



Linux Embedded System Design Workshop

Designing with Texas Instruments ARM and ARM+DSP Systems

Discussion Notes

Workshop Student Notes
Revision 3.0;
Sept 2011

Technical Training Organization

Notice

Creation of derivative works unless agreed to in writing by the copyright owner is forbidden. No portion of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission from the copyright holder.

Texas Instruments reserves the right to update this Guide to reflect the most current product information for the spectrum of users. If there are any differences between this Guide and a technical reference manual, references should always be made to the most current reference manual. Information contained in this publication is believed to be accurate and reliable. However, responsibility is assumed neither for its use nor any infringement of patents or rights of others that may result from its use. No license is granted by implication or otherwise under any patent or patent right of Texas Instruments or others.

Copyright © 2006 - 2011 by Texas Instruments Incorporated.
All rights reserved.

Training Technical Organization

Texas Instruments, Incorporated
6500 Chase Oaks Blvd, MS 8437
Plano, TX 75023
(214) 567-0973

Revision History

October 2006, Version 0.80 (alpha)
December 2006, Versions 0.90/0.91 (alpha2)
January 2007, Version 0.92 (beta)
February 2007, Version 0.95
March 2008, Versions 0.96 (errata)
April 2008, Version 0.98 (chapter rewrites & errata)
September 2008, Version 1.30 (beta 1 & 2)
October 2008, Version 1.30 (beta 3)
February 2010, Version 2.00
August 2010, Version 2.10
Oct 2010, v3.03; Jan 2011, v3.05; Apr 2011, v3.06a; July 2011, v3.07
Aug 2011, Version 3.08

Welcome Topics

<i>Workshop Goals</i>	0-4
<i>Where To Get Additional Information/Training?</i>	0-6
<i>Workshop Outline</i>	0-7
<i>About You</i>	0-8
<i>Administrative Topics</i>	0-8

Workshop Goals

Workshop Goal

“Learn how to put together a ARM-based Linux embedded system.”

- ◆ Given that you already know:
 - ◆ How to program in C
 - ◆ Basics of Linux
 - ◆ Understand basic embedded system concepts
- ◆ Provided with:
 - ◆ Linux Distribution for ARM
 - ◆ TI foundation software, incl: Codec Engine, Codecs (i.e. Algo's)
 - ◆ Tools: EVM, SDK/DVSDK
- ◆ In this workshop, you'll learn the essential skills required to put together a Linux ARM or ARM+DSP based system

So, the focus of the workshop is...

Workshop Prerequisites

Pre-requisite Skills

- ◆ Required
 - ◆ Some knowledge of C programming
 - ◆ Basic Linux skills (i.e. shell commands, etc.)
- ◆ Recommended
 - ◆ Embedded system basics (memory map, linking, etc.)
 - ◆ Basic Linux programming (processes, threads, etc.)
- ◆ Nice to have
 - ◆ C6x DSP programming
 - ◆ Understanding of Linux device drivers
 - ◆ Video Application/System Knowledge

Not Required

- ◆ No H/W design experience required

Focus of Linux/ARM Workshop

This workshop focuses on applications which plan to use TI's ARM or ARM+DSP software model:

- ◆ ARM running embedded [Linux](#)
- ◆ DSP running [DSP/BIOS](#)
- ◆ Signal processing (and IPC) via [Codec Engine](#) (VISA API)
- ◆ Signal Processing Layer (CODECs and algorithms) built using [xDM/xDAIS](#) API
- ◆ Building programs with [GNU Make](#) (gMake) and TI's Real-Time Software [XDC](#) tools

[Where to get more info...](#)

What Will You Accomplish?

When you leave the workshop, you should be able to...

- ◆ Describe the [basic silicon features](#) and options of the TI high-performance ARM and ARM+DSP processors
- ◆ Draw a diagram describing the [building blocks](#) used in TI's ARM+DSP foundation [software](#)
- ◆ Build a Linux application which uses:
 - ◆ The audio and video [Linux drivers](#) provided in the Linux distribution
 - ◆ [Invoke local](#) (ARM-based) and [remote](#) (DSP-based) signal processing [algorithms](#) via Codec Engine (VISA API)
- ◆ Create the building-blocks used by the Codec Engine:
 - ◆ Use gMake & Configuro to build a signal processing [Engine](#) and [DSP Server](#)
- ◆ Algorithms/Codecs:
 - ◆ Describe the [xDM/xDAIS](#) API's used to access algorithms
 - ◆ [Write an algorithm](#) following the XDM API
- ◆ Tools:
 - ◆ [Setup the DVEVM](#) hardware tool by configuring various [U-Boot](#) parameters

Where To Get Additional Information/Training?

Why Don't We Cover Everything?

- In 4 days, it is impossible to cover everything. However, we do cover about an equivalent of a college semester course on the DM644x.
- We provide the following lists as a starting place for additional information.

Where can I get additional skills? (from TI)

Texas Instruments Curriculum

◆ Building Linux based Systems (ARM or ARM+DSP processors)	DaVinci / OMAP / Sitara System Integration Workshop using Linux (4-days) www.ti.com/training
◆ Building BIOS based Systems (DSP processors)	System Integration Workshop using DSP/BIOS (4-days) www.ti.com/training
◆ Developing Algo's for C6x DSP's (Are you writing/optimizing algorithms for latest C64x+ or C674x DSP's CPU's)	C6000 Optimization Workshop (4-days) www.ti.com/training
Online Resources:	
● OMAP / Sitara / DaVinci Wiki http://processors.wiki.ti.com	● DaVinci Open-Source Linux Mail List http://linux.davincidsdp.com/mailman/listinfo/davinci-linux-open-source
● TI E2E Community (videos, forums, blogs) http://e2e.ti.com	● Gstreamer and other projects http://linux.davincidsdp.com or https://gforge.ti.com/gf/
● This workshop presentation & exercises http://processors.wiki.ti.com/index.php/OMAP™/DaVinci™_System_Integration_using_Linux_Workshop	● TI Software http://www.ti.com/dvevmupdates , http://www.ti.com/dms http://www.ti.com/myregisteredsoftware

Where can I get additional skills? (Non-TI)

Non-TI Curriculum

A few references, to get you started:

◆ Linux	<ul style="list-style-type: none"> • "Linux For Dummies", by Dee-Ann LeBlanc • "Linux Pocket Guide", by Daniel J. Barrett • free-electrons.com/training • www.linux-tutorial.info/index.php • www.oreilly.com/pub/topic/linux • The Linux Documentation Project: www.tldp.org • Rute Linux Tutorial: http://rute.2038bug.com/index.html.gz
◆ Embedded Linux	<ul style="list-style-type: none"> • "Building Embedded Linux Systems", by Karim Yaghmour • "Embedded Linux Primer", by Christopher Hallinan • free-electrons.com/training
◆ Linux Application Programming	<ul style="list-style-type: none"> • "Beginning Linux Programming" Third Edition, by Neil Matthew and Richard Stones
◆ ARM Programming (not required for Linux based designs)	<ul style="list-style-type: none"> • http://www.arm.com/
◆ Writing Linux Drivers	<ul style="list-style-type: none"> • "Linux Device Drivers" Third Edition, by Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman http://lwn.net/Kernel/LDD3/ • www.adeneo.com
◆ Video	<ul style="list-style-type: none"> • "The Art of Digital Video", John Watkinson • "Digital Television", H. Benoit • "Video Demystified", Keith Jack • "Video Compression Demystified", Peter Symes

Workshop Outline

Linux Embedded System Design Workshop

- | | |
|-------------------------------|---|
| Introduction | <ul style="list-style-type: none"> 0. Welcome 1. Device Families Overview 2. TI Foundation Software 3. Introduction to Linux/U-Boot 4. Tools Overview |
| Application Coding | <ul style="list-style-type: none"> 5. Building Programs with gMake 6. Device Driver Introduction 7. Video Drivers 8. Multi-Threaded Systems |
| Using the Codec Engine | <ul style="list-style-type: none"> 9. Local Codecs : Given an Engine 10. Local Codecs : Building an Engine 11. Remote Codecs : Given a DSP Server 12. Remote Codecs : Building a DSP Server |
| Algorithms | <ul style="list-style-type: none"> 13. xDAIS and xDM Authoring 14. (Optional) Using DMA in Algorithms 15. (Optional) Intro to DSPLink |



Copyright © 2011 Texas Instruments. All rights reserved.

Lab Exercises

- | | |
|--------------------------------|--|
| Introduction | <ul style="list-style-type: none"> 3. Configure U-Boot and boot the DVEVM |
| Application Programming | <ul style="list-style-type: none"> 5. Building programs with GMAKE (and Configuro) 6. <i>Given:</i> File ? <i>Audio</i>; <i>Build:</i> Audio In ? Audio Out 7. Setup an On-Screen Display (scrolling banner)
Video In ? Video Out 8. Concurrently run audio and video loop-thru programs |
| Using the Codec Engine | <ul style="list-style-type: none"> 9. Use a provided Engine (containing local codecs) 10. Build an Engine (given local codecs) 11. Use remote codecs (using a provided DSP Server)
Swap out video_copy codec for real H.264 codec 12. Build a DSP Server (given DSP-based codecs)
Use dot-product routine from C6Accel |
| Algorithms | <ul style="list-style-type: none"> 13. Build a DSP algorithm and test it in CCS (in Windows), then put your algo into a DSP server and call it from Linux |



Copyright © 2011 Texas Instruments. All rights reserved.

About You

About your Experience – A Show Of Hands

(No right or wrong answers, we just want to know where you're coming from)

- ◆ **Who is building a Video application**
 - ◆ If not, what other types of applications
- ◆ **Experienced with Linux**
 - ◆ Linux command line (Bash)
 - ◆ Mounting drives, NFS
- ◆ **Linux C Programmer**
 - ◆ GCC, gMake
 - ◆ Linux threads (processes, pThread)
- ◆ **Previous TI DSP developer?**
 - ◆ Another TI processor
 - ◆ Competitor's processor
- ◆ **Experience building Embedded Systems**
 - ◆ Memory maps and linking
 - ◆ Bootloading a processor

Administrative Topics

Administrative Topics

- ◆ **Name Tags**
- ◆ **Start & End Times**
- ◆ **Bathrooms**
- ◆ **Phone calls**
- ◆ **Lunch !!!**
- ◆ **Let us know if you'll miss part of the workshop**



TI ARM / ARM+DSP Devices

Introduction

In this chapter a cursory overview of Texas Instruments ARM and ARM+DSP devices by examining the CPU core, H/W Accelerator, and Peripheral options.

Along the way we will introduce a few topics that will be important as we work our way through the rest of the workshop chapters

Learning Objectives

At the conclusion of this chapter, you should be able to:

- List the major CPU cores offered by TI in their ARM and ARM+DSP families
- List the hardware accelerators offered by TI
- Describe the PRU, SCR and EDMA3 peripherals
- Describe the benefits and constraints of the MMU, pin-muxing, and ARM+DSP access to peripherals
- Choose a device based on the above criteria

Chapter Topics

TI ARM / ARM+DSP Devices	1-1
<i>TI Embedded Processors Portfolio</i>	<i>1-3</i>
<i>System Examples</i>	<i>1-4</i>
<i>What Processing Do You Need?</i>	<i>1-5</i>
ARM Core	1-6
DSP Core	1-7
Accelerator : 3D Graphics	1-11
Accelerator : Audio/Video Algorithms.....	1-12
Accelerators : Video Port (VPSS) / Display (DSS)	1-14
<i>Peripherals.....</i>	<i>1-19</i>
PRU – Programmable Real-time Unit	1-20
Moving Data Around – SCR / EDMA3.....	1-21
<i>Final Considerations</i>	<i>1-22</i>
Memory Map & MMU	1-22
Access to Peripherals.....	1-24
Pin Muxing	1-24
Choosing a Device : Web Tool.....	1-25

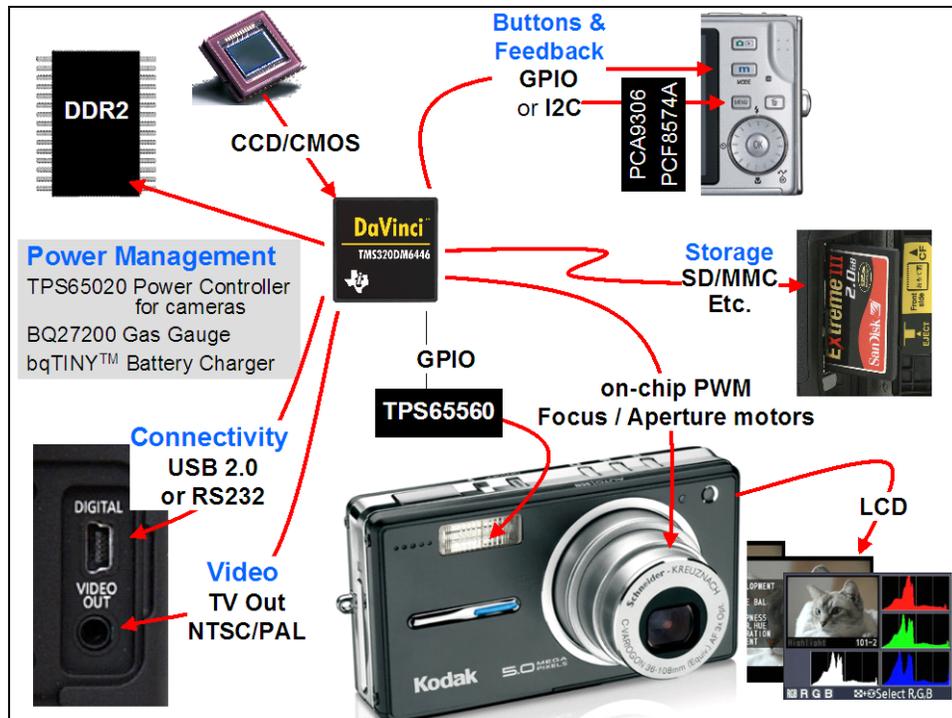
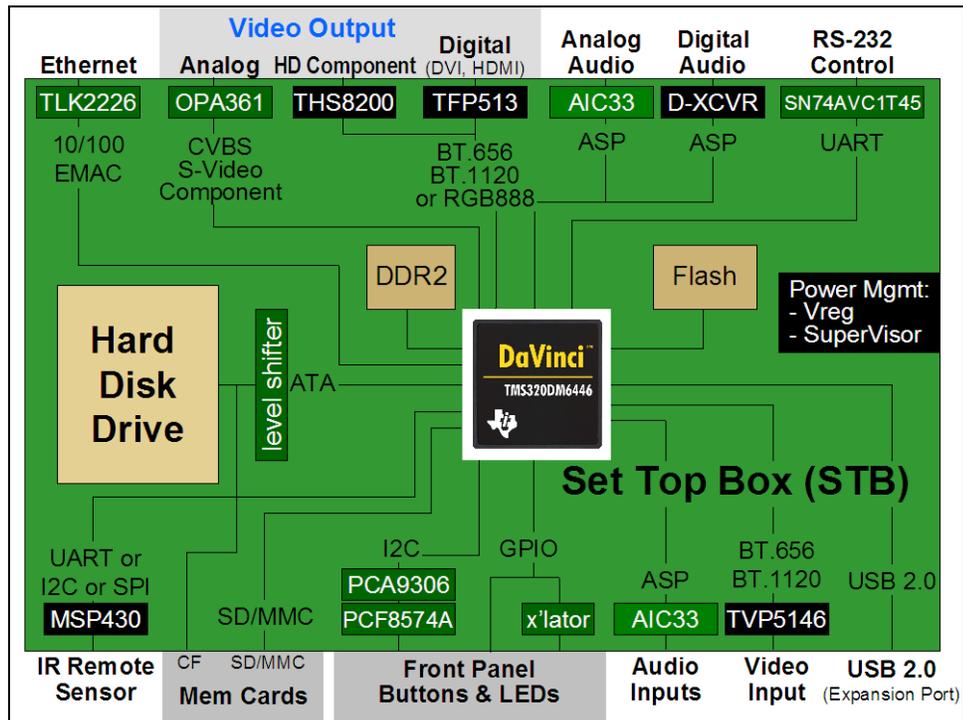
TI Embedded Processors Portfolio

TI Embedded Processors Portfolio

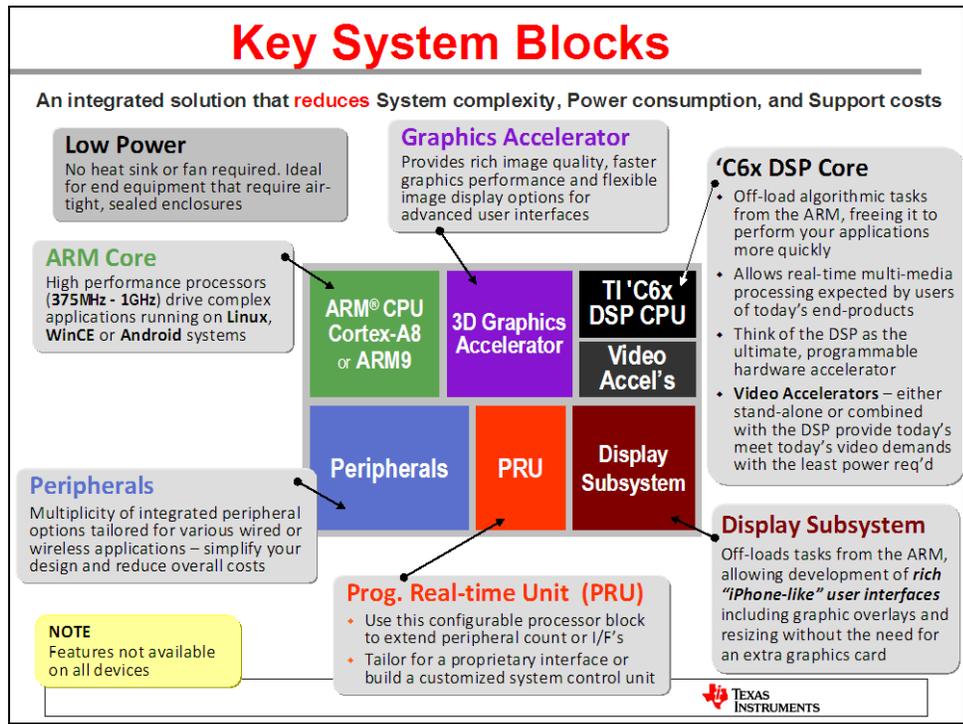
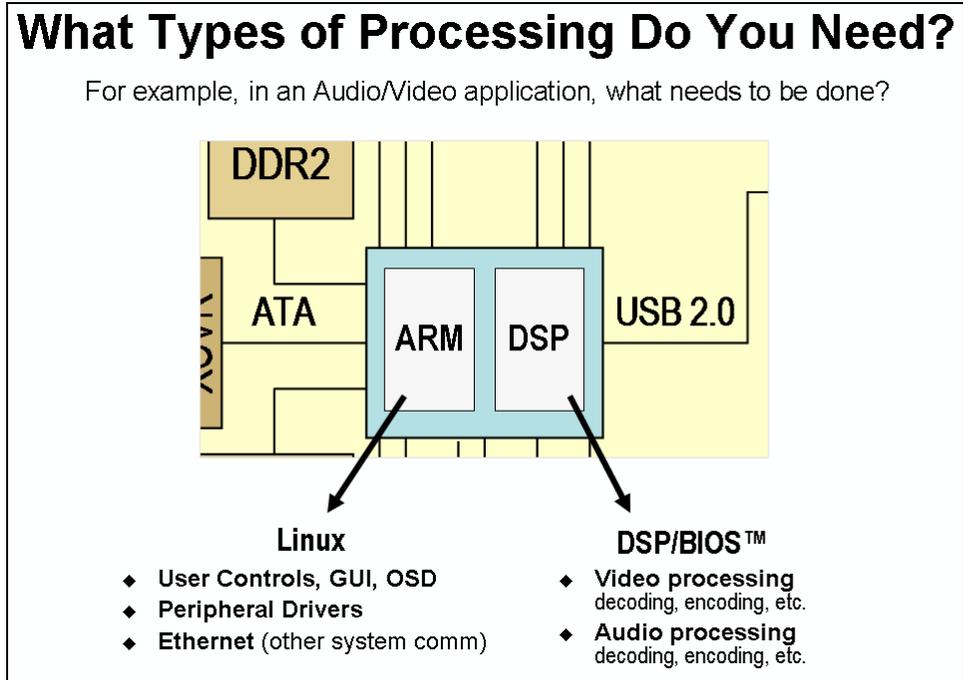
Microcontrollers			ARM-Based		DSP
16-bit	32-bit Real-time	32-bit ARM	ARM+	ARM + DSP	DSP
MSP430 Ultra-Low Power Up to 25 MHz Flash 1 KB to 256 KB Analog I/O, ADC, LCD, USB, RF Measurement, Sensing, General Purpose \$0.49 to \$9.00 	C2000™ Fixed & Floating Point Up to 300 MHz Flash 32 KB to 512 KB PWM, ADC, CAN, SPI, I ² C Motor Control, Digital Power, Lighting, Sensing \$1.50 to \$20.00 	ARM Industry Std Low Power <100 MHz Flash 64 KB to 1 MB USB, ENET, ADC, PWM, SPI Host Control \$2.00 to \$8.00 	ARM9 Cortex A-8 Industry-Std Core, High-Perf GPP Accelerators MMU USB, LCD, MMC, EMAC Linux/WinCE User Apps \$5.00 to \$35.00 	C64x+ plus ARM9/Cortex A-8 Industry-Std Core + DSP for Signal Proc. 4800 MMACs/1.07 DMIPS/MHz MMU, Cache VPSS, USB, EMAC, MMC Linux/Win + Video, Imaging, Multimedia \$12.00 to \$85.00 	C647x, C64x+, C674x, C55x Leadership DSP Performance 24,000 MMACS Up to 3 MB L2 Cache 1G EMAC, SRIO, DDR2, PCI-66 Comm, WiMAX, Industrial/Medical Imaging \$4.00 to \$99.00+ 



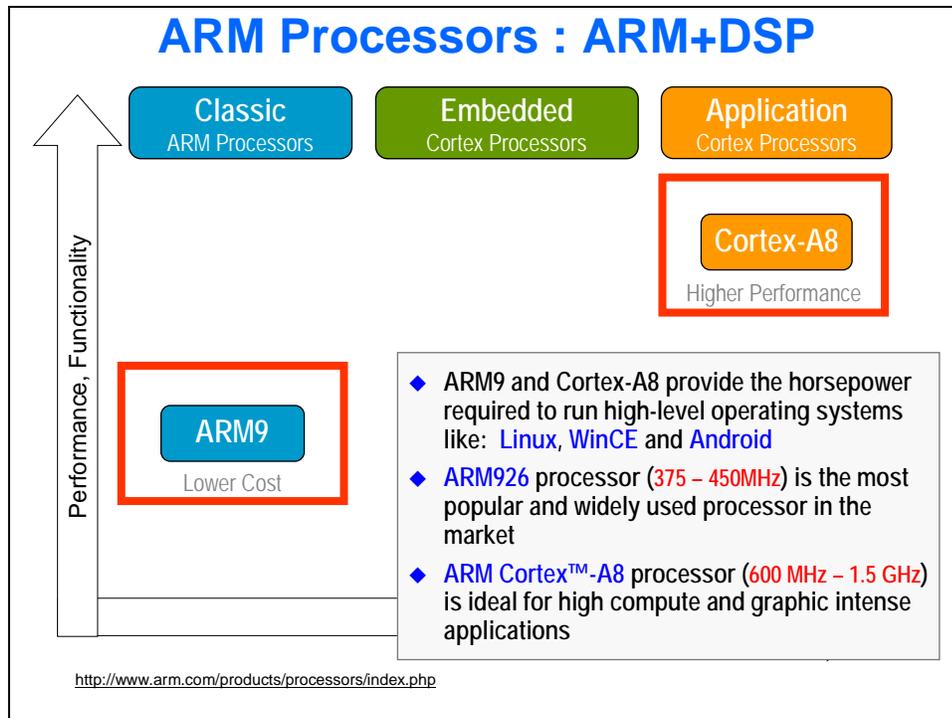
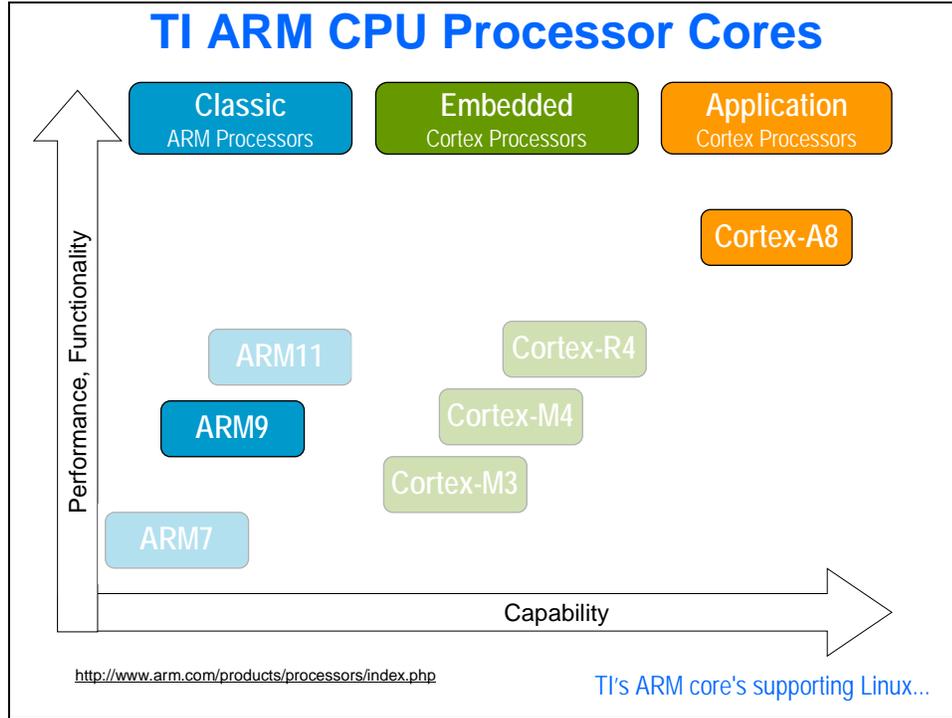
System Examples



What Processing Do You Need? (CPU, H/W Accelerators)



ARM Core

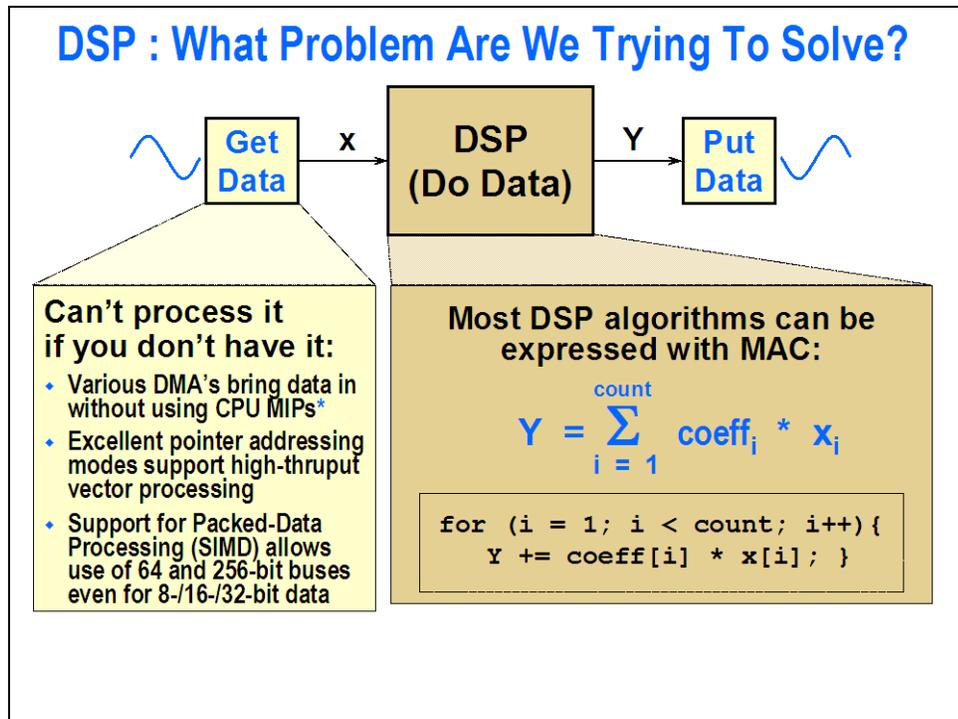


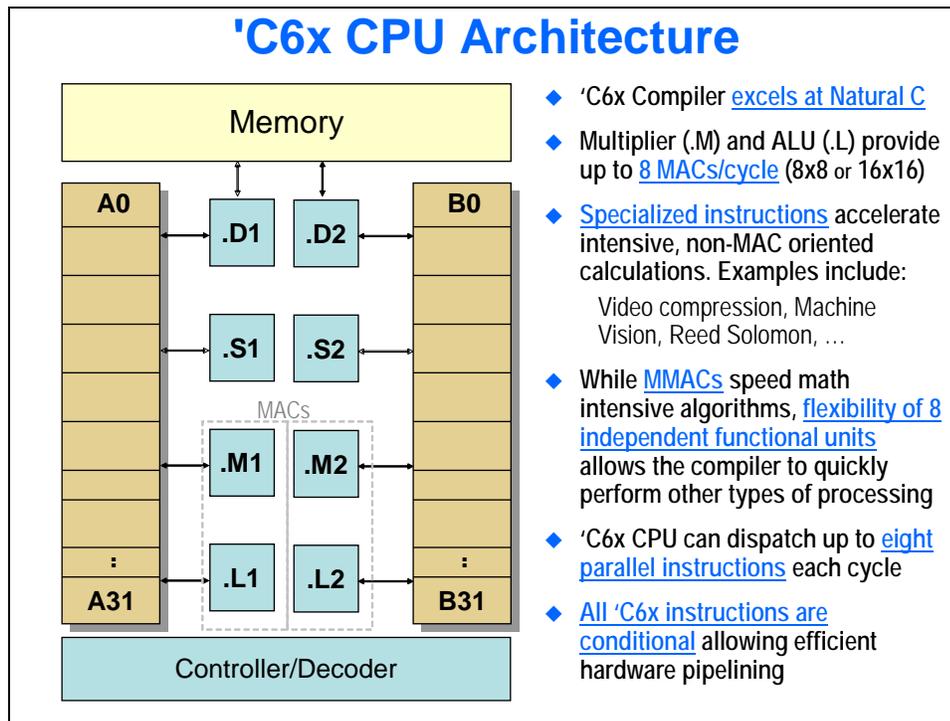
TI ARM Devices

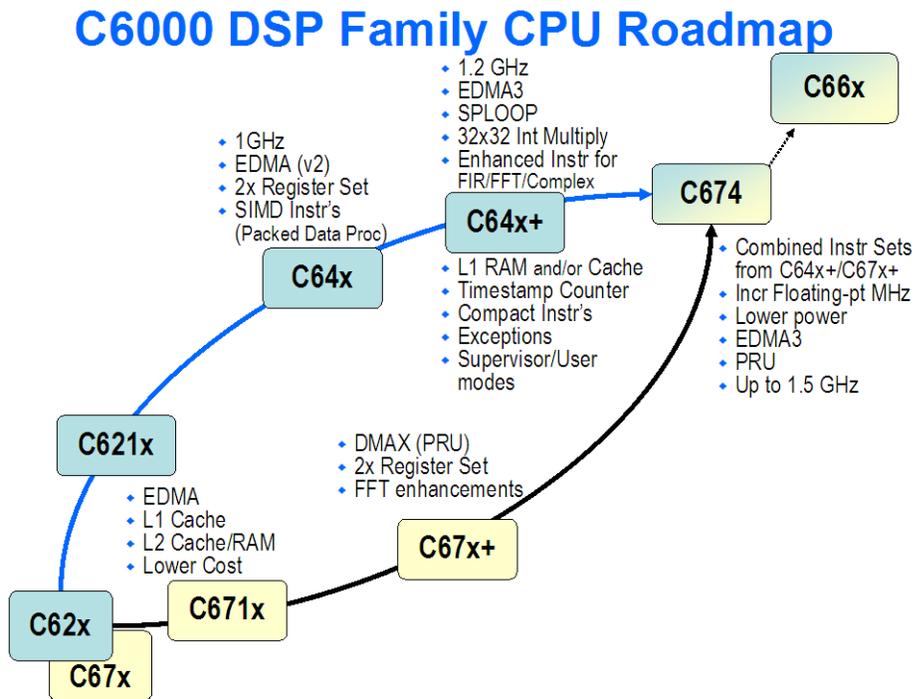
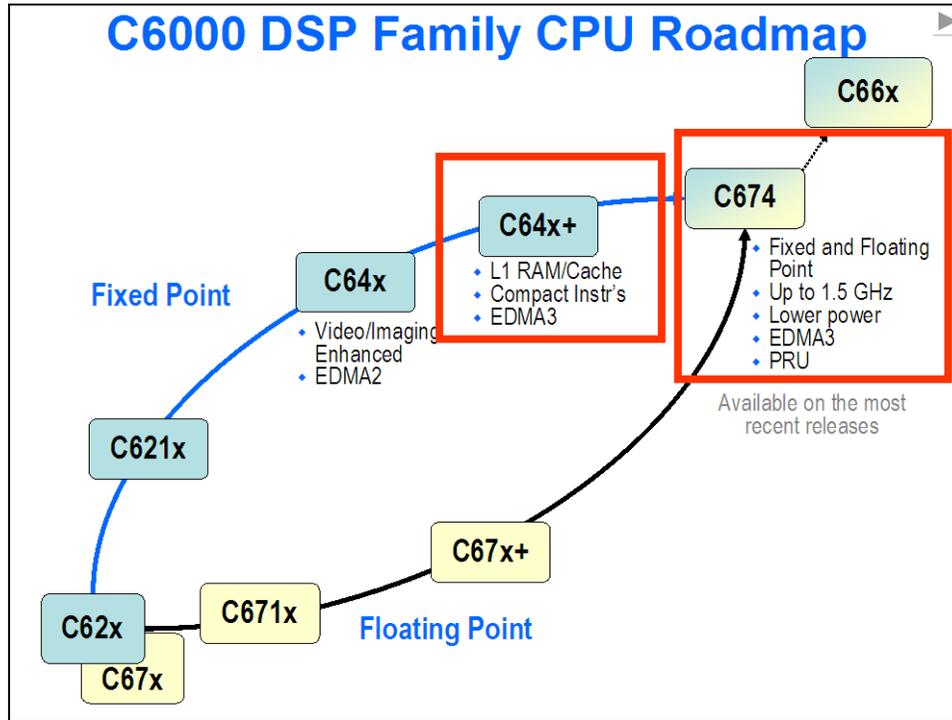
	ARM	ARM+DSP	DSP
ARM926	AM1705	OMAP-L137	C6747
	AM1707		
	AM1806	OMAP-L138	C6748
	AM1808		
	DM355	DM644x	DM6437
	DM365	DM6467	
Cortex A8	OMAP3503	OMAP3525	
	OMAP3515	OMAP3530	
	AM3505		
	AM3515		
	AM3703	DM3725	
	AM3715	DM3730	

Pin-for-Pin Compatibility

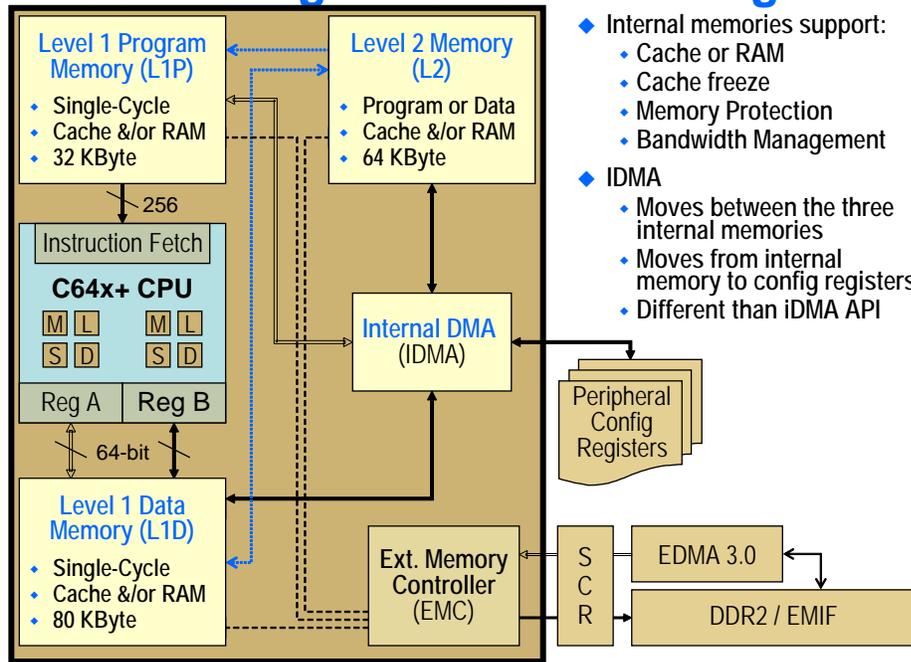
DSP Core







C64x+ MegaModule Block Diagram



Accelerator : 3D Graphics

Video ≠ Graphics

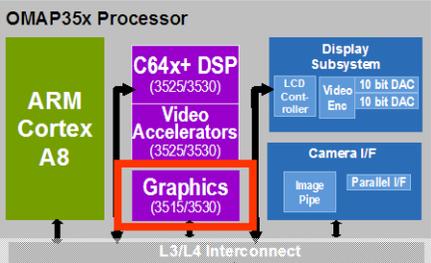
Video	Graphics
<ul style="list-style-type: none"> ◆ Captured ◆ Motion <ul style="list-style-type: none"> ▪ Integrated image ▪ Base image + motion ◆ YUV/YCbCr format <ul style="list-style-type: none"> ▪ Convert to RGB for display ◆ Video codec : block-based <ul style="list-style-type: none"> ▪ MPEG, WMV, JPG, etc. ◆ Natural attributes <ul style="list-style-type: none"> ▪ Digitized analog image 	<ul style="list-style-type: none"> ◆ Generated ◆ Animation <ul style="list-style-type: none"> ▪ Multi-layered image ▪ New delta images ◆ RGB format <ul style="list-style-type: none"> ▪ Ready for display ◆ Graphics : pixel-based <ul style="list-style-type: none"> ▪ BMP, GIF, etc. ◆ Calculated attributes <ul style="list-style-type: none"> ▪ Synthesized digital image





◆ With modern technology one can approach video performance and resolution, but the integrated effect of video is yet to be simulated in graphics.
 ◆ With increased HD video resolution, hardware acceleration is required; higher resolution – as well as demand for smooth 3D rendering – drives need for graphics acceleration

OMAP35x : ARM+DSP+Graphics



OMAP35x Processor

ARM Cortex A8, C64x+ DSP (3525/3530), Video Accelerators (3525/3530), Graphics (3515/3530), Display Subsystem (LCD Cont. roller, Video Enc, 10 bit DAC), Camera I/F (Image Pipe, Parallel I/F), System (Timers GPx12, WDT x2)

OMAP3515	OMAP3530
ARM® Cortex™ A8	ARM® Cortex™ A8
N/A	C64x+ DSP & video accelerator
2D/3D graphics accelerator – IMG SGX530	2D/3D graphics accelerator – IMG SGX530

PowerVR SGX Graphics Engine

- One of the highest performance graphics core in the market providing faster graphics performance without the need for an extra graphics card
- Off-loads tasks from ARM Cortex-A8 core and reduces system complexity

Also available on:

- AM3515
- AM3517
- DM3715
- DM3730



3D in SW only = Slower UI



3D HW Accelerated = Fast UI

Oct 2008

31

Accelerator : Audio/Video Algorithms

DM35x/DM36x

TMS320DM365 digital media processor

Performance

- 1080p : 10fps - H.264
- 720p : 30fps
 - H.264 HP
 - MPEG4 SP
 - MPEG2 MP@ML
 - WMV9/VC-1 AP
- JPEG enc/dec at 50 MPix/sec

Applications include:

- IP security camera, multi-channel DVR, baby monitor, video doorbell, digital signage, personal media player, HD web cam

Features

- Core**
 - ARM926EJ-S™ core, 300 MHz
 - H.264 coprocessor (HCP)
 - MPEG-4/JPEG coprocessor (MJCP)
 - Video Processing Subsystem (VPSS)
- Half power consumption in key modes**
- Peripheral Highlights**
 - EMAC 10/100
 - USB 2.0 HS OTG/mini-host w/PHY
 - External Memory Interface (EMIF)
 - Mobile DDR/DDR2
 - Keyscan/ADC
 - Audio Voice Codec

Video Processing Subsystem

Front End: CCD Controller Video Interface, Histogram/3A, Resizer, Image Signal Processing (ISP)

Back End: Enhanced On-Screen Display, Video Enc, 10b HD/SD DAC, 10b HD/SD DAC, 10b HD/SD DAC

ARM Subsystem: ARM9 CPU 300 MHz

MJCP Co-Processor: MPEG-4, JPEG

HCP Co-Processor: HCP

Peripherals: EDMA, Keyscan/ADC, EMAC 10/100, USB 2.0 HS OTG, Timer x6, WDT, PWM x4, ASP x2, IPC, HPI, mDDR/DDR2 EMIF, NAND/EC/EMIF, MMC/SDIO x2, RTC, Voice Codec

Connectivity: System

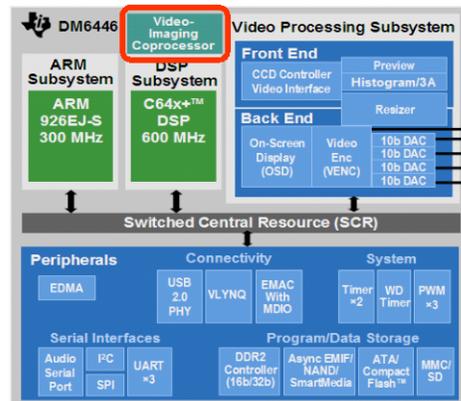
DMA Data & Configuration Bus

TEXAS INSTRUMENTS

DM6446 - VICP

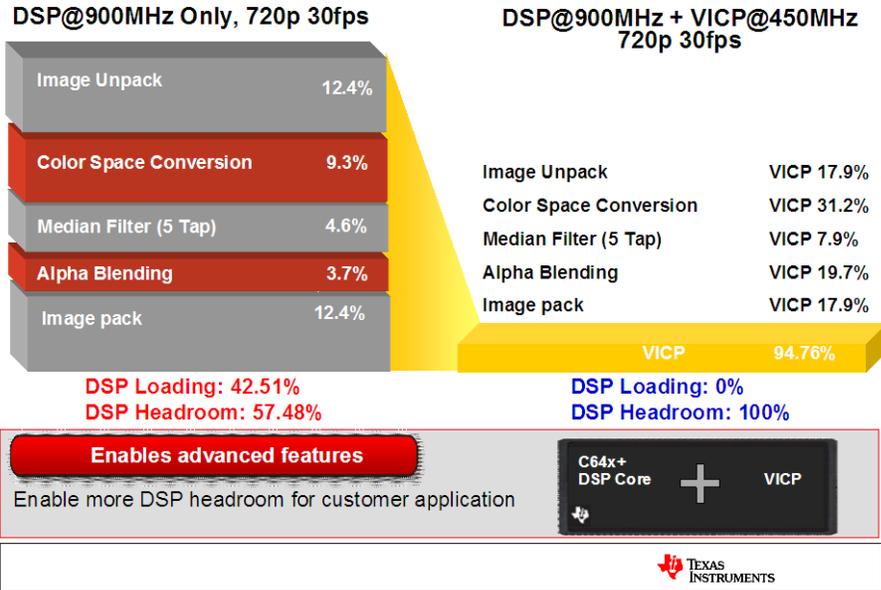
TMS320DM6446 : ARM9 + DSP + VICP

- Cores**
 - TMS320C64x+™ DSP Core at 810 MHz
 - VICP Accelerator at 405 MHz
 - Real-Time Encode /Decode
 - Signal Processing APIs
- Performance**
 - At 810MHz:**
 - H.264 BP 720p30 Decode
 - Simultaneous H.264 BP D1 Enc/Dec
 - Dual H.264 BP D1 Decode
 - At 594MHz:**
 - H.264 BP D1 Encode
 - Simultaneous H.264 BP CIF Encode
 - H.264 MP 30fps D1 Decode
 - VC1/WMV9 D1 Decode
 - MPEG2 MP D1 Decode
 - MPEG4 ASP D1 Decode
 - Etc.



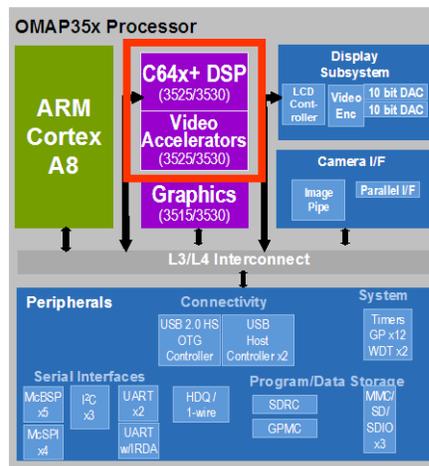
Signal Processing Libraries...

VICP enable additional DSP headroom for customer application



OMAP35xx – IVA2.2

OMAP35x Processors: ARM+DSP+Graphics



OMAP3525	OMAP3530
ARM® Cortex™ A8	ARM® Cortex™ A8
C64x+ DSP & video accelerator	C64x+ DSP & video accelerator
	2D/3D graphics accelerator – IMG SGX530
L2 256KB L1P 16KB L1D 16KB	L2 256KB L1P 16KB L1D 16KB
LPDDR@166MHz	LPDDR@166MHz
Neon float support	Neon float support
MPEG4 720p 24fps/30fps encode/decode H.264 MP VGA decode H.264BP/VC1/WMV9 D1 encode/decode	MPEG4 720p 24fps/30fps encode/decode H.264 MP VGA decode H.264BP/VC1/WMV9 D1 encode/decode
32 ch DMA, SSI, 5 McBSP, 2-3 UART, 4 I2C, IrDA, 4 SPI, MMC/SD, USB	32 ch DMA, SSI, 5 McBSP, 2-3 UART, 4 I2C, IrDA, 4 SPI, MMC/SD, USB

Pin-for-pin compatible

Oct 2008

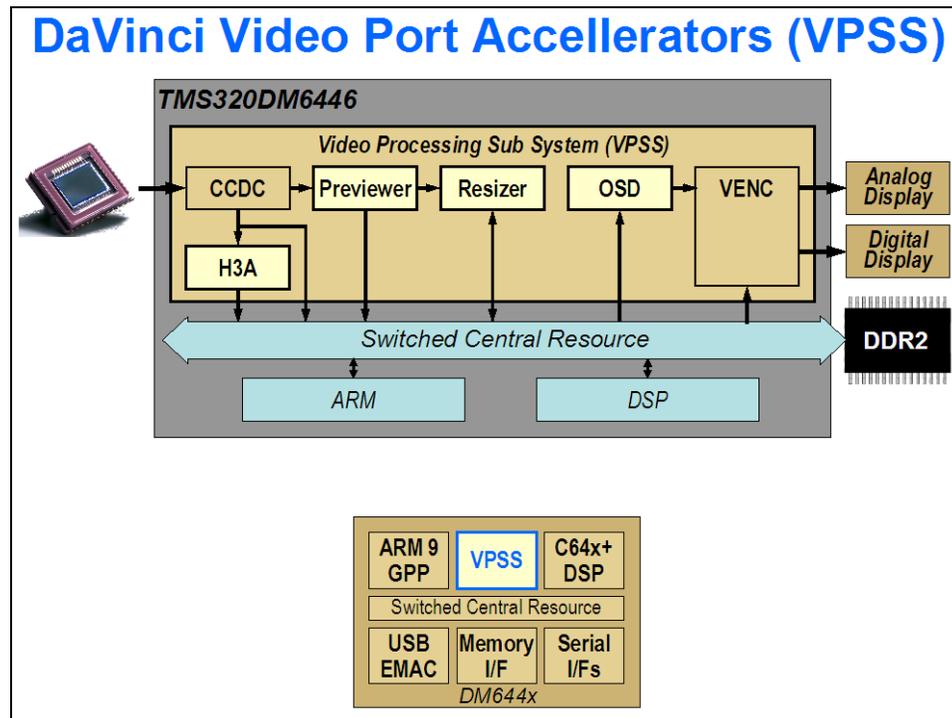
TEXAS INSTRUMENTS

42

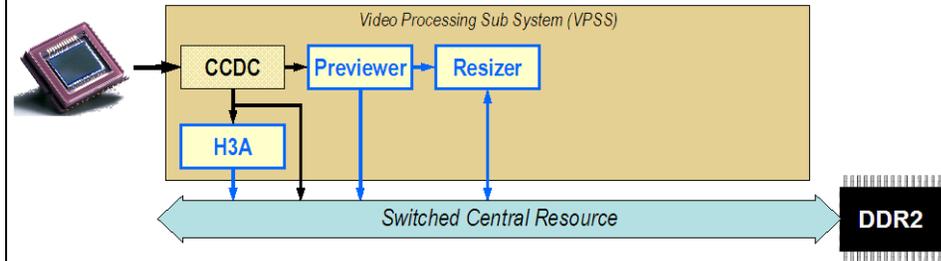
Accelerators : Video Port (VPSS) / Display (DSS)

The Video Port SubSystem (from DaVinci team) and Display SubSystem (from OMAP3) each contain a number of hardware accelerators to off-load a variety of video-related tasks from the ARM and/or DSP CPU's.

