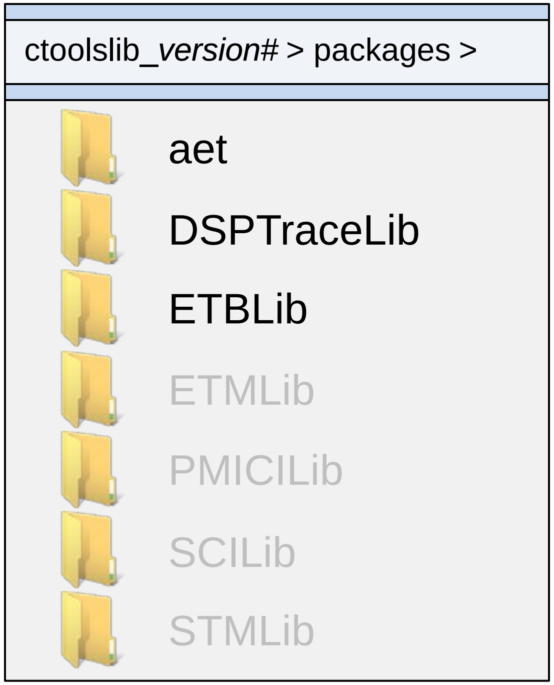
NOTE: Not all features are available for all devices. Usage is dependent on core and device hardware.

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.



ctoolslib_*version#* > packages >

aet

DSPTraceLib

ETBLib

ETMLib

PMICILib

SCILib

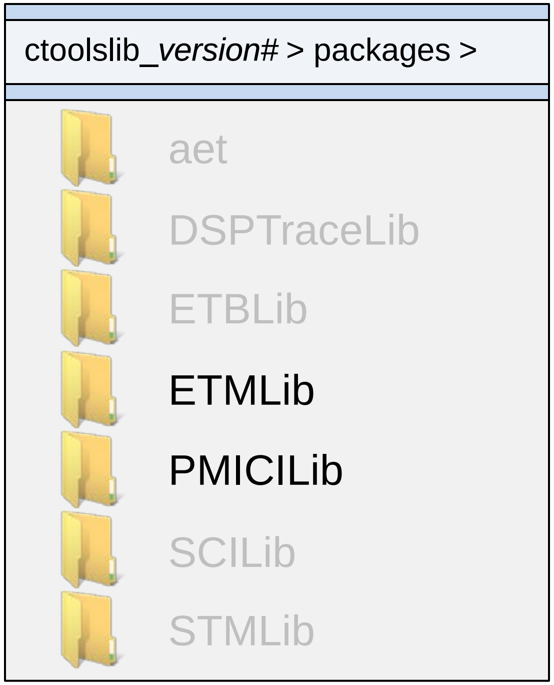
STMLib

More details at http://processors.wiki.ti.com/index.php/CToolsLib

Run-Time Diagnostic Elements (2/3)

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

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



ctoolslib_*version#* > packages >

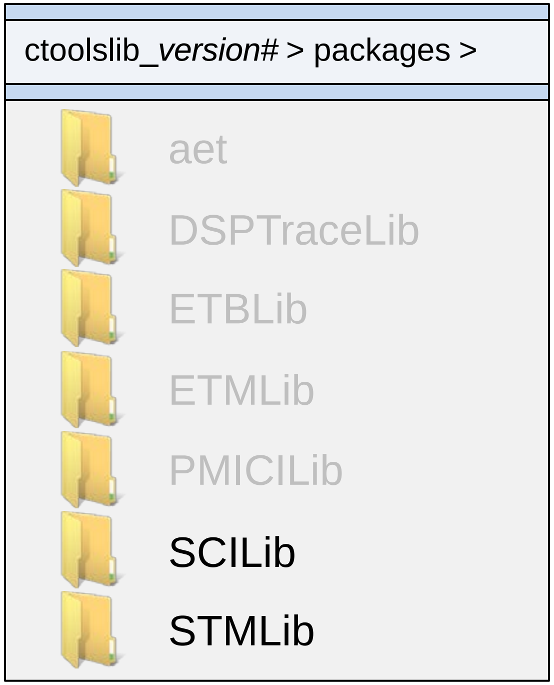
aet
DSPTraceLib
ETBLib
ETMLib
PMICILib
SCILib
STMLib

More details at http://processors.wiki.ti.com/index.php/CToolsLib

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.



ctoolslib_*version#* > packages >

aet
DSPTraceLib
ETBLib
ETMLib
PMICILib
SCILib
STMLib

More details at http://processors.wiki.ti.com/index.php/CToolsLib

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.