VPSS (Video Port SubSystem)



Front End : Resizer, Previewer, H3A



H3A

- ◆ Statistical engine for calculating image properties
- ◆ Histogram:
 - Histogram data collection (in RGB color space)
 - ARM + DSP can access these statistics
- ◆ Automatic Focus Control
- ◆ Automatic White Balance Correction
- ◆ Automatic Exposure Compensation

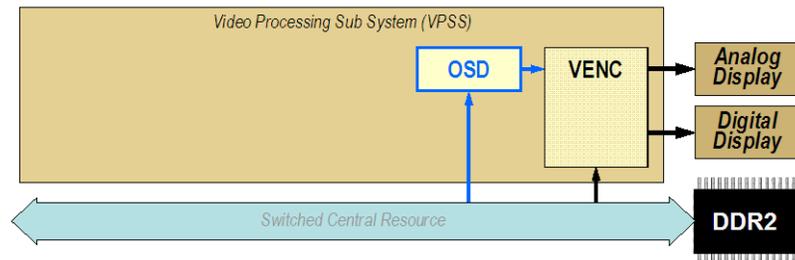
Previewer

- ◆ Bayer RGB to YCbCr 4:2:2 color space conversion
- ◆ Programmable noise filter
- ◆ Offloads processing effort

Resizer

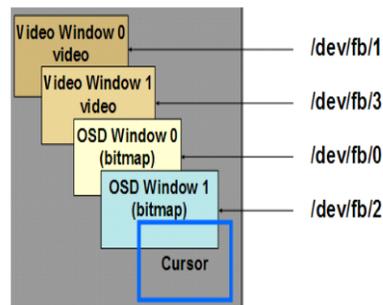
- ◆ 4x to 1/4x Resizing N/256 Zoom step
- ◆ Linear and Bi-Cubic Resize Algorithms
- ◆ Automatic Video Rescale
- ◆ Offloads processing effort

Back End : On-Screen Display (OSD)



Hardware On-Screen Display (OSD)

- ◆ 2 separate video windows
- ◆ 2 separate OSD windows
 - One can be used as attribute window for alpha-blending between video and OSD windows
- ◆ 1 rectangular cursor window
- ◆ 1 background color



OSD Usage : Set-Top Box Example

Video0 - Background

Video1 - Overlay (e.g. PIP)

OSD0 -
on-screen menu

OSD1 -
alpha-blending/
pixel-by-pixel
OSD attribute

Cursor -
as selection

OSD Attribute Window

- ◆ Allows Pixel by Pixel Blending of OSD0 and Video Windows
- ◆ Uses a 4-bit, Bit-map Window

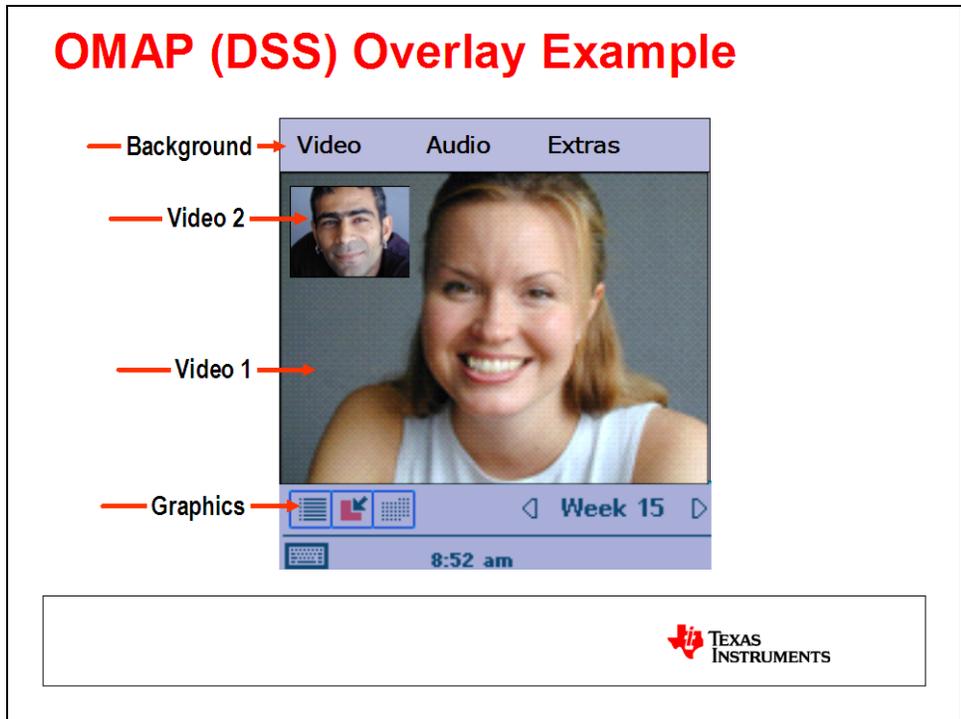
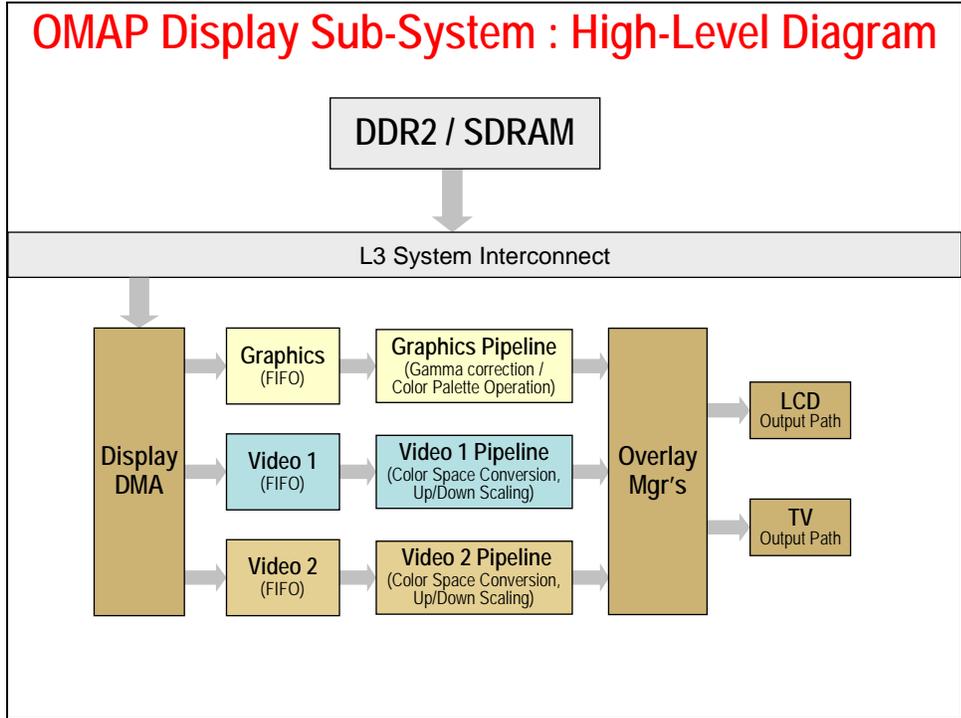
Blinking

8-level blending

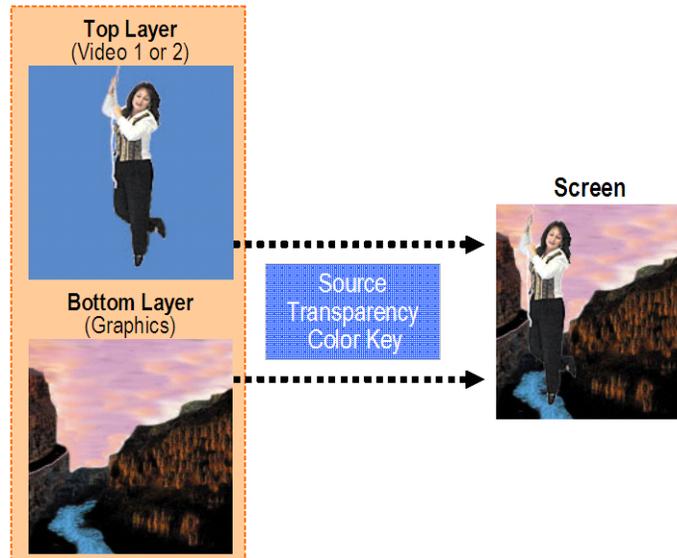
000:	00.0%,	100% Video
001:	12.5%,	87.5% Video
010:	25.0%,	75.0% Video
...		
110:	75.0%,	25.0% Video
111:	100%,	00.0% Video

111 110 101 100 011 010 001 000

DSS (Display SubSystem)



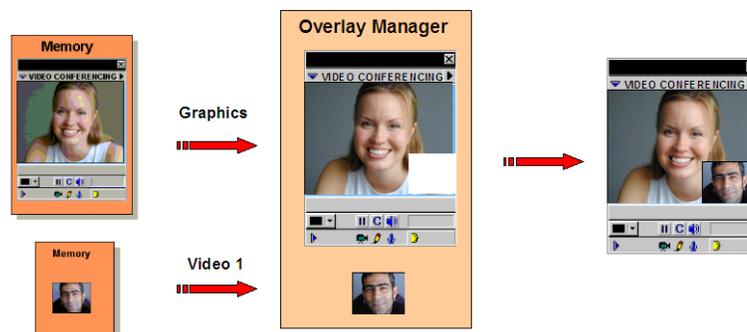
OMAP (DSS) Overlay Manager Color Key Example



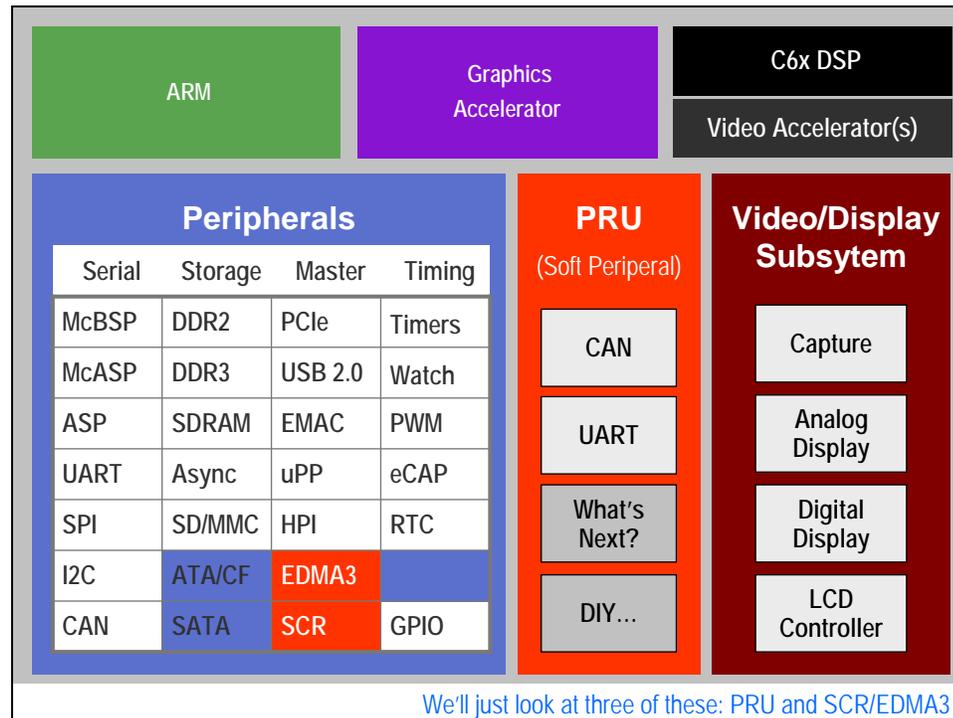
OMAP Display Overlay Manager

Overlay Optimization

- ◆ Only fetch needed pixels from memory
 - At least video window 1 and graphics window must be enabled
 - The graphics pixels under the video 1 will not be fetched from the memory
 - The transparency color key must be disabled
- ◆ Reduces the peak bandwidth
 - Only visible pixels from graphics and video buffers are fetched and displayed



Peripherals



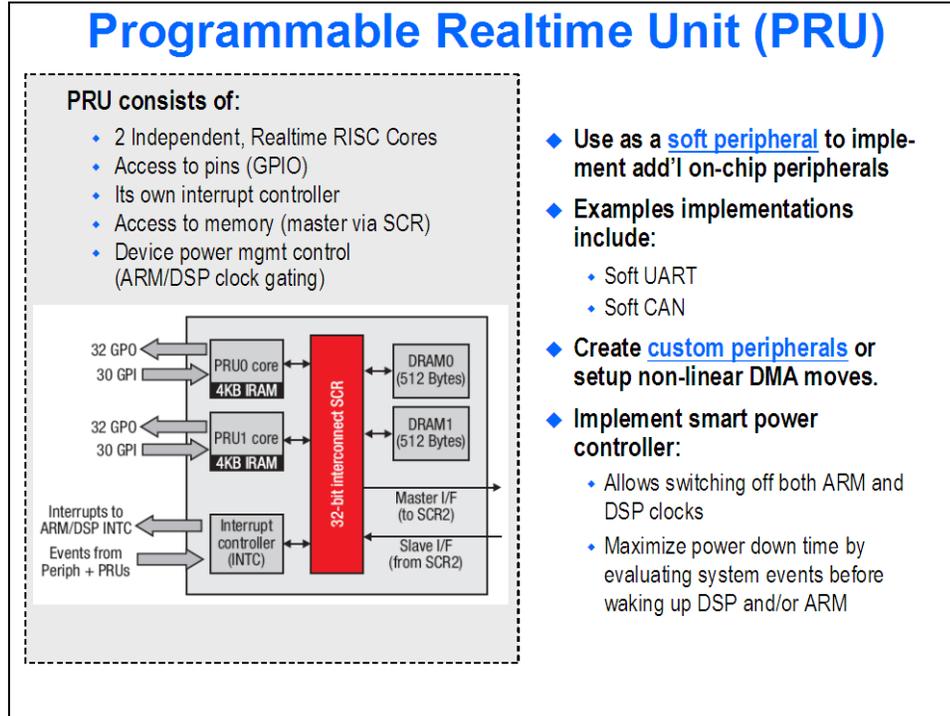
Between the enormity of examining the features of every one of these peripherals, and the fact that each device has a different subset of peripherals, we just don't have the time to dig into each one of them.

For this reason, we'll only take a brief look at three of them:

- PRU – Programmable Real-time Unit
- SCR – Switched Central Resource (i.e. internal bus crossbar)
- EDMA3 – Enhanced DMA controller (version 3)

The first two are probably not very self-explanatory. The third, conveniently, is part of the SCR discussion.

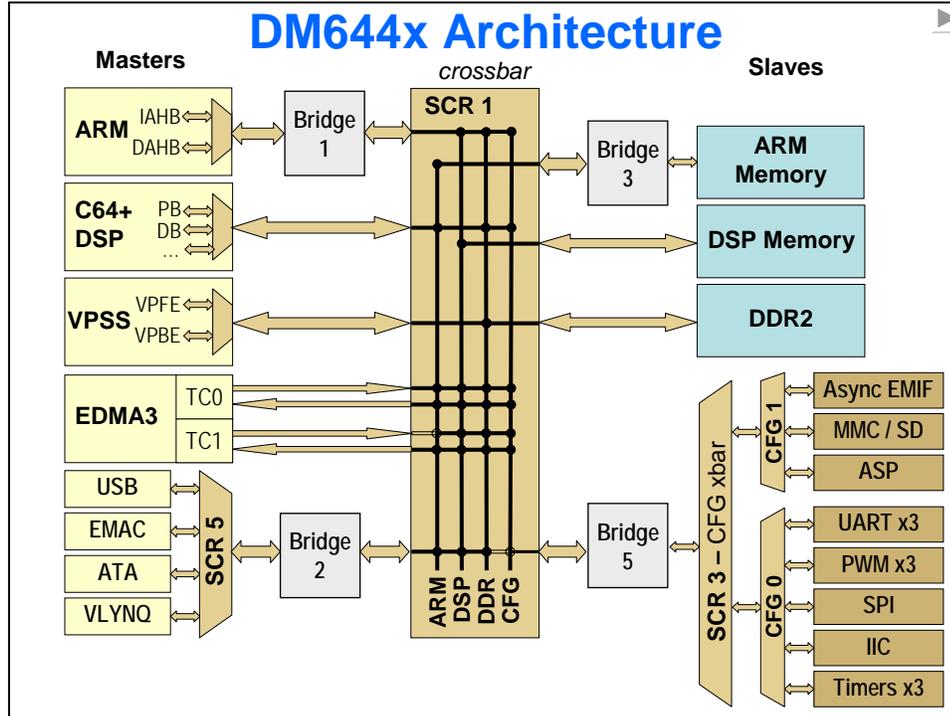
PRU – Programmable Real-time Unit



PRU SubSystem : IS / IS-NOT

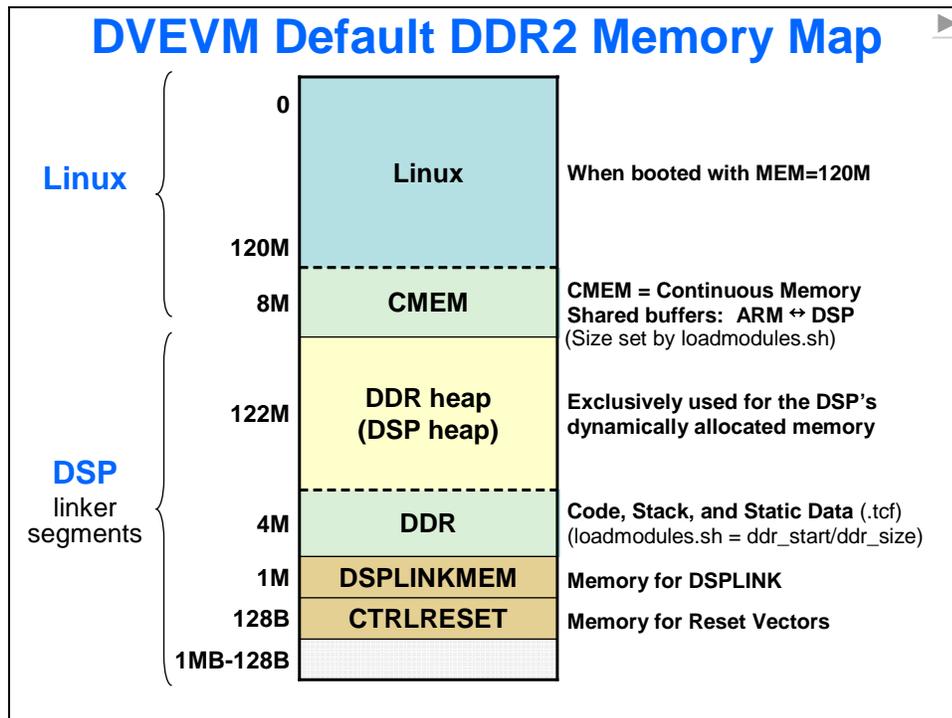
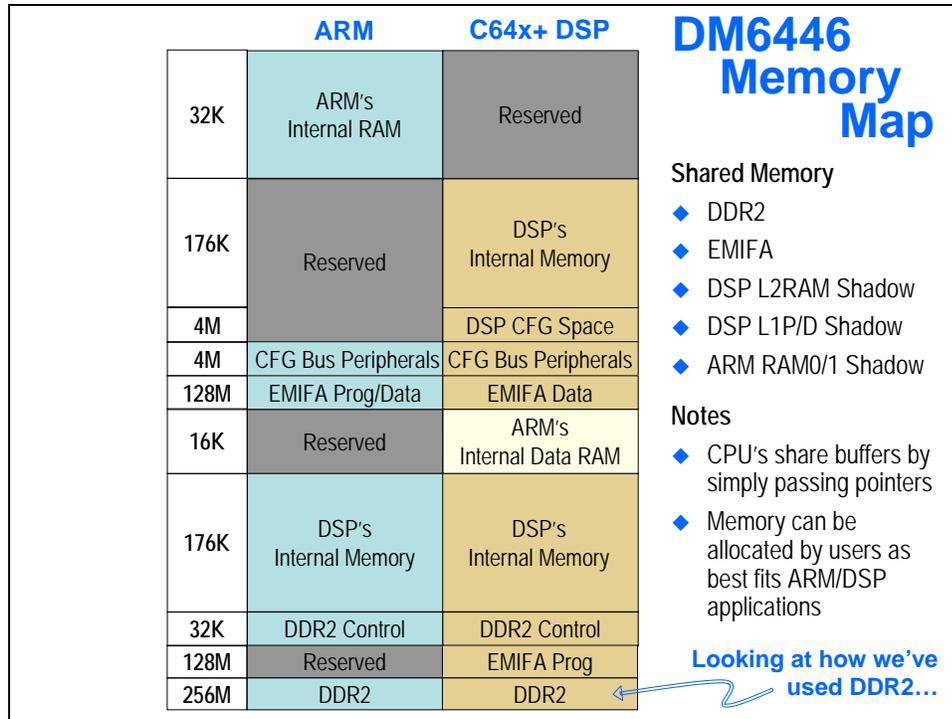
Is	Is-Not
Dual 32-bit RISC processor specifically designed for manipulation of packed memory mapped data structures and implementing system features that have tight real time constraints.	Is not a H/W accelerator used to speed up algorithm computations.
Simple RISC ISA: <ul style="list-style-type: none"> ▪ Approximately 40 instructions ▪ Logical, arithmetic, and flow control ops all complete in a single cycle 	Is not a general purpose RISC processor: <ul style="list-style-type: none"> ▪ No multiply hardware/instructions ▪ No cache or pipeline ▪ No C programming
Simple tooling: Basic command-line assembler/linker	Is not integrated with CCS. Doesn't include advanced debug options
Includes example code to demonstrate various features. Examples can be used as building blocks.	No Operating System or high-level application software stack

Moving Data Around – SCR / EDMA3



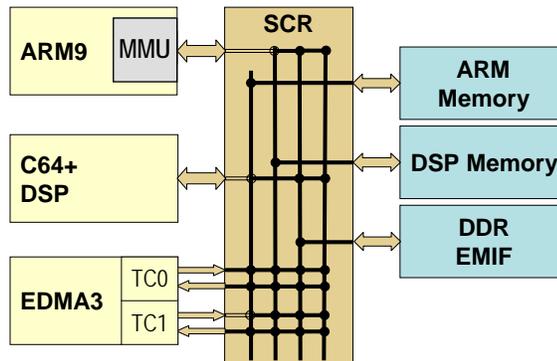
Final Considerations

Memory Map & MMU



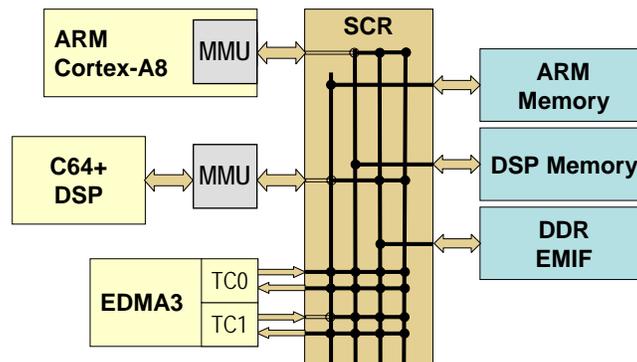
MMU

DM6446 Memory Mgmt Unit (MMU)



- ◆ ARM's memory accessed thru MMU
 - Benefits: virtual memory, memory protection
 - MMU discussed briefly in Chapters 8 and 11
- ◆ DSP has no MMU (only memory protection which is not commonly used)
 - DSP can access ARM's memory – easy buffer sharing (but less robust?)
 - No MMU in memory path increases performance
 - Address translation req'd during buffer passing from ARM to DSP – thankfully, Codec Engine performs this task

OMAP35x MMU's



- ◆ ARM's memory still accessed thru MMU
- ◆ DSP now has an MMU in its memory path
 - Common usage: "Set & Forget"
 - Setting it only once at DSP loadtime protects *Linux-space* while minimizing MMU performance degradation
 - Again, Codec Engine framework sets the MMU for us at loadtime

Access to Peripherals

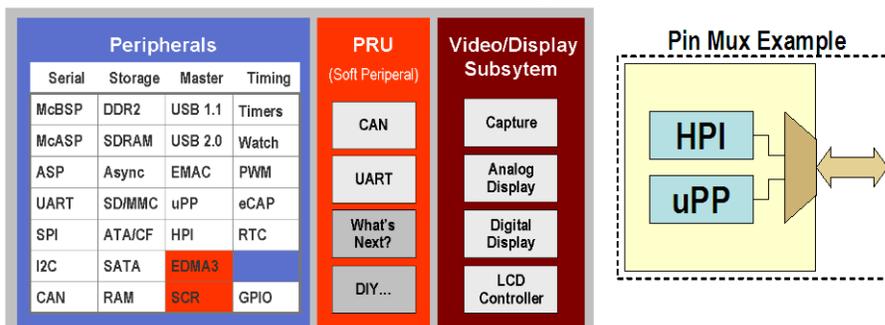
TMS320DM6446 Peripheral Ownership

Peripheral	ARM	DSP
VPSS (Video capture/display)	✓	
EMAC/MDIO (Ethernet)	✓	
USB 2.0	✓	
ARM Interrupt Controller	✓	
ATA/CF, MMC/SD	✓ ✓	
GPIO, PWM0/1/2, Watchdog Timer	✓ ✓ ✓ ✓ ✓	
I2C, SPI, UART0/1/2	✓ ✓ ✓ ✓ ✓	
EDMA	✓	✓
Timer0/1	✓	✓
ASP (Audio Serial Port)	✓	✓
DDR2, EMIF	✓ ✓	✓ ✓

- ◆ Some devices provide full access to all peripherals from the ARM and DSP
- ◆ ARM-only access: OMAP35, DM37x, DM6467, DM644x*
- ◆ ARM/DSP access: OMAP-L137, OMAP-L138

Pin Muxing

What is Pin Multiplexing?



- ◆ How many pins is on your device?
- ◆ How many pins would all your peripheral require?
- ◆ Pin Multiplexing is the answer – only so many peripherals can be used at the same time ... in other words, to reduce costs, peripherals must share available pins
- ◆ Which ones can you use simultaneously?
 - ◆ Designers examine app use cases when deciding best muxing layout
 - ◆ Read datasheet for final authority on how pins are muxed
 - ◆ Graphical utility can assist with figuring out pin-muxing...

[Pin mux utility...](#)

Pin Muxing Tools

- ◆ Graphical Utilities For Determining which Peripherals can be Used Simultaneously
- ◆ Provides Pin Mux Register Configurations
- ◆ http://wiki.davincisp.com/index.php?title=Pinmux_Uilities_for_Davinci_Processors

Choosing a Device : Web Tool

DSP & ARM MPU Selection Tool

130 Results found

Part Number	ARM Processor	ARM MHz (Max.)	Operating System	Application	Instruction Set	DSP MHz (Max.)	16x16 MMACS	Real Time OS	SDRAM Interface	On Chip Memory	Video Capability	Video Coders	Video Resolution	Audio Coders	Video Ports (8-bit)	Video Ports (16-bit)	Video Interface
TMS320DM648-900	No	0	No	No	C64x	900	720x	DSP/BI	DDR2	576	Decode,Encode	JPEG,MPEG2,MPEG4-SP,MPEG4-ASP,VC1,H.2	D1 or Le	G.711,MI	10	5	BT
TMS320DM648-720	No	0	No	No	C64x	720	576x	DSP/BI	DDR2	576	Decode,Encode	JPEG,MPEG2,MPEG4-SP,MPEG4-ASP,VC1,H.2	D1 or Le	G.711,MI	10	5	BT
TMS320DM647-900	No	0	No	No	C64x	900	720x	DSP/BI	DDR2	320	Decode,Encode	JPEG,MPEG2,MPEG4-SP,MPEG4-ASP,VC1,H.2	D1 or Le	G.711,MI	10	5	BT
TMS320DM647-720	No	0	No	No	C64x	720	576x	DSP/BI	DDR2	320	Decode,Encode	JPEG,MPEG2,MPEG4-SP,MPEG4-ASP,VC1,H.2	D1 or Le	G.711,MI	10	5	BT
TMS320DM643-600	No	0	No	No	C64x	600	480x	DSP/BI	SDRAM	288	Decode,Encode	JPEG,MPEG2,MPEG4-SP,MPEG4-ASP,VC1,H.2	D1 or Le	G.711,MI	4	2	BT
TMS320DM643-500	No	0	No	No	C64x	500	200x	DSP/BI	SDRAM	288	Decode,Encode	JPEG,MPEG2,MPEG4-SP,MPEG4-ASP,VC1,H.2	D1 or Le	G.711,MI	4	2	BT

http://focus.ti.com/en/multimedia/flash/selection_tools/dsp/dsp.html

To force proper pagination, this page was intentionally left almost blank.

Intro to TI's Foundation Software

Introduction

At this point we have seen an overview of TI's ARM and ARM+DSP devices (Sitara, DaVinci, OMAP, Integra). Next we will explore the basic components of the TI software model. Each of the concepts outlined here will be discussed in further detail in succeeding chapters, but in this chapter we hope to give you a look at the big picture.

Learning Objectives

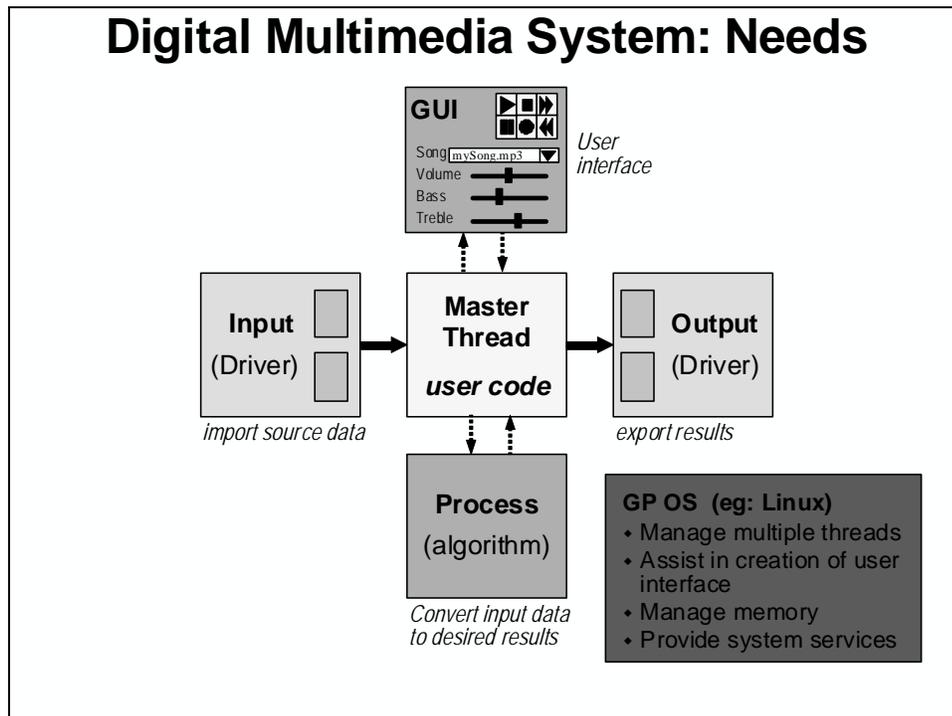
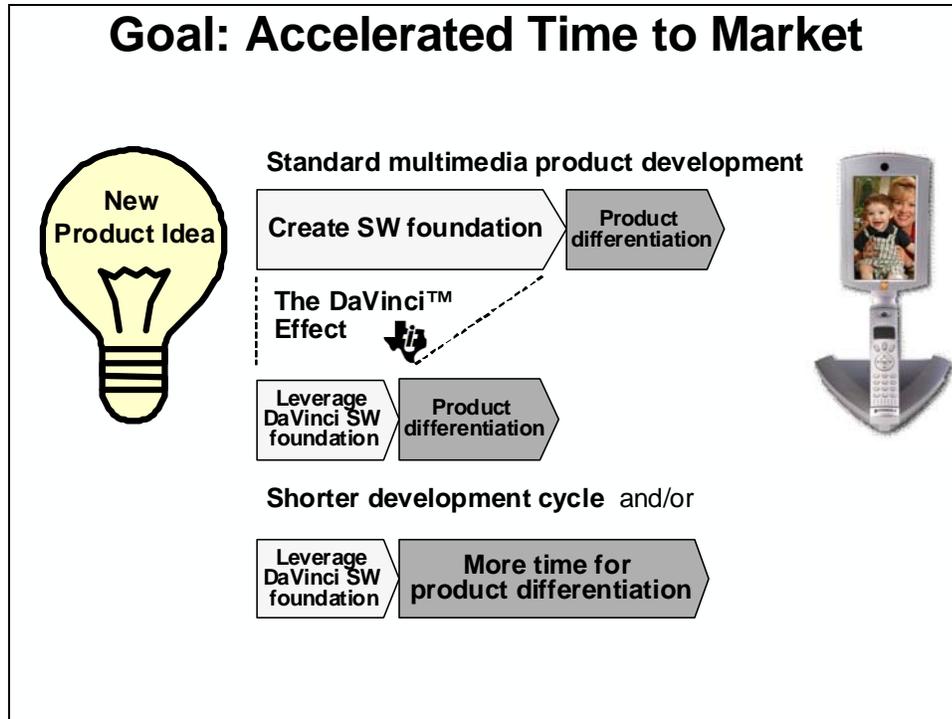
◆ Linux Applications	Concepts Driver Interface (Linux) VISA Interface (Codec Engine)
◆ Algorithms/Codecs (Signal Processing)	xDAIS (eXpressDSP™ Algorithm Standard) xDM (extending xDAIS with Classes) VISA vs xDM Where to Get Your Algorithms
◆ Codec Engine Details	ARM or ARM+DSP RPC - Remote Procedure Call Code Review
◆ Software Summary	

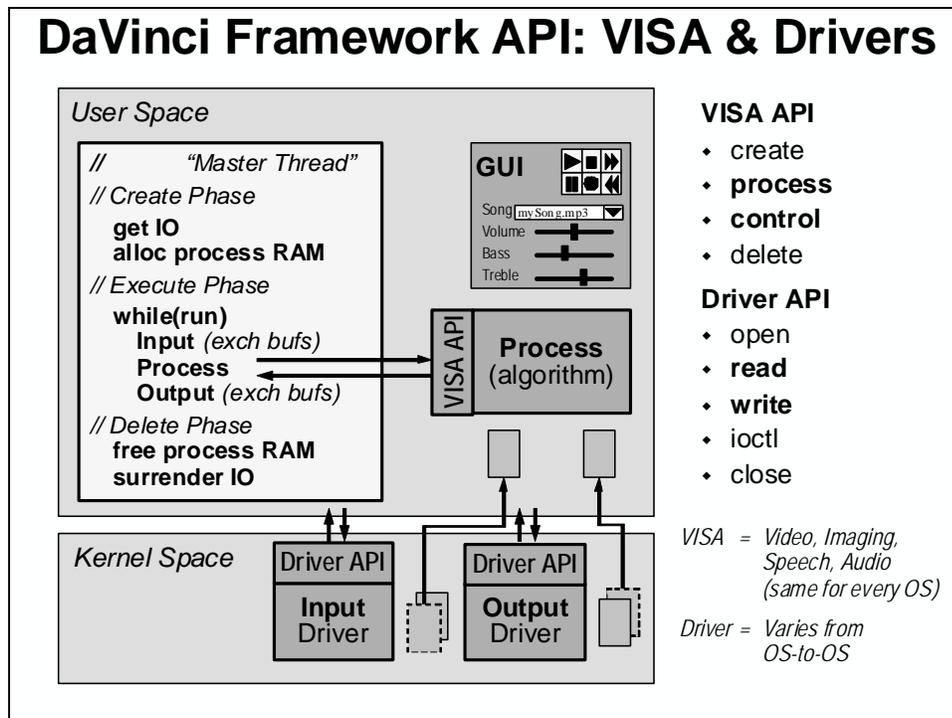
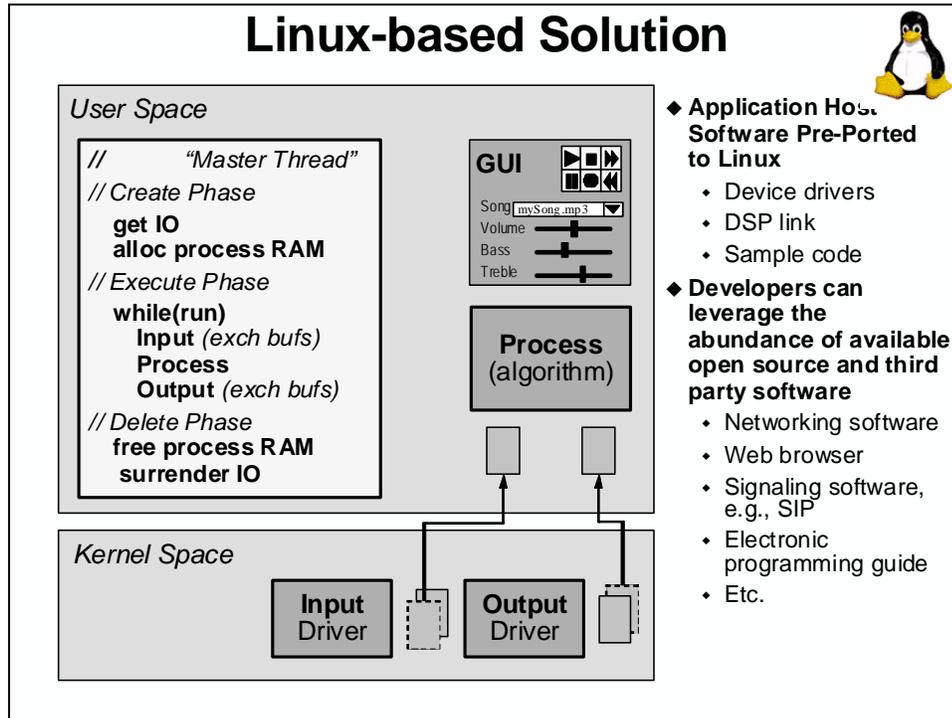
Chapter Topics

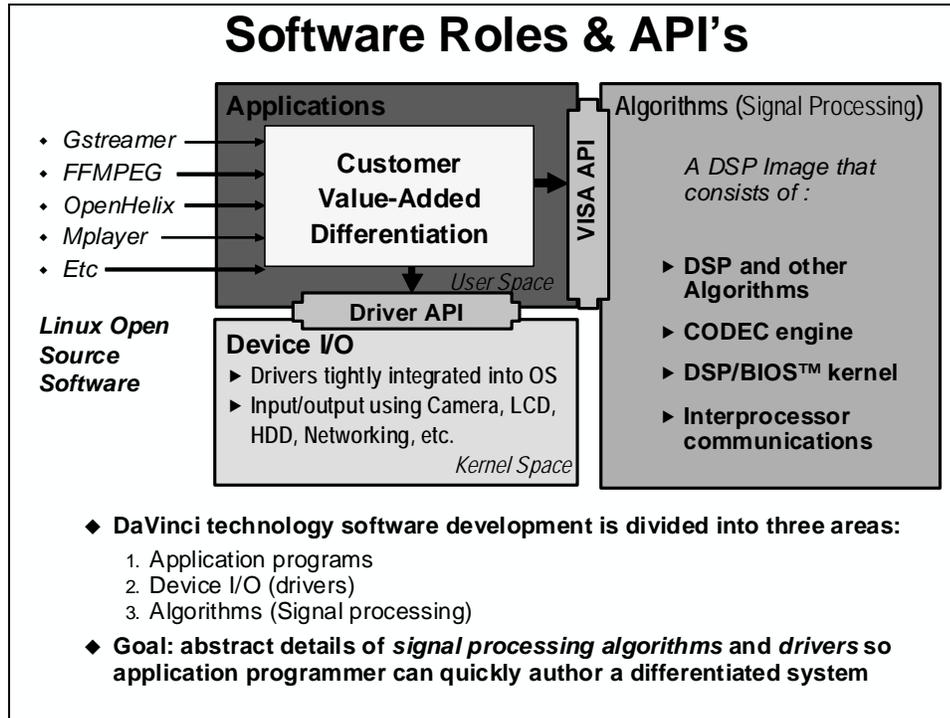
Intro to TI's Foundation Software.....	2-1
<i>Application Layer.....</i>	<i>2-3</i>
Concepts	2-3
I/O Layer - Linux Device Drivers	2-6
Access Signal Processing Horsepower using the Codec Engine's VISA Interface	2-7
Adding VISA to our "Master Thread":.....	2-9
<i>Signal Processing Layer – Algorithms and Codecs.....</i>	<i>2-10</i>
What is a "Codec"?.....	2-10
xDAIS (eXpressDSP™ Algorithm Standard)	2-11
xDM (Extending xDAIS with Algorithm Classes).....	2-13
VISA vs. xDM.....	2-14
<i>Where to Get Your Algorithms.....</i>	<i>2-17</i>
<i>CODEC Engine Details.....</i>	<i>2-19</i>
RPC – Remote Procedure Call	2-20
Code Review	2-22
<i>Software Summary.....</i>	<i>2-26</i>
<i>For More Information.....</i>	<i>2-28</i>

Application Layer

Concepts







I/O Layer - Linux Device Drivers

Linux Device Drivers

- ◆ **Driver API's**
 - Collection of drivers chosen to support the I/O ports for each O/S
 - Where available, common O/S drivers are chosen (Linux - see below)
 - Where no common driver exists, TI must create its own driver API

- ◆ **Linux Drivers**
 - Storage - ATA/IDE, NAND, NOR, MMC/SD
 - Audio - ALSA, OSS Audio driver
 - Video - V4L2 for Capture/Display, FBDev for OSD graphics, IPIPE, Resizer, Previewer, H3A
 - Network - Ethernet
 - USB - Mass storage - Host and Gadget drivers
 - Serial - UART, I2C, SPI
 - Other - PWM, Watchdog, EDMA, GPIO
 - Boot - Das U-Boot (open source Linux boot-loader)



Linux – Basic File I/O & Character Driver API

Basic Linux file I/O usage in user programs is via these API:

```
myFileFd = fopen("/mnt/harddrive/myfile", "rw");
fread ( aMyBuf, sizeof(int), len, myFileFd );
fwrite( aMyBuf, sizeof(int), len, myFileFd );
fclose( myFileFd );
```

Additionally, you can use `fprintf()` and `fscanf()` for more feature-rich file read/writes.

Simple drivers use the same format as files...

```
soundFd = open ("/dev/dsp", O_RDWR);
read ( soundFd, aMyBuf, len );
write( soundFd, aMyBuf, len );
close( soundFd );
```

Additionally, drivers use I/O control (`ioctl`) commands to set driver characteristics

```
ioctl( soundFd, SNDCTL_DSP_SETFMT, &format );
```

- Basic drivers will be covered in more detail in Chapter 6.
- Some Linux drivers (such as V4L2 and FBDEV video drivers) typically use `mmap` and `ioctl` commands instead of `read` and `write` that pass data by reference instead of by copy. These will be studied in greater detail in the Chapter 7.

Master Thread – Accessing I/O

```
idevfd = open("/dev/xxx", O_RDONLY);
ofilefd = open("./fname", O_WRONLY);
ioctl(idevfd, CMD, &args);
```

```
while( doRecordVideo == 1 ) {
    read(idevfd, &rd, sizeof(rd));
```

```
    write(ofilefd, &wd, sizeof(wd));
}
```

```
close(idevfd);
close(ofilefd);
```

```
// Create Phase
// get input device
// get output device
// initialize IO devices...
```

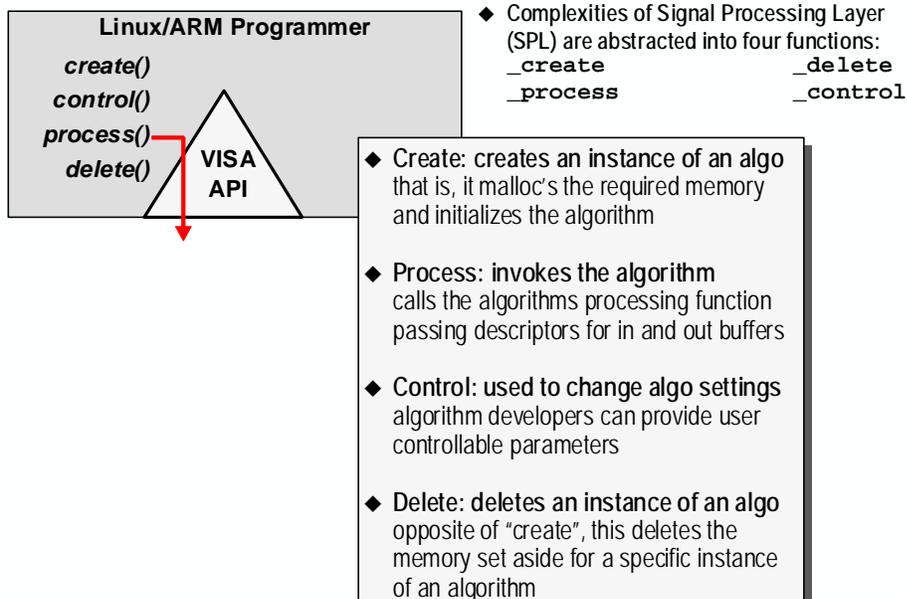
```
// Execute phase
// read/swap buffer with Input device
```

```
// pass results to Output device
```

```
// Delete phase
// return IO devices back to OS
```

Access Signal Processing Horsepower using the Codec Engine's VISA Interface

VISA – Four SPL Functions



VISA Benefits

Application Author Benefits

- ◆ App author enjoys benefits of signal processing layer without need to comprehend the complexities of the DSP algo or underlying hardware
- ◆ Application author uses only *one* API for a given media engine class
- ◆ Changing CODEC within the class involves *no* changes to app level code
- ◆ All media engine classes have a similar look and feel
- ◆ Adapting any app code to other engines and API is very straight forward
- ◆ Example apps that use VISA to manage xDM CODECs provided by TI
- ◆ Customers can create multimedia frameworks that will leverage VISA API
- ◆ VISA contains hooks allowing additional functionalities within CODECs
- ◆ Authoring app code, multimedia frameworks & end equipment expertise is what customers do best, and want to focus on - VISA optimizes this

Algorithm Author Benefits

- ◆ CODEC engine authors have a known standard to write to
- ◆ CODEC authors need have no knowledge of the end application
- ◆ CODECs can be sold more readily, since they are easy to apply widely
- ◆ Each class contains the information necessary for that type of media
- ◆ VISA, and xDAIS-DM, build on xDAIS – an established algo interface
- ◆ Tools exist today to adapt algos to xDAIS, and may include –DM soon (?)

Adding VISA to our “Master Thread”:

Master Thread Key Activities ▶

<pre> idevfd = open("/dev/xxx", O_RDONLY); ofilefd = open("./fname", O_WRONLY); ioctl(idevfd, CMD, &args); myCE = Engine_open("vcr", myCEAttrs); myVE = VIDENC_create(myCE, "videnc", params); while(doRecordVideo == 1) { read(idevfd, &rd, sizeof(rd)); VIDENC_process(myVE, ...); //VIDENC_control(myVE, ...); write(ofilefd, &wd, sizeof(wd)); } close(idevfd); close(ofilefd); VIDENC_delete(myVE); Engine_close(myCE); </pre>	<pre> // Create Phase // get input device // get output device // initialize IO devices... // prepare VISA environment // prepare to use video encoder // Execute phase // read/swap buffer with Input device // run algo with new buffer // optional: perform VISA algo ctrl // pass results to Output device // Delete phase // return IO devices back to OS // algo RAM back to heap // close VISA framework </pre>
---	--

◆ See Chapter 9 for more details of VISA functions (i.e. prototypes)

Signal Processing Layer – Algorithms and Codecs

What is a “Codec”?

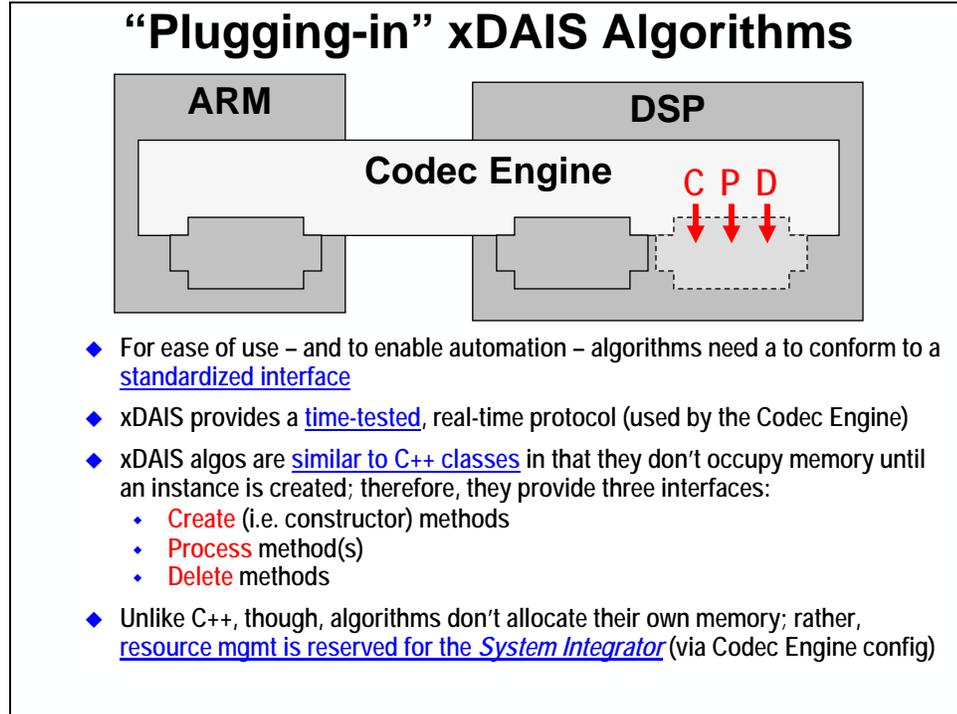
Signal Processing Layer : What is a Codec?

1. Compression / Decompression algorithm
2. Single device containing both an analog-to-digital (A/D) and digital-to-analog (D/A) converter
3. In the world of TI's DaVinci software, we often use the term “codec” to refer to any real-time algorithm

Should the framework have been called the “Algo Engine”, or how about the “xDAIS Engine”?

In the end, “Codec Engine” won out.

xDAIS (eXpressDSP™ Algorithm Standard)



xDAIS/xDM for Algorithm Authors

- ◆ xDAIS provides a consistent, straight-forward set of methods for algorithm authors to specify an algorithm's resource needs ...
... therefore, using xDAIS compliant algos allows *users to easily manage resources*, such as system memory, DMA, and accelerators via a consistent API
- ◆ Algorithms *cannot take resources*, but must request them – this is done thru standard xDAIS required functions
- ◆ Codec Engine's VISA classes match up to xDM classes – xDM (xDAIS for Digital Media) is an extension to xDAIS which defines the *plug-n-play* multimedia algorithm classes
- ◆ Similar to C++ classes, algo's shouldn't use global variables, but rather bundle them into an *instance object* (i.e. class object)

Required Algorithm Functions

	Description	Function
Instance Creation (i.e. Constructor) e.g. VIDENC_create()	Specify Memory requirements	(algAlloc)
	Specify DMA requirements	(dmaAlloc)
	Initialize algorithm	(algInit, dmaInit)
Process VIDENC_process()	Prepare scratch memory	(algActivate)
	Run algorithm	(process)
	Save scratch data	(algDeactivate)
Delete VIDENC_delete()	Free resources	(algFree, dmaFree)

iAlg Functions Summary

◆ Create Functions (i.e. Constructor Functions)

- **algNumAlloc** - Tells application (i.e. CODEC engine) how many blocks of memory are required; it usually just returns a number
- **algAlloc** - Describes properties of each required block of memory (size, alignment, location, scratch/persistent)
- **algInit** - Algorithm is initialized with specified parameters and memory

◆ Execute Functions

- **algActivate** - Prepare scratch memory for use; called prior to using algorithm's process function (e.g. prep history for filter algo)
- **algDeactivate** - Store scratch data to persistent memory subsequent to algo's process function
- **algMoved** - Used if application relocates an algorithm's memory

◆ Delete Function

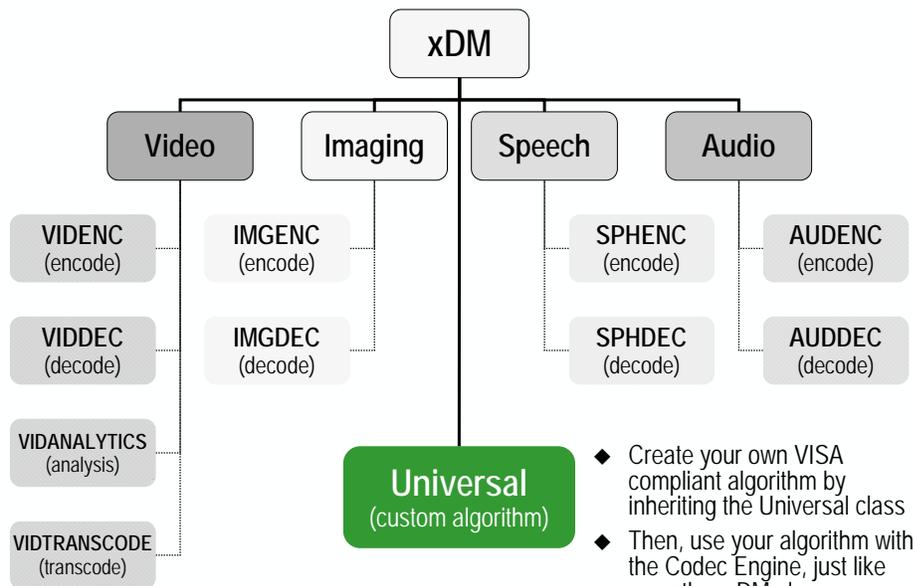
- **algFree** - Algorithm returns descriptions of memory blocks it was given, so that the application can free them

xDM (Extending xDAIS with Algorithm Classes)

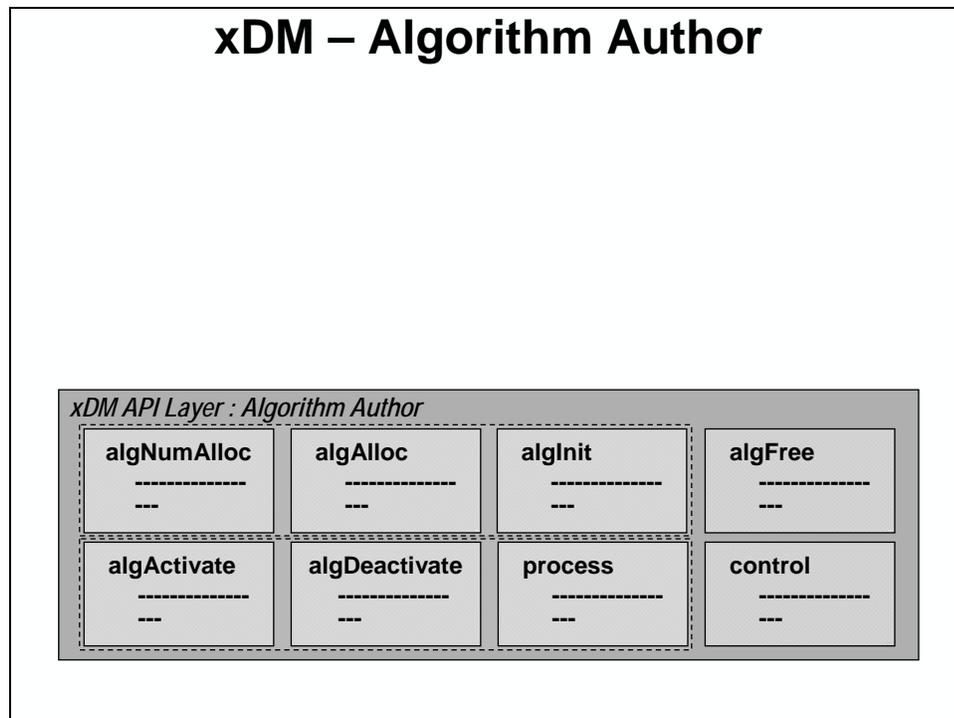
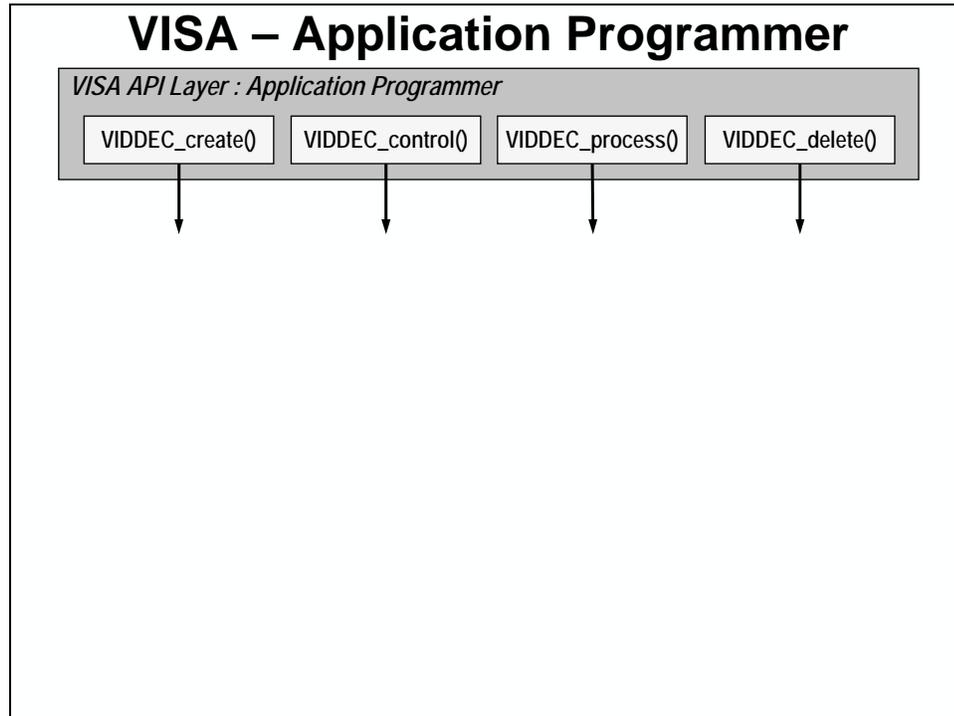
xDM: Extending xDAIS

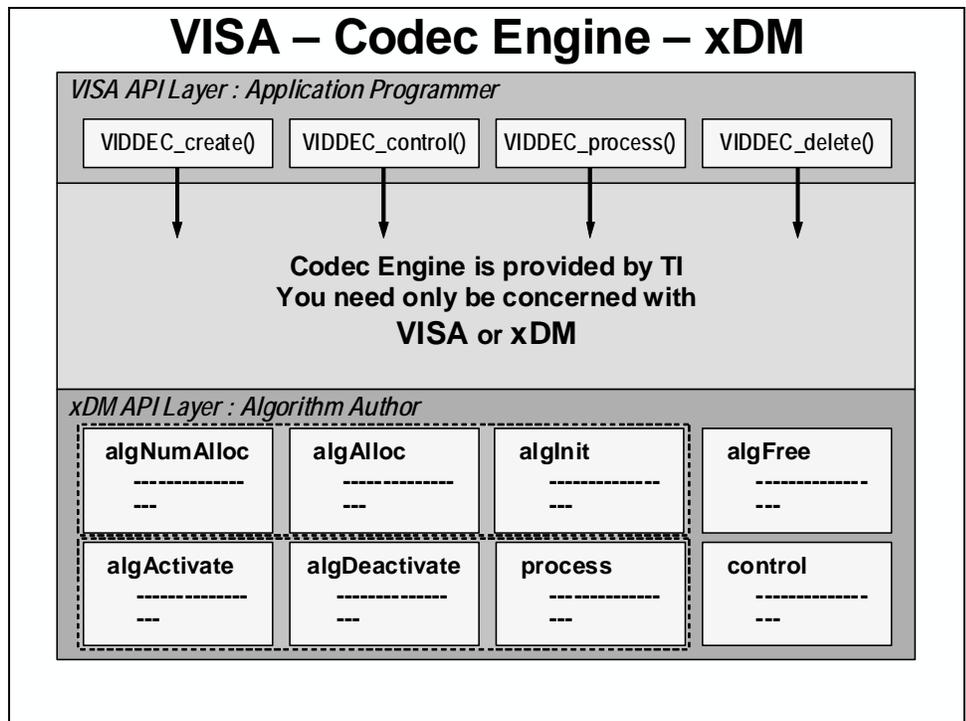
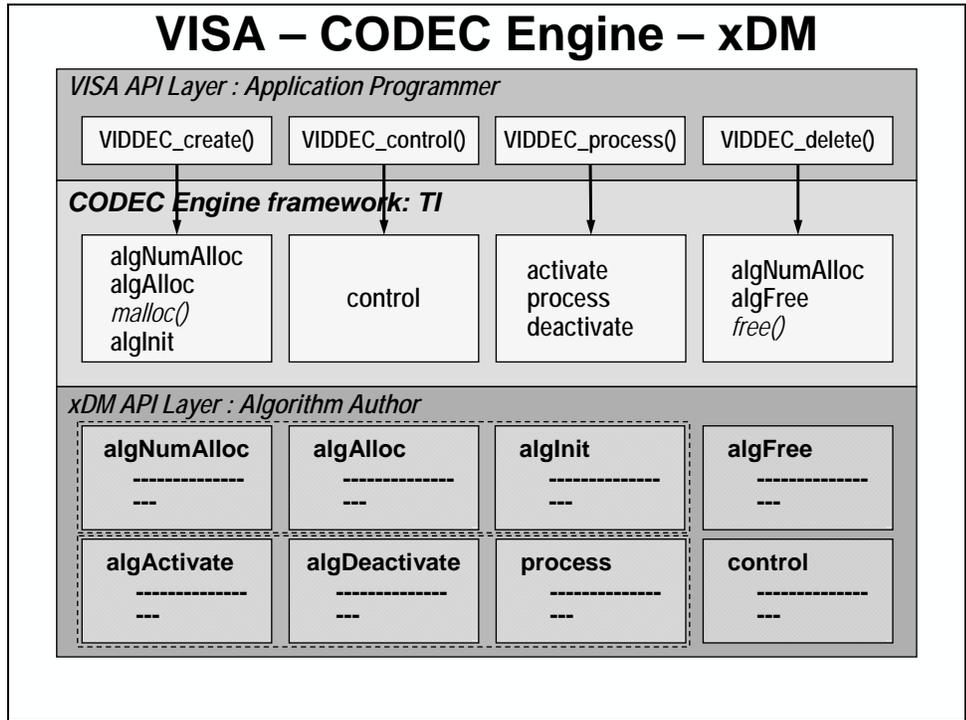
- ◆ Digital Media extensions for xDAIS “xDAIS-DM” or “xDM”
- ◆ Enable **plug + play** ability for multimedia CODECs across implementations / vendors / systems
- ◆ **Uniform** across domains...video, imaging, audio, speech
- ◆ Can be extended for custom / vendor-specific functionality
- ◆ **Low overhead**
- ◆ Insulate application from component-level changes
 - Hardware changes should not impact software
 - PnP ...enable ease of replacement for versions, vendors
- ◆ xDM is still **Framework and O/S Agnostic**
- ◆ Create code faster – Published API enables **early and parallel development**
 - System level development in parallel with component level algo development
 - Reduces integration time for system developers
- ◆ **Published and Stable interface helps:**
 - TI, third parties, and customers
 - Support backward compatibility

Eleven xDM Classes



VISA vs. xDM



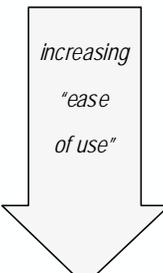


Almost blank ...

Where to Get Your Algorithms

Where to Get Algorithms

1. **Make your own** xDM algorithms
(discussed in Chapter 13)
2. **Obtain free algorithms/CODEC's from TI**
(www.ti.com/dms)
3. **Purchase Individual CODECs**
from TI or TI Third Party
4. **Contract a Complete DSP Executable**
from one of TI's ASP's



*increasing
"ease
of use"*

Algorithm / Codec Inventory

Texas Instruments' Digital Media Software Inventory*						
Codecs	Target Hardware					
	TMD320DM646x	TMD320DM644x	TMD320DM643x	TMD320DM648	TMD320DM3x	OMAP35x
Video & Imaging						
H.264 Video Decoder	●	●	●	●		●
H.264 Video Encoder	●	●	●	●		●
VICP	●	●				
JPEG Imaging Decoder	●	●	●	●	●	●
JPEG Imaging Encoder	●	●	●	●	●	●
MPEG-2 Video Decoder	●	●	●	●		●
MPEG-2 Video Encoder		●				
MPEG-4 Video Decoder		●	●	●	●	●
MPEG-4 Video Encoder		●	●		●	●
VC1 Video Decoder		●	●			

Digital Media Software: www.ti.com/dms



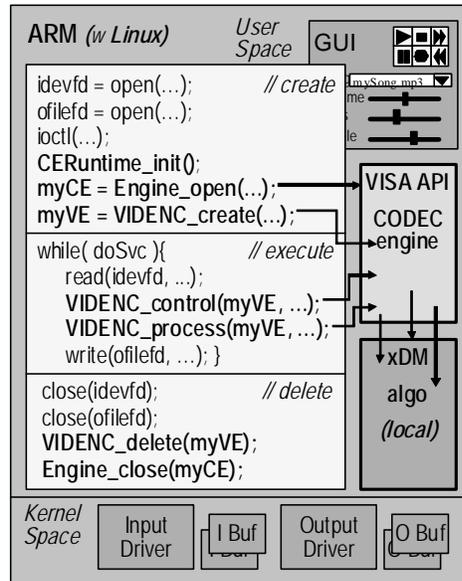
Available eXpressDSP™ Compliant Algorithms

Audio	Digital Motor Control	Speech	Encryption	VB Modem	Video & Imaging
3D Stereo		Adaptive Speech Filter	3-DES	ACG	BioKey
AAC Decoder/Encoder	Position Control	ASR Densifier	AES	BELL 103, 202	Blip Stream
Acoustic Echo Canceller	RMS Signal Measurement	Broadband Noise Cancel	Assembly	V.21, V.22, V.23	Fingerprint
Adaptive Noise Canceller	Speed Control	Caller ID Text to Speech	Diffie-Hellman	V.32, V.34, V.42, V.90	Biometrics
Chorus Effect	Torque Control	Clear Voice Capture	ELGAMAL	More ...	H.261, H.263
MP3 Decoder/Encoder	Vector PWN	Full Duplex/Noise Suppress	HMAC	Telephony	JPEG
MPEG2	More ...	MPEG4 Speech Decoder	MD5	2100Hz Tone Dec	HAAR
MPEG4	Vocoders	Voice Recognition	RSA	Call Progress Analysis	MJPEG
Noise Reduction	G.165, G.168, G.711	More ...	More ...	DTMF	MJPEG, 2, 2
Reverb	G.722, G.723, G.726		Wireless	Echo Canceller	RMS Compression
More ...	G.728, G.729, G.729A		2.28 bps/Hz PTM	Line Echo Canceller	VP4 Decoder
	G.7.29AB, G.723.1		Cyclic Redundancy Check	More ...	More ...
	Voice Activity		Deinterleaver Multiplexer	Protocol Stacks	
	More ...		Viterbi Decoder	HDLX Gen Lev 2	
			More ...	HDLC Receiver	
				HDLC Transmitter	
				TC/IP	
				More ...	

- ◆ More than 1000 TMS320 algorithms provided by TI third parties
- ◆ eXpressDSP Compliance Program available for greater confidence
- ◆ Ensures:
 - Interoperability
 - Portability
 - Maintainability
 - Measurability
- ◆ eXpressDSP™ Algorithm Standard Developer's Kit available to help develop your own compliant IP

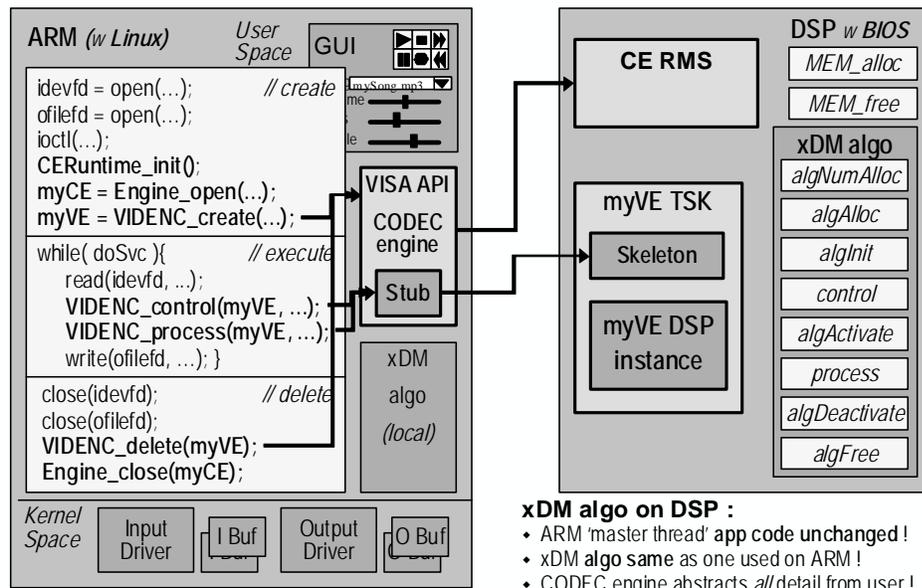
CODEC Engine Details

DaVinci Technology Framework: ARM Only



- ◆ **xDM algo on ARM :**
 - Familiar layout to 1000's of Linux programmers
 - Optimal for low to medium demand algos
- ◆ **Will all algos run successfully on the ARM?**
 - MIPS
 - Power efficiency
 - Separate I/O interruptions from DSP processing
 - Non-determinism of GP OS's
- ◆ **So, in many cases, hard real-time high demand DSP work needs a DSP to implement the processing phase of the system**
- ◆ **How much extra work will be imposed on the application author to locate the xDM algo on DaVinci technology-based DSPs?**

DaVinci Technology Framework: ARM + DSP

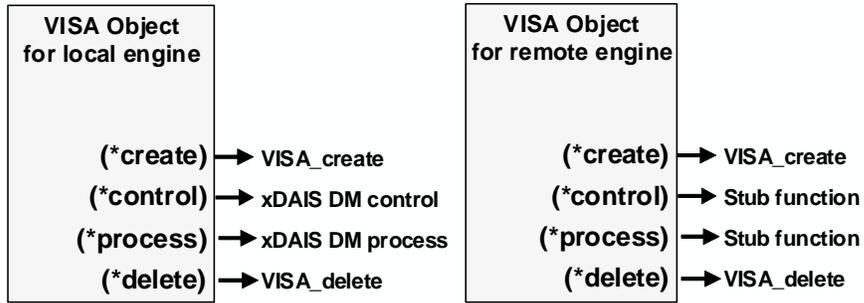


- xDM algo on DSP :**
- ARM 'master thread' app code unchanged !
 - xDM algo same as one used on ARM !
 - CODEC engine abstracts *all* detail from user !

RPC – Remote Procedure Call

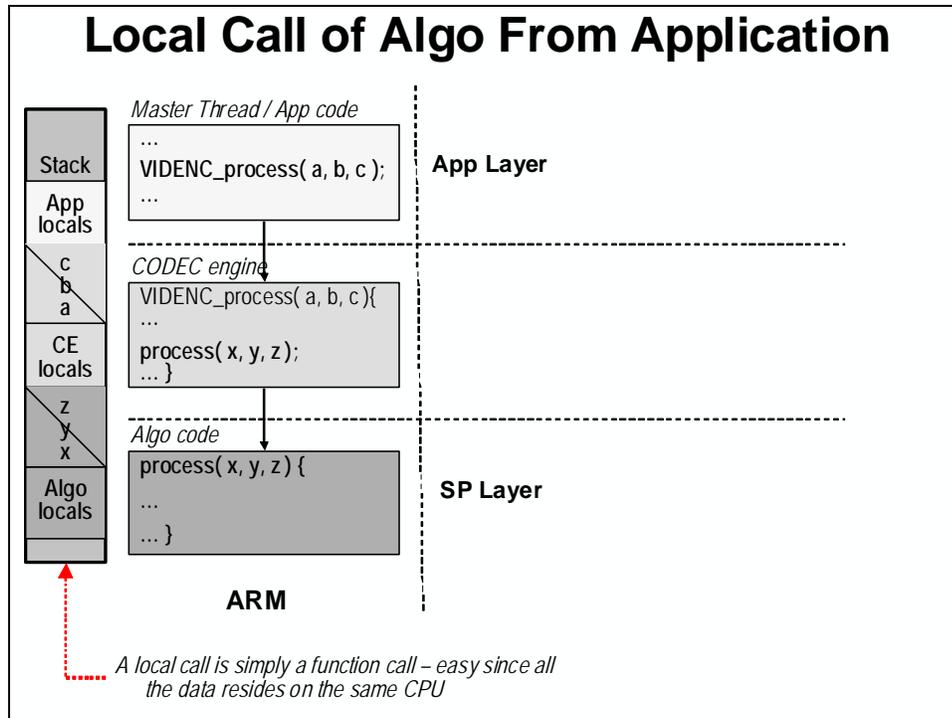
Local and Remote Algo Objects

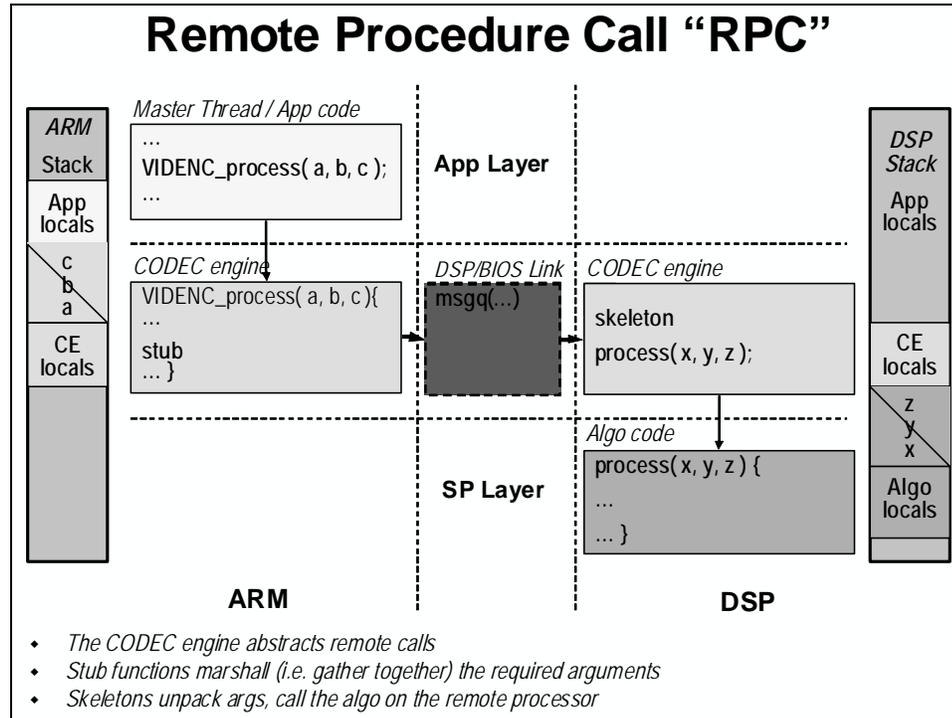
How does a VISA_process call know whether to call a local function or to call a remote function via a stub?



VISA_create and VISA_delete test a bit in the engine object to determine if engine is local or remote and proceed accordingly

Local Call of Algo From Application





DaVinci Technology System SW Component Details

Application Code

- No changes for local vs remote algo
- Serves as 'master thread'
- Controls all other components in system

IO Drivers

- No change to peripherals
- No change to application code
- Drivers accessible to Linux community
- Data does *not* pass through app on way to algo – no extra layer of overhead
- Buffers are in shared memory, equally accessible to DSP

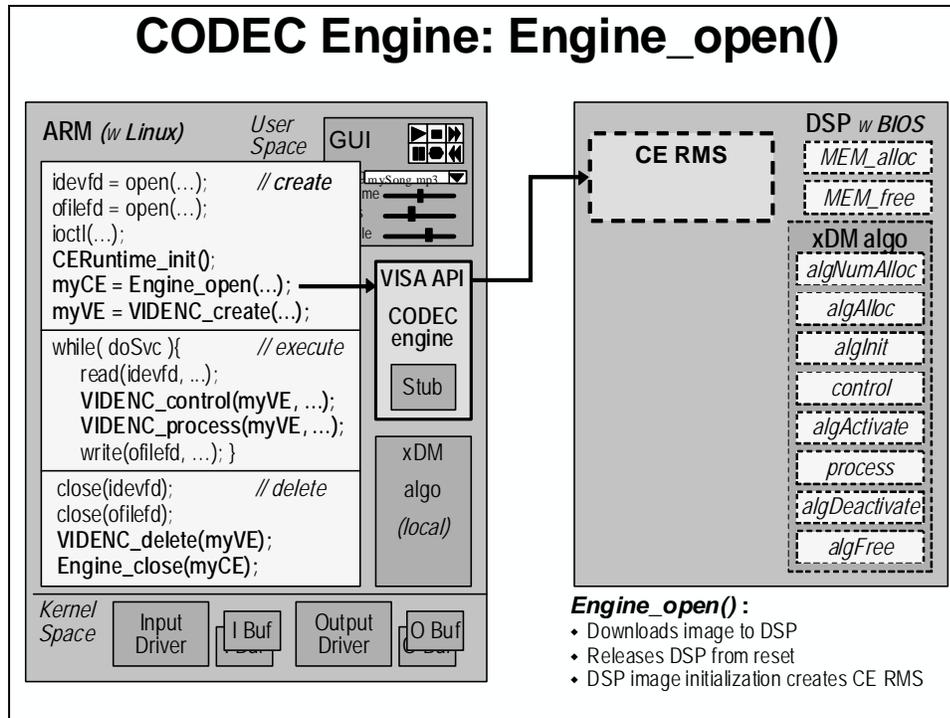
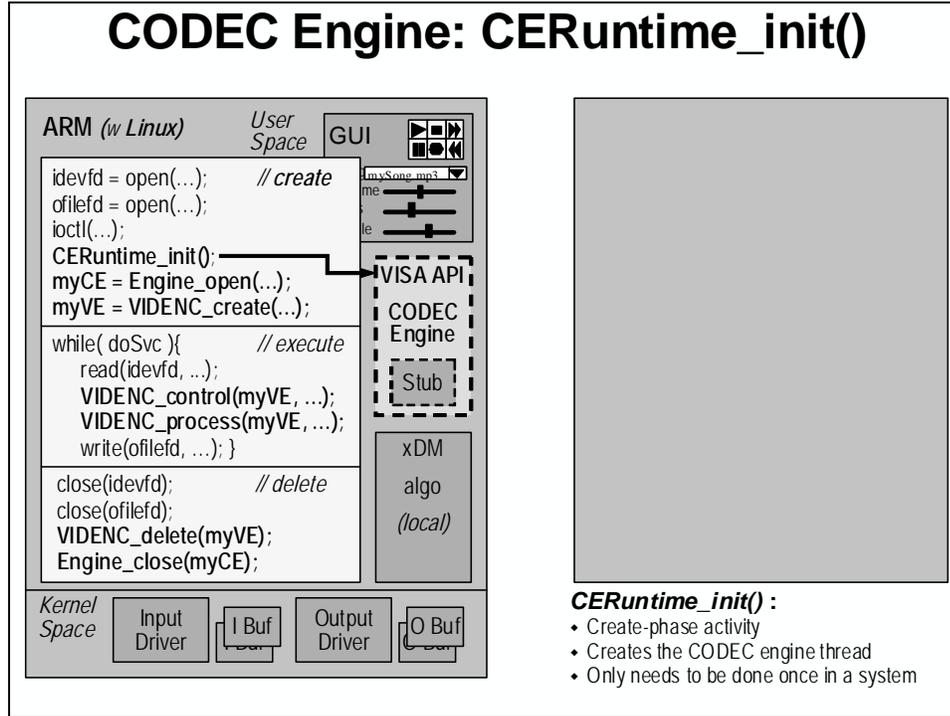
Algorithm / DSP Task

- No change to algo code to run on DSP
- No change to algo code
- DSP Task is a pure 'data transducer':
 - no direct control over peripherals
 - 'slave' to app code control
 - not the 'master' of the application
- Algo inside TSK to provide priority, context
- Algo can use ACPY, DMAN to bring buffer data from shared mem to local RAM

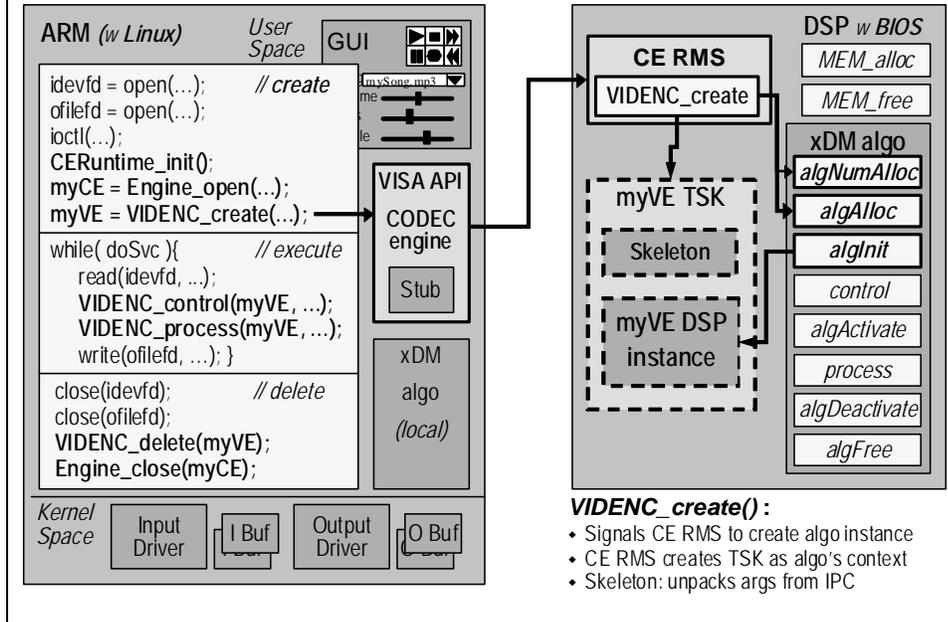
Interprocessor Communication

- CODEC engine abstracts all IPC details
- App/algo unaware of location of algo
- Infrastructure provided by TI for DaVinci technology

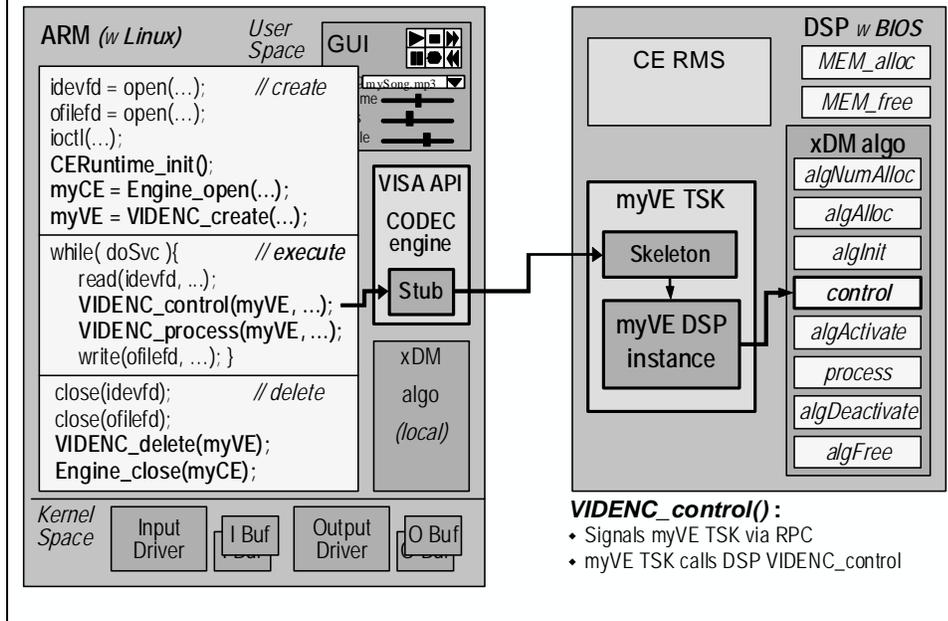
Code Review

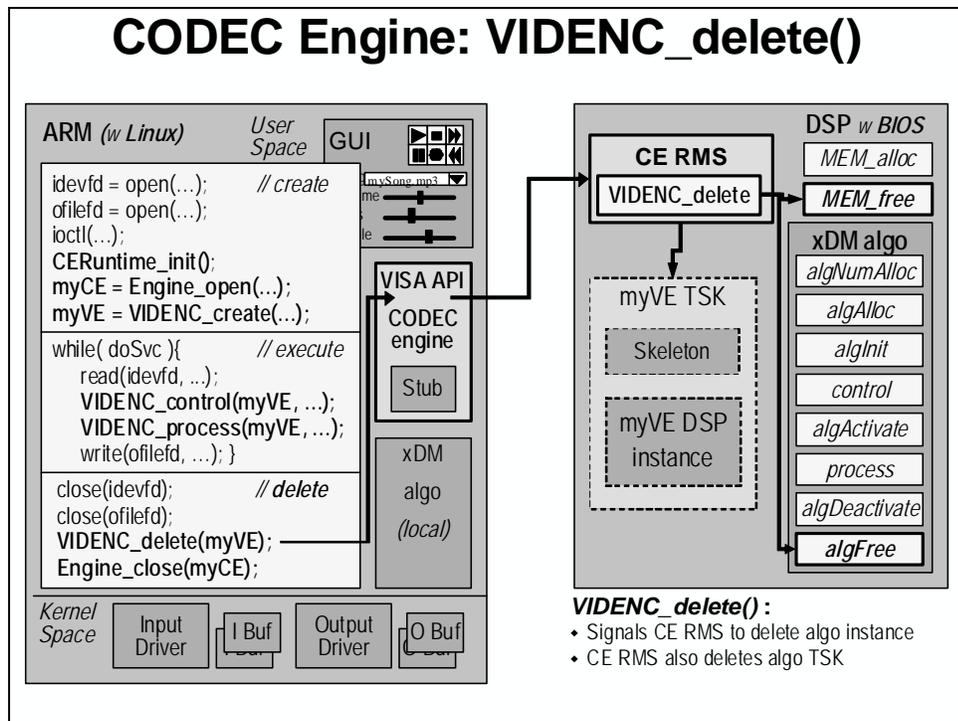
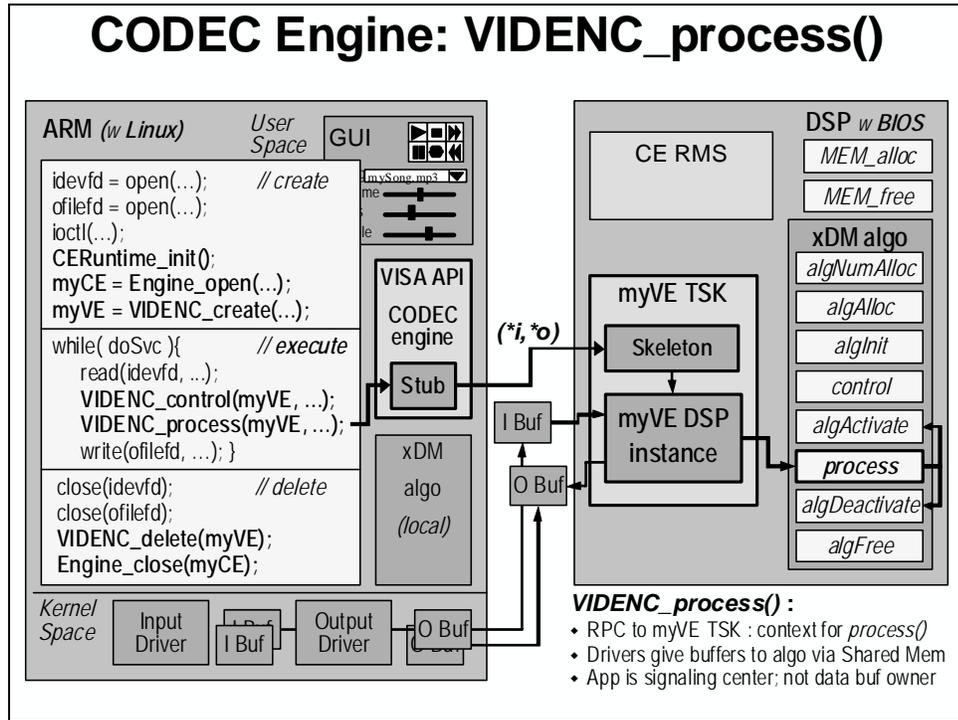


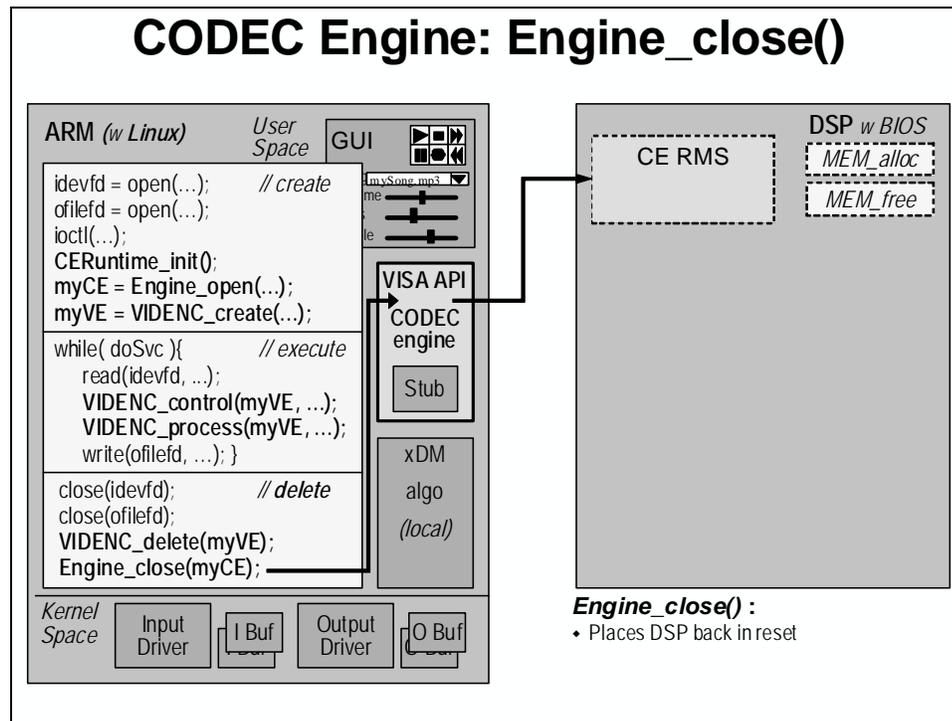
CODEC Engine: VIDENC_create()



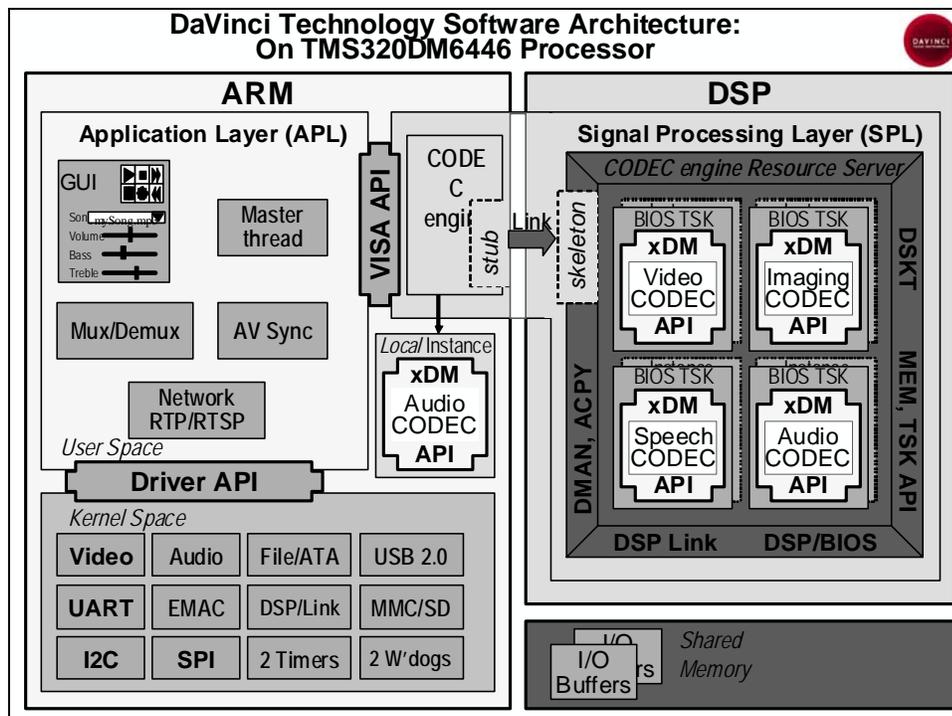
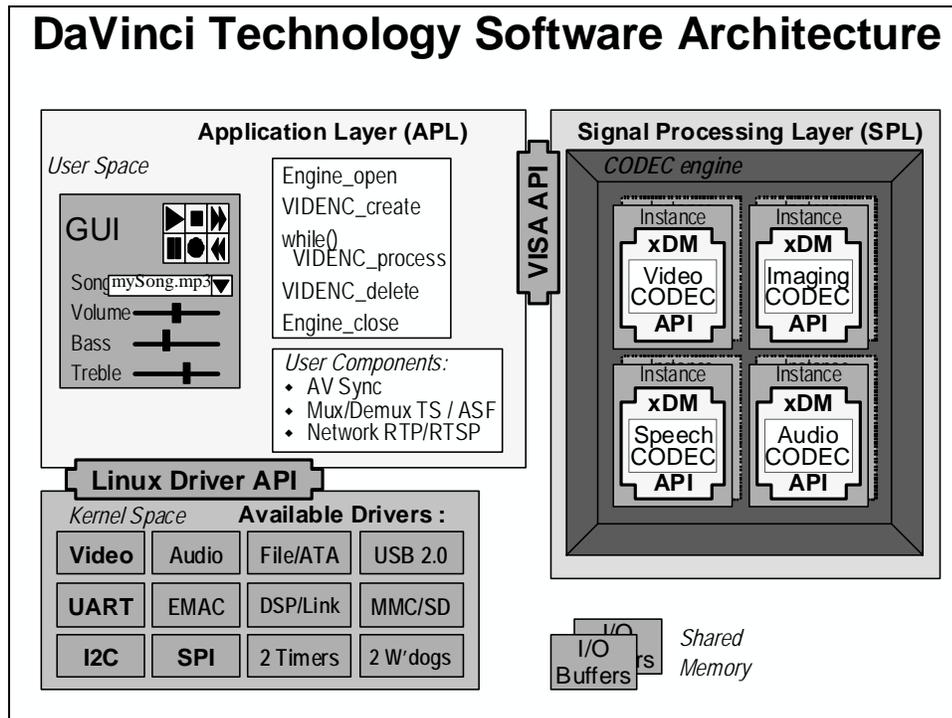
CODEC Engine: VIDENC_control()

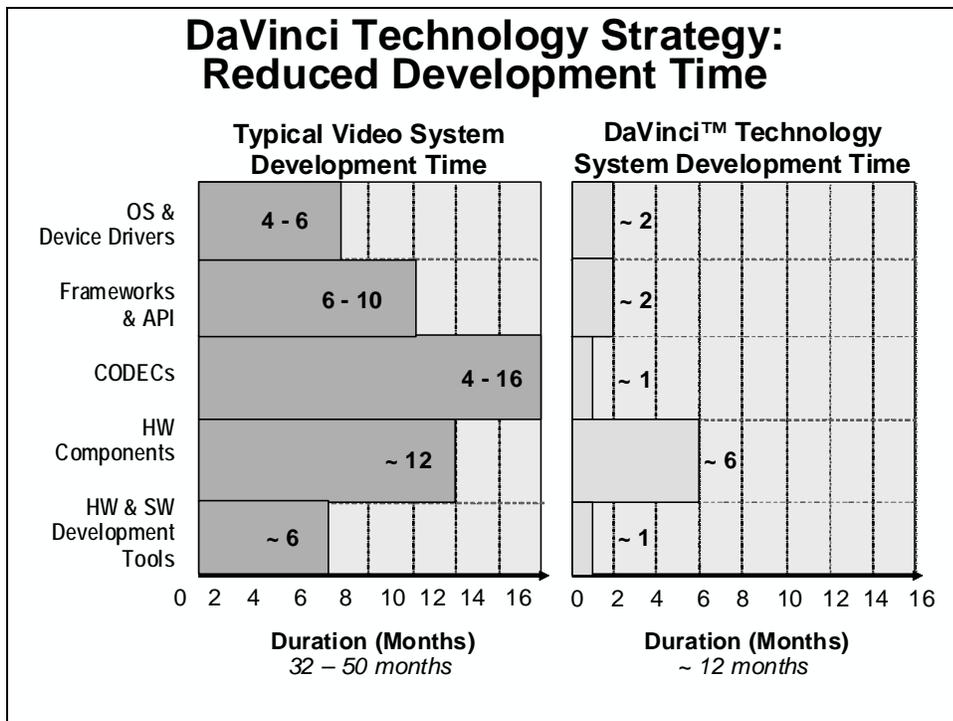
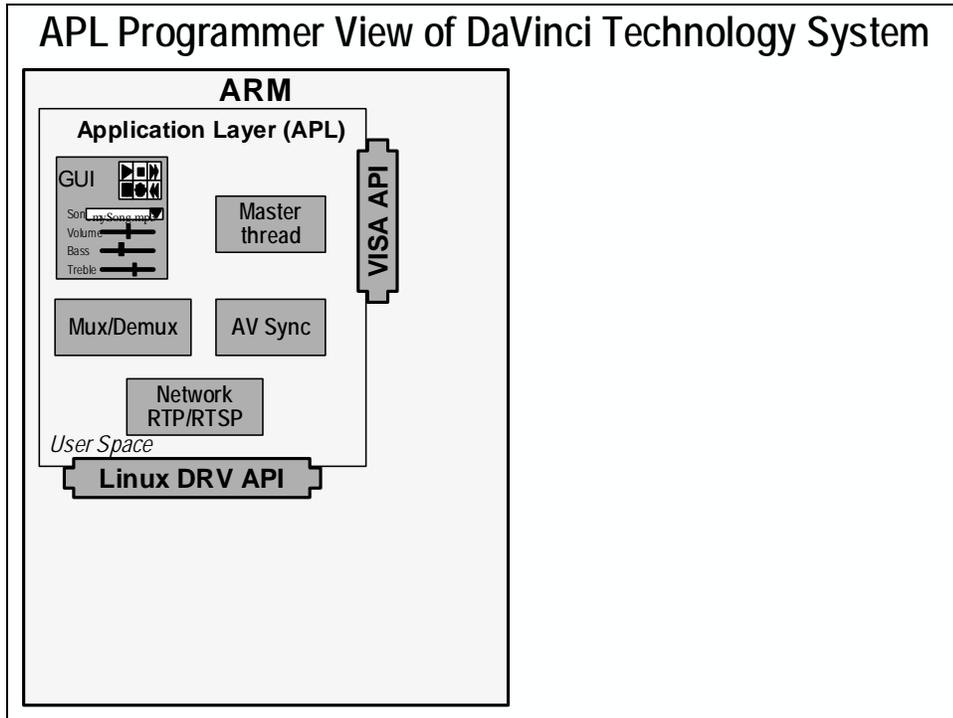






Software Summary





For More Information

For More Information

- ◆ *Codec Engine Application Developer User's Guide* (sprue67d.pdf)
- ◆ *Codec Engine Algorithm Creator User's Guide* (sprued6c.pdf)
- ◆ *Codec Engine Server Integrator's User's Guide* (sprued5b.pdf)
- ◆ *Prog. Details of Codec Engine for DaVinci Tech.* (spr091.pdf)
- ◆ Multimedia Frameworks Products (MFP) Product Folder:
<http://focus.ti.com/docs/toolsw/folders/print/tmdmfp.html>

Getting Started with the Tools

Introduction

This chapter begins with a very brief introduction to Linux. While we hope that most of you already are familiar with some Linux basics, we'll spend a few minutes covering its high points.

In the second part of the chapter, we'll review the VMware program and how we can use it to create a virtual Linux PC inside our Windows environment. If you can dedicate a PC to be used solely for Linux, that's great. For many of us, who don't have that option, VMware easily lets us use one – inside the other.

Finally, we end this chapter with a brief introduction to Das U-Boot, the Linux bootloader used to bootstrap TI's Linux-based devices. Its ability to be configured through RS-232 makes it a handy choice. You'll further explore some of the U-Boot config options in the lab accompanying this chapter.

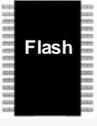
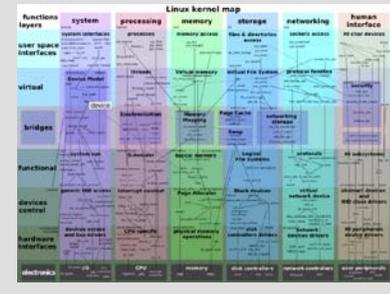
Chapter Topics

Getting Started with the Tools	3-1
<i>Introduction to Linux</i>	<i>3-2</i>
Linux Fundamentals	3-2
Basic Linux Commands – <i>You should already know</i>	3-5
<i>VMware - Linux box in a Windows PC.....</i>	<i>3-6</i>
<i>Booting the Device.....</i>	<i>3-13</i>
Getting to U-boot.....	3-13
Where to find boot files.....	3-14
Das U-Boot.....	3-16
Configuring U-Boot.....	3-17
Boot Variations.....	3-19
Configure U-Boot with Tera Term Macro's.....	3-22
<i>For More Information.....</i>	<i>3-22</i>

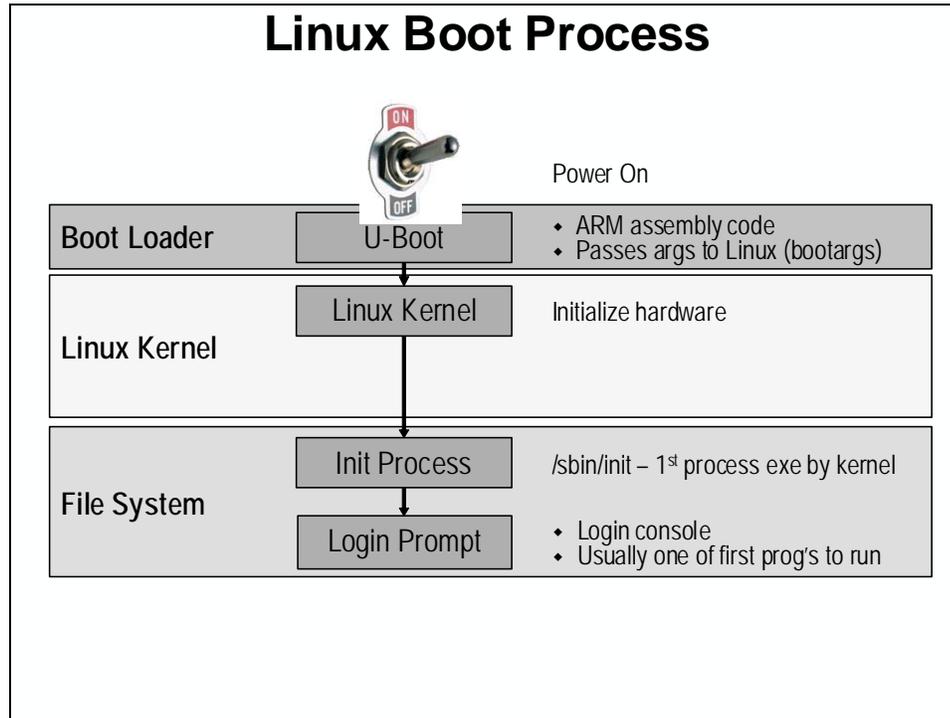
Introduction to Linux

Linux Fundamentals

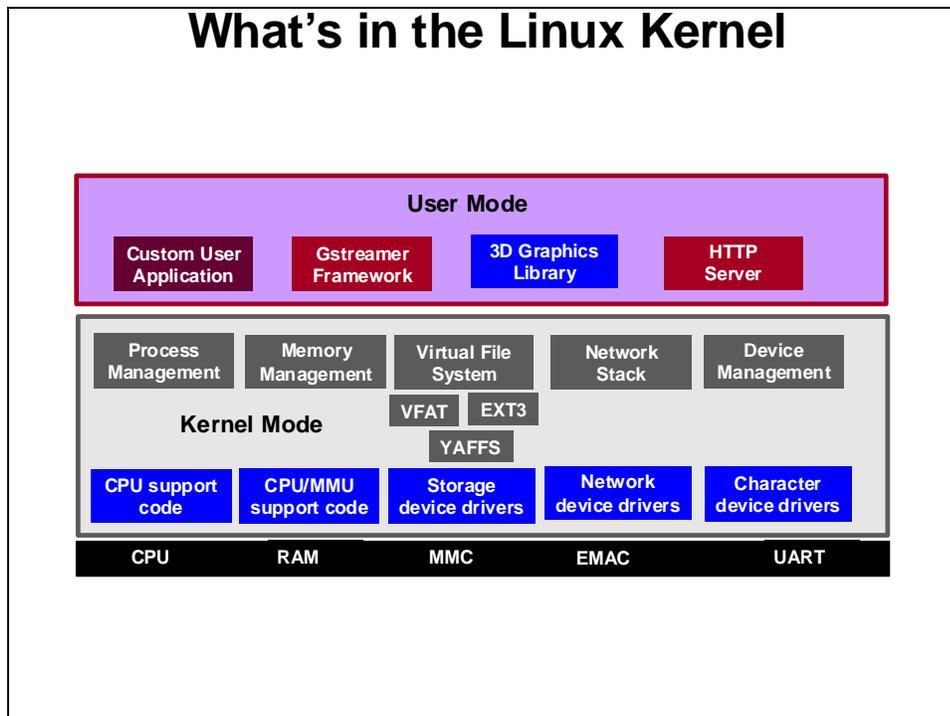
Linux in Three Parts

- 1 Bootloader**
 - Provides rudimentary h/w init
 - Calls Linux kernel and passes boot arguments
- 2 Kernel**
 - Initializes the system (and device)
 - Manages system resources
 - Provides services for user programs
- 3 Filesystem**
 - Single filesystem (/ root)
 - Stores all system files
 - After init, kernel looks to filesystem for "what's next"
 - bootarg tells linux where to find root filesystem

Steps in Booting Linux

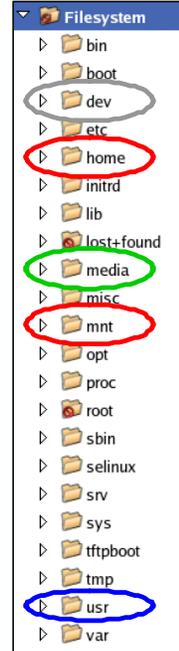


Linux Kernel



Linux Root Filesystem

Red Hat / Ubuntu : Root File System



Some folders common to Linux:

/dev – Common location to list all device drivers

/home - Storage for user's files

- ◆ Each user gets their own folder (e.g. /home/user)
- ◆ Similar to "My Documents" in Windows
- ◆ DVSDK GSG directory for TI tools, examples, working directory
- ◆ "root" user is different, that user's folder is at /root

/media – Usually find CDROM drive(s) mounted here

/mnt – Common location to mount other file systems

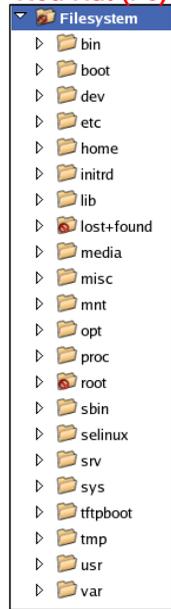
- ◆ Linux only allows one filesystem
- ◆ Add other disks (physical, network, etc) by mounting them to an empty directory in the root filesystem
- ◆ Windows adds new filesystems (C:, D:, etc.) rather than using a single one

/usr – Storage for user binaries

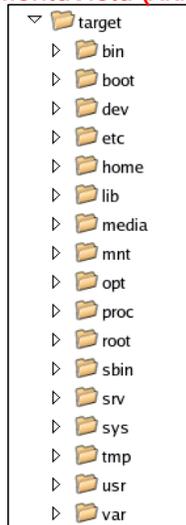
- ◆ X86 Compiler for Ubuntu programs (gcc) is stored in here

Filesystems: Red Hat vs. MontaVista

Red Hat (PC)



MontaVista (ARM)



- ◆ **Tools/Host filesystem:** location our dev'l tools
- ◆ **Target filesystem:** filesystem to run on TI processors
- ◆ Notice the similarities between the two different Linux filesystems
- ◆ When releasing to production, it's common to further reduce the target filesystem to eliminate cost

Basic Linux Commands – You should already know

Linux Command Summary

Some commands used in this workshop:

File Management

- ls and ls -la
- cd
- cp
- ln and ln -s
- mv
- rm
- pwd
- tar (create, extract .tar and tar.gz files)
- chmod
- chown
- mkdir
- mount, umount (in general, what is “mounting” and how do you do it?)
- alias
- touch

Network

- /sbin/ifconfig, ifup, ifdown
- ping
- nfs (What is it? How to share a folder via NFS. Mounting via NFS.)

VMware Shared Folders

- /mnt/hgfs/<shared name>

Program Control

- <ctrl>-c
- ps, top
- kill
- renice

Kernel

- insmod, rmmod

Linux Users

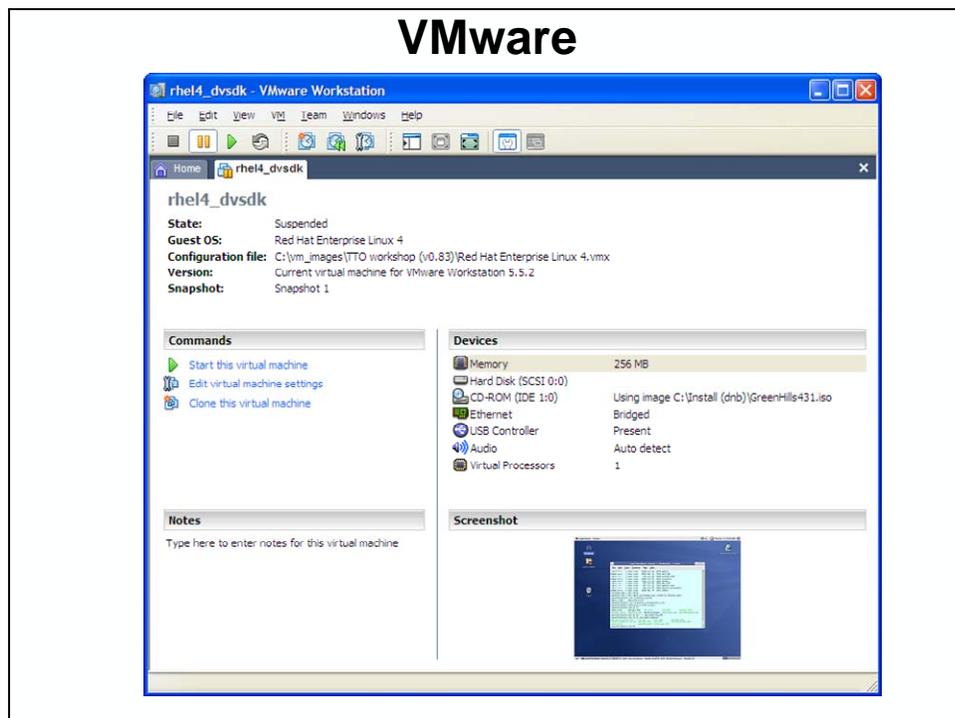
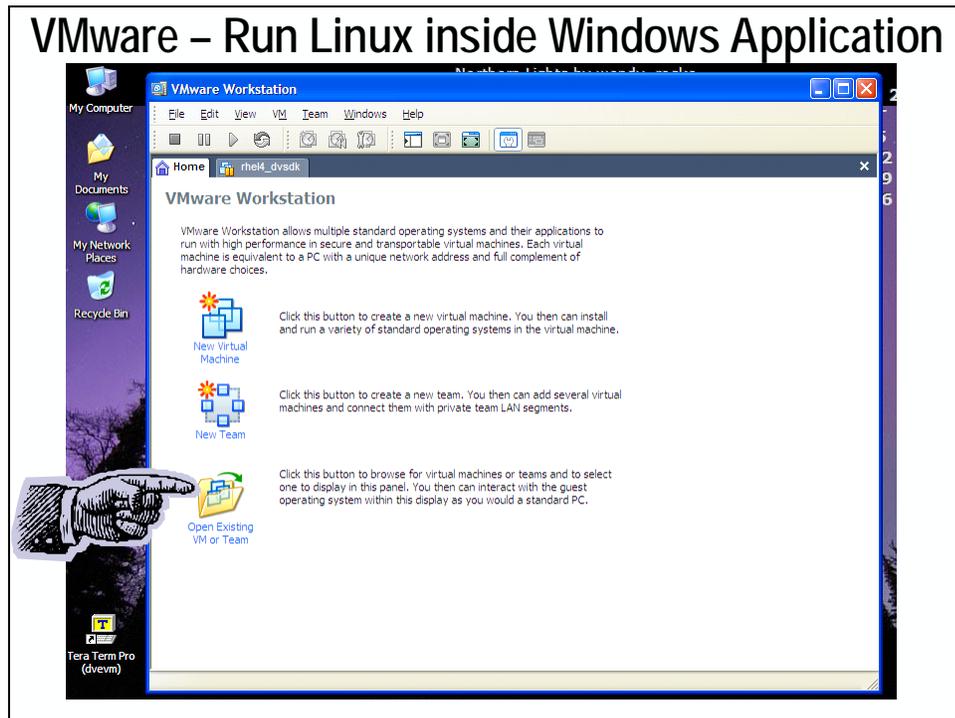
- root
- user
- su (... exit)

BASH

- What is BASH scripting
- What are environment variables
- How to set the PATH environment variable
- What is .bashrc? (like DOS autoexec.bat)
- man pages
- change command line prompt

In this course, we will try to provide the necessary Linux commands needed to run the tools and such. Even so, if there are commands listed here that you don't recognize, we recommend that you study up on them. To this end, the “*Linux Pocket Guide*” reference from the “**0. Welcome Chapter**” might be handy.

VMware - Linux box in a Windows PC



VMware – Virtual Machine

Why VMware?

- ◆ Allows simultaneous use of Linux and Windows with one PC
- ◆ Virtual Linux PC within a Windows application
- ◆ VMware provides free “player” version of their software
- ◆ Virtual PC settings and hard disc image are stored inside any Windows folder
- ◆ Easily distribute virtual Linux PC with all DaVinci tools pre-installed
- ◆ By keeping known “good” copy of VMware image, you can easily reset Linux PC

Virtual Image Used In This Workshop

Workshop VMware Image

Workshop VMware Images

Notes:

- ◆ Screensaver & Firewall off
- ◆ NFS, TFTP, GIMP installed
- ◆ VMware toolbox installed

DM6446 Labs:

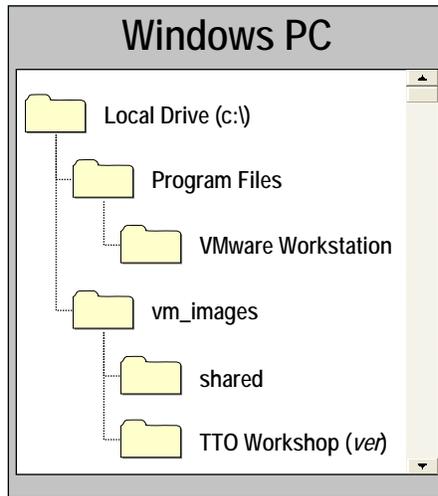
- ◆ Ubuntu 10.04
- ◆ id = user, psw = useruser
- ◆ Software installation following DVSDK *Getting Started Guide*:
 - MV Linux 5.0 (Linux tools)
 - DaVinci LSP (kernel)
 - Target Filesystem

OMAP3530/AM3517 Labs:

- ◆ Ubuntu 10.04
- ◆ id = user, psw = none
- ◆ DVSDK/SDK Tools:
 - Community Linux (Arago)
 - CodeSourcery Toolset

VMware : Free Player vs Full Workstation

VMware – Free Player vs. Full Workstation



Full Workstation

- ◆ Can build VMware PC image from scratch
- ◆ “Snapshot” feature allows you to save & restore previous machine states (handy!)
- ◆ “Shared Folders” feature makes it easy to share files between Linux and Windows
- ◆ Not free, but small discount with current users referral code
- ◆ Workstation users get both full/free

Free Player

- ◆ Free
- ◆ Someone else has to create original VMware image (and do h/w mods)
- ◆ No snapshot feature

VMware : Snapshots

VMware : Snapshot

Actions

- ◆ Create new
- ◆ Delete
- ◆ Go To

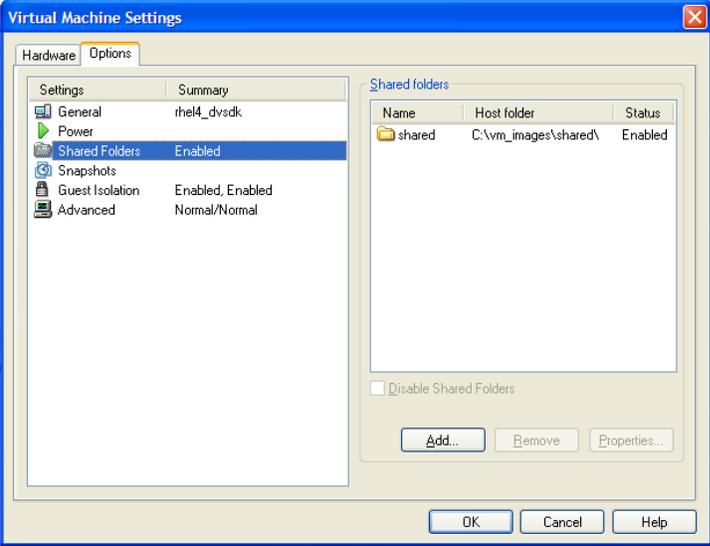
Deleting snaps

- ◆ Deleting last one sets you back to previous state
- ◆ Delete middle one combines snapshots

Performance

- ◆ Too many snapshots can diminish perf.
- ◆ Too few and you could get stuck :-)

VMware : Shared Folders



The screenshot shows the 'Virtual Machine Settings' dialog box with the 'Options' tab selected. The 'Shared Folders' section is highlighted in the left sidebar and is set to 'Enabled'. The main area displays a table of shared folders:

Name	Host folder	Status
shared	C:\vm_images\shared\	Enabled

Below the table, there is a checkbox for 'Disable Shared Folders' (unchecked) and buttons for 'Add...', 'Remove', and 'Properties...'. At the bottom of the dialog are 'OK', 'Cancel', and 'Help' buttons.

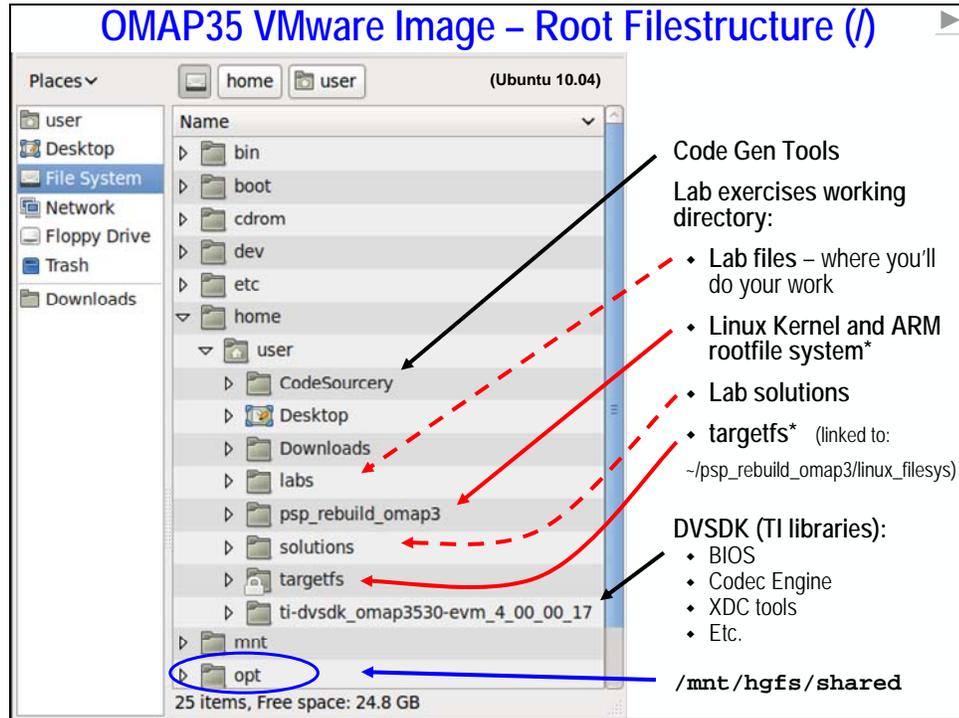
Sharing folders

- VMware shared folders
- NFS
- Samba

VMware Shared Folders

- Easiest method
- Access from: /mnt/hgfs/shared

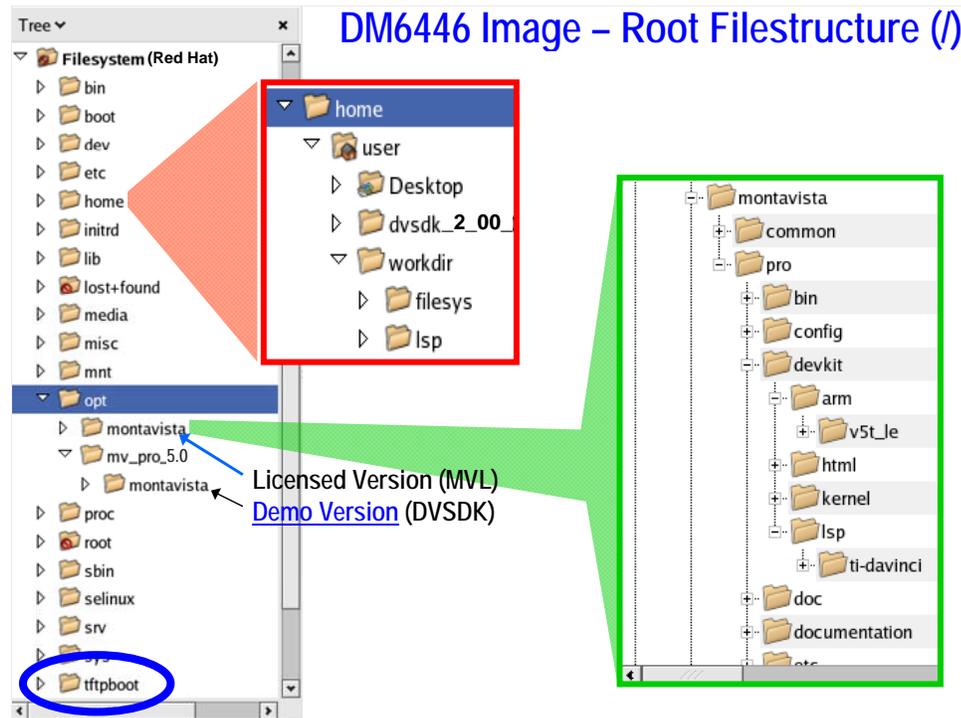
VMware Filesystem – for OMAP35/AM35 (DVSDK 4.xx)



Some important points in the preceding filesystem:

1. By default, all TI tools are installed into the user's home folder – in our case, that would be the `/home/user` directory.
2. The two installers that come with DVSDK 4.x create the folders:
 - `CodeSourcery` – where the ARM code generation tools are located
 - `ti-dvSDK_<platform>_<dvSDK_version>` - where all the TI libraries and software are located. This directory also includes the PSP Linux directory – which contains the Linux Kernel, boot files (MLO, U-Boot), as well as the target (i.e. ARM) filesystem
3. The `psp_rebuild_omap3` folder is a replication of the Linux PSP folder. We rebuilt the Linux kernel and modules and wanted to keep our working copy in a separate folder.
4. To make things easier, we ask you to create a (Linux) symbolic link called `targetfs` to the `linux_filesys` directory (which contains the ARM root filesystem). The link will point into the `psp_rebuild_omap3` folder.
5. The `labs` and `solutions` folders will be installed by you in Lab 3. The `labs` directory contains the working files for each of the exercises in the course. As its title implies, the `solutions` directory contains the solution files.

VMware Filesystem for DM6446 Tools (DVSDK 2.xx)



Some important points in the preceding filesystem:

6. The directory structure on the left shows the root filesystem of the VMware image filesystem.
7. The first callout shows the `/home/user` folder. It contains a few interesting things:
 - `/home/user/dvevm_2_xx_xx` contains all the TI DVEVM and DVSDK content (cgt, bios, xdc, codec engine, examples, etc.). We placed this content here based on the DVEVM and DVSDK Getting Started Guides (GSG) instructions.
 - `/home/workdir` – a working directory, into it we made a copy of the Linux Support Package (LSP) and the default DM644x target root filesystem created by MV for the DVEVM.
 - `/home/user/workshop` and `/home/user/solutions` are the workshop lab folders. *These directories are not shown on this screen capture.*

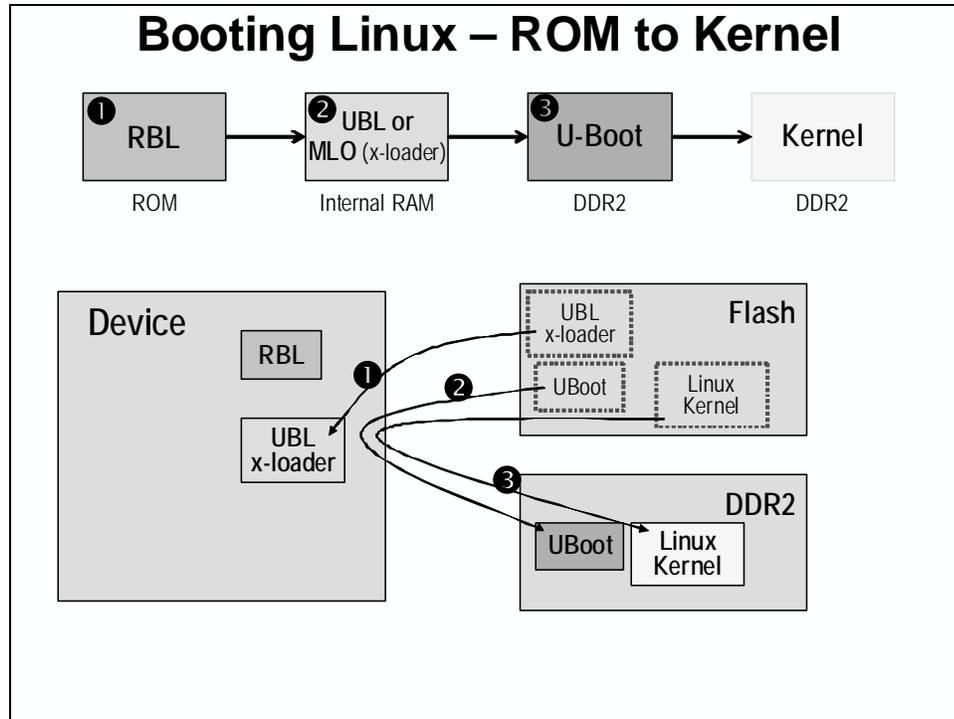
8. The lower callout highlights the MontaVista tools in the /opt folder. In it:
 - /opt/montavista – contains the full version of MV Linux. The full version is sold by MontaVista.
 - /opt/mv_pro_5.0/montavista – contains the **demo** version of MV Linux which ships with the DVSDK. It was placed in a different folder (1 level down) to keep it from conflicting with the full version of MV Linux. *We installed this as part of the workshop VMware image.*

If you install both the MontaVista licensed software, and all the DVSDK software, you may end up with both of these in your filesystem (as shown above). *For our workshop VMware image, though, we only installed the demo version at /opt/mv_pro_5.0/montavista.*

9. Finally, in the next topic we discuss booting the DVEVM. For Linux to boot, the bootloader (Das U-Boot) needs to find the Linux kernel; this is usually found either in the flash memory on the board, or the bootloader can download it from a network. When the latter is chosen, the default is to TFTP the kernel (“uImage” file) from the /tftpboot folder.

Booting the Device

Getting to U-boot



Bootloader Components

Boot stage	Operations	User Config'd	DaVinci	OMAP3x
First-level	This is ROM'd code for detecting desired boot type (NAND, UART, ...) and loading executable code of second-level bootloader from selected peripheral/interface	No	RBL	RBL
Second-level	The primary function of this boot loader is to initialize external memory and system clocks so that a larger, more advanced boot loader (in this case U-boot) can be loaded.	Board Designer	UBL	XLDR
Linux boot	"Das U-boot" is the standard open-source Linux boot loader for ARM. It supports networking for TFTP/NFS booting. It is used to locate, load and execute the Linux kernel in ulmage format and is also responsible for passing arguments to the kernel	Yes	U-boot	U-Boot

Customizing UBL / XLDR

1. Configure system clocks
2. Setup memory interfaces

* In this workshop we will only configure the 3rd level bootloader (Das U-boot).

Where to find boot files...

To Boot Linux, You Need...

1. Bootloader (U-Boot)	◆ At reset, U-Boot bootloader is executed
2. Linux Kernel	◆ U-Boot loads O/S kernel into DDR2 memory; then,
3. Filesystem	◆ Connects to the root filesystem If you don't know what this is, think of it as the 'c:\' drive of in Windows PC

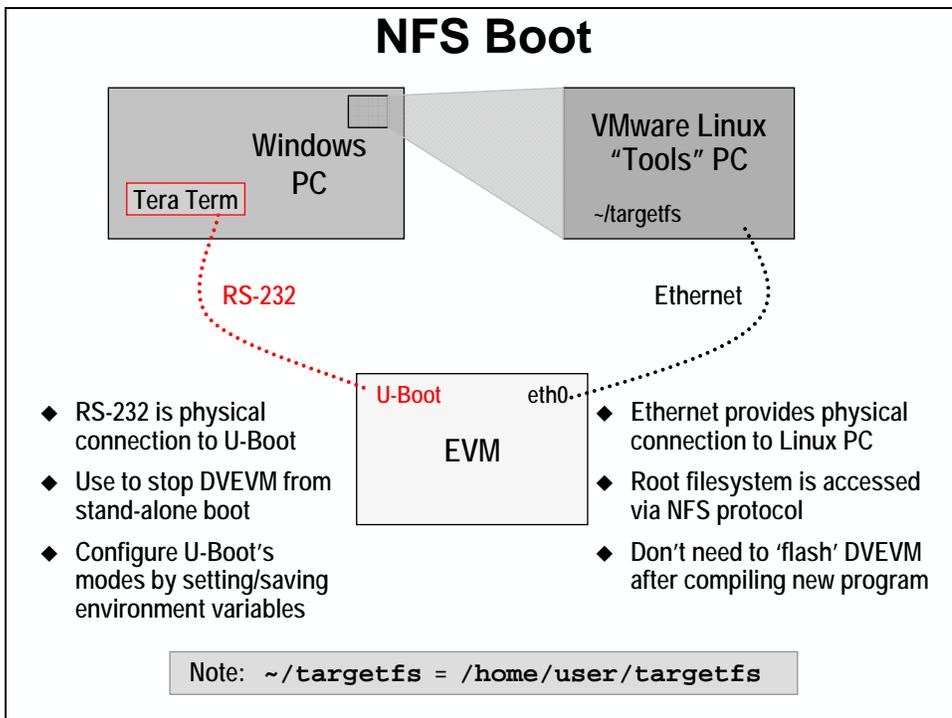
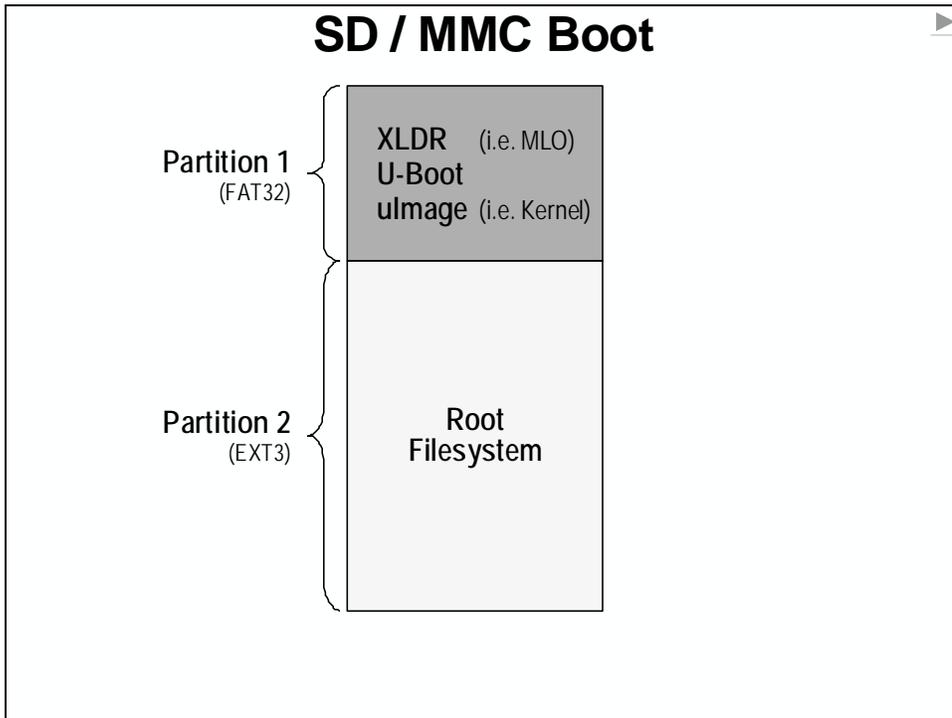
Where Do You Find ...

Where located:	DM6446 EVM Default	AM3517 1-day Wkshp	Good for Development
1a. UBL or Xloader/MLO	Flash	MMC	Flash or MMC
1b. Bootloader (U-Boot)	Flash	MMC	Flash or MMC
2. Linux Kernel	Flash	MMC	TFTP (from Ubuntu)
3. Filesystem	Hard Drive	MMC	NFS (from Ubuntu)

"HDD boot"
"MMC boot"
"NFS boot"

- ◆ By default, the DM6446 DVEVM ships in "HDD boot" mode; this allows the demo applications to run "out-of-the-box"
- ◆ OMAP3530 & AM3517 ship with boot code in NAND. An MMC card demo also ships with the EVM's. Also, the SDK provides an MMC image
- ◆ "NFS boot" (network boot) is good for application development

For MMC boot, the boot file(s) are contained on one (or two) MMC partitions. Here's an example:



Note, for the DM6446 lab exercises, we installed the tools according to the *DM6446 DVEVM Getting Started Guide*, which located the root filesystem at: `/home/user/workdir/filesys`

Das U-Boot

Das U-Boot

- ◆ The Linux PSP SDK board is delivered with the open-source boot loader: Das U-Boot (U-Boot)
- ◆ At runtime, U-Boot is usually loaded in on-board Flash or an SD/MMC card
- ◆ In general, U-Boot performs the functions:
 1. Initializes the DaVinci EVM hardware
 2. Provides boot parameters to the Linux kernel
 3. Starts the Linux kernel

Additional U-Boot Features

In addition, it provides some convenient features that help during development:

- ◆ Ping IP addresses
- ◆ Reads and writes arbitrary memory locations
- ◆ Uploads new binary images to memory via serial, or Ethernet
- ◆ Copies binary images from one location in memory to another

Configuring U-Boot

Configuring U-Boot and Starting Linux (5 Steps)

1. Connect an RS232 serial cable and start a Tera Term
2. Power on the DVEVM and press any key in TeraTerm to abort the boot sequence
3. Set U-Boot variables to select how Linux will boot (save changes to flash to retain settings after power cycle)
4. Boot Linux using either:
 - ◆ the U-Boot "boot" command
 - ◆ power-cycle the DVEVM
5. After Linux boots, log in to the DVEVM target as "root"
 - Note, login with: "user" for the Tools Linux PC
"root" for the DVEVM target
 - You can use any RS-232 comm application (Linux or Win), we use Tera Term for its macro capability

U-Boot Variables & Commands

Configuring U-Boot

Common Uboot Commands:

- ◆ printenv - prints one or more uboot variables
- ◆ setenv - sets a uboot variable
- ◆ saveenv - save uboot variable(s)
- ◆ run - evaluate a uboot variable expression
- ◆ ping - (debug) use to see if Uboot can access NFS server

Common Uboot Variables:

- ◆ You can create whatever variables you want, though some are defined either by Linux or common style
 - bootcmd - where Linux kernel should boot from
 - bootargs - string passed when booting Linux kernel
e.g. tells Linux where to find the root filesystem
 - serverip - IP address of root file system for NFS boot
 - nfspath - Location on serverip for root filesystem

U-Boot Macros

At times, it can be handy to create “macros” in U-boot. These are really nothing more than variables that can be expanded when called with the U-boot “run” command.

U-Boot Macros

- ◆ Variables can reference each other

for example, to keep original bootargs settings, try:

```
setenv bootargs_original $(bootargs)
```

- ◆ “Run” command – Force uboot to evaluate expressions using run

For example, evaluate this expression:

```
setenv set_my_server `setenv $(serverip):$(nfspath)`
```

Using the run command:

```
setenv serverip 192.168.1.40
setenv nfspath /home/user/workdir/filesys
run set_my_server
```

Common U-Boot Commands and Variables

U-Boot Commands/Variables

U-Boot Commands: setenv, saveenv, printenv, run, ping, dhcp, tftp

Example: `setenv ipaddr 192.168.1.41`

Common variables used to configure the behaviour of U-Boot:

autoload	If set to “no”, only lookup performed, no TFTP boot
autostart	If “yes”, a loaded image is started automatically
baudrate	Baudrate of the terminal connection, defaults to 115200
bootargs	Passed to the Linux kernel as boot “command line” arguments
bootcmd	Command string that is executed when the initial countdown is not interrupted
bootdelay	After reset, U-Boot waits ‘bootdelay’ seconds before executing bootcmd; abort with any keypress
bootfile	Name of the default image to load with TFTP
cpuclock	Available for processors with adjustable clock speeds
ethaddr	Ethernet MAC address for first/only ethernet interface (eth0 in Linux); additional ethernet i/f’s use eth1addr, eth2addr, ...
initrd_high	Used to restrict positioning of initrd ramdisk images
ipaddr	IP address; needed for tftp command
loadaddr	Default load address for commands like tftp or loads
loads_echo	If 1, all characters recv’d during a serial download are echoed back
pram	Size (kB) to reserve for “Protected RAM” if the pRAM feature is enabled
serverip	TFTP server IP address; needed for tftp command; (also use for nfs root mnt’s)
serial#	Hardware identification information, usually during manufacturing of the board
silent	If option is enabled for your board, setting this will suppress all console msgs
verify	

Boot Variations

Here are a variety of ways to boot Linux, along with some of the U-Boot settings which enable them.

Boot Variations (kernel)

Mode	IP	Linux Kernel	Root Filesystem
1.	dhcp	Flash	HDD
2.	dhcp	Flash	NFS
3.	dhcp	TFTP	HDD
4.	dhcp	TFTP	NFS
6.	dhcp	MMC	MMC

U-Boot's `bootcmd` variable specifies the root filesystem

Flash	<code>setenv bootcmd bootm 0x2050000</code>
MMC	<code>setenv bootcmd "mmc init; fatload mmc 0 \${loadaddr} uImage; run mmcargs; bootm \${loadaddr}"</code>
TFTP	<code>setenv bootcmd 'dhcp;bootm'</code>

Boot Variations (filesystem)

Mode	IP	Linux Kernel	Root Filesystem
1.	dhcp	Flash	HDD
2.	dhcp	Flash	NFS
3.	dhcp	TFTP	HDD
4.	dhcp	TFTP	NFS
5.	dhcp	MMC	MMC

U-Boot's `bootargs` variable specifies the root filesystem

HDD	<code>setenv bootargs console=ttyS0,115200n8 noinitrd rw ip=dhcp root=/dev/hda1, nolock mem=120M</code>
MMC	<code>setenv bootargs console=ttyS0,115200n8 noinitrd root=/dev/mmcblk0p2 rootfstype=ext3 rootwait nolock mem=120M</code>
NFS	<code>setenv bootargs console=ttyS0,115200n8 noinitrd rw ip=dhcp root=/dev/nfs nfsroot=\$(serverip):\$(nfspath),nolock mem=120M</code>

Configuring U-Boot Kernel from **Flash**, Filesystem from **HDD**

```
[rs232]# baudrate 115200
[rs232]# setenv stdin serial
[rs232]# setenv stdout serial
[rs232]# setenv stderr serial
[rs232]# setenv bootdelay 3
[rs232]# setenv bootfile uImage
[rs232]# setenv serverip 192.168.2.101
[rs232]# setenv nfspath /home/user/workdir/filesys
[rs232]# setenv bootcmd bootm 0x2050000 ← Kernel
[rs232]# setenv bootargs console=ttyS0,115200n8 ← from
noinitrd rw ip=dhcp root=/dev/hda1 HDD
mem=120M
[rs232]# saveenv
```

Configuring U-Boot Kernel via **TFTP**, Filesystem from **NFS (network)**

```
[rs232]# baudrate 115200
[rs232]# setenv stdin serial
[rs232]# setenv stdout serial
[rs232]# setenv stderr serial
[rs232]# setenv bootdelay 3
[rs232]# setenv bootfile uImage
[rs232]# setenv serverip 192.168.2.101
[rs232]# setenv nfspath /home/user/workdir/filesys
[rs232]# setenv bootcmd 'dhcp;bootm'
[rs232]# setenv bootargs console=ttyS0,115200n8
noinitrd rw ip=dhcp root=/dev/nfs
nfsroot=$(serverip):$(nfspath),nolock
mem=120M
[rs232]# saveenv
```

Static vs. Dynamic Boot

Booting with Static IP Addresses			
Mode	IP	Linux Kernel	Root Filesystem
1.	dhcp	Flash	HDD
2.	dhcp	Flash	NFS
3.	dhcp	TFTP	HDD
4.	dhcp	TFTP	NFS
5.	static	Flash	HDD
6.	static	Flash	NFS
7.	static	TFTP	HDD
8.	static	TFTP	NFS

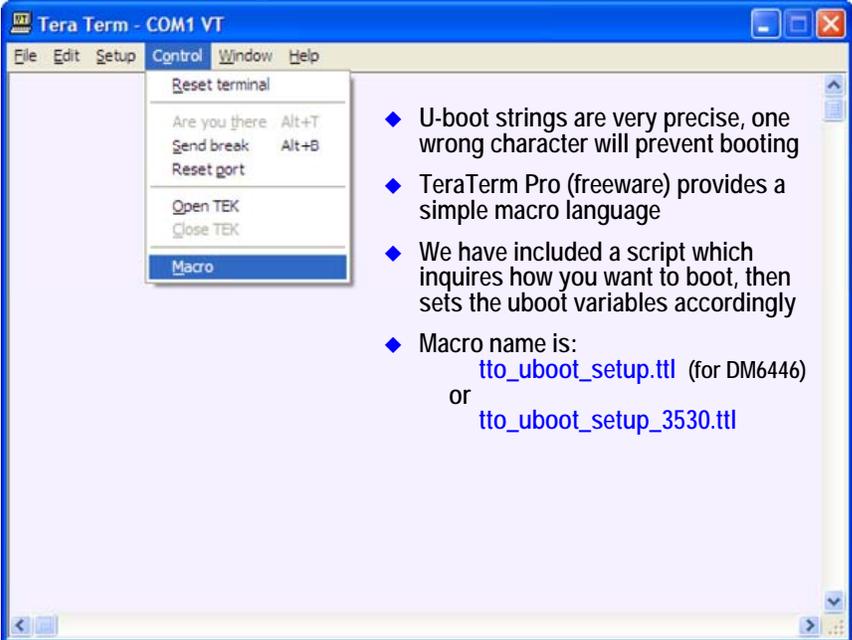
U-Booting : Static vs Dynamic IP

- ◆ You must specify the IP addresses
- ◆ Everywhere we previously had `dhcp` must now reference the static ip addresses
- ◆ This example creates a variable called `myip` and used it in place of the previous `dhcp` entries in `bootargs`

```
[rs232]# setenv serverip 192.168.13.120
[rs232]# setenv ipaddr 192.168.13.121
[rs232]# setenv gateway 192.168.13.97
[rs232]# setenv netmask 255.255.255.224
[rs232]# setenv dns1 156.117.126.7
[rs232]# setenv dns2 157.170.1.5
[rs232]# setenv myip $(ipaddr):$(gateway):$(netmask):$(dns1):$(dns2)::off
[rs232]# setenv nfspath /home/user/workdir/filesys
[rs232]# setenv bootcmd bootm 0x2050000
[rs232]# setenv bootargs console=ttyS0,115200n8 noinitrd rw
ip=$(myip) root=/dev/nfs nfsroot=$(serverip):$(nfspath)
,nolock mem=120M $(videocfg)
[rs232]# saveenv
```

Configure U-Boot with Tera Term Macro's

Using Tera Term Macros



The screenshot shows the Tera Term window titled 'Tera Term - COM1 VT'. The 'Control' menu is open, displaying options: 'Reset terminal', 'Are you there Alt+T', 'Send break Alt+B', 'Reset port', 'Open TEK', 'Close TEK', and 'Macro'. The 'Macro' option is highlighted in blue.

- ◆ U-boot strings are very precise, one wrong character will prevent booting
- ◆ TeraTerm Pro (freeware) provides a simple macro language
- ◆ We have included a script which inquires how you want to boot, then sets the uboot variables accordingly
- ◆ Macro name is:
`tto_uboot_setup.ttl` (for DM6446)
or
`tto_uboot_setup_3530.ttl`

For More Information

For More Information

- ◆ *The Linux Filesystem Explained*
<http://www.freeos.com/articles/3102/>

Tools Overview

Introduction

There are a wide range of tools available for ARM and ARM+DSP platforms. The list grows even longer when we include Linux and Linux distributions into our discussion of tools. Needless to say, in the short span of this chapter we'll only be able to examine a small sampling of tools – here, we'll try to focus on some of the tools directly supported by TI and some of its third parties.

We'll start out by briefly discussing the types of hardware development platforms available from TI; after which we'll examine the Software Development Kits from TI.

Next we'll take a quick look at some of the IDE tools supporting both ARM and DSP development.

Finally, we will try to define Linux distributions, as well as provide a summary of commercial and community options supporting devices from Texas Instruments.

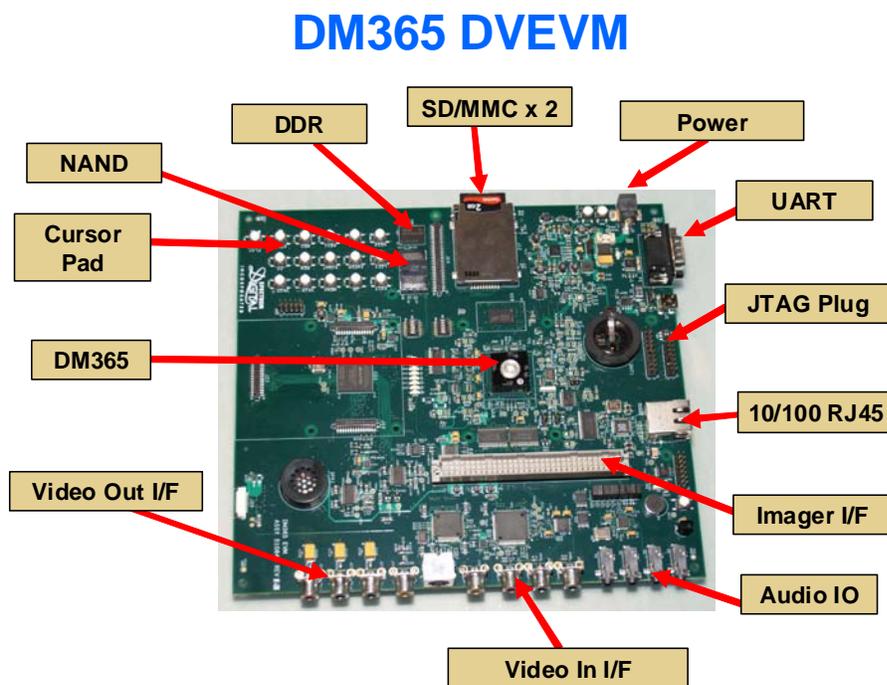
Along the way we'll see that it is important to think about what your role is on your development team, as that can help us narrow down which tools will best support our development needs.

Chapter Topics

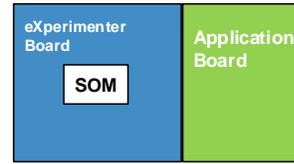
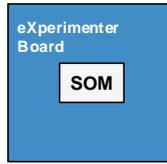
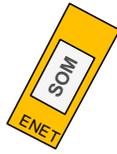
- Tools Overview 4-1**
- Development Platforms..... 4-2*
 - Hardware 4-2
 - Software Development Kits (SDK, DVSDK) 4-5
- Linux Distributions 4-6*
 - What are distributions? 4-6
 - O/S Choices 4-8
 - Community Options 4-9
 - Commercial Options..... 4-12
- (Optional) How Do I Rebuild Linux ... and Why? 4-14*
- (Optional) Software Developer Roles & Tools 4-16*
 - Linux Tools 4-16
 - DSP Tools..... 4-18

Development Platforms

Hardware



Modular EVM Kits – AM3517 Example



SOM Module

AM3517 SOM-M2

Price: < \$100

SW Development

- ◆ 1.6" x 2"
- ◆ Features:
 - 256 MB DDR2 SDRAM
 - 512 MB NAND flash
 - Wired Ethernet
 - Wireless 802.11b/g/n*
 - Bluetooth 2.1 + EDR IF*
- ◆ Self-boot Linux image
- ◆ Purchase – Logic via Arrow, Avnet, Digikey
- ◆ Support – Logic

eXperimenter Kit

SDK-XAM3517-10-256512R

Price: \$199

SW and H/W Dev't

- ◆ 5" x 6"
- ◆ Features SOM features +
 - HDMI (video only)
 - MMC/SD card slot
 - Network/USB/Serial/JTAG /Logic-LCD Connectors
 - Built-in XDS100 emulation
- ◆ Purchase – Logic via Arrow, Avnet, Digikey
- ◆ Support – Logic
- ◆ SW: Linux, WinCE

EVM

TMDXEVM3517

Price: \$999

Full Development Platform

- ◆ EVM additionally includes:
 - LCD
 - Multimedia In/Out
 - KeyPad
 - Connect: CAN, RJ45, USB, UART, stacked SD
- ◆ Channel – TI & distribution
- ◆ Support – TI & Logic
- ◆ Linux and WinCE SDK's (from TI); Android SDK is in development

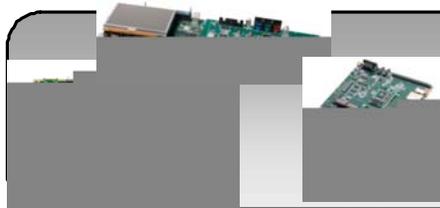
Hardware Development Environments

4 Types of Hardware Development Tools

System-on- Module	Community Board	eXperimenter Kit	Evaluation Module
			
Use Case	Use Case	Use Case	Use Case
<ul style="list-style-type: none"> ▪ Simplify system board design ▪ Medium for Prototype or Production end equipment 	<ul style="list-style-type: none"> ▪ Evaluation of processor functionality ▪ Application development with limited peripheral access ▪ NOT supported by TI 	<ul style="list-style-type: none"> ▪ Evaluation of processor functionality ▪ Application development with limited peripheral access 	<ul style="list-style-type: none"> ▪ Touch-screen application development with full peripheral access ▪ Application specific development



Evaluation and Development Kits



Development Kit Contents:

- Evaluation board and Documentation
- Software Development Kits
- Development Tools

Tool	Part Number	Price	Availability
AM37x EVM	TMDXEVM3715	\$1495	TI / Mistral
AM/DM37x Eval Module	TMDX3730EVM	\$1495	TI / Mistral
OMAP35x EVM	TMDSEVM3530	\$1495	TI / Mistral
AM3517 EVM	TMDXEVM3517	\$999	TI / Logic
AM18x EVM	TMDXEVM1808L	\$1150	TI
OMAP-L138 EVM	TMDXOSKL138BET	\$849	TI / Logic
AM17x EVM	TMDXEVM1707	\$845	TI
AM18x Experimenter Kit	TMDXEXP1808L	\$445	TI

Community Boards & Modules



To Access:

Contact TI partners for more information or click link to buy now

	Tool	Part Number	Price *	Availability
Low Cost Kits	Beagle Board (OMAP35x)	Beagle	\$149	Community
	Hawkboard (OMAP-L138)	ISSPLHawk	\$89	Community
SOM	OMAP35x System on Module	OMAP35x SOM-LV	\$99	Logic
	Overo OMAP35x Computer on Module	Overo	\$149-\$219	Gumstix
	KBOC OMAP35x System on Module	KBOC	\$139	KwikByte

* Prices subject to change



Software Development Kits (SDK, DVSDK)

Software Development Kits		
S/W Dev'l Kit	Description	Processor(s)
Linux PSP SDK	Small Linux Distro supporting TI ARM devices	<ul style="list-style-type: none"> • OMAP35, AM35, AM18 • OMAP-L1 • DM644x, DM6467, DM3xx
"DVSDK"	TI provided libraries, examples, demos Codec Engine (VISA), DSPlink, Codecs/Algos (XDM), BIOS, XDC, Linux utilities, etc.	<ul style="list-style-type: none"> • All TI SOC's: ARM, DSP, ARM+DSP • Obviously, not all devices require all the s/w components
Code Gen Tools (not really "kits" per se)	<ul style="list-style-type: none"> ♦ Linux GNU Compiler (CodeSourcery) ♦ C6000 DSP Compiler (TI) 	• All TI ARM and DSP devices where appropriate
Graphics SDK	Graphix SVSGX development kit OPENGL ES / VG demos, drivers, targetfs, Getting Started Guide	<ul style="list-style-type: none"> • OMAP3515, OMAP3530 • AM3517

- ♦ PSP is a TI specific acronym that represents the name of the group inside of Texas Instruments which "owns" the kernel and driver development activities: Platform Support Package team
- ♦ Wireless SDK is available independently of these other kits to support the TI WLxxxx Bluetooth/WiFi devices

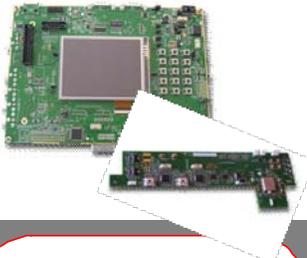
Wireless SDK : Getting started with WL1271 on OMAP35x EVM

Software

- Pre-integrated with TI's SDK
- WLAN and *Bluetooth*® software support (FM support not included)
- Pre-tested against WiFi and *Bluetooth*® specifications
- Open Source Linux drivers
 - Kernel 2.6.x
 - TI WLAN driver
 - BlueZ *Bluetooth*® stack
- Windows® CE 6.0 drivers
 - Available in mid 2010
 - Microsoft WiFi and *Bluetooth*® stacks
 - Adeneo's *Bluetooth* Manager

Documentation

- User Guides
- Complete API reference
- Application Notes
- Demo applications and sample code



- Evaluate 802.11b/g/n and *Bluetooth*® capability, and begin SW development
- Included in EVM box: Mar 2010
- Standalone Connectivity Card upgrade available from Mistral <http://www.mistralsolutions.com/WL1271>
- WL1271 module available from LS Research and its distributors http://www.lsr.com/products/radio_modules/802.11_BGN_BT/tiwi.shtml

Hardware

- Wireless Connectivity Card
 - WL1271 module with integrated TCXO
 - 2.4GHz chip antenna (default configuration)
 - U.FL antenna connector (optional configuration)
 - Plugs into EVM's Expansion Connector (supported on EVM Rev G)

Development tools and partners

- Compatible with EVM's toolchain
- Wireless Connectivity Card reference schematics
- Command Line Interface (CLI) to configure and exercise WLAN & *Bluetooth*® applications
- Partners
 - LS Research: WL1271 module
 - Mistral: Linux System Integrator
 - Adeneo: WinCE Syst. Integrator



Linux Distributions

What are distributions?

Build It Yourself ?

Quote from kernel.org:

If you're new to Linux, you don't want to download the kernel, which is just a component in a working Linux system. Instead, you want what is called a *distribution of Linux, which is a complete Linux system.*

There are numerous distributions available for download on the Internet as well as for purchase from various vendors; some are general-purpose, and some are optimized for specific uses.

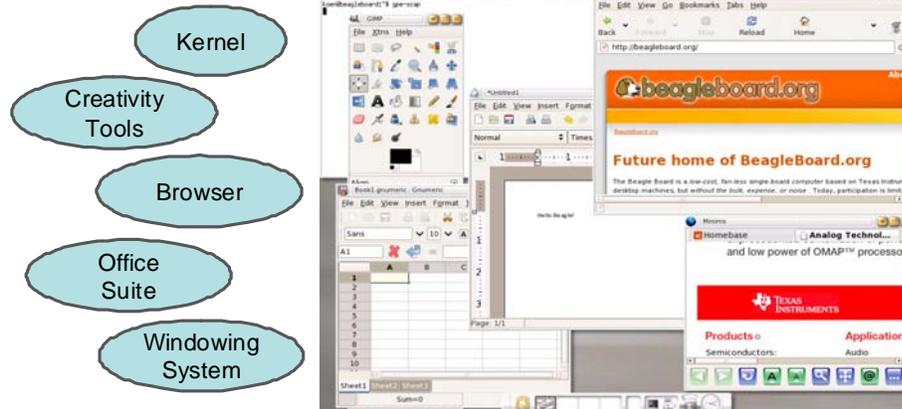
- ◆ This may be a bit of an understatement – even experienced users usually use a distribution
- ◆ Creating a distribution takes a lot of effort
- ◆ Maintaining a distribution ... takes even more effort
- ◆ In fact, using a distribution even takes quite a bit of effort

What Is a 'Linux Distribution'

A 'Linux distribution' is a combination of the components required to provide a working Linux environment for a particular platform:

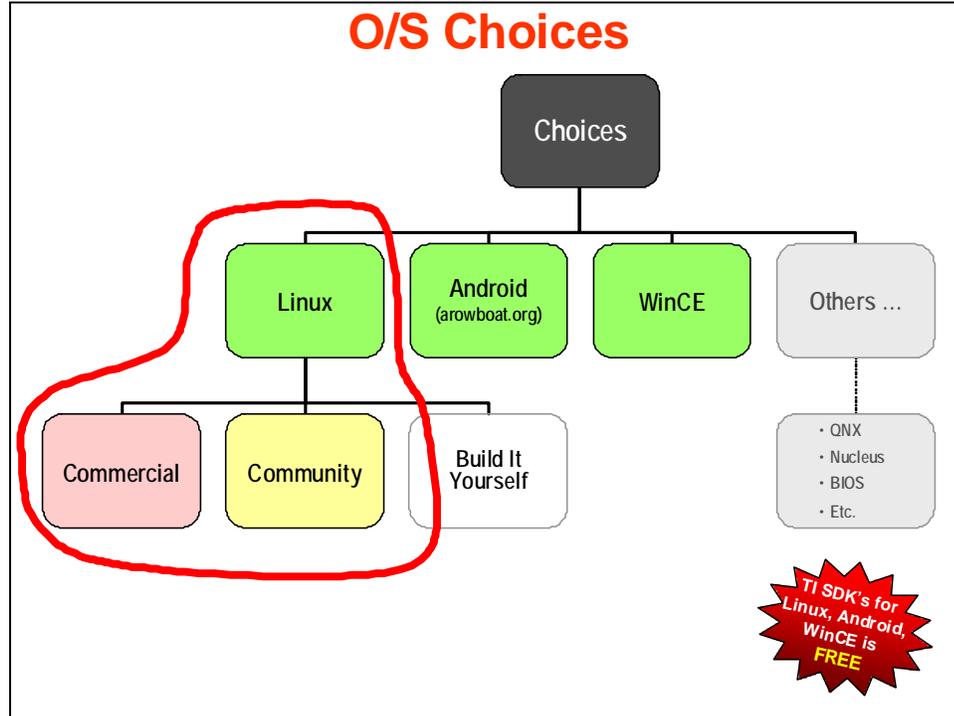
1. Linux kernel port
 - A TI LSP or Linux PSP is a Linux kernel port to a device, not just a set of device drivers
2. Bootloader
 - Uboot is the standard bootloader for ARM Linux
3. Linux 'file system'
 - This does NOT mean a specific type of file system like FAT file system or flash file system ... rather, it more like the "C:" drive in Windows
 - It refers to all the 'user mode' software that an application needs such as graphics libraries, network applications, C run-time library (glibc, uclibc), codec engine, dynamically-loaded kernel modules (CMEM, DSPLINK)
4. Development tools
 - CodeSourcery - GCC, GDB
 - MV DevRocket, CCSv5 (beta), GHS Multi, etc.

Linux Distributions



- ◆ Linux isn't complete without a distribution
- ◆ [MontaVista](#) and [Timesys](#), for example, provide commercial (i.e. production) distribution for TI's DaVinci / OMAP processors
- ◆ A few distributions supporting the open-source BeagleBoard (OMAP35x-based) include: [OpenEmbedded](#), Ubuntu, Fedora, Android, Gentoo, ARMedslack and ALIP

O/S Choices



Linux Distributions Options for TI

Custom (Build it Yourself)		Community		Commercial
Custom from Sources	Open Embedded (OE)	TI SDK (PSP)	Ångström	<ul style="list-style-type: none"> • Timesys • MontaVista • Mentor • RidgeRun
<ul style="list-style-type: none"> • "GIT" from kernel.org, and others 	<ul style="list-style-type: none"> • Bit-Bake • Recipies 	<ul style="list-style-type: none"> • OE / GIT • Binary Updated for each SDK release 	<ul style="list-style-type: none"> • Binary • Narcissus (online tool) • OE 	<ul style="list-style-type: none"> • Source • Binary (Update patches)

Ease of Use →

- Expert User (only)
- Latest

- Easy
- Tested

Community Options

Community Options

- ◆ **TI Linux SDK (PSP)**
 - ◆ Pre-built snapshot of Linux tested against specific version of TI Software Development Kits
 - ◆ Updated at each new SDK/DVSDK release
 - ◆ PSP = Platform Support Package (name of TI team)
 - ◆ Currently, a “BusyBox-based” bare-bones distro (“lean/mean”)
 - ◆ Arago open-source OE project
 - ◆ Advantage of OE – recipes can be reused by Angstrom (or custom OE) users
 - ◆ In general, users shouldn’t (re)build using OE; no reason to, because if you want more features, we recommend you go with Angstrom (also built using OE)

- ◆ **Ångström**
 - ◆ Open-source, full-featured Linux distro targeted for embedded systems
 - ◆ Get it from:
 - ◆ User-compiled binaries widely available for many targets
 - ◆ **Narcissus** (<http://www.angstrom-distribution.org/narcissus>)
Web-based tool creates binary vers (w/ your own package sel'n)
 - ◆ Built using OE (user community can re-use TI OE recipes)

Ångström : Narcissus

Welcome!

This is an online tool to create so called 'rootfs' images for your favourite device. This page will guide through the basic options and will close to let you select the additional packages you want.

Base settings:

Select the machine you want to build your rootfs image for:

Choose your image name.
This is used in the filename offered for download, makes it easier to distinguish between rootfs images after downloading.

Choose the complexity of the options below.
simple will hide the options most users don't need to care about and advanced will give you lots of options to fiddle with.

User environment selection:

Console gives you a bare commandline interface where you can install a GUI into later on. X11 will install an X-window environment and present you with a Desktop Environment option below. Ople is a qt/e 2.0 based environment for PDA style devices.

X11 Desktop Environments:

Enlightenment
 GNOME

Current configuration:
 Machine: am3517-evm
 Image name: T10_Example
 Image type: tbz2

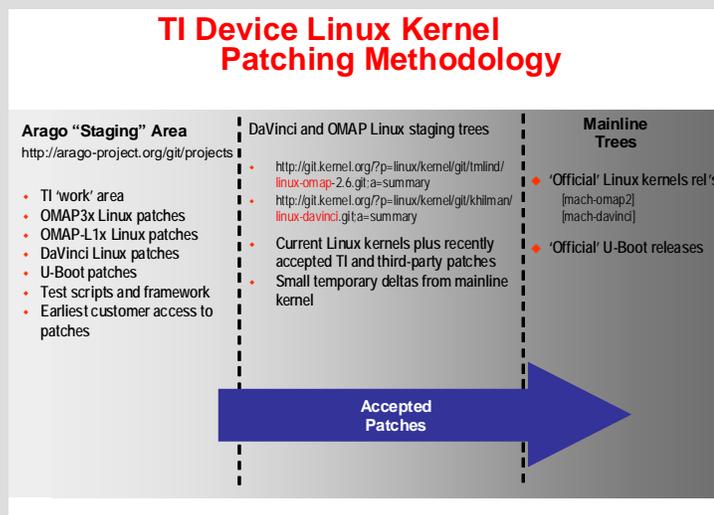
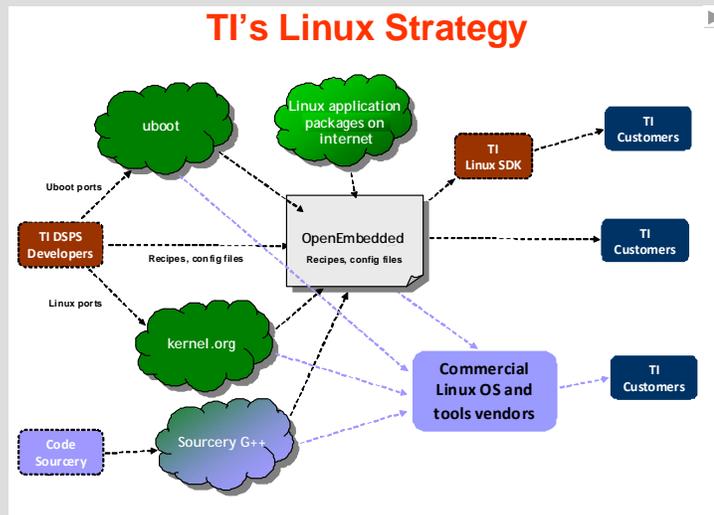
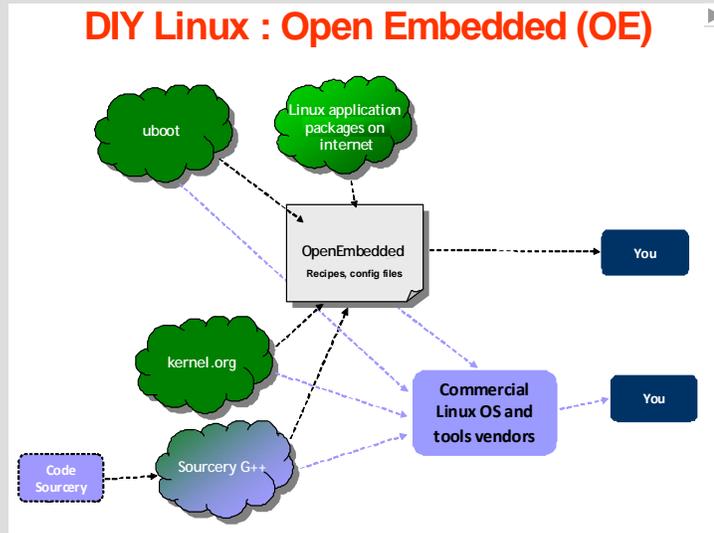
Additional Packages:
 angstrom-task-gnome
 bash
 shadow

(<http://www.angstrom-distribution.org/narcissus>)

DIY Options

- ◆ **Open-Embedded (OE)**
 - ◆ Build Linux from source using OE's Bit-Bake recipe(s)
 - ◆ Many recipes available for various architectures, including many variations for a given device
 - ◆ Builds full-up distro including Kernel and Filesystem
 - ◆ TI builds it's PSP Linux distro's via OE
- ◆ **Build from Sources (roll your own)**
 - ◆ Build directly from sources, such as kernel.org
 - ◆ Use GIT, as well as SVN and others to get sources from repo's
 - ◆ *Are you nuts? Either you want to waste your life redoing what OE did, or you're so advanced you don't need this presentation.*

Side-bar : Open- Embedded



Commercial Options

Commercial O/S Vendors

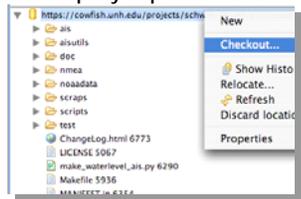
◆ Linux <ul style="list-style-type: none"> ◆ TimeSys ◆ MontaVista ◆ Wind River ◆ Mentor ◆ Ridgerun 	
◆ WinCE <ul style="list-style-type: none"> ◆ Adeneo ◆ Mistral ◆ MPC Data ◆ BSQUARE 	◆ RTOS <ul style="list-style-type: none"> ◆ Green Hills ◆ Wind River (VxWorks) ◆ ELogic (ThreadX) ◆ QNX ◆ Mentor (Nucleus)

Linux Partner Strategy

- ◆ **Commercial:** provide support, off-the-shelf Linux distributions or GNU tools
- ◆ **Consultants:** provide training, general embedded Linux development expertise, or specific expertise for developing drivers or particular embedded applications
- ◆ http://www.tiexpressdp.com/index.php/Linux_Consumers_and_Commercial_Linux_Providers

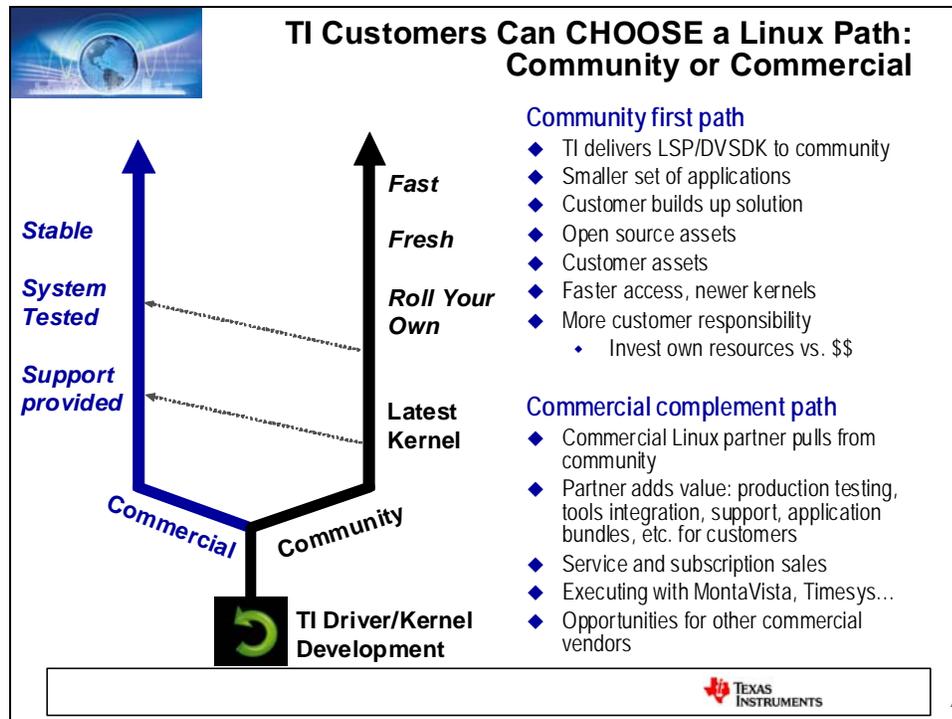
Commercial vs Community

- ◆ **Commercial**
 - ◆ **Less effort** – another team does the work of porting new kernel to the distribution ... and then laboriously testing it over-and-over again
 - ◆ **More robust** – many customers generating more inputs/errata to team testing and maintaining the distribution
 - ◆ **More secure** – among other reasons, many homebrew distributions don't get around to adding security patches due to effort and time
 - ◆ **Latest features?** Many vendors *backport* new features into their kernels – thus, you get the stability of a known kernel but with new features
 - ◆ **Good choice if:** you don't need the absolute latest features; you have a many projects to amortize the costs; you're a Linux wiz who really knows what they're doing.
 - ◆ **Bottom Line** – Commercial distributions trade cost (and the bleeding edge features) for robustness and less-effort. What is it worth, if your company depends on this product?
- ◆ **Community (to Git or not)**
 - ◆ Access to **latest improvements** in Linux kernel
 - ◆ Want to know exactly how it is all put together
 - ◆ Maximum involvement in Linux community
 - ◆ **No costs** ... (unless you count your labor)
 - ◆ **Bottom Line** – Choose this option if you have the Linux expertise and labor is cheaper than NRE; or, you need access to the very latest features



Example Comparison : [MVL Pro 5.0](#) vs [GIT](#)

	MVL 5.0 Pro	Community Linux
Kernel Version	Uses 2.6.18, which is almost 3 years old	Uses latest available kernel
Kernel bug-fixes	Applied to 2.6.18, so no need to change kernel versions	Applied to current release, which changes every few months. User may need new kernel to get a fix.
File System	Comprehensive host and target file systems with GUI tools for optimization.	Not part of kernel. TI is addressing through Arago. Initially may be less user-friendly than MVL.
Linux run-time Licensing	Demo copy and LSP open source, but original licensing has created confusion.	TI offering is clearly free as GIT Linux distributions are open source.
Tools licensing	GNU Tools free. IDE requires annual subscription.	GNU Tools free. IDE requires annual subscription.
Third-party support	MV and its partners	Multi-vendor, including MV



(Optional) How Do I Rebuild Linux ... and Why?

How Do I Build It, Let Me Count the Ways...

1. **Don't** ... find a pre-built Linux ulmage
2. **Build Default Linux**
 - a. **make defconfig**
 - b. **make ulmage**

Why Re-Build Linux Kernel?

- ◆ TI SDK's often support various ARM CPU's, thus GSG directs you to specify target processor and rebuild kernel
- ◆ You made changes to a Linux source file (i.e. driver, etc.)

Change to Kernel's Directory (TI/MontaVista LSP Example)

```
> cd ti-davinci/linux-2.6.18_pro500
```

Configure the Kernel

```
> make ARCH=arm CROSS_COMPILE=arm_v5t_le- davinci_dm644x_ defconfig
```

Build the Kernel

```
> make ARCH=arm CROSS_COMPILE=arm_v5t_le- uImage
```

Configure the Kernel

```
host $ cd ti-davinci/linux-2.6.18_pro500
```

```
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- davinci_dm644x_defconfig
```

Verify Kernel

```
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- checksetconfig
```

Customize the Kernel

```
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- menuconfig
```

Build the Kernel

```
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- uImage
```

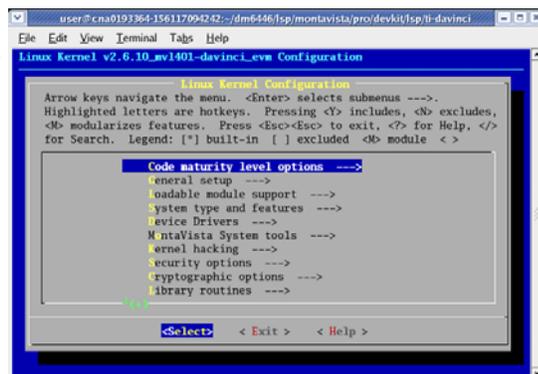
Build Loadable Modules (i.e. dynamic "insmod" modules)

```
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le- modules
```

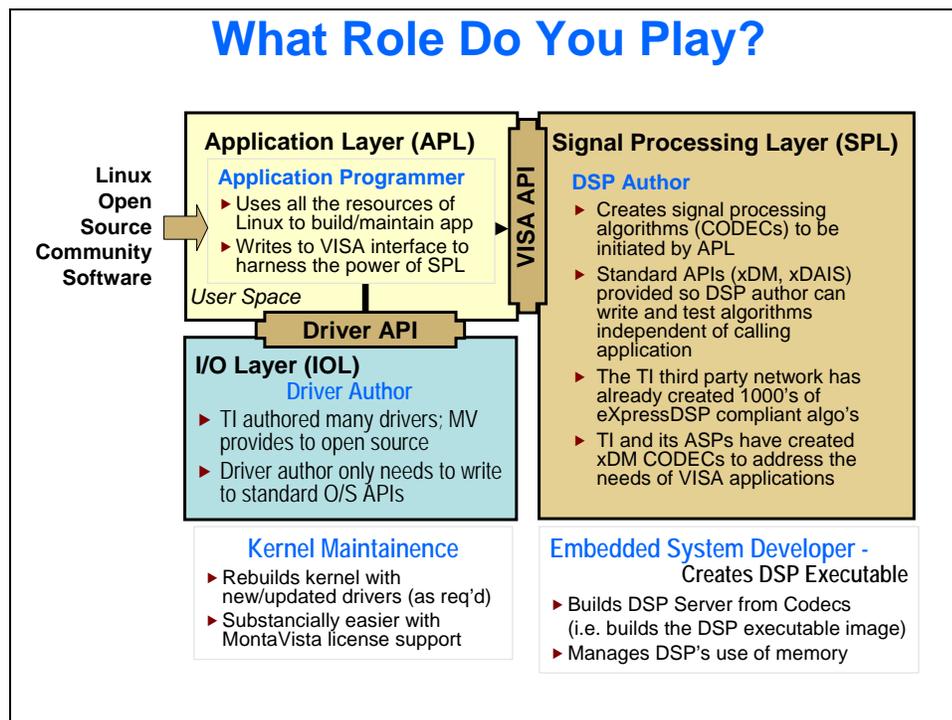
```
host $ make ARCH=arm CROSS_COMPILE=arm_v5t_le-  
INSTALL_MOD_PATH=/home/<useracct>/workdir/filesys  
modules_install
```

How Do I Build It, Let Me Count the Ways...

1. Don't ... find a pre-built Linux ulmage
2. Build Default Linux
 - a. make defconfig
 - b. make ulmage
3. Build 'Custom' Linux
 - a. make defconfig
 - b. make menuconfig
 - c. make ulmage



(Optional) Software Developer Roles & Tools



Linux Tools

This section contains information on the Linux tools used to for these developer roles.

- ▶ Application Tools (ARM)
- ▶ Driver Development
- ▶ Kernel Maintenance

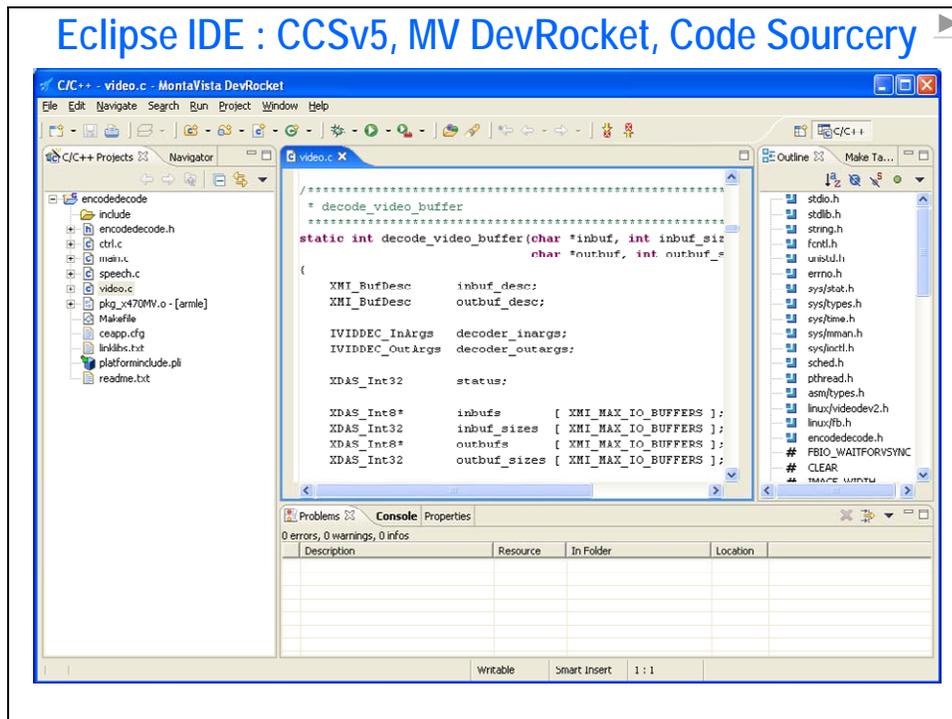
You'll find subsections that address tools for these roles by their vendor.

Linux Tools Summary

Linux Development IDE Tools			
ARM/Linux Tools	Application Author	Driver Development	Linux Kernel & Filesystem
MontaVista (MVL) DevRocket	<ul style="list-style-type: none">• “Best in class” for Linux application development• GNU ARM compiler (GCC)• GDB-based debug• Eclipse-based IDE	<ul style="list-style-type: none">• IDE debugger for driver code development• KGDB based debug	<ul style="list-style-type: none">• “Best in class”• DevRocket’s Kernel & Platform (filesystem) projects make quick and easy for this category
Green Hills (GHS) Multi IDE	<ul style="list-style-type: none">• Also “Best in class” for Linux application development• Supports multiple OSs: INTEGRITY, Linux, etc.• GHS <i>optimized</i> ARM compiler	<ul style="list-style-type: none">• “Best in class” debug support for Linux driver development• IDE debugger for driver code development• Advanced debug target server – use KGDB or JTAG to connect to target	<ul style="list-style-type: none">• Rudimentary support for building the Linux Kernel
Texas Instruments Code Composer (CCStudio v5 Alpha)	<ul style="list-style-type: none">• Alpha/beta versions are free via the TI website• Add’s GDB debugging and Linux awareness	<ul style="list-style-type: none">• IDE debugger for driver code development• JTAG to connect to target	<ul style="list-style-type: none">• Not applicable
Open Source	<ul style="list-style-type: none">• GDB• DDD• Eclipse	<ul style="list-style-type: none">• KGDB	<ul style="list-style-type: none">• Standard command-line (or ascii-based gui) support for building the linux kernel

IDE Examples

Eclipse : CCSv5, MV DevRocket, Code Sourcery



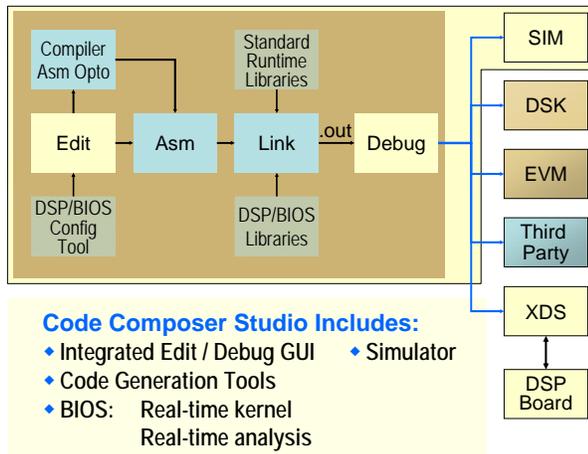
DSP Tools

DSP Development Tools

DSP Tools	DSP Programmer	Embedded System Development
MontaVista DevRocket	<ul style="list-style-type: none"> • Not Applicable 	<ul style="list-style-type: none"> • Not Applicable
Green Hills (GHS) Multi IDE	<ul style="list-style-type: none"> • Single IDE for ARM and DSP • Supports ARM + DSP devices • Only support basic DSP/BIOS visualization • GHS uses TI's Optimizing DSP Compiler 	<ul style="list-style-type: none"> • TI Code Generation tools provide system configuration and building of executable DSP embedded image • Multi-proc (ARM-DSP) support may aide with finding problems with final system integration
Texas Instruments Code Composer Studio (CCStudio)	<ul style="list-style-type: none"> • Highly-integrated development environment • Full DSP/BIOS aware debugging <ul style="list-style-type: none"> • BIOS Visualization Tools • Kernel Object Viewer • BIOS Configuration Tool • Supports ARM+DSP and DSP-only devices • Optimizing TI Optimizing DSP Compiler • Wizards to help with building and verifying xDAIS/xDM algorithms (i.e. codecs) 	<ul style="list-style-type: none"> • TI Code Generation tools provide system configuration and building of executable DSP embedded image

- Bottom Line:
- ◆ Go with GHS for an all-in-one ARM+DSP development tool
 - ◆ Choose TI for "Best-in-Class" DSP development at a lower price point

Code Composer Studio



Code Composer Studio Includes:

- ◆ Integrated Edit / Debug GUI
- ◆ Code Generation Tools
- ◆ BIOS: Real-time kernel Real-time analysis
- ◆ Simulator

Multi IDE: Simultaneous Dual-Core DaVinci™ Technology ARM/C64x+ Debugging

The screenshot shows the Multi IDE interface with two separate development environments side-by-side. The left window shows code for 'core 1' and the right window shows code for 'core 2'. Both windows have their own debug consoles and toolbars. A central window displays the 'Tasks for "response\ghnu\da20\BoardMemory_192_168_200_62"' and a table of objects under 'agency'.

Object Id	Name	Objects under "agency"
Agency	Agency	Agency
ADSP06 (core 1)	ADSP06 (core 1)	ADSP06 (core 1)
CE64x (core 1)	CE64x (core 1)	CE64x (core 1)
ADSP06 (core 2)	ADSP06 (core 2)	ADSP06 (core 2)
CE64x (core 2)	CE64x (core 2)	CE64x (core 2)
MA TI:	MA TI:	MA TI:
41	41	41

- ◆ Advanced target debug server
- ◆ Loading and debugging images on both cores
- ◆ Multi-core debug
 - Individual core instruction stepping
 - Processor context debugging and viewing (e.g. registers, variables,...)
- ◆ Single instance of MULTI debugger
- ◆ OS aware debugging on all cores (Linux, INTEGRITY, DSP/BIOS™ kernel, ...)

Building Programs with gMake

Introduction

DaVinci software can be built using GNU's standard gMake utility. In the future, when we introduce building codec engine servers, we'll invoke another tool – XDC to build the application. In one part of the lab, you will be introduced to Configuro, and XDC tool that allows you to consume packaged content from TI (or 3rd parties). It will create some files used by gMake to build the final application.

This chapter introduces the GNU gMake utility and how to build a makefile from scratch. You will then have a chance to analyze the makefile that will be used throughout the rest of the workshop.

Learning Objectives

Outline

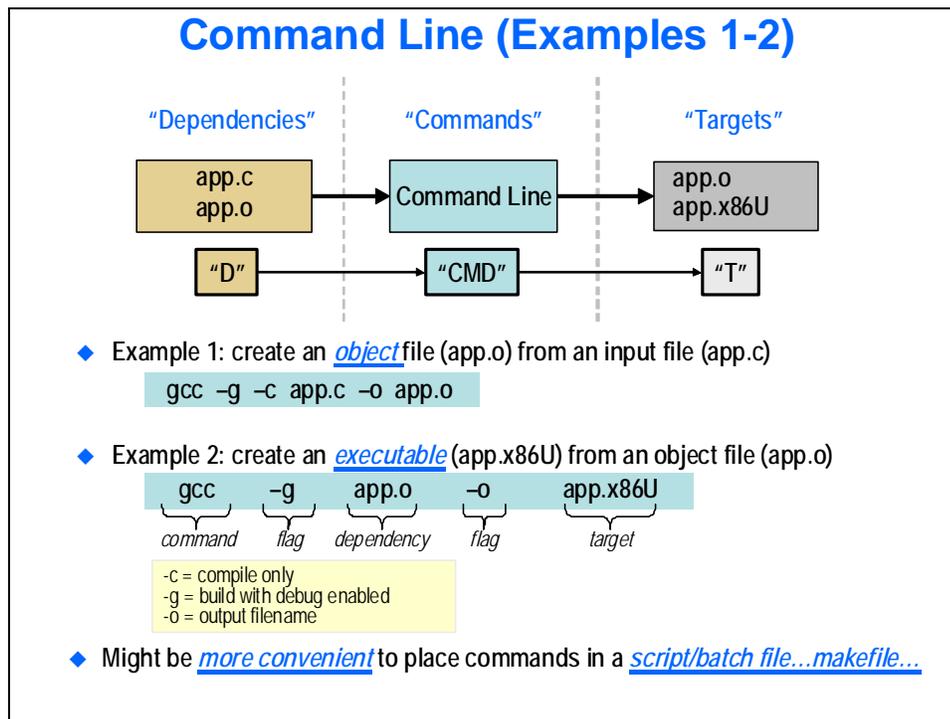
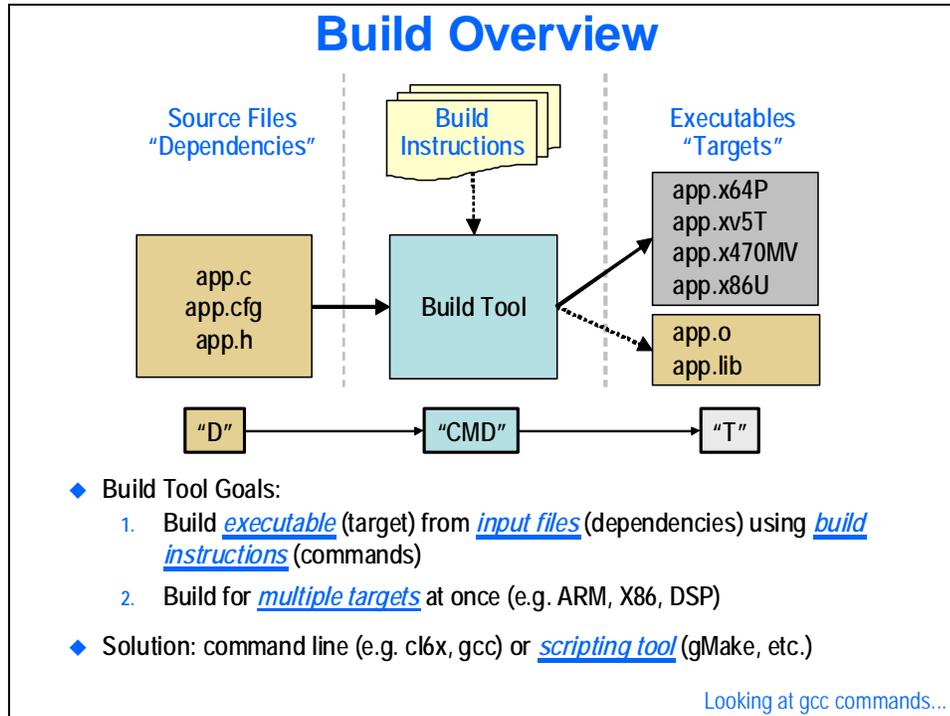
- ◆ Brief overview of [gcc](#) for compiling and linking
- ◆ Understand how to [build targets](#) using gmake
- ◆ Use [rules](#) and [variables](#) (built-in, user-defined) in makefiles
- ◆ Learn how to add “[convenience](#)” and “[debug](#)” rules
- ◆ Learn how to handle C (header file) [dependencies](#)
- ◆ Use [Configuro](#) to consume packaged content
- ◆ [Debug](#) a Linux/ARM program using [GDB](#) with [CCSv5/Eclipse](#)

Chapter Topics

Building Programs with gMake	5-1
<i>Building Programs With gMake</i>	5-3
Big Picture	5-3
Creating/Using a Makefile.....	5-4
Using Variables and Printing Debug Info.....	5-6
Wildcards and Pattern Substitution	5-9
Basic Makefile – Code Review	5-10
Handling Header File Dependencies	5-11
<i>Using Configuro to Consume Packages</i>	5-12
Current Flow.....	5-12
What is a Package?	5-13
Configuro – How it Works	5-13
Using Configuro in the Upcoming Lab	5-14
MakeFile Example – Using Configuro.....	5-15
Looking Inside Compiler.opt & Linker.cmd	5-16
<i>Debugging with CCSv5 (Eclipse).....</i>	5-17
Debugging & Debuggers.....	5-17
GNU Debugger.....	5-17
Graphical Debugging with GDB	5-18
Setting Up CCSv5 for GDB	5-19
<i>Appendix</i>	5-21

Building Programs With gMake

Big Picture



Creating/Using a Makefile

Basic Makefile with Rules

- ◆ One of the more common *“scripting”* tools is GNU Make, aka gMake, aka Make...
- ◆ gMake uses “rules” to specify build commands, dependencies and targets
- ◆ Generically, a **RULE** looks like this:


```

            TARGET : DEPENDENCY
            [TAB]  COMMANDS...
```
- ◆ Remember Example 2? Let’s make this into a simple Makefile rule:


```

            gcc      -g      app.o      -o      app.x86U
            command  flag      dependency  flag      target
```
- ◆ Becomes....


```

            app.x86U : app.o
                   gcc -g app.o -o app.x86U
```

} **RULE**

Creating Your First Makefile

Command Lines

```

            gcc -c -g app.c -o app.o
            gcc -g app.o -o app.x86U
```

Makefile

```

            # Makefile for app.x86U (goal)

            app.x86U : app.o
                   gcc -g app.o -o app.x86U

            app.o : app.c
                   gcc -g -c app.c -o app.o
```

Running gMake

- ◆ To run gMake, you can use the following commands:
 - `make` (assumes the makefile name is "makefile", runs FIRST rule only)
 - `make app.x86U` (specifies name of "rule" – e.g. `app.x86U`)
 - `make -f my_makefile` (can use custom name for makefile via forcing flag... -f)
- ◆ gMake looks at timestamps for each target and dependency. If the target is newer than its dependencies, the rule (and associated commands) will not be executed.
- ◆ To "rebuild all", use the "clean" rule to remove intermediate/executable files...

Looking at convenience rules...

"Convenience" Rules

- ◆ Convenience rules (e.g. `all`, `clean`, `install`) can be added to your makefile to make building/debug easier.
- ◆ For example, a "clean" rule can delete/remove existing intermediate and executable files prior to running gMake again.
- ◆ If the rule's target is NOT a file, use the .PHONY directive to tell gMake not to search for that target filename (it's a phony target).
- ◆ Let's look at three common convenience rules (to use, type "`make clean`"):

"Build All Targets"	<code>.PHONY : all</code> <code>all : app.x86U ...(all "goals" listed here)</code>
"Remove Unwanted Files"	<code>.PHONY : clean</code> <code>clean :</code> <code> rm -rf app.o</code> <code> rm -rf app.x86U</code>
"Copy Executable to the install directory"	<code>.PHONY : install</code> <code>install : app.x86U</code> <code> cp app.x86U /dir1/install_dir</code>

Note: "all" rule is usually the first rule because if you type "make", only the first rule is executed

gMake Rules Summary



```

graph LR
    D["D"] --> CMD["CMD"]
    CMD --> T["T"]
    
```

- ◆ 3 common uses of rules include:
 - [.x] – final executable
 - [.o] – intermediate/supporting rules
 - [.PHONY] – convenience rules such as clean, all, install
- ◆ Examples:


```

.x      app.x86U : app.o
        gcc -g app.o -o app.x86U
-----
.o      app.o : app.c
        gcc -g -c app.c -o app.o
-----
.PHONY clean :
clean :
        rm -rf app.x86U
            
```
- ◆ Run:
 - `make` (assumes makefile name is "makefile" or "Makefile" and runs the first rule only)
 - `make app.x86U` (runs the rule for app.x86U and all supporting rules)
 - `make clean`

Using Variables and Printing Debug Info

Using Built-in Variables



```

graph LR
    D["D"] --> CMD["CMD"]
    CMD --> T["T"]
    
```

- ◆ Simplify your makefile by using these built-in gMake variables:
 - \$@ = Target
 - \$^ = All Dependencies
 - \$< = 1st Dependency Only
- ◆ Scope of variables used is the current rule only.
- ◆ Example:

Original makefile...

```

app.x86U: app.o
        gcc -g app.o -o app.x86U
            
```

Becomes...

```

app.x86U: app.o
        gcc -g $^ -o $@
            
```

User-Defined Variables & Include Files



- ◆ [User-defined variables](#) simplify your makefile and make it more readable.
- ◆ [Include files](#) can contain, for example, [path statements](#) for build tools. We use this method to place absolute paths into one file.
- ◆ If "-include path.mak" is used, the "-" tells gMake to keep going if errors exist.
- ◆ Examples:

makefile

```
include path.mak
CC := $(CC_DIR)gcc
CFLAGS := -g
LINK_FLAGS := -o

app.x86U : app.o
    $(CC) $(CFLAGS) $^ $(LINK_FLAGS) $@
```

path.mak

```
CC_DIR := /usr/bin/
...
# other paths go here...
```

Printing Debug/Warning Info



- ◆ Two common commands for printing info to stdout window:
 - `echo` – command line only, flexible printing options ("`@`" suppresses echo of "echo")
 - `$(warning)` – can be placed [anywhere in makefile](#) – provides filename, line number, and message
- ◆ Examples:

```
app.x86U : app.o
    $(CC) $(CFLAGS) $^ $(LINK_FLAGS) $@
    @echo
    @echo $@ built successfully; echo

$(warning Source Files: $(C_SRCS))
app.x86U : app.o $(warning now evaluating dep's)
    $(CC) $(CFLAGS) $^ $(LINK_FLAGS) $@
    $(warning $@ built successfully)
```

- ◆ `$(warning)` does *not* interrupt gMake execution
- ◆ A similar function: "`$(error)`" stops gMake and prints the error message.

Quiz

- ◆ Fill in the blanks below assuming (start with .o rule first):
 - Final "goal" is to build: `main.x86U`
 - Source files are: `main.c`, `main.h`
 - Variables are: `CC` (for `gcc`), `CFLAGS` (for compiler flags)

```
CC := gcc
CFLAGS := -g
# .x rule
_____ : _____
_____ -o _____

# .o rule
_____ : _____
_____ -c _____ -o _____
```

Wildcards and Pattern Substitution

Using “Wildcards”

- ◆ [Wildcards \(*\)](#) can be used in the [command](#) of a rule. For example:

```
clean :
  rm -rf *.o
```

Removes all .o files in the current directory.

- ◆ Wildcards (*) can also be used in a [dependency](#). For example:

```
print : *.c
  lpr -p $?
```

Prints all .c files that have changed since the last print.

Note: automatic var "\$?" used to print only changed files

- ◆ However, wildcards (*) can [NOT](#) be used in [variable declarations](#). For example:

```
OBJS := *.o
```

OBJS = the string value ".o" – not what you wanted*

To set a [variable](#) equal to a list of object files, use the following [wildcard function](#):

```
OBJS := $(wildcard *.o)
```

Simplify Your MakeFile Using “%”

- ◆ Using pattern matching (or pattern substitution) can help [simplify your makefile](#) and help you remove explicit arguments. For example:

Original Makefile

```
app.x86U : app.o main.o
  $(CC) $(CFLAGS) $^ -o $@

app.o : app.c
  $(CC) $(CFLAGS) -c $^ -o $@

main.o : main.c
  $(CC) $(CFLAGS) -c $^ -o $@
```

Makefile Using Pattern Matching

```
app.x86U : app.o main.o
  $(CC) $(CFLAGS) $^ -o $@

%.o : %.c
  $(CC) $(CFLAGS) -c $^ -o $@
```

- ◆ The .x rule depends on the .o files being built – that’s what kicks off the .o rules
- ◆ % is a shortcut for \$(patsubst ...), e.g. \$(patsubst .c, .o)

Basic Makefile – Code Review

Basic gMake Makefile – Review (1)

Include file that contains tool paths (e.g. the path to gcc)

User-defined variables

“all” rule

Main “goal” of makefile... rule for app.x86U

Intermediate .o rule. Notice the use of pattern matching.

```
# -----
# ----- includes -----
# -----
include ./path.mak

# -----
# ----- user-defined vars -----
# -----
CC := $(X86_GCC_DIR)gcc
CFLAGS := -g
LINKER_FLAGS := -lstdc++

# -----
# ----- make all -----
# -----
.PHONY : all
all : app.x86U

# -----
# ----- executable rule (.x) -----
# -----
app.x86U : app.o
$(CC) $(CFLAGS) $(LINKER_FLAGS) $^ -o $@
@echo; echo $@ successfully created; echo

# -----
# ----- intermediate object files rule (.o) -----
# -----
%.o : %.c
$(CC) $(CFLAGS) -c $^ -o $@
```

Basic gMake Makefile – Review (2)

“clean” rule. Removes all files created by this makefile. Note the use of .PHONY.

“printvars” rule used for debug. In this case, it echos the value of variables such as “CC”, “CFLAGS”, etc.

```
# -----
# ----- clean all -----
# -----
.PHONY : clean
clean :
rm -rf app.x86U
rm -rf app.o

# -----
# ----- basic debug for makefile -----
# ----- example only -----
# -----
.PHONY : printvars
printvars:
@echo CC = $(CC)
@echo X86_GCC_DIR = $(X86_GCC_DIR)
@echo CFLAGS = $(CFLAGS)
@echo LINKER_FLAGS = $(LINKER_FLAGS)
```

Handling Header File Dependencies

Handling Header File Dependencies (1)

- ◆ Handling header files dependencies *can be difficult* – especially if you have to edit the header filenames manually.
- ◆ If a header file gets updated, but is not specified in the makefile, the change *will not trigger a new .o target* to be rebuilt.
- ◆ Let's see how to handle this properly...

app.c

```
#include "app.h"
...
```

makefile

```
app.x86U : app.o
    $(CC) $(CFLAGS) $^ -o $@

app.o : app.c app.h
    $(CC) $(CFLAGS) -c $< -o $@
```

- ◆ Which dependency is missing from the makefile above?
- ◆ Do you want to *manually specify* all the header files for all .c programs in your application? Or would something automatic be of interest? ...

Handling Header File Dependencies (2)

- ◆ Which *common build tool* is really good at locating header files in .c files and building a list? *gcc*, of course.
- ◆ So, let's *use the compiler* to create a list of header file dependencies and place these in a corresponding .d file:

```
%d : %.c
    $(CC) -MM $(CFLAGS) $< > $@
    @sed -e 's|.*:|$*.o:|' < $@ >> $@

-include app.d
```

- ◆ *-MM* option *creates a list of header files* from a .c file and echos them to the stdout window – which we pipe into a corresponding .d file (using > \$@).
- ◆ The cryptic “sed” command is used to *reformat the list of files* in the .d file (e.g. app.h) to look like a rule for app.o:

app.d

```
app.o : app.h
```

- ◆ Three notes about the *-include*:
 - Includes the new app.d file which contains the proper header file dependencies.
 - The “-” prevents us from getting an error on first pass before .d file is created.
 - Remember, any time an included file is regenerated, make processing starts over.

Code for Handling Dependencies

```

C_SRCS := $(wildcard *.c)           # Create list of .c files
OBJS   := $(subst .c,.o,$(C_SRCS))
DEPS   := $(subst .c,.d,$(C_SRCS)) # Create list of .d files from .c files

# 3. Dependency Rule (.d)
# -----
%.d : %.c
    @echo "Generating dependency file"
    $(CC) -MM $(CFLAGS) $< > $@      # Use gcc to generate list of dep's
    @echo "Formatting dependency file"
    $(call format_d, $@, $(PROFILE))  # Call macro to format dep file list

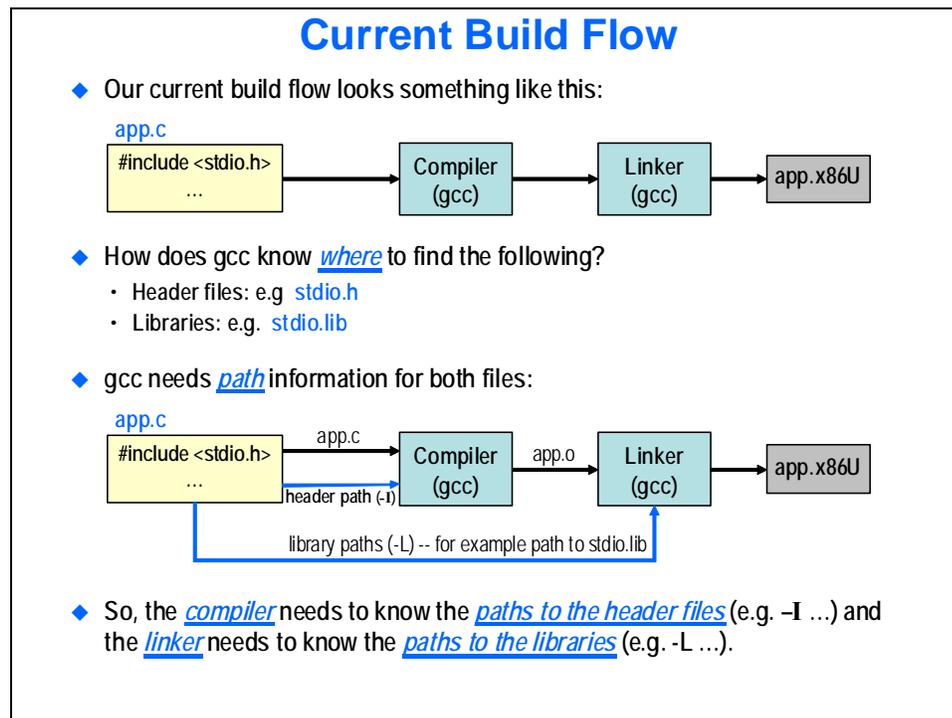
# Define Macro which formats .d file as gmake rule
define format_d
    @mv -f $1 $1.tmp                 # Macro has two parameters:
    @sed -e 's|.*:|$$*.o:|' < $1.tmp > $1 # Dependency File (.d): $1
    @rm -f $1.tmp                     # Profile: $2
endef

# Include dependency files
# Don't include (.d) files if "clean" is the target
ifneq ($(filter clean,$(MAKECMDGOALS)),clean)
    -include $(DEPS)
endif

```

Using Configuro to Consume Packages

Current Flow



What is a Package?

Goal: Consume “Packages” from TI & 3rd Parties

- ◆ TI (and 3rd party) content is delivered as a “package”.
- ◆ Package = library + metadata = “smart” library

Packages contain:

- ◆ Payload (i.e. library)
- ◆ Program metadata
 - Version Info
 - Dependencies
 - Library/header references
 - Module (interface description)

- ◆ gcc STILL needs path information from this “package”:

- ◆ How does gcc know where to get this path information from?

Configuro – How it Works

Using Configuro

- ◆ Configuro is a tool that helps users consume/use packaged content.

- ◆ Configuro creates two files:
 - compiler.opt – includes compiler options and header file paths
 - linker.cmd – includes linker options and library file paths
- ◆ Configuro needs four other inputs to help it perform properly:
 - .cfg – indicates which packages to include
 - XDCPATH – list of paths where all the packages are located
 - Platform – e.g. ti.platforms.evmDM6446
 - Target – e.g. gnu.targets.MVArm9

Let's try using a function from a packaged library...

Using Configuro in the Upcoming Lab

Using Configuro

- ◆ We're using Configuro in the upcoming lab for two reasons:
 1. To use a more efficient printf() – `System_printf()`. This will help us understand the [basics](#) of using Configuro.
 2. TI and our 3rd parties deliver content as “packages” – so, you'll need to know the [proper use of Configuro](#) to consume them (many more examples come later in the workshop).
- ◆ Shown below are: `app.c` and `app_cfg.cfg`
 - ◆ Notice the inclusion of “System.h” – which comes from a “package” delivered by TI. This library allows the use of `System_printf()`:

`app.c`

```
#include <stdio.h>
#include <xdc/runtime/System.h>
#include "app.h"

int main(void)
{
    System_printf ( "Hello World (xdc)\n" );
    printf ( "Hello World %d (std)\n", YYYY);
    return 0;
}
```

`app_cfg.cfg`

```
/* app_cfg.cfg */
var mySys = xdc.useModule('xdc.runtime.System');
```

What does the makefile look like when using Configuro?

Makefile Example – Using Configuro

Makefile Example – Using Configuro

```

#-----
#-- Configuro variables ---
#-----
XDCROOT := $(XDC_INSTALL_DIR)
CONFIGURO := $(XDCROOT)/xs xdc.tools.configuro
export XDCPATH := /home/user/rtsc_primer/examples;$(XDCROOT)
CONFIG := app_cfg
TARGET = gnu.targets.Linux86
PLATFORM = ti.platforms.PC

#-----
#-- .cfg rule --
#-----
%/linker.cmd %/compiler.opt : %.cfg
    $(CONFIGURO) -c $(CC_ROOT) -t $(TARGET) -p $(PLATFORM) -o $(CONFIG) $<

```

- ◆ How are the created files (linker.cmd, compiler.opt) used in the makefile? ...

Makefile Example – Using Configuro

```

#-----
#-- Configuro variables ---
#-----
XDCROOT := $(XDC_INSTALL_DIR)
CONFIGURO := $(XDCROOT)/xs xdc.tools.configuro
export XDCPATH := /home/user/rtsc_primer/examples;$(XDCROOT)
CONFIG := app_cfg
TARGET = gnu.targets.Linux86
PLATFORM = ti.platforms.PC

#-----
#-- .x and .o rules --
#-----
app.x86U : app.o $(CONFIG)/linker.cmd
    $(CC) $(CFLAGS) $^ -o $@

%.o : %.c $(CONFIG)/compiler.opt
    $(CC) $(CFLAGS) $(shell cat $(CONFIG)/compiler.opt) -c $< -o $@

#-----
#-- .cfg rule --
#-----
%/linker.cmd %/compiler.opt : %.cfg
    $(CONFIGURO) -c $(CC_ROOT) -t $(TARGET) -p $(PLATFORM) -o $(CONFIG) $<

```

Looking Inside Compiler.opt & Linker.cmd

compiler.opt / linker.cmd

compiler.opt

```
-I"/home/user/rtsc_primer/examples"  
-I"/home/user/dvSDK_1_30_00_40/xdc_3_05"  
-I"/home/user/dvSDK_1_30_00_40/xdc_3_05/packages"  
-I"/home/user/dvSDK_1_30_00_40/xdc_3_05/packages"  
-I"."  
-Dxdc_target_types__="gnu/targets/std.h"  
-Dxdc_target_name__=MV Arm9  
-Dxdc_cfg_header__="/home/user/lab05d_standard_make/app/debug/mycfg/package/cfg/mycfg_x470MV.h"
```

linker.cmd

```
INPUT(  
  /home/user/lab05d_standard_make/app/debug/mycfg/package/cfg/mycfg_x470MV.o470MV  
  /home/user/rtsc_primer/examples/acme/utls/lib/acme.utls.a470MV  
  /home/user/dvSDK_1_30_00_40/xdc_3_05/packages/gnu/targets/rts470MV/lib/gnu.targets.rts470MV.a470MV  
)
```

Debugging with CCSv5 (Eclipse)

Debugging & Debuggers

Debugging & Debugger's

- ◆ **User Mode Debugging**
 - When debugging user mode programs, you often only want to debug – hence stop – one thread or program.
 - GDB (GNU Debugger) works well for this. (GDB discussed on next slide)
 - Connection is usually Ethernet or serial-port
- ◆ **Kernel Mode Debug**
 - Debugging kernel code requires complete system access.
 - You need KGDB (Ethernet) or scan-based (JTAG) debuggers for this.
- ◆ **A debugger lets you**
 - Pause a program
 - Examine and change variables
 - Step through code
- ◆ **Code Composer Studio (CCSv5)**
 - Latest version of TI's graphical software debugger (i.e. IDE)
 - IDE – integrated development environment: which combines editing, building and debugging into a single tool
 - Built on Eclipse platform; can install on top of standard Eclipse
 - Allows debugging via GDB (Ethernet/serial) or JTAG (scan-based)
 - Free license for GDB debugging

GNU Debugger

GDB : GNU Debugger

- ◆ Open source debugger that is often supplied with the toolchain
- ◆ In this workshop, it's included in the Code Sourcery package
- ◆ GDB has a client/server nature:

GDB Server:

- First start *gdbserver*, specifying connection and app to be debugged
- Server then runs your app, following gdb commands

```

Tera Term
EVM> gdbserver 192.168.1.122:10000 myApp
Listening on port 10000
Remote debugging from host 192.168.1.1
Hello World
                    
```

GDB Server

Runs your app for you, based on the GDB commands you send

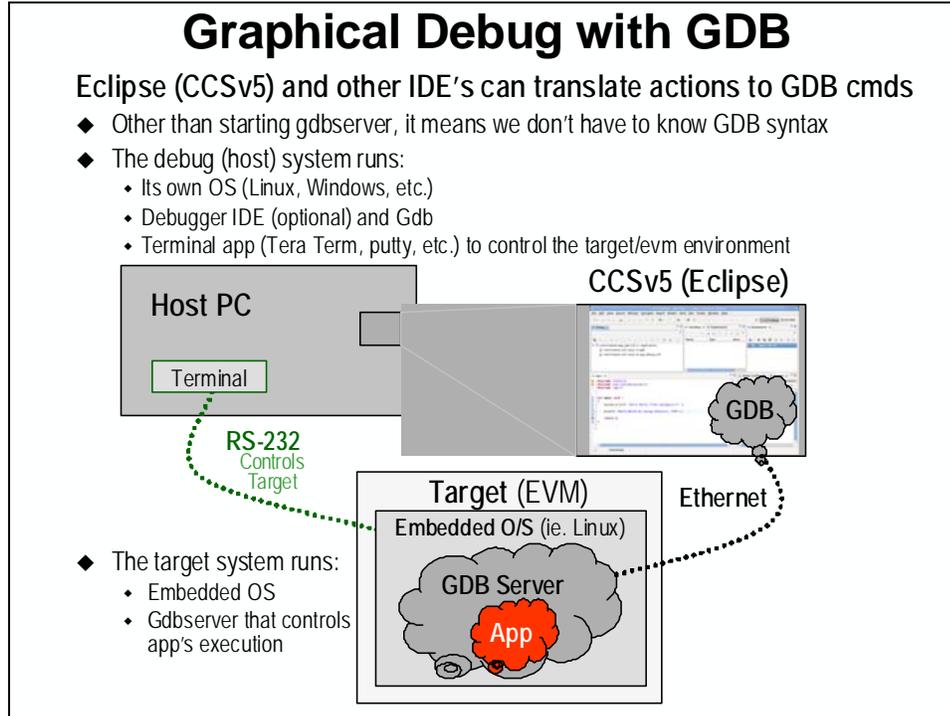
GDB (i.e. debug client)

- Command-line oriented debugger
- Telnet, Bash terminal, IDE

```

(gdb) target remote 192.168.1.122:10000
(gdb) step
(gdb) b 43
(gdb) run
                    
```

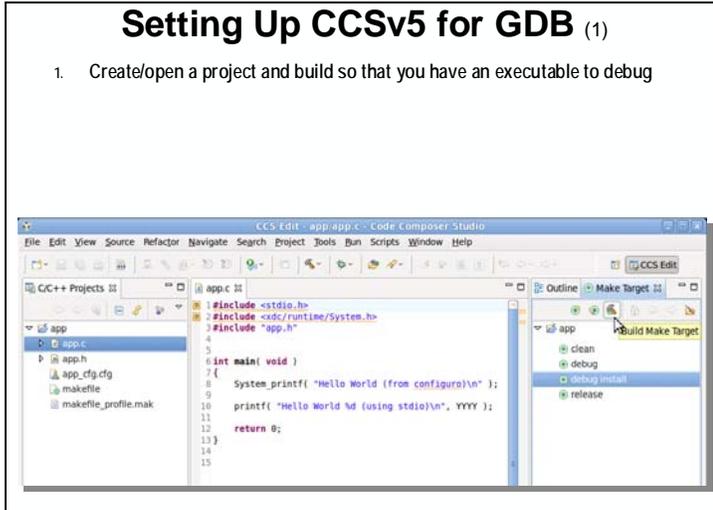
Graphical Debugging with GDB



Setting Up CCSv5 for GDB

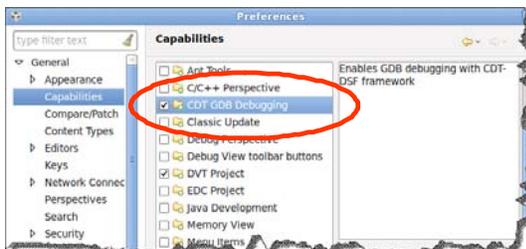
Setting Up CCSv5 for GDB (1)

1. Create/open a project and build so that you have an executable to debug



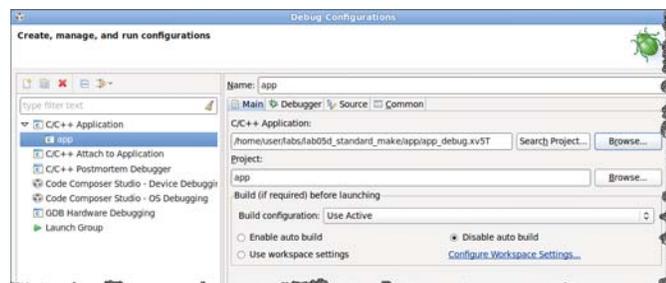
Setting Up CCSv5 for GDB (2)

1. Create/open a project and build so that you have an executable to debug
2. By default, CCSv5 doesn't have remote/GDB debugging turned on



Setting Up CCSv5 for GDB (3)

1. Create/open a project and build so that you have an executable to debug
2. By default, CCSv5 doesn't have remote/GDB debugging turned on
3. Once enabled, create a new "Debug Connection" for GDB



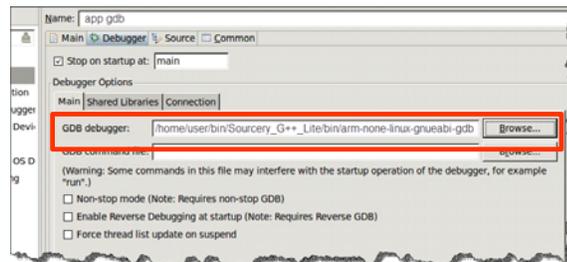
Setting Up CCSv5 for GDB (4)

1. Create/open a project and build so that you have an executable to debug
2. By default, CCSv5 doesn't have remote/GDB debugging turned on
3. Once enabled, create a new "Debug Connection" for GDB
4. Eclipse assumes "native" debugging, so you need to re-configure for remote



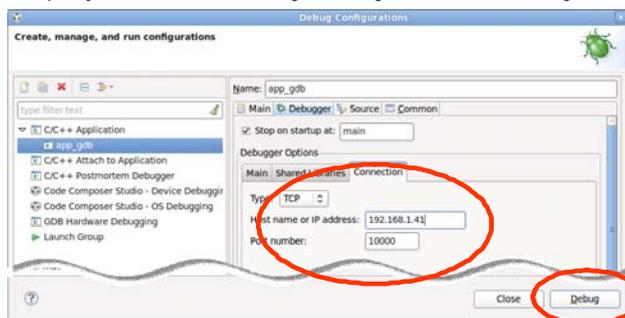
Setting Up CCSv5 for GDB (5)

1. Create/open a project and build so that you have an executable to debug
2. By default, CCSv5 doesn't have remote/GDB debugging turned on
3. Once enabled, create a new "Debug Connection" for GDB
4. Eclipse assumes "native" debugging, so you need to re-configure for remote
5. Point your connection to GDB for the target (remote) processor



Setting Up CCSv5 for GDB (6)

1. Create/open a project and build so that you have an executable to debug
2. By default, CCSv5 doesn't have remote/GDB debugging turned on
3. Once enabled, create a new "Debug Connection" for GDB
4. Eclipse assumes "native" debugging, so you need to re-configure for remote
5. Point your connection to GDB for the target (remote) processor
6. Specify the method of connecting to the target ... then click 'Debug'



Appendix

Here are the answers to the quiz from the chapter material:

Quiz

- ◆ Fill in the blanks below assuming (start with .o rule first):
 - Final "goal" is to build: `main.x86U`
 - Source files are: `main.c`, `main.h`
 - Variables are: `CC` (for `gcc`), `CFLAGS` (for compiler flags)

```

CC := gcc
CFLAGS := -g
# .x rule
main.x86U : main.o
           $(CC) $(CFLAGS) $^ -o $@

# .o rule
main.o : main.c main.h
        $(CC) $(CFLAGS) -c $< -o $@

```

- ◆ Could `$<` be used in the .x rule? What about `$^` in the .o rule?

Yes, the `$<` can be used in the .x rule because there is only ONE dependency. However, the .o rule has two dependencies and would therefore need `$^`.

Answers to the lab quiz questions:

Study the .o rule for a moment. Look at the command that contains `$(CC)`. Just after the `-c` on this line, you should see a `$<` to indicate first dependency only. And, if you use `$^` to indicate both dependencies, `gMake` will fail. Explain:

The .o rule is running the COMPILER. It only knows how to compile a .c file. `Compiler.opt` is NOT a .c file, so we must use `$<` to indicate just the first dependency. If `$^` is used, the compiler will attempt to compile the `compiler.opt` file and crash. However, we need `compiler.opt` contents to show up on the command line, hence the `$(shell cat ...)`.

Now look at the .x rule. Study the command that contains `$(CC)`. Notice that this time we use `$^` to indicate both dependencies. If you use `$<`, `gMake` will not produce the proper output. Explain:

Both dependencies are needed. The `linker.cmd` file is an INPUT to the linker and therefore is required. So, `$^` must be used and `$<` (first dependency only) would fail.

We hate wasting paper, too. Even so, blank pages like this provide necessary padding to keep the odd page out.

Intro to Device Drivers

Introduction

By the end of this chapter, we should have an audio driver setup to capture and/or playback audio data. Along the way, we will also learn about the basic concepts of Linux processes and device drivers.

Outline

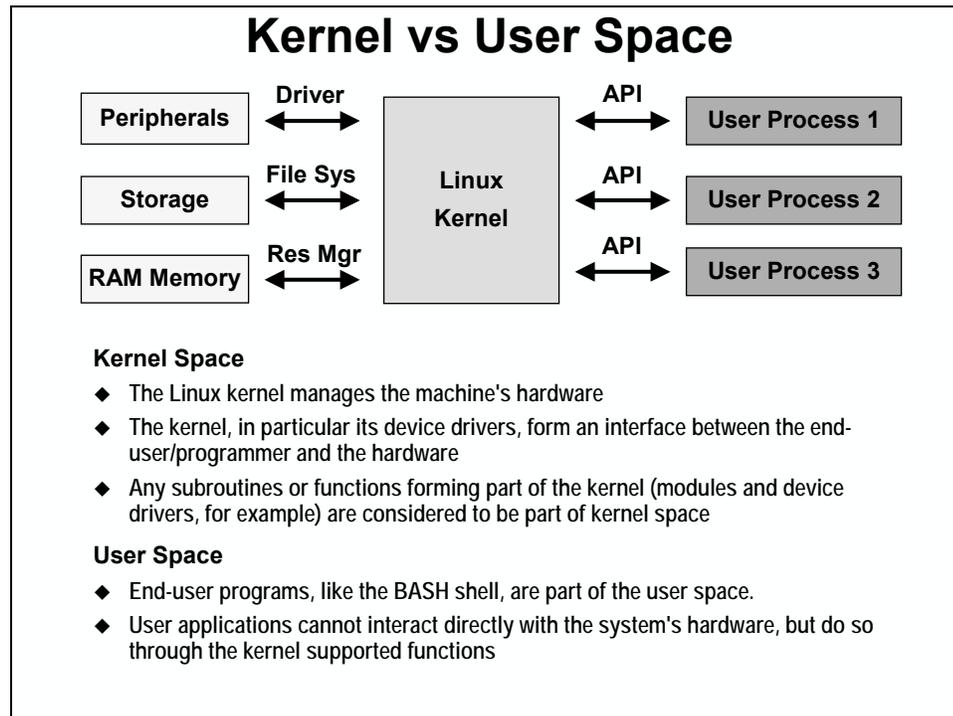
Learning Objectives

- ◆ Differential between Kernel and User Space
- ◆ Understand two methods for adding modules to the Linux kernel
- ◆ Define nodes in Linux and why they are useful
- ◆ Describe why a filesystem is needed
- ◆ Describe the basic file and driver I/O interfaces
- ◆ Describe what DMAI library is used for
- ◆ List the supported modules of ALSA and the DMAI functions to read and write data to it
- ◆ Build an audio pass-thru application – given audio input and output examples

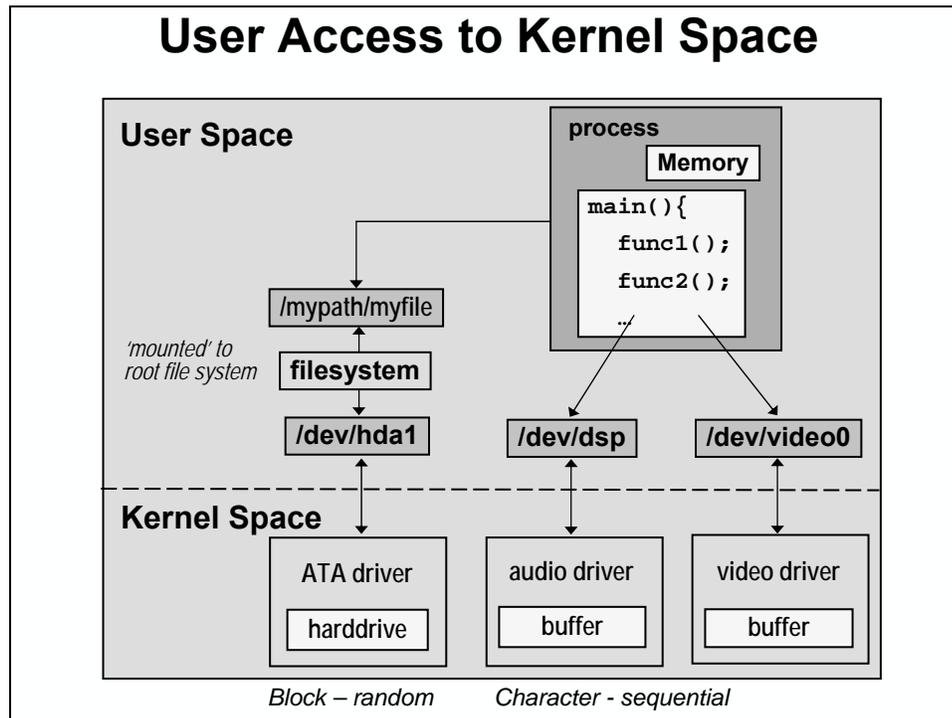
Chapter Topics

Intro to Device Drivers.....	6-1
<i>Kernel vs User Space.....</i>	<i>6-3</i>
<i>Linux Drivers - Basic Concepts</i>	<i>6-4</i>
(1) Load the driver code into the kernel	6-5
(2) Create a virtual file reference (node)	6-6
(3) Mount block drivers using a filesystem.....	6-8
(4) Access resources using open/close/read/write	6-9
<i>Linux Audio Drivers.....</i>	<i>6-10</i>
Open Sound System (OSS).....	6-10
Advanced Linux Sound Architecture (ALSA)	6-11
<i>DMAI with ALSA Driver Example</i>	<i>6-12</i>
<i>Linux Signals.....</i>	<i>6-16</i>

Kernel vs User Space



Linux Drivers - Basic Concepts



Four steps are required for users to access kernel space drivers:

Four Steps to Accessing Drivers

1. Load the driver's code into the kernel (insmod or static)
2. Create a virtual file to reference the driver using mknod
3. Mount block drivers using a filesystem (block drivers only)
4. Access resources using open, read, write and close

(1) Load the driver code into the kernel

Kernel Object Modules

How to add modules to Linux Kernel:

1. Static (built-in)

Linux Kernel

oss
fbdev
httpd

v4l2
nfsd
dsp
ext3

Kernel Module Examples:

fbdev	frame buffer dev
v4l2	video for linux 2
nfsd	network file server dev
dsp	oss digital sound proc.
audio	alsa audio driver

- Linux Kernel source code is broken into individual modules
- Only those parts of the kernel that are needed are built in

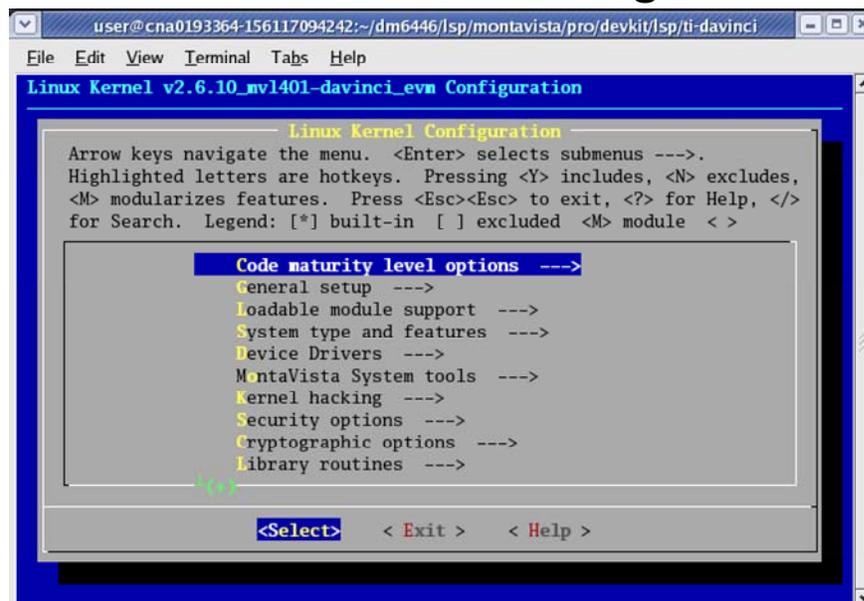
2. Dynamic (insmod)

.ko = kernel object

```
# insmod <mod_name>.ko [mod_properties]
```

- Use insmod (short for insert module) command to dynamically add modules into the kernel
- Keep statically built kernel small (to reduce size or boot-up time), then add functionality later with insmod
- Insmod is also handy when developing kernel modules
- Later we'll insert two modules (cmem.ko, dsplink.ko) using a script: loadmodules.sh

Static Linux Kernel Configuration



```
#> make ARCH=arm CROSS_COMPILE=arm_v5t_le- menuconfig
```

(2) Create a virtual file reference (node)

Linux Driver Registration

```
# mknod <name> <type> <major> <minor>

<name>:  Node name (i.e. virtual file name)
<type>:  b      block
         c      character
<major>: Major number for the driver
<minor>: Minor number for the driver
```

```
Example:  mknod /dev/fb/3 c 29 3
```

```
Usage:    Fd = open("/dev/fb/3", O_RDWR);
```

- ◆ Register new drivers with *mknod* (i.e. Make Node) command.
- ◆ Major number determines which driver is used (the name does not affect which driver is used). Most devices have number assigned by Linux community.
- ◆ Minor number is significant for some drivers; it could denote instance of given driver, or in our example, it refers to a specific buffer in the FBdev driver.

Block and Character Drivers

Block Drivers:

<code>/dev/hda</code>	ATA → harddrive, CF
<code>/dev/ram</code>	external RAM

Character Drivers:

<code>/dev/dsp</code>	sound driver
<code>/dev/video0</code>	v4l2 video driver
<code>/dev/fb/0</code>	frame buffer video driver

- ◆ Block drivers allow out of order access
- ◆ Block devices can be mounted into the filesystem
- ◆ Character drivers are read as streams in a FIFO order
- ◆ Networking drivers are special drivers

Linux Device Registration

- ◆ Linux devices are registered in /dev directory
- ◆ Two ways to view registered devices:
 - cat /proc/devices
 - ls -lsa command (as shown below) to list available drivers

```

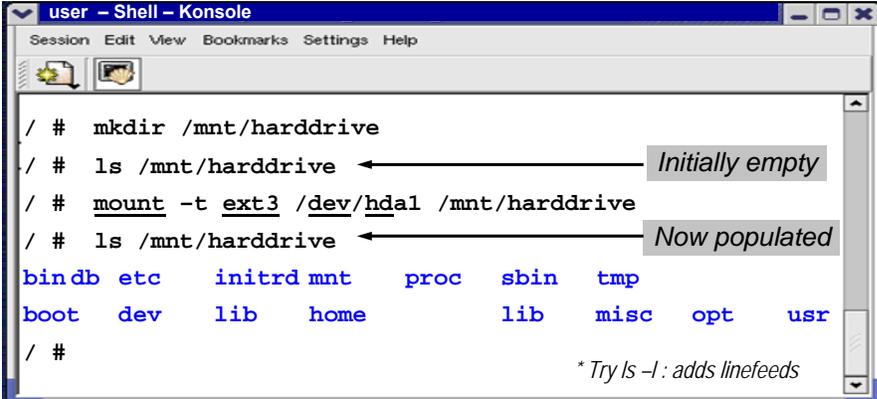
/ # cd /dev
/dev # ls -lsa
0 brw-rw---- 1 root disk 0, 0 Jun 24 2004 /dev/hda
0 crw-rw---- 1 root uucp 4, 64 Mar 8 2004 /dev/ttyS0
0 crw----- 1 user root 14, 3 Jun 24 2004 /dev/dsp
0 crw----- 1 user root 29, 0 Jun 24 2004 /dev/fb/0
0 crw----- 1 user root 29, 1 Jun 24 2004 /dev/fb/1
  
```

Annotations:

- Permissions (user,group,all)
- Major number
- Minor number
- Name
- /dev directory
- block vs char

(3) Mount block drivers using a filesystem

Mounting Block Devices



```

user - Shell - Konsole
Session Edit View Bookmarks Settings Help

/ # mkdir /mnt/harddrive
/ # ls /mnt/harddrive ← Initially empty
/ # mount -t ext3 /dev/hda1 /mnt/harddrive
/ # ls /mnt/harddrive ← Now populated
bindb etc  initrd mnt  proc  sbin  tmp
boot  dev  lib  home          lib  misc  opt  usr
/ #
*Try ls -l: adds linefeeds

```

- ◆ Unlike Windows, there is only one filesystem – therefore you must mount to a mount point (i.e. empty directory) in the root filesystem
- ◆ Mounting a block driver into the filesystem gives access to the files on the device as a new directory
- ◆ Easy manipulation of flash, hard drive, compact flash and other storage media
- ◆ Use mkfs.ext2, mkfs.jffs2, etc. to format a device with a given filesystem
- ◆ The above example shows mounting the DM6446 DVEVM into an NFS root filesystem

The hard disc drive (HDD) on the DaVinci DM6446 DVEVM comes formatted with ext3 file system. This robust filesystem helps to prevent errors when the system is shut down unexpectedly (which happens quite often when developing embedded systems). Other filesystems include:

MontaVista : Supported File Systems Types

Harddrive File systems:

- ext2** Common general-purpose filesystem
- ext3** Journalled filesystem -
Similar to ext2, but more robust against unexpected power-down
- vfat** Windows "File Allocation Table" filesystem

Memory File systems:

- jffs2** Journaling flash filesystem (NOR flash)
- yaffs** yet another flash filesystem (NAND flash)
- ramfs** Filesystem for RAM
- cramfs** Compressed RAM filesystem

Network File systems:

- nfs** Share a remote linux filesystem
- smbfs** Share a remote Windows® filesystem

(4) Access resources using open/close/read/write

Accessing Files

Manipulating files from within user programs is as simple as...

File descriptor / handle
Directory previously mounted
File to open...
Permissions

```

• myFileFd = fopen("/mnt/harddrive/myfile", "rw");
• fread ( aMyBuf, sizeof(int), len, myFileFd );
• fwrite( aMyBuf, sizeof(int), len, myFileFd );
• fclose(myFileFd );

```

Array to read into / write from
size of item
of items
File descriptor / handle

Additionally, use printf and fscanf for more feature-rich file read and write capability

Using Character Device Drivers

Simple drivers use the same format as files:

```

• soundFd = open("/dev/dsp", O_RDWR);
• read ( soundFd, aMyBuf, len );           len always in # of bytes
• write( soundFd, aMyBuf, len );
• close( soundFd );

```

Additionally, drivers use I/O control (ioctl) commands to set driver characteristics

```

• ioctl( soundFd, SNDCTL_DSP_SETFMT, &format);

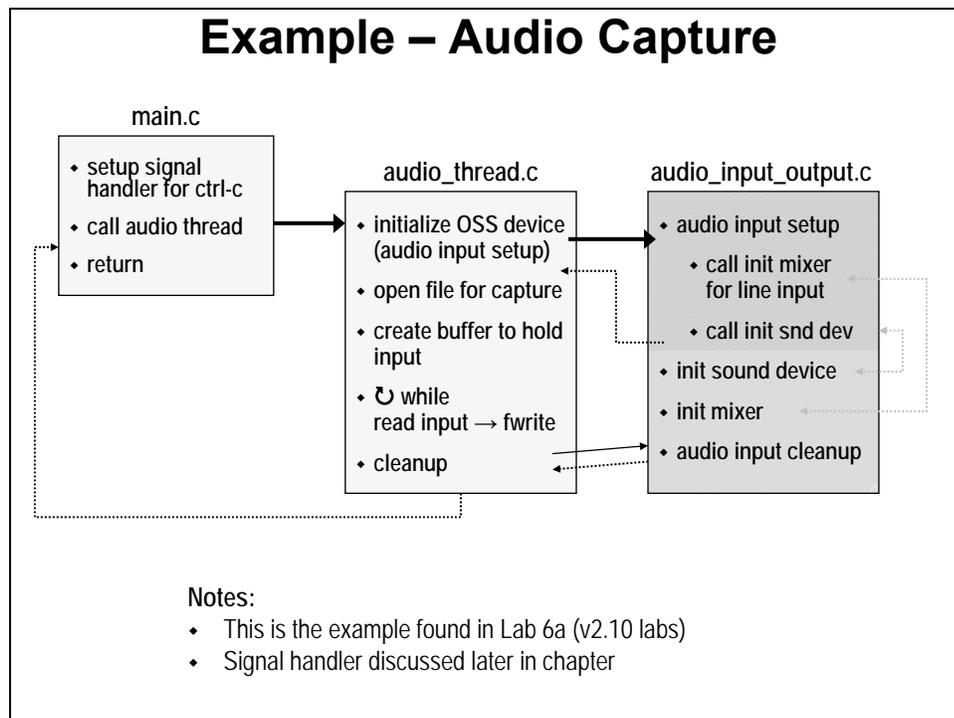
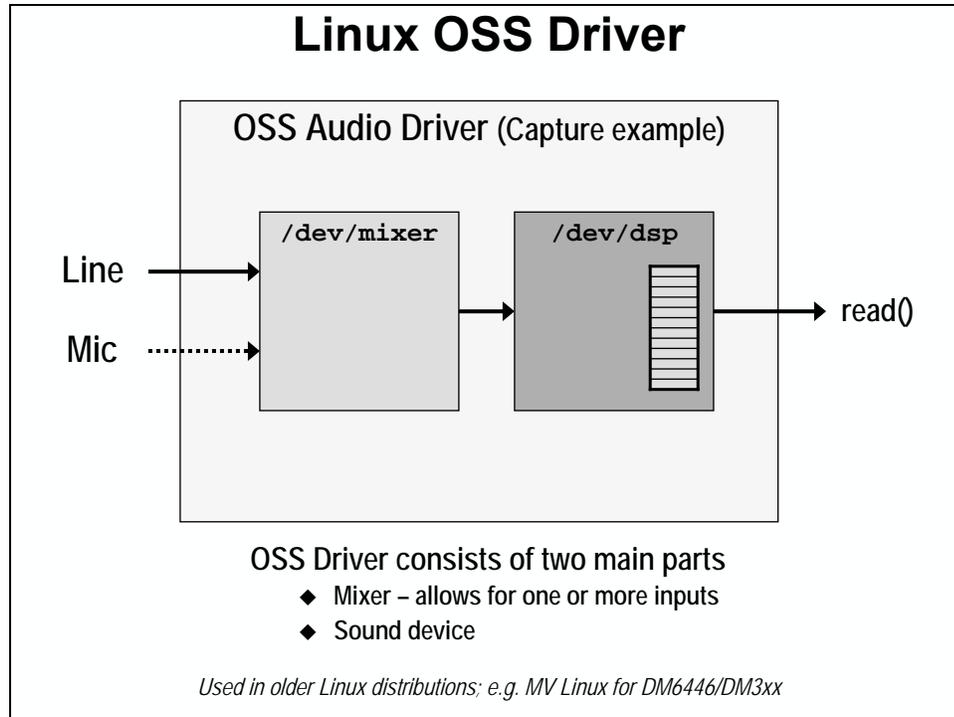
```

Notes:

- ◆ More complex drivers, such as V4L2 and FBDEV video drivers, have special requirements and typically use ioctl commands to perform reads and writes
- ◆ /dev/dsp refers to the "digital sound processing" driver, not the C64x+ DSP

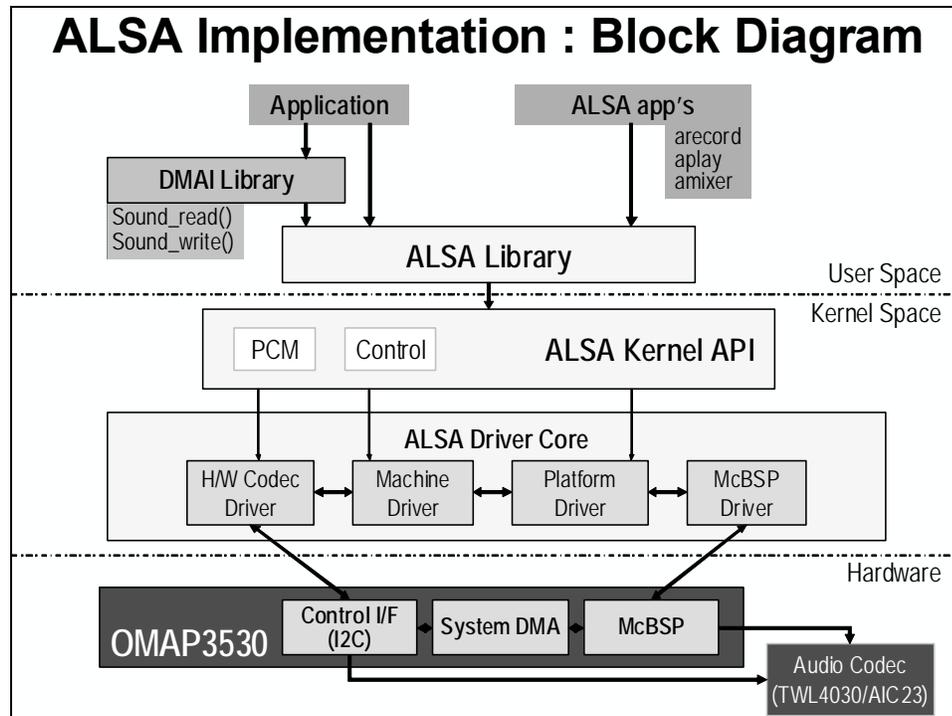
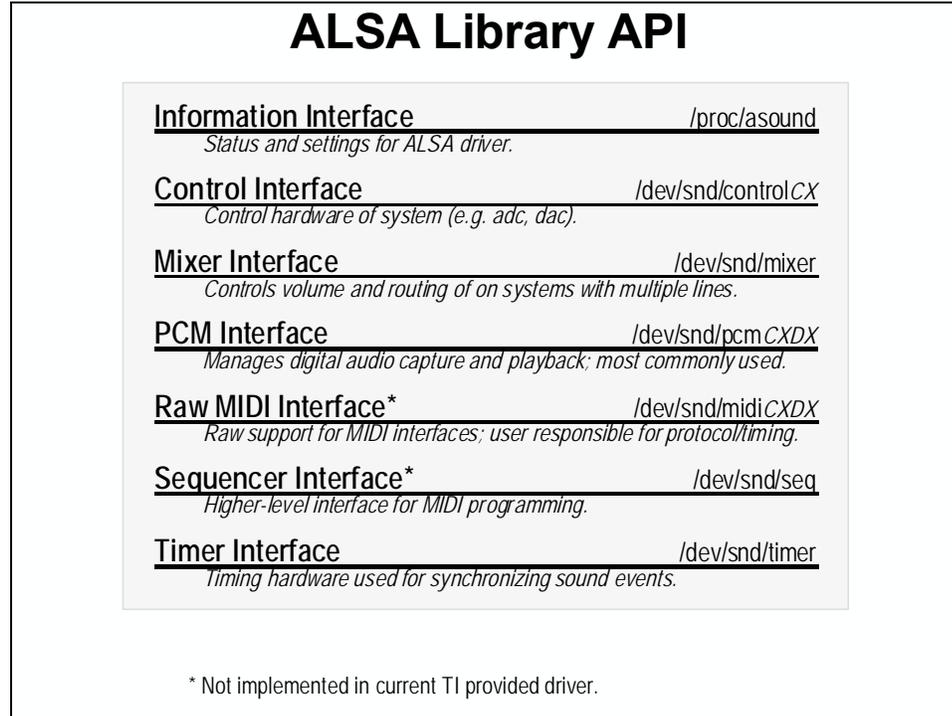
Linux Audio Drivers

Open Sound System (OSS)

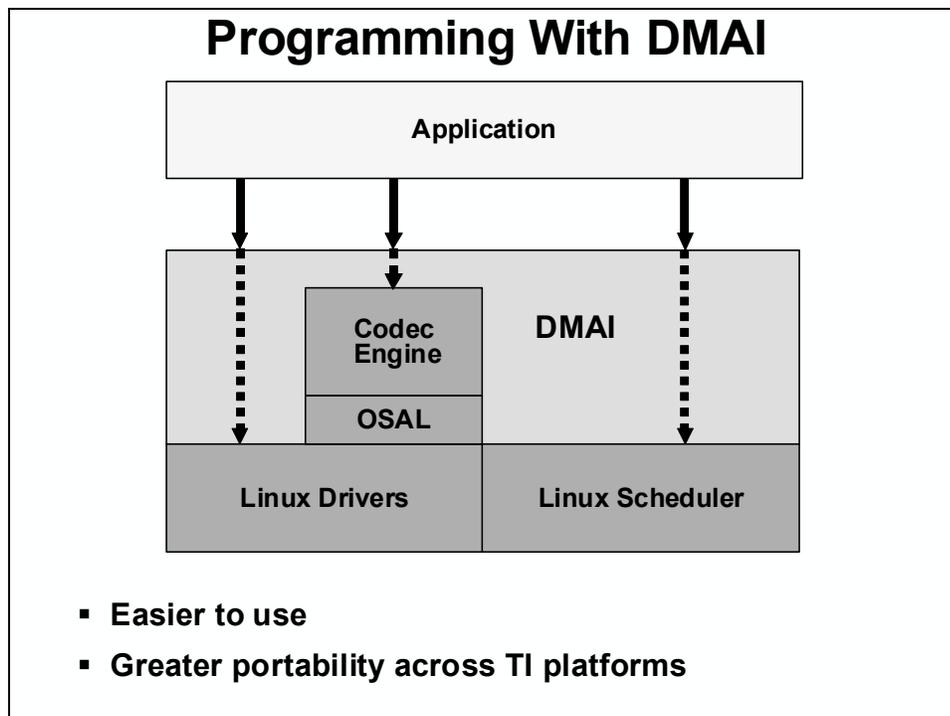
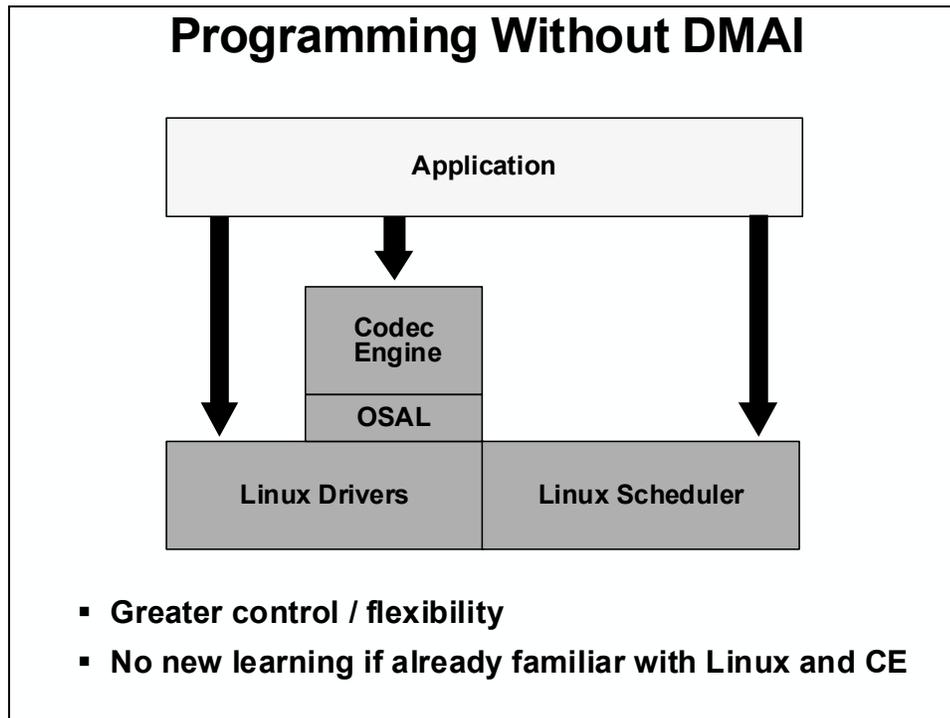


The signal handler in main() is covered a little later.

Advanced Linux Sound Architecture (ALSA)



DMAI with ALSA Driver Example



DMAI Driver Modules

DMAI Module	What it Abstracts
Sound	OSS or ALSA audio driver
Capture	V4L2 video capture
Display	V4L2 or fbdev video display driver
Resize	Hardware resizer driver
ColorSpace	Color spaces
VideoStd	Defines video standards

DMAI Codec Engine Modules

Adec, Aenc:	AUDDEC, AUDENC
Idec, lenc:	IMGDEC, IMGENC
Sdec, Senc:	SPHDEC, SPHENC
Vdec, Venc:	VIDDEC, VIDENC
Adec1, Aenc1:	AUDDEC1, AUDENC1
Idec1, lenc1:	IMGDEC1, IMGENC1
Sdec1, Sdec1:	SPHDEC1, SPHENC1
Vdec2, Venc1:	VIDDEC2, VIDENC1

DMAI Buffer Modules

- ◆ **Buffer** – Reference a virtually or physically contiguous memory array
- ◆ **BufferGfx** – Extends generic Buffer module with graphics attributes
- ◆ **BufTab** – Reference a table of buffers (multiple buffers)
- ◆ **Ccv** – Color space conversion of a buffer

- ◆ In addition to a **memory pointer**, Buffer objects store the memory array's **size**, **bytes used**, and a usage/**ownership** mask
- ◆ BufferGfx objects additionally store x and y **offset**, horizontal and vertical **resolution** and **line length**

DMAI Scheduler Modules

- ◆ **Pause:**
Block a thread until an external condition occurs
- ◆ **Rendezvous:**
Used to synchronize the initialization of multiple threads

Example Creation – Sound Object

```

Sound_Handle hSound;
Sound_Attrs sSoundAttrs = Sound_Attrs_STEREO_DEFAULT
.
.
.
sSoundAttrs.channels      = 2;
sSoundAttrs.mode          = Sound_Mode_FULLDUPLEX;
sSoundAttrs.soundInput    = Sound_Input_LINE;
sSoundAttrs.sampleRate    = 44100;
sSoundAttrs.soundStd      = Sound_Std_ALSA;
sSoundAttrs.bufSize       = 4096;

hSound = Sound_create( &sSoundAttrs );

```

Example Usage – Sound Object

```

Buffer_Attrs bAttrs = Buffer_Attrs_DEFAULT;
Buffer_Handle hBuf   = NULL;
.
.
.
hBuf = Buffer_create( sSoundAttrs.bufSize,&bAttrs );
while( env->quit != TRUE )
{
    /* Read input buffer from ALSA input device */
    Sound_read( hSound, hBuf );

    /* Write output buffer into ALSA output device */
    Sound_write( hSound, hBuf );
}

```

Linux Signals

Linux Signals

A signal is an event generated by Linux in response to some condition, which may cause a process to take some action when it receives the signal.

- ◆ "Raise" indicates the generation of a signal
- ◆ "Catch" indicates the receipt of a signal

Linux Signals

- ◆ A signal may be raised by error conditions such as:
 - ◆ Memory segment violations
 - ◆ Floating-point processor errors
 - ◆ Illegal instructions
- ◆ A signal may be generated by a shell and terminal handlers to cause interrupts
- ◆ A signal may be explicitly sent by one process to another

Signals defined in signal.h

Signal	Value	Action	Comment
SIGHUP	1	Term	Hangup detected on controlling terminal or death of controlling process
SIGINT	2	Term	Interrupt from keyboard
SIGQUIT	3	Core	Quit from keyboard
SIGILL	4	Core	Illegal Instruction
SIGABRT	6	Core	Abort signal from abort(3)
SIGFPE	8	Core	Floating point exception
SIGKILL	9	Term	Kill signal

** Note, this is not a complete list*

Raising / Catching a Signal

◆ Raising:

- ◆ A foreground process can be sent the SIGINT signal by typing Ctrl-C
- ◆ Send to background process using the kill command:

example: `kill -SIGKILL 3021`

◆ Receiving/Catching:

- ◆ If a process receives a signal without first arranging to catch it, the process is terminated
- ◆ SIGKILL (9) cannot be caught, blocked, or ignored

Handling a Signal

A program can handle signals using the signal library function:

```
#include <signal.h>

void (*signal (int sig, void(*func)(int)));
```

integer signal
to be caught or ignored

function to be called
when the specific
signal is received

main.c

```
int main(int argc, char *argv[ ])
{
    int status = EXIT_SUCCESS;
    void *audioThreadReturn;

    /* Set the signal callback for Ctrl-C */
    signal(SIGINT, signal_handler);

    /* Call audio thread function */
    audioThreadReturn = audio_thread_fxn((void *) &audio_env);

    if(audioThreadReturn == AUDIO_THREAD_FAILURE){
        DBG("audio thread exited with FAILURE status\n");
        status = EXIT_FAILURE;
    }
    else
        DBG("audio thread exited with SUCCESS status\n");

    exit(status);
}

/* Callback called when SIGINT is sent to the process (Ctrl-C). */
void signal_handler(int sig)
{
    DBG("Ctrl-C pressed, cleaning up and exiting..\n");
    audio_env.quit = 1; // used as while loop condition
}
```

• setup signal handler for ctrl-c

• call audio thread

• return

Video Driver Details

Introduction

This chapter explores the video system drivers. The labs demonstrate the Linux V4L2 and FBdev drivers as well as basic file I/O, through four small applications: on-screen display (OSD), video recorder, video player, and video loop-thru (video-capture copied to video-display).

Outline

- V4L2 Capture Driver
 - Using mmap
 - V4L2 Coding
- FBDev Driver
 - Video Planes
 - FBDev Coding
- DMAI Buffers
- DMAI Display Driver
- Video Display Boot Arguments
- Lab Exercise
 - Video OSD
 - Video Recording
 - Video Playback
 - Video Loophtru

Chapter Topics

Video Driver Details.....	7-1
<i>v4L2 Capture Driver.....</i>	<i>7-3</i>
Overview	7-3
How v4L2 Works	7-4
Using mmap.....	7-6
(Optional) v4L2 Coding	7-8
<i>FBdev – Display Driver.....</i>	<i>7-9</i>
Overview	7-9
Video Planes.....	7-9
Using Pan feature for Ping-Pong Buffers	7-11
FBdev Coding.....	7-12
<i>DMAI for Video Drivers</i>	<i>7-13</i>
DMAI Buffer Module.....	7-13
DMAI Display Module.....	7-15
<i>Setting Video Display Properties.....</i>	<i>7-17</i>
DVSDK 2.xx (DM6446)	7-17

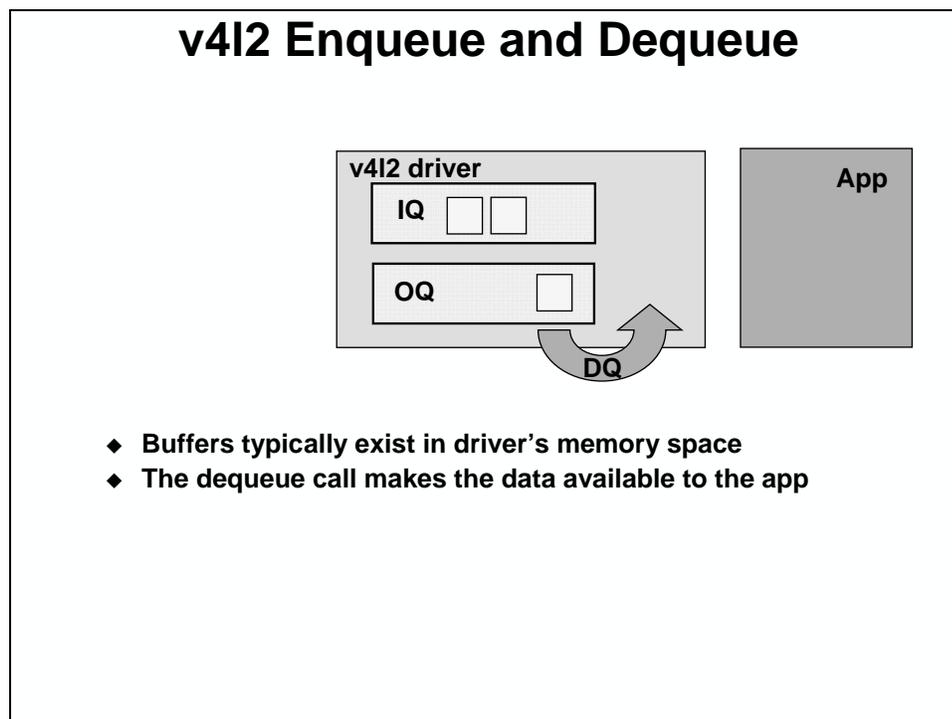
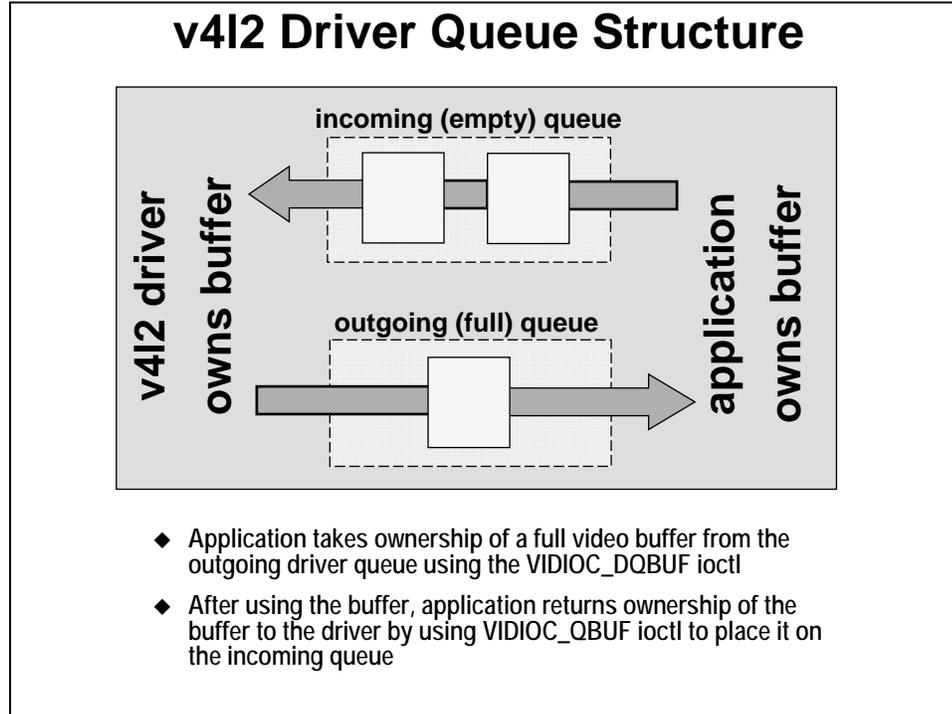
v4L2 Capture Driver

Overview

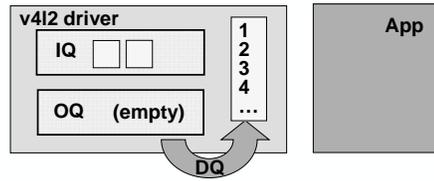
v4l2 Driver Overview

- ◆ v4l2 is a standard Linux video driver used in many linux systems
- ◆ Supports video input and output
 - In this workshop we use it for video input only
- ◆ Device node
 - Node name: `/dev/video0`
 - Uses major number 81
- ◆ v4l2 spec: <http://bytexsex.org/v4l>
- ◆ Driver source location:
`.../lsp/ti-davinci/drivers/media/davinci_vpfe.c`

How v4L2 Works

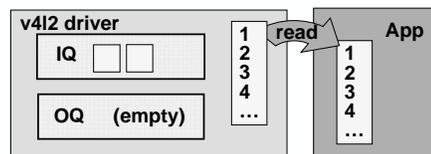


v4I2 Enqueue and Dequeue



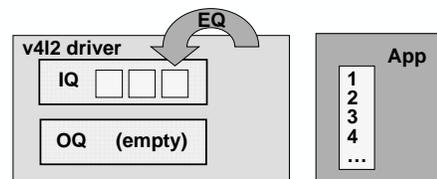
- ◆ Buffers typically exist in driver's memory space
- ◆ The dequeue call makes the buffer available to the app
- ◆ Even after DQ, buffers still exist in the driver's memory space but not the application's

v4I2 Enqueue and Dequeue



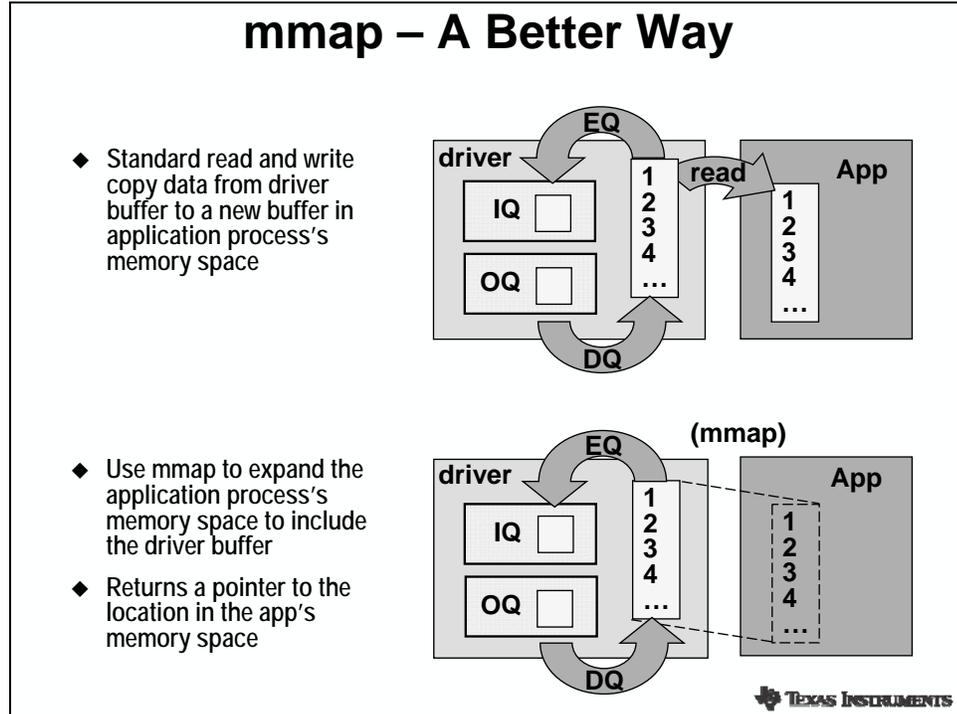
- ◆ Buffers typically exist in driver's memory space
- ◆ The dequeue call makes the buffer available to the app
- ◆ Even after DQ, buffers still exist in the driver's memory space but not the application's
- ◆ read and write operations copy buffers from driver's memory space to app's or vice-versa

v4I2 Enqueue and Dequeue

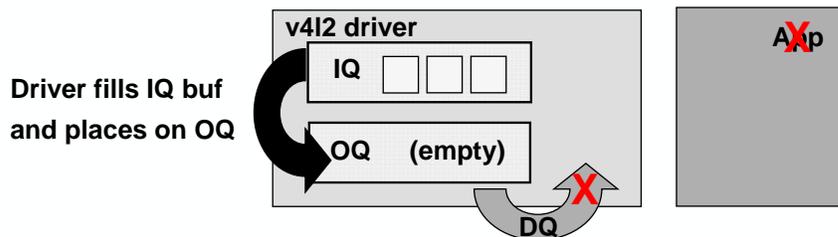


- ◆ Buffers typically exist in driver's memory space
- ◆ These buffers exist in the driver's memory space but not the application's
- ◆ read and write operations copy buffers from driver's memory space to app's or vice-versa

Using mmap

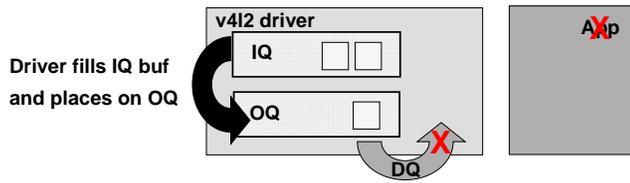


v4l2 Queue Synchronization



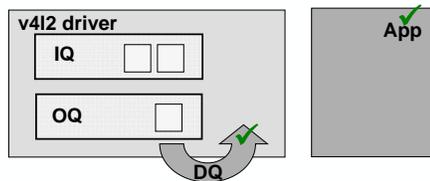
- ◆ The VIDIOC_DQBUF ioctl blocks the thread's execution waits if until a buffer is available on the output queue
- ◆ When driver adds a new, full buffer to the output queue, the application process is released
- ◆ Dequeue call completes and application resumes with the following set of commands

v4l2 Queue Synchronization



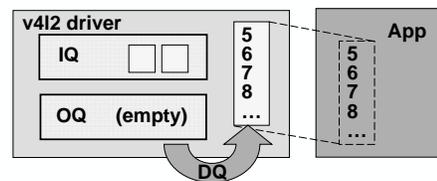
- ◆ The VIDIOC_DQBUF ioctl blocks the thread's execution waits if until a buffer is available on the output queue
- ◆ When driver adds a new, full buffer to the output queue, the application process is released
- ◆ Dequeue call completes and application resumes with the following set of commands

v4l2 Queue Synchronization



- ◆ The VIDIOC_DQBUF ioctl blocks the thread's execution waits if until a buffer is available on the output queue
- ◆ When driver adds a new, full buffer to the output queue, the application process is released
- ◆ Dequeue call completes and application resumes with the following set of commands

v4l2 Synchronization



- ◆ The VIDIOC_DQBUF ioctl blocks the thread's execution waits if until a buffer is available on the output queue
- ◆ When driver adds a new, full buffer to the output queue, the application process is released
- ◆ Dequeue call completes and application resumes with the following set of commands

(Optional) v4L2 Coding

v4I2 Buffer Passing Procedure

```
while(cond){
    ioctl(v4l2_input_fd, VIDIOC_DQBUF, &buf);
    bufPtr = mmap(NULL,                               // "start" address (usually 0)
                  buf.length,                          // length
                  PROT_READ | PROT_WRITE,             // allowed use for map'd memory
                  MAP_SHARED,                          // allow sharing of mapped bufs
                  v4l2_input_fd,                       // driver/file descriptor
                  buf.m.offset);                       // offset requested from "start"
    doSomething(bufPtr, buf.length, ...);
    munmap(bufPtr, buf.length);
    ioctl(v4l2_input_fd, VIDIOC_QBUF, &buf);
}
```

- ◆ A simple flow would be: (1) DQBUF the buffer, (2) map it into user space, (3) use the buffer, (4) unmap it, (5) put it back on the driver's queue
- ◆ More efficient code would map each driver buffer once during initialization, instead of mapping and unmapping within the loop
- ◆ Alternatively, later versions of the driver allow you to create the buffer in 'user' space and pass it to the driver

Commonly Used v4I2 ioctl's

Data Structures

```
struct v4l2_requestbuffers req;
req.count;           // how many buffers to request
req.type;           // capture, output, overlay
req.memory;         // mmap, userptr, overlay

struct v4l2_buffer buf;
buf.index;          // which driver buffer
buf.type;           // matches req.type
buf.memory;         // matches req.memory
buf.m.offset;       // location of buffer in driver mem
```

Request the driver allocate a new buffer

```
ioctl(fd, VIDIOC_REQBUFS, &req);
```

Get information on a driver buffer

```
ioctl(fd, VIDIOC_QUERYBUF, &buf);
```

Enqueue and Dequeue buffers to/from driver

```
ioctl(fd, VIDIOC_QBUF, &buf);
Ioctl(fd, VIDIOC_DQBUF, &buf);
```

FBdev – Display Driver

Overview

FBdev Driver Overview

- ◆ FBdev is a standard Linux video output driver used in many linux systems
- ◆ Can be used to map the frame buffer of a display device into user space
- ◆ Device nodes have major number 29
- ◆ Device nodes have a minor number x
- ◆ Uses /dev/fb/x node naming convention

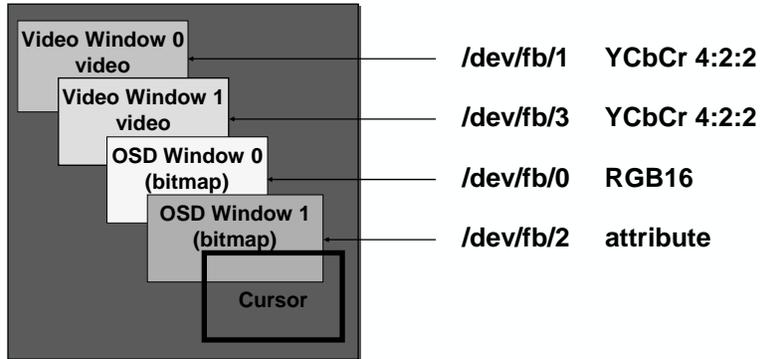
Video Planes

Multiple Video/OSD Windows

The diagram illustrates the layering of video and OSD windows. It shows a 3D perspective of a display area. At the bottom is the 'VIDEO AREA' showing a green field. Above it is the 'BACKGROUND' layer. On top of the background is an 'APPLICATION WINDOW' containing text and graphics. The entire structure is divided into two horizontal sections: the 'UPPER LAYER' at the top and the 'LOWER LAYER' at the bottom.

Source: DirectFBOverview.pdf

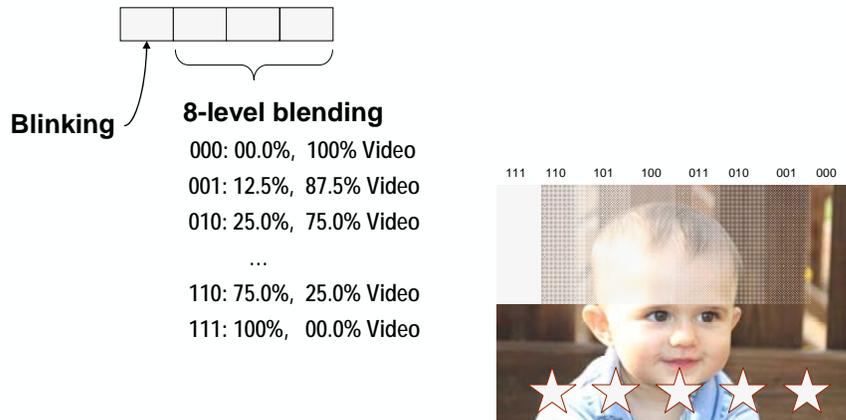
Video Port Back End Features



- ◆ Two video windows for picture-in-picture
- ◆ Two OSD windows or one OSD window + attribute window
- ◆ /dev/fb2 attribute mode provides pixel-level alpha blending

OSD Attribute Window

- ◆ Allows Pixel by Pixel Blending of OSD0 and Video Windows
- ◆ Uses a 4-bit, Bit-map Window



Using Pan feature for Ping-Pong Buffers

For Each /dev/fb/x Video Plane

$xres$ $yres$ $xres_virtual$ $yres_virtual$
 $xoffset$ $yoffset$

- ◆ Named FBdev because it gives direct access to the display device's frame buffer
- ◆ FBIOPAN_DISPLAY allows users to pan the active display region within a virtual display buffer

Ping-pong Buffers with FBdev

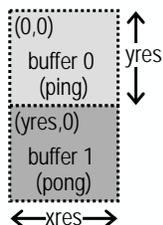
$offset = (0,0)$ $buffer0$ (ping)
 $offset = (yres,0)$ $buffer1$ (pong)
 $xres_virtual$ $yres_virtual$

- ◆ FBdev has no video buffer queue (provides direct access to the display device's frame buffer)
- ◆ Use FBIOPAN_DISPLAY to switch between 2 or more buffers in the virtual buffer space
- ◆ Use FBIO_WAITFORVSYNC to block process until current buffer scan completes, then switch.

Ping-pong Buffers with FBdev

$offset = (0,0)$ $buffer0$ (ping)
 $offset = (yres,0)$ $buffer1$ (pong)
 $xres_virtual$ $yres_virtual$

FBdev Coding



Buffer Synchronization

```
pVirtualBuf = mmap(NULL, display_size * NUM_BUFS,
                    PROT_READ | PROT_WRITE,
                    MAP_SHARED,
                    FbdevFd, 0);

while(cond){
    // map next frame from virtual buffer
    display_index = (display_index + 1) % NUM_BUFS;
    ioctl(pFbdevFd, FBIOGET_VSCREENINFO, &vinfo);
    vinfo.yoffset = vinfo.yres * display_index;

    // write pixels into next video frame
    genPicture(...);

    // switch to next frame
    ioctl(pFbdevFd, FBIOPAN_DISPLAY, &vinfo);

    // wait for current frame to complete
    ioctl(pFbdevFd, FBIO_WAITFORVSYNC, NULL);
}
```

Commonly Used FBdev ioctl

Data Structures

```
struct fb_fix_screeninfo myFixScreenInfo;
myFixScreenInfo.smem_len;    // length of framebuffer

struct fb_var_screeninfo myVarScreenInfo;
myVarScreenInfo.xres;       // visible pic resolution
myVarScreenInfo.xres_virtual; // virtual pic resolution
myVarScreenInfo.xoffset;    // from virtual to vis
```

Get or put variable screen information

```
ioctl(fd, FBIOGET_VSCREENINFO, &myVarScreenInfo);
ioctl(fd, FBIOPUT_VSCREENINFO, &myVarScreenInfo);
```

Get fixed screen information

```
ioctl(fd, FBIOGET_FSCREENINFO, &myFixScreenInfo);
```

We use Pan to switch output buffers

```
ioctl(fd, FBIOPAN_DISPLAY, &myVarScreenInfo);
```

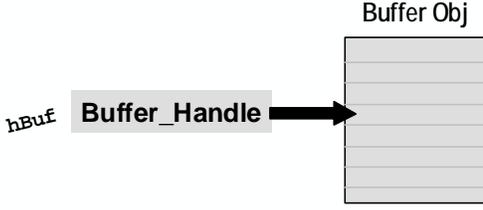
After writing buffer and pan_display, wait for current to finish

```
ioctl(fd, FBIO_WAITFORVSYNC, NULL); // arg 3 is not used
```

DMAI for Video Drivers

DMAI Buffer Module

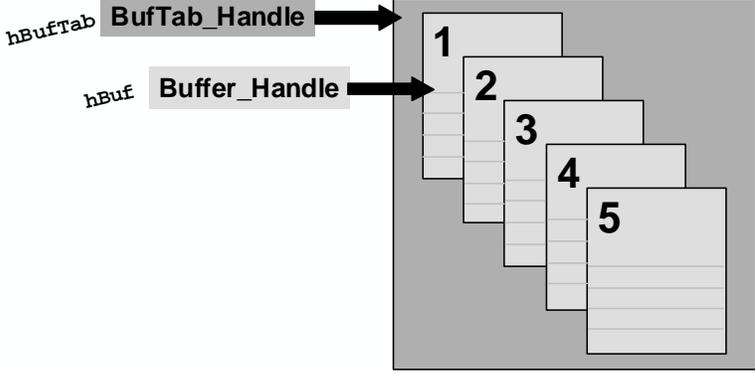
DMAI : Buffers and BufferGfx



The diagram illustrates the relationship between a Buffer Handle and a Buffer Object. On the left, a box labeled 'hBuf' contains the text 'Buffer_Handle'. A thick black arrow points from this box to a vertical stack of six rectangular boxes on the right, labeled 'Buffer Obj'.

- ◆ DMAI includes a Buffer module which can create/delete buffers; a handle is returned by the *Buffer_create()* function
- ◆ *Buffers* objects include: length, size, attributes
- ◆ Basic *Buffer* attributes include: inUse, type, memory allo info
- ◆ Even further, BufferGfx (graphics buffer) extends the basic *Buffer* to include additional video/graphics metadata

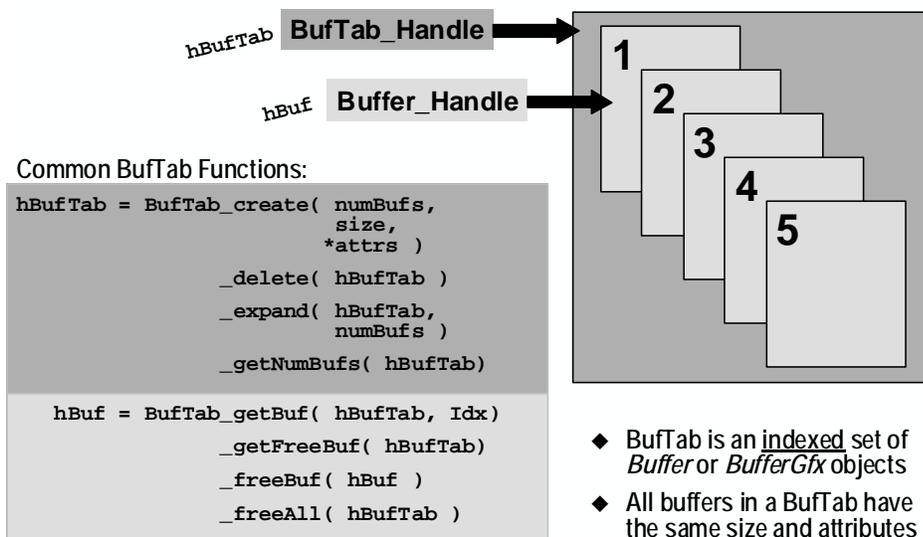
DMAI : BufTables



The diagram illustrates the structure of a BufTable. On the left, a box labeled 'hBufTab' contains the text 'BufTab_Handle'. A thick black arrow points from this box to a large gray rectangular area on the right. Inside this area, five overlapping rectangular boxes are stacked vertically, labeled '1', '2', '3', '4', and '5'. Below the '1' box, a box labeled 'hBuf' contains the text 'Buffer_Handle', with a thick black arrow pointing to the '1' box.

- ◆ BufTab is an indexed set of *Buffer* or *BufferGfx* objects
- ◆ *BufTab_create()* can allocate an array of buffers (returning a handle)
- ◆ Check out a buffer: *_get()*, *_getFreeBuf()*
- ◆ Free buffers using: *_free()*, *_freeAll()*

Buffer Tables (BufTab)



Simple BufTab Example

```

#include <xdc/std.h>
#include <ti/sdo/dmai/Dmai.h>
#include <ti/sdo/dmai/BufTab.h>

BufTab_Handle hBufTab;
Buffer_Attrs bAttrs = Buffer_Attrs_DEFAULT;
Buffer_Handle hBuf;

Dmai_init(); // Always call b4 using any DMAI module

// Allocate an array of 10 basic Buffers, size 1KB
hBufTab = BufTab_create( 10, 1024, &bAttrs );

// Use the Buffers in the BufTab
hBuf = BufTab_getFreeBuf( hBufTab );
dosomething( hBuf );
BufTab_freeBuf( hBuf );

// Delete the buffer table
BufTab_delete( hBufTab );

```

DMAI Display Module

Create Display Obj Example (Driver Buffers)

```
#define          NUM_DISPLAY_BUFS  3
Display_Attrs  sDAttrs  = Display_Attrs_O3530_VID_DEFAULT;
Display_Handle hDisplay = NULL;

:
:
sDAttrs.videoStd = Capture_getVideoStd(hCapture);
sDAttrs.numBufs  = NUM_DISPLAY_BUFS;
sDAttrs.rotation = 90;

hDisplay = Display_create( hBufTabDisplay, &sDAttrs);
```

- Video display driver supports two types of buffers – *Driver* or *User* created

Create Display Obj Example (User Bufs)

```
#define          NUM_DISPLAY_BUFS  3
Display_Attrs  sDAttrs  = Display_Attrs_O3530_VID_DEFAULT;
Display_Handle hDisplay = NULL;
BufferGfx_Attrs gfxAttrs = BufferGfx_Attrs_DEFAULT;
BufTab_Handle  hBufTabDisplay = NULL;

:
:
sDAttrs.videoStd = Capture_getVideoStd(hCapture);
sDAttrs.numBufs  = NUM_DISPLAY_BUFS;
sDAttrs.rotation = 90;

hBufTabDisplay = BufTab_create( NUM_DISPLAY_BUFS, bufSize,
                               BufferGfx_getBufferAttrs( &gfxAttrs ));

hDisplay = Display_create( hBufTabDisplay, &sDAttrs);
```

- Make sure the buffers are contiguous in memory – use either CMEM or DMAI's BufTab to create them.
- *User Space Buffers* are passed to the driver during open(). In our example, this is done by DMAI's Display_create() function.

Using Display Obj Example

```

BufTab_Handle  hDispBuff = NULL;

:
while( env->quit != TRUE )
{
    /* Acquire empty buffer from display device */
    Display_get( hDisplay, &hDispBuf );

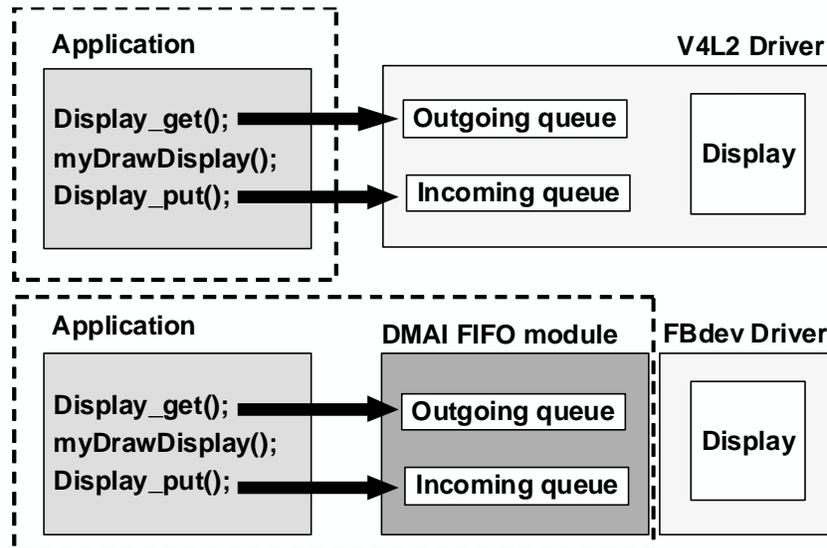
    /* Following is not a real function, placeholder */
    myDrawDisplay( Buffer_getUserPtr( hDispBuf ),
                  Buffer_getNumBytesUsed( hDispBuf ) );

    /* Pass output buffer to display device */
    Display_put( hDisplay, hDispBuf );
}

```

- Once created, using the display consists of a simple get() and put() calls
- The myDrawDisplay() function is pseudo code to just show something happening to the display buffer between the get and put

Display Driver Abstraction



- DMAI provides common i/f (`Display_get()`, `Display_put()`) for v4l2 and fbDev.
- FIFO module adds buffering for fbDev, so that they have a common i/f.

Setting Video Display Properties

DVSDK 2.xx (DM6446)

Video Display Driver Properties (1)

- ◆ Set resolution, bit-depth, size-of-buffer, and offset for FBdev's OSD and video drivers
- ◆ Video display properties can be set in one of 3 ways:

1. Linux boot arguments (examine bootargs as set in Lab 4)

ex: video=davincifb:osd0=720x480x16,1350K:osd1=720x480,1350K:vid0=720x480,2025K:...

2. From the command-line shell

example: fbset -fb /dev/fb/1 -xres 720 -yres 480 -vxres 720 -vyres 1440 -depth 16 -nonstd 1

3. In your program (look in video_output.c and video_osd.c)

examples: **vidInfo.xres** = 720;
 attrInfo.xres_virtual = 768;

* Note: Default properties are OK for all settings except xres_virtual, which now defaults to 720, but must be set to 768 for NTSC/PAL. (Small migration issue, as the previous driver defaulted to 768.)

Video Display Driver Properties (2)

The user can choose the following, as well:

- ◆ Display mode: NTSC, PAL
- ◆ Display type: Composite, S-video, Component

Set mode and type with one of three ways:

1. Linux boot arguments (examine bootargs set in Lab 4)

example: davinci_enc_mgr.ch0_output=COMPOSITE
 davinci_enc_mgr.ch0_mode=NTSC:...

2. Command-line (modifying /sysfs)

example: Write appropriate value to text file found at either:
 /sys/class/davinci_display/ch0/output
 /sys/class/davinci_display/ch0/mode

3. In your program

ex: fd = fopen("/sys/class/davinci_display/ch0/output", O_RDWR);
 write(fd, "COMPOSITE", 10)

Page left intentionally blank.

Multi-Threaded Applications

Introduction

In this chapter an introduction to Linux will be provided. Those currently using Linux should already be familiar with the information presented here. Anyone new to Linux should find this foundational information helpful in providing context for concepts that come in later chapters.

Learning Objectives

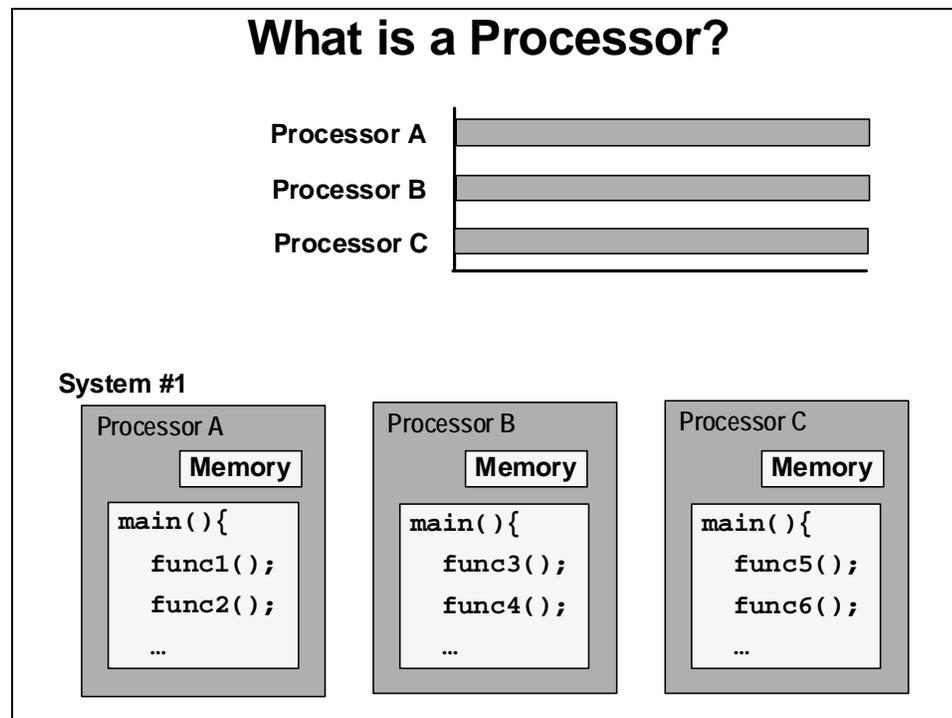
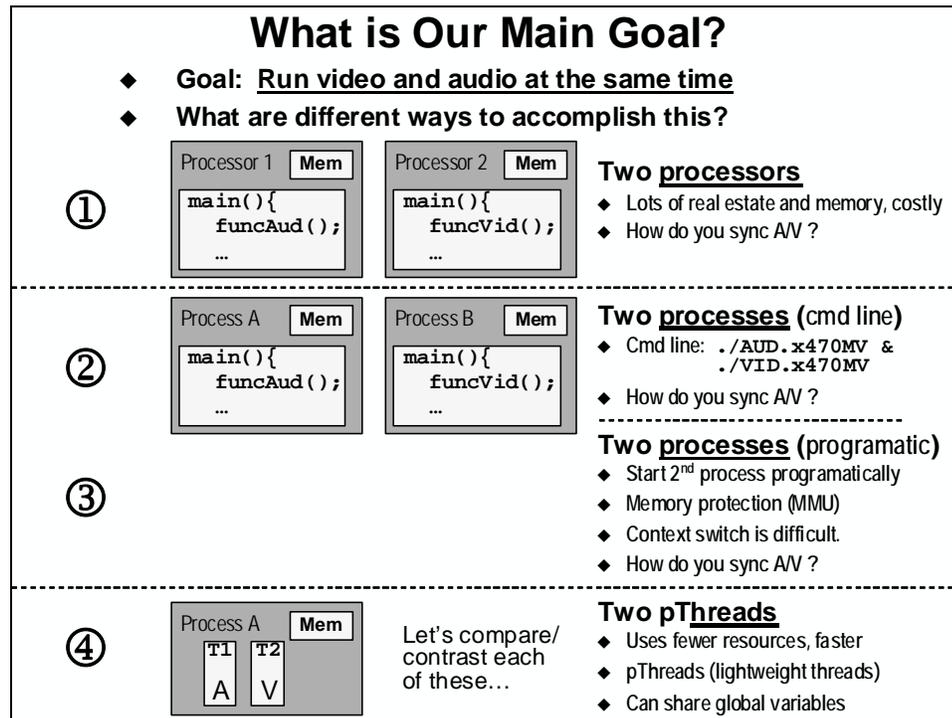
At the conclusion of this chapter, you should be familiar with the basics of:

- What are Linux processes – and how are they like processors
- What are Linux threads – and how do they differ from processes
- How does Linux implement thread synchronization
- Using Linux real-time thread scheduling

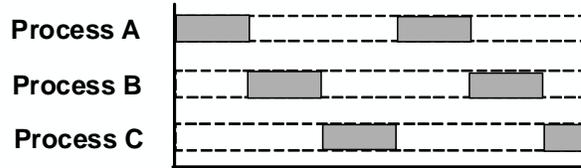
Chapter Topics

Multi-Threaded Applications	8-1
<i>Linux Processes</i>	8-3
<i>Linux Threads</i>	8-8
<i>Thread Synchronization</i>	8-12
<i>Using Real-Time Threads</i>	8-15

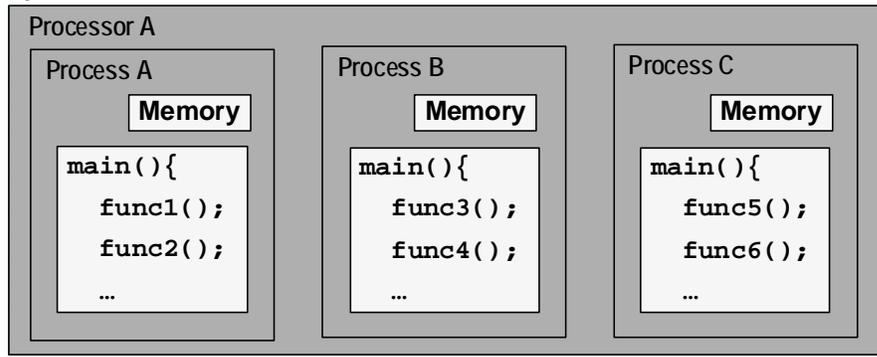
Linux Processes



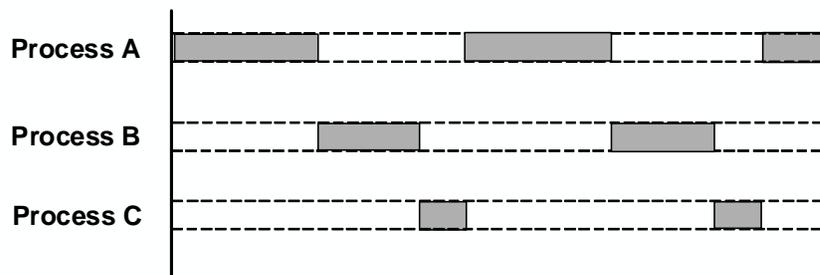
What is a Process?



System #2



Linux Time-Slice Scheduler



- ◆ Processes are time-sliced with more time given to lower niceness
- ◆ Linux dynamically modifies processes' time slice according to process behavior
- ◆ Processes which block are rewarded with greater percentage of time slice total

Scheduling Methodologies

Time-Slicing with Blocking

Scheduler shares processor run time between all threads with greater time for higher priority

- ✓ No threads completely starve
- ✓ Corrects for non-"good citizen" threads
- ✗ Can't guarantee processor cycles even to highest priority threads.
- ✗ More context switching overhead

Linux Default

Thread Blocking Only

Lower priority threads won't run unless higher priority threads block (i.e. pause)

- ✗ Requires "good citizen" threads
- ✗ Low priority threads may starve
- ✓ Lower priority threads never break high priority threads
- ✓ Lower context-switch overhead

BIOS

- Notes:**
- ◆ Linux threads provide extensions for real-time thread behavior as well; however, time-slicing is the default
 - ◆ Similarly, you can setup BIOS to time-slice threads (TSK's), but this is not the default for BIOS (i.e. real-time) systems

The Usefulness of Processes

Option 1: Audio and Video in a single Process

```
// audio_video.c
// handles audio and video in
// a single application

int main(int argc, char *argv[])
{
    while(condition == TRUE){
        callAudioFxn();
        callVideoFxn();
    }
}
```

Option 2: Audio and Video in separate Processes

```
// audio.c, handles audio only

int main(int argc, char *argv[]) {
    while(condition == TRUE)
        callAudioFxn();
}
```

```
// video.c, handles video only

int main(int argc, char *argv[]) {
    while(condition == TRUE)
        callVideoFxn();
}
```

Splitting into two processes is helpful if:

1. audio and video occur at different rates
2. audio and video should be prioritized differently
3. multiple channels of audio or video might be required (modularity)
4. memory protection between audio and video is desired

Terminal Commands for Processes

# <code>ps</code>	Lists currently running user processes
# <code>ps -e</code>	Lists all processes
# <code>top</code>	Ranks processes in order of CPU usage
# <code>kill <pid></code>	Ends a running process
# <code>renice +5 -p <pid></code>	Changes time-slice ranking of a process (range +/- 20)

Launching a Process – Terminal

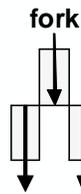
```

user:~/workdir/bootcampstarter/lab_soln/lab6_soln - Shell - Konsole
Session Edit View Bookmarks Settings Help
root@92.168.10.20:/mnt/bootcamp/lab_soln/lab6_soln/release# ./lab6_soln &
[1] 979
root@92.168.10.20:/mnt/bootcamp/lab_soln/lab6_soln/release# /dev/fb/0 initiali
ied with resolution 720x480 and 16 bpp.
root@92.168.10.20:/mnt/bootcamp/lab_soln/lab6_soln/release# ps
  PID TTY          TIME CMD
  975 pts/0    00:00:00 bash
  979 pts/0    00:00:09 lab6_soln
  980 pts/0    00:00:00 ps
root@92.168.10.20:/mnt/bootcamp/lab_soln/lab6_soln/release# kill 979
root@92.168.10.20:/mnt/bootcamp/lab_soln/lab6_soln/release# ps
  PID TTY          TIME CMD
  975 pts/0    00:00:00 bash
  981 pts/0    00:00:00 ps
[1]+  Terminated                  ./lab6_soln
root@92.168.10.20:/mnt/bootcamp/lab_soln/lab6_soln/release#

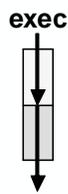
```

Side Topic – Creating New Processes in C

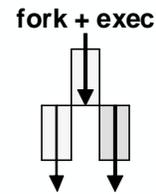
We won't actually need this for our lab exercises, though, we found it interesting enough to include it here.



Splits one executing process into two with same environment



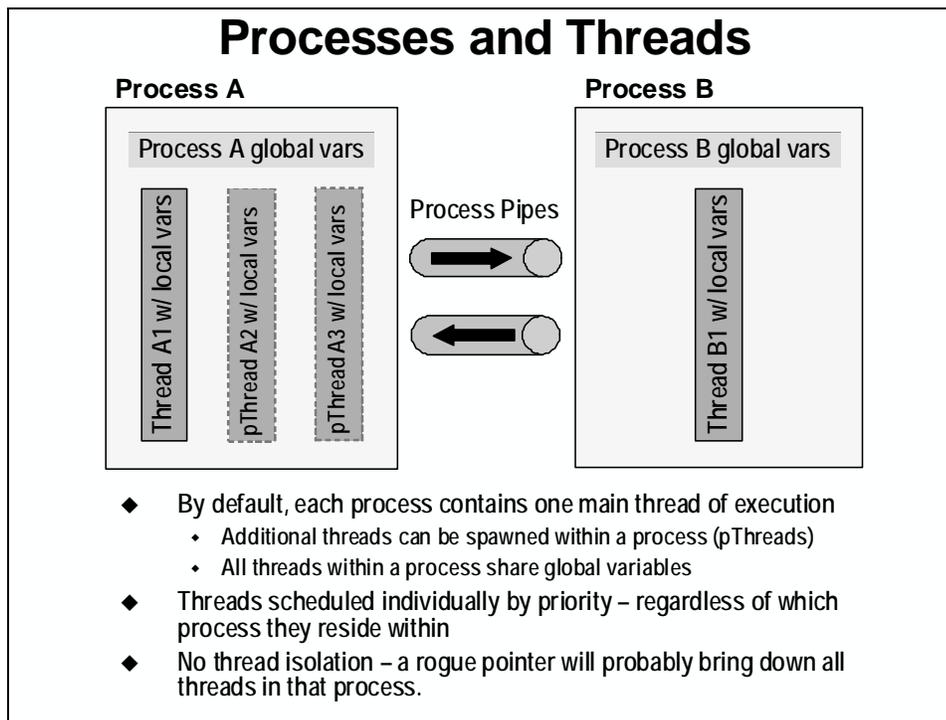
New process replaces old but maintains previous environment



Launch new process and keep previous process

- ◆ All processes are *split-off* from the original process created at startup
- ◆ When using fork, both processes run the same code; to prevent this, test if newly created process and run another program – or exec to another program
- ◆ To review, a *process* consists of:
 - ◆ Context (memory space, file descriptors)
 - ◆ One (or more) threads

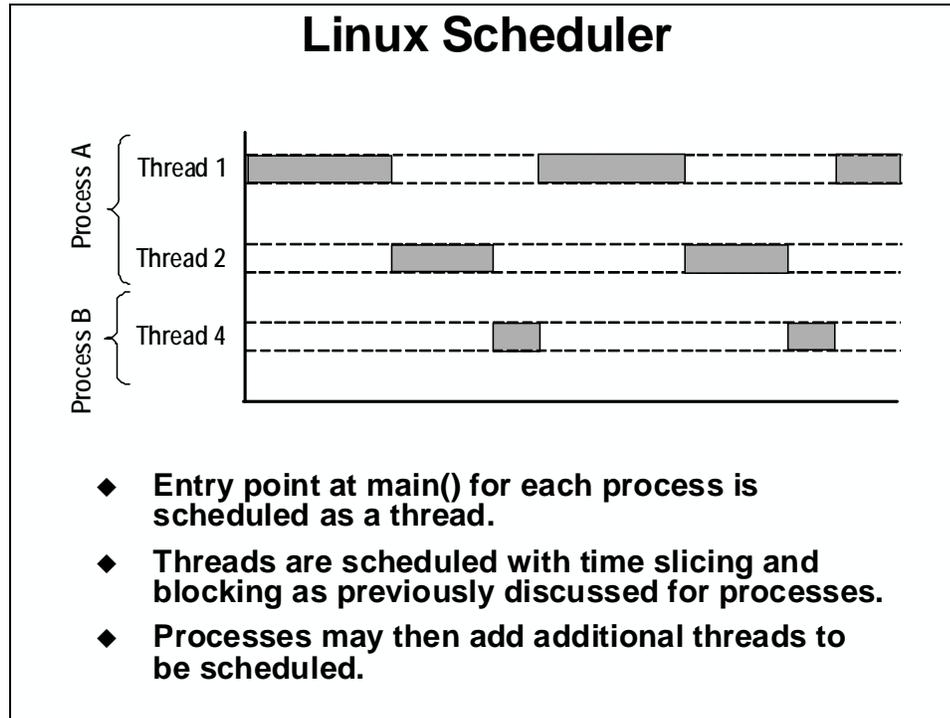
Linux Threads



Threads vs Processes

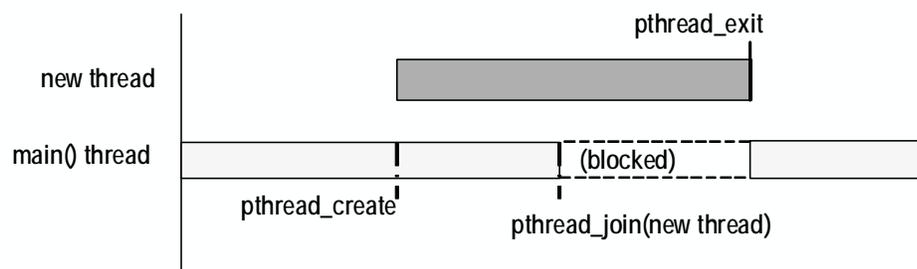
	Processes	Threads
Memory protection	✓	✗
Ease of use	✓	✗
Start-up cycles	✗	✓
Context switch	✗	✓
Codec Engine can span	✗ *Note	✓
Shared globals	no	yes
Environment	program	function

*Note: New Codec Engine feature now supports CE across processes when using LAD (Link Arbitor Daemon).



Thread Functions

Threads can only be launched from within a C program
(not from a shell)



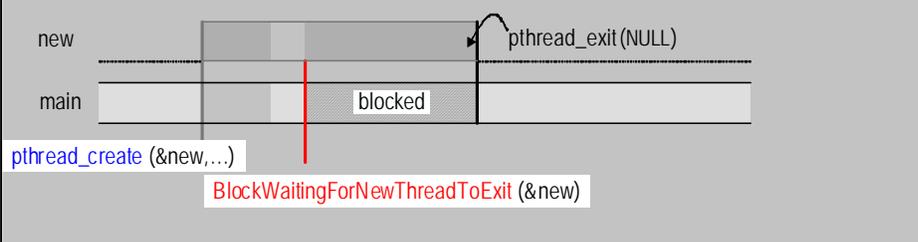
```
#include <pthread.h>
int pthread_create(pthread_t *thread, pthread_attr_t *attr,
void *(*start_routine)(void *), void *arg);
pthread_join(pthread_t thread, void **retval);
pthread_exit(void *retval);
```

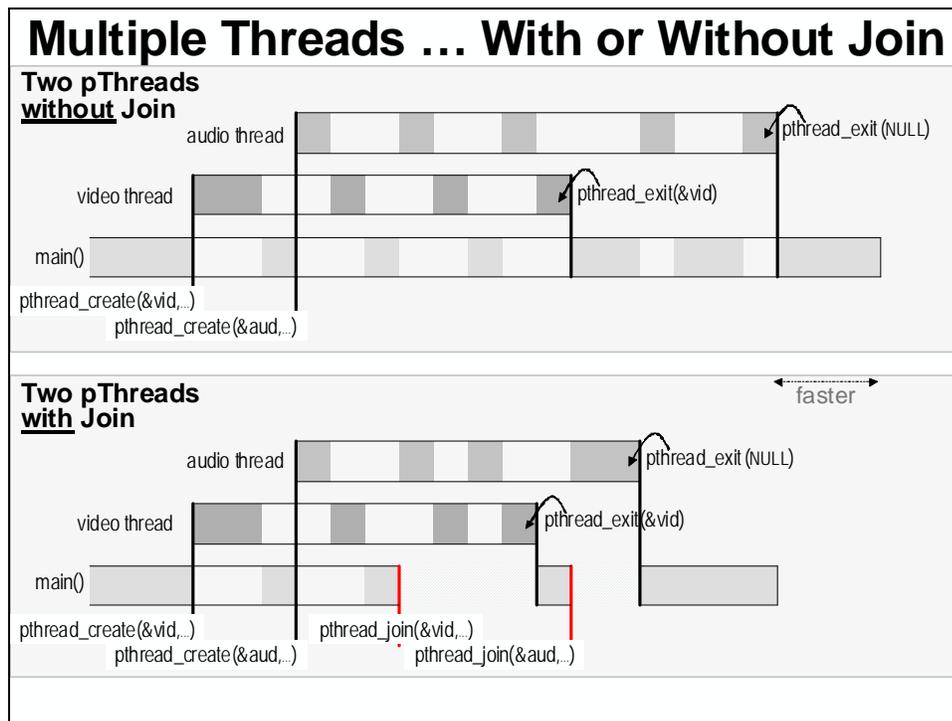
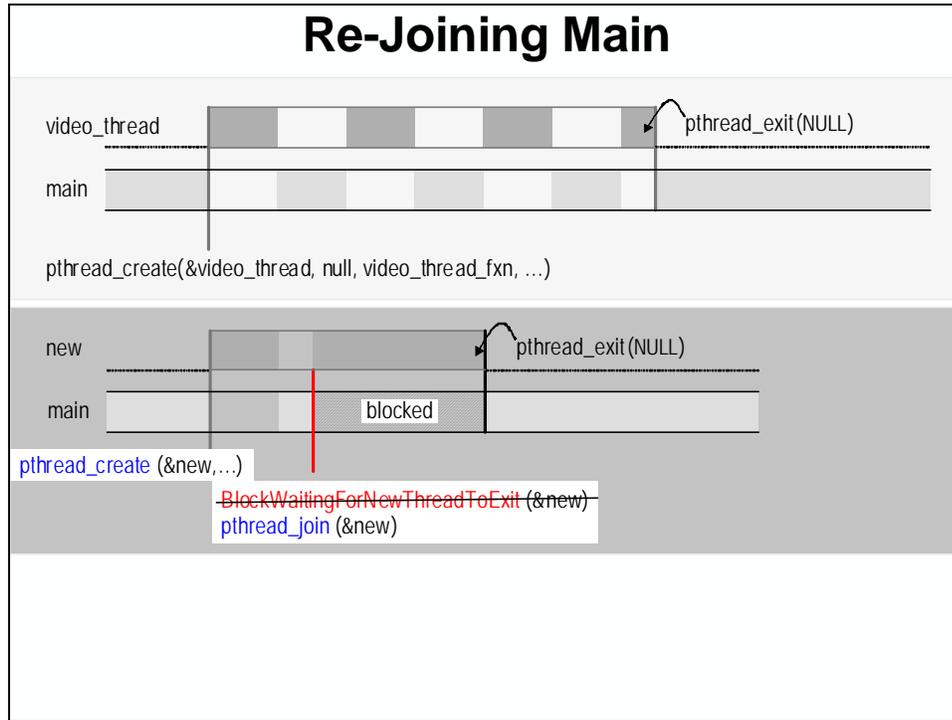
pThread Functions – Create & Exit



- ◆ Use `pthread_create()` to kickoff a new thread (i.e. child)
 - Starts new thread executing in the same process as its parent
 - As shown, both threads now compete for time from the Linux scheduler
 - Two important arguments – thread object, function to start running upon creation
- ◆ `pthread_exit()` causes child thread end
 - If `_create`'s starting function exits, `pthread_exit()` is called implicitly

Waiting for the Child to Exit





Thread Synchronization

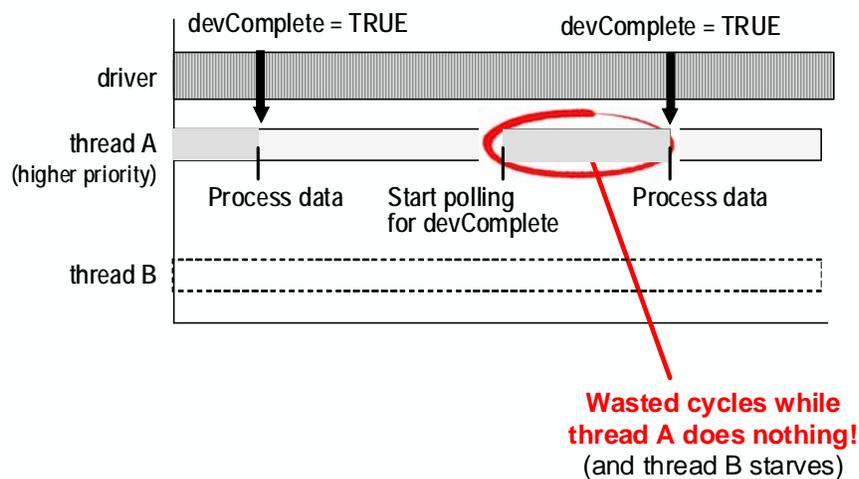
Thread Synchronization (Polling)

```
void *threadA(void *env){
int test;
while(1){
    while(test != TRUE) {
        test = (volatile int) env->driverComplete;
    }
    doSomething(env->bufferPtr);
}
}
```

} Polling Loop

- ◆ Thread A's doSomething() function should only run after the driver completes reading in a new buffer
- ◆ Polling can be used to halt the thread in a spin loop until the driverComplete flag is thrown.
- ◆ But **polling is inefficient** because it wastes CPU cycles while the thread does nothing.

Thread Synchronization (Polling)



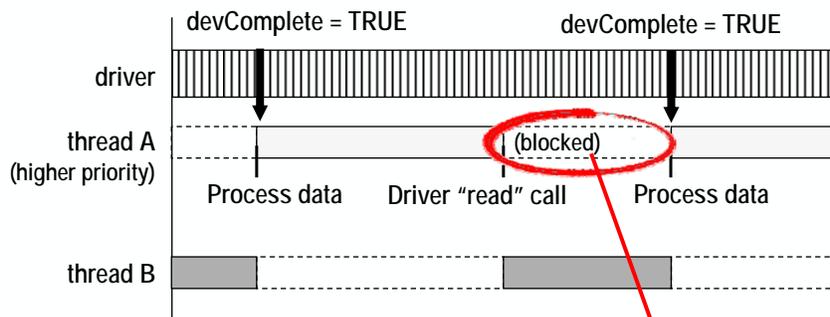
Thread Synchronization (Blocking)

```
void *threadA(void *env){
while(1){
    read(env->audioFd, env->bufferPtr, env->bufsize);
    doSomethingNext(env->bufferPtr);
}
```

Blocks
(waits till complete)

- ◆ Instead of polling on a flag, the thread blocks execution as a result of the driver's read call
- ◆ More efficient than polling because thread A doesn't waste cycles waiting on the driver to fill the buffer

Thread Synchronization (Blocking)



- ◆ Semaphores are used to block a thread's execution until occurrence of an event or freeing of a resource
- ◆ Much more efficient system

**Thread blocks until driver fills buffer.
No wasted cycles!**
(thread B gets to fill time)

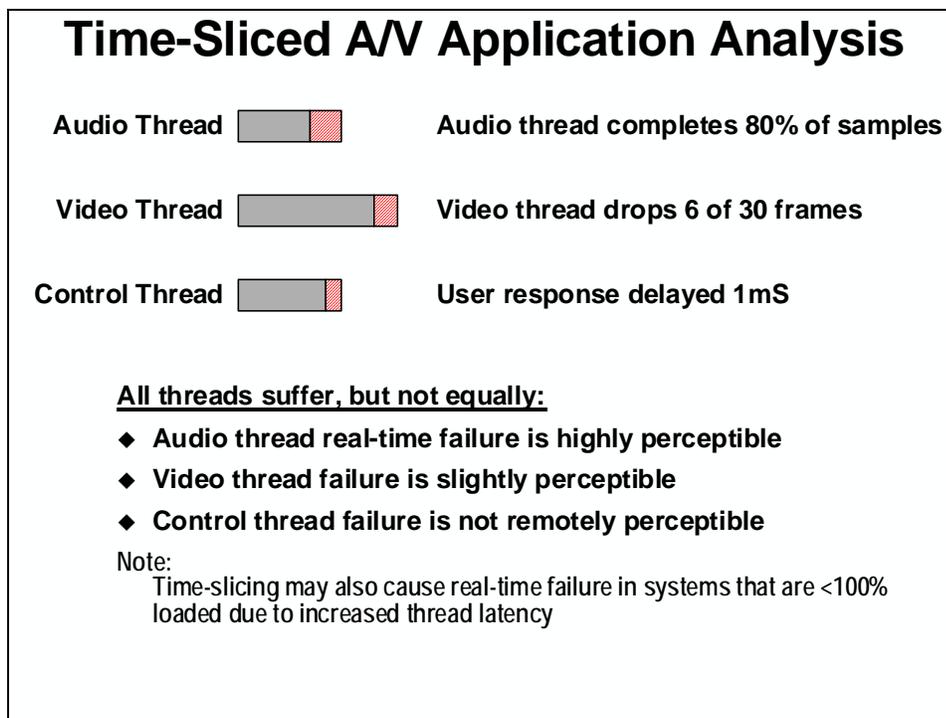
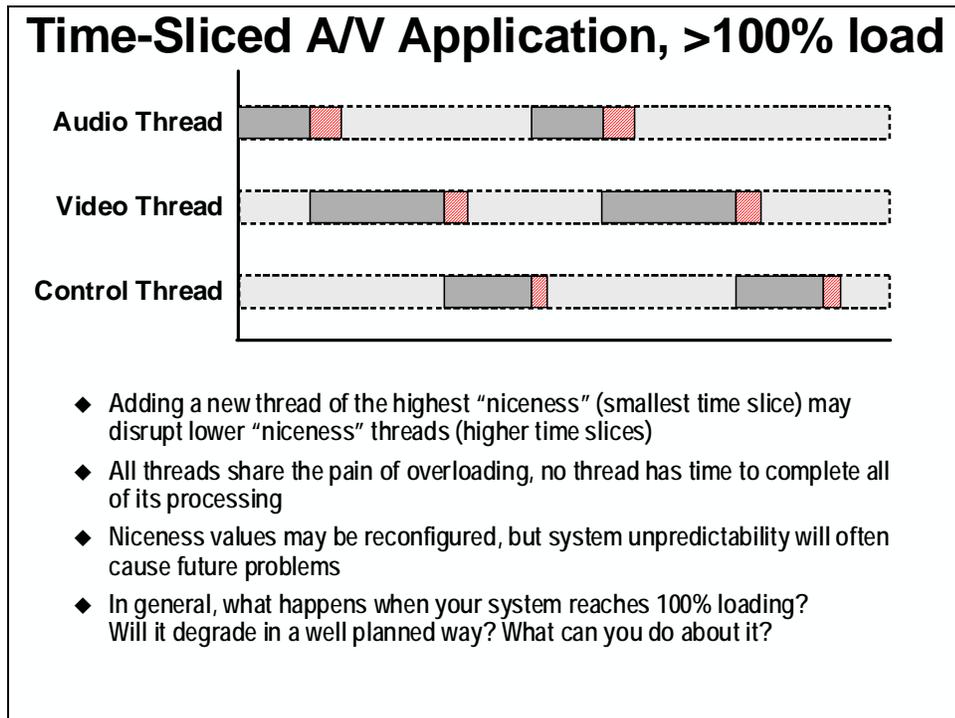
Synchronization with Peripherals

OSS driver: read function is blocking
 write function blocks if outgoing buffer full

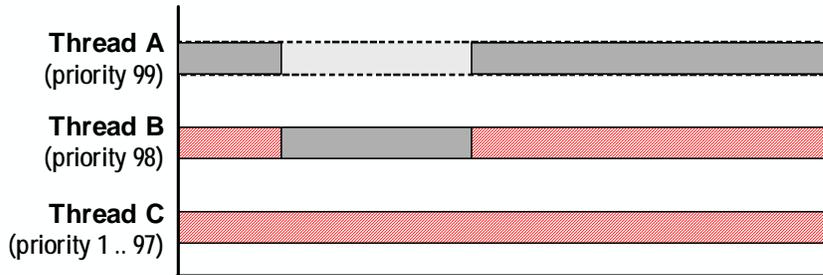
V4L2 driver: VIDIOC_DQBUF ioctl is blocking

FBDEV driver: FBIO_WAITFORVSYNC ioctl is blocking

Using Real-Time Threads

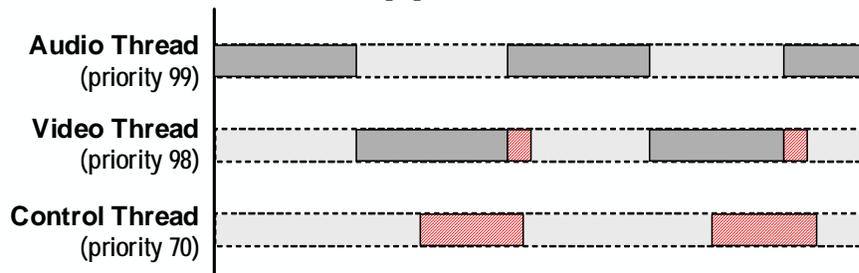


Linux Real-Time Scheduler (Generic)



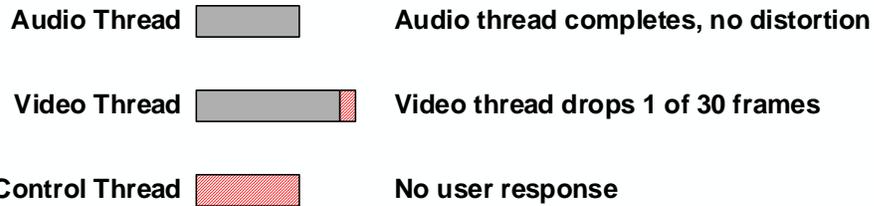
- ◆ In Linux, Real-Time threads are scheduled according to priority (levels 1-99, where time-slicing is effectively level 0)
- ◆ The highest priority thread always “wins” and will run 100% of the time unless it blocks

Real-time A/V Application, >100% load



- ◆ Audio thread is guaranteed the bandwidth it needs
- ◆ Video thread takes the rest
- ◆ Control thread never runs!

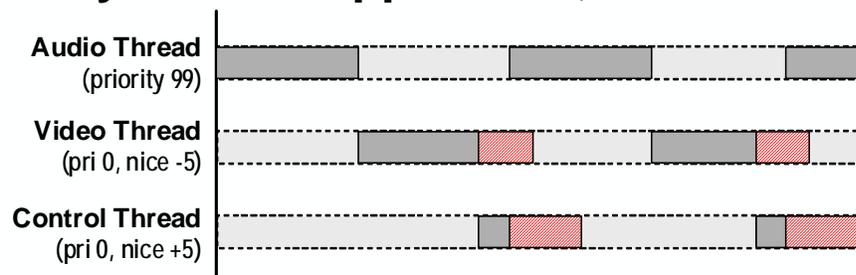
Time-Sliced A/V Application Analysis



Still a problem:

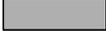
- ◆ Audio thread completes as desired
- ◆ Video thread failure is practically imperceptible
- ◆ Control thread never runs – User input is locked out

Hybrid A/V Application, >100% load



- ◆ Audio thread is guaranteed the bandwidth it needs
- ◆ Video thread takes *most* of remaining bandwidth
- ◆ Control thread gets a small portion of remaining bandwidth

Hybrid A/V Application Analysis

Audio Thread  Audio thread completes, no distortion

Video Thread  Video thread drops 2 of 30 frames

Control Thread  User response delayed 100ms

A good compromise:

- ◆ Audio thread completes as desired
- ◆ Video thread failure is barely perceptible
- ◆ Control thread delayed response is acceptable
- ◆ Bottom Line: We have designed the system so that it degrades gracefully

Default Thread Scheduling

```

#include <pthread.h>
...
pthread_create(&myThread, NULL, my_fxn,
              (void *) &audio_env);
    
```

◆ Setting the second argument to **NULL** means the pthread is created with default attributes

pThread attributes:	NULL / default value:
stacksize	PTHREAD_STACK_MIN
...	...
detachedstate	PTHREAD_CREATE_JOINABLE
schedpolicy	SCHED_OTHER (time slicing)
inheritsched	PTHREAD_INHERIT_SCHED
schedparam.sched_priority	0

Scheduling Policy Options

	SCHED_OTHER	SCHED_RR	SCHED_FIFO
Sched Method	Time Slicing	Real-Time (RT)	
RT priority	0	1 to 99	1 to 99
Min niceness	+20	n/a	n/a
Max niceness	-20	n/a	n/a
Scope	root or user	root	root

- ◆ Time Sliced scheduling is specified with SCHED_OTHER:
 - Niceness determines how much time slice a thread receives, where higher niceness value means less time slice
 - Threads that block frequently are rewarded by Linux with lower niceness
- ◆ Real-time threads use preemptive (i.e. priority-based) scheduling
 - Higher priority threads always preempt lower priority threads
 - RT threads scheduled at the same priority are defined by their policy:
 - SCHED_FIFO: When it begins running, it will continue until it blocks
 - SCHED_RR: "Round-Robin" will share with other threads at it's priority based on a deterministic time quantum

Real-time Thread Creation Procedure

Create attribute structure

```
// Initialize the pthread_attr_t structure audioThreadAttrs
pthread_attr_init(&audioThreadAttrs);
```

Set attribute to real-time priority 99

```
// Set the inheritance value in audioThreadAttrs structure
pthread_attr_setinheritsched(&audioThreadAttrs,
                             PTHREAD_EXPLICIT_SCHED);
```

```
// Set the scheduling policy for audioThreadAttrs structure
pthread_attr_setschedpolicy(&audioThreadAttrs, SCHED_RR);
```

```
// Set the scheduler priority via audioThreadParams struct
audioThreadParams.sched_priority = 99;
pthread_attr_setschedparam(&audioThreadAttrs,
                           &audioThreadParams);
```

Create thread with given attributes

```
// Create the new thread using thread attributes
pthread_create(&audioThread, &audioThreadAttrs,
              audio_thread_fxn, (void *) &audio_env);
```

*** HTTP ERROR 404 – PAGE NOT FOUND ***

Local Codecs - Using a Given Engine

Introduction

In this chapter the steps required to use a given engine will be examined

Learning Goals

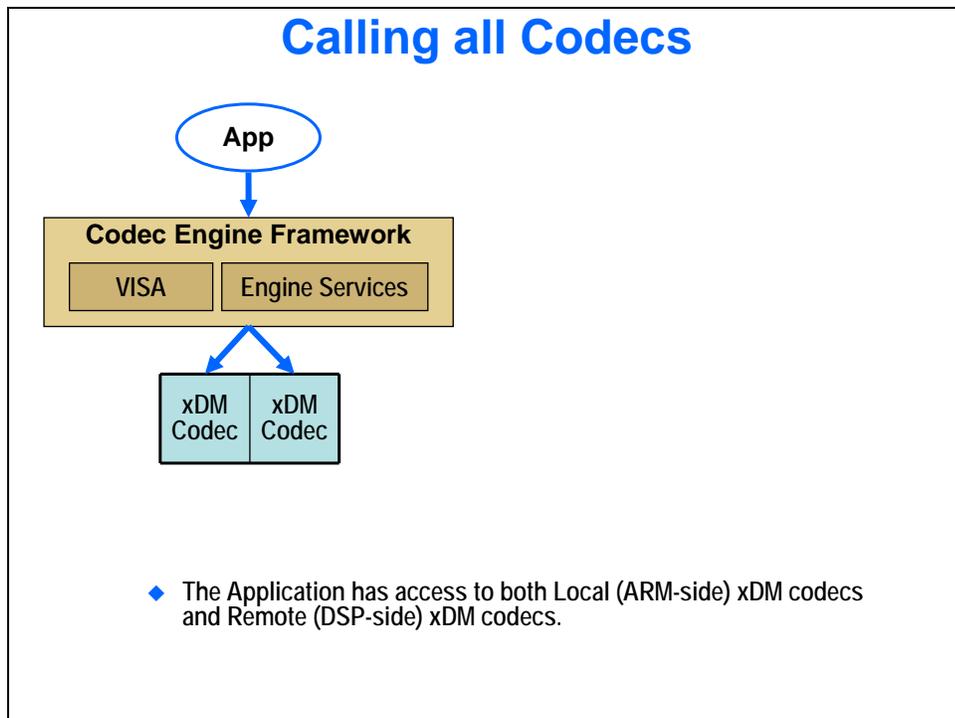
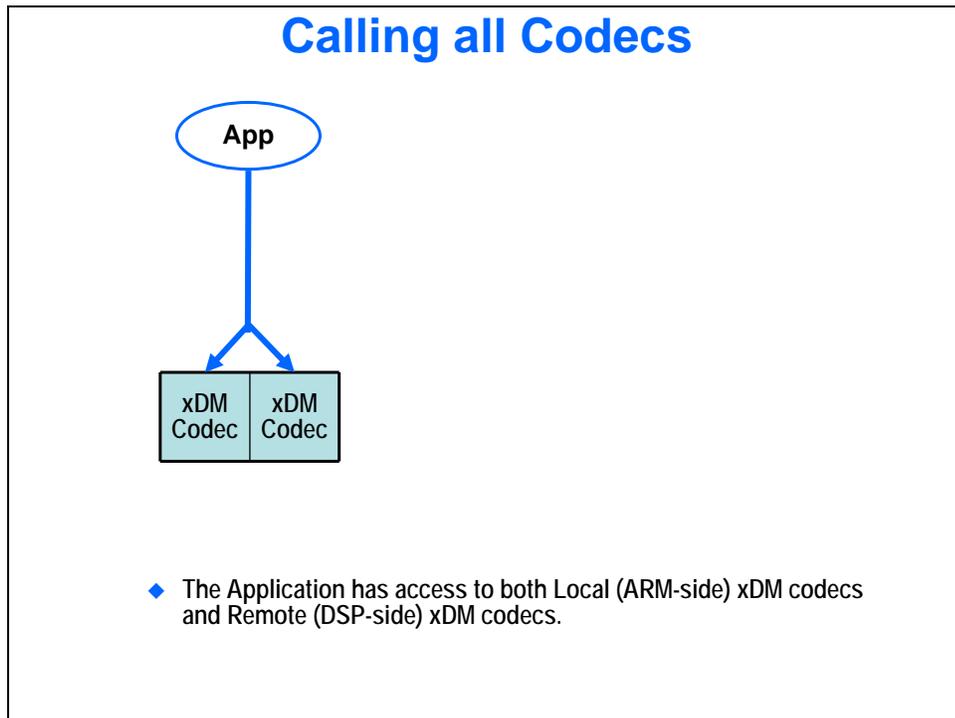
In this chapter the following topics will be presented:

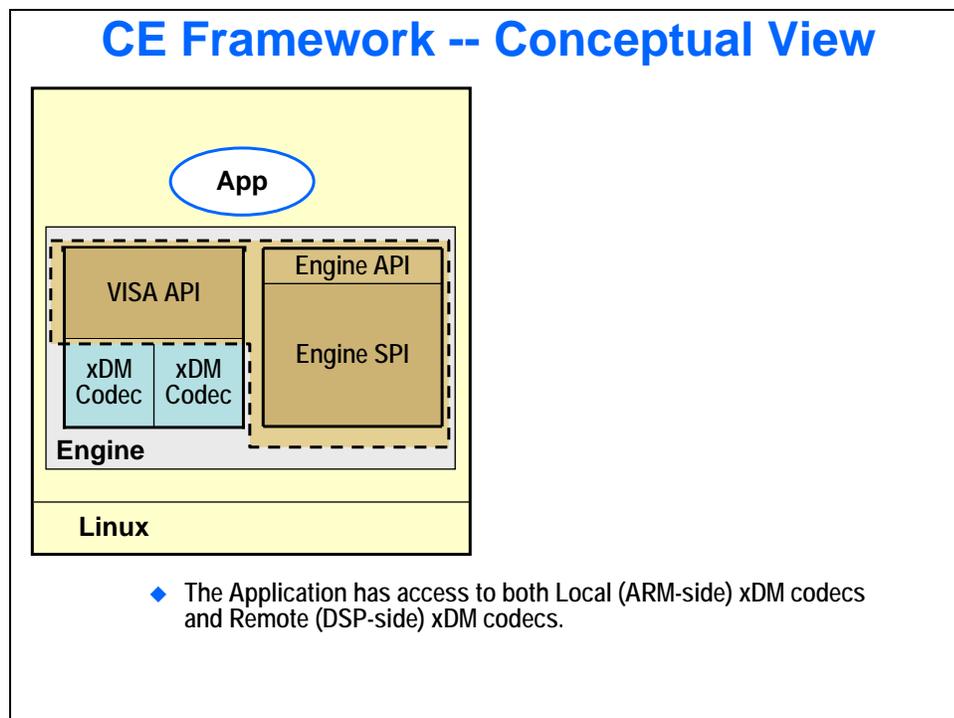
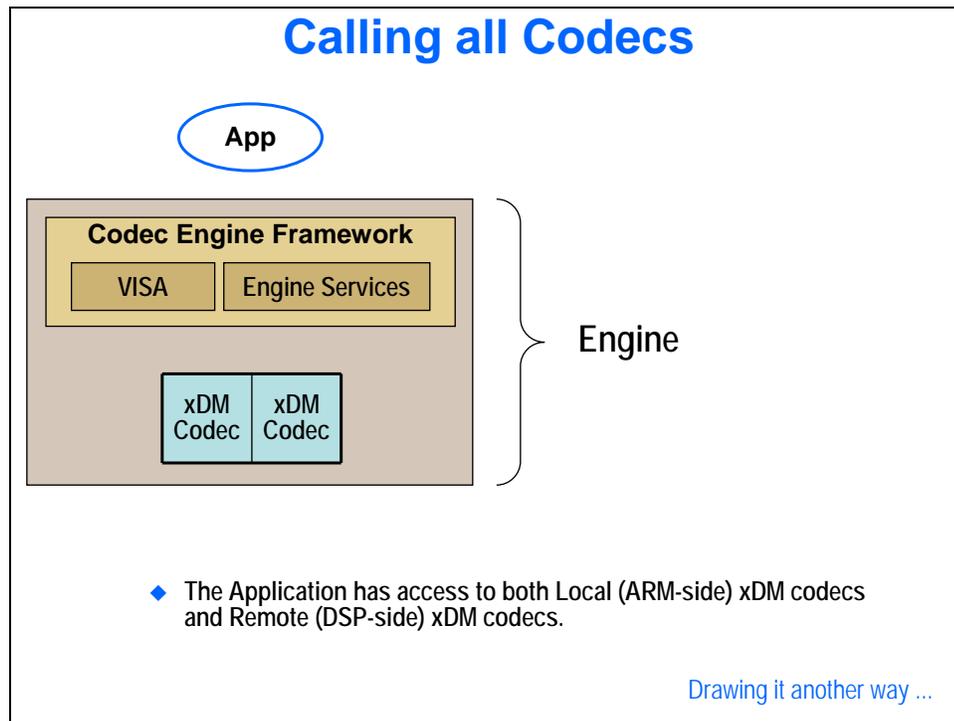
- VISA (Class) API
- Under the Hood of the Codec Engine
- Detailed Look at VISA Functions
- Using the Codec Engine within Multiple Threads

Chapter Topics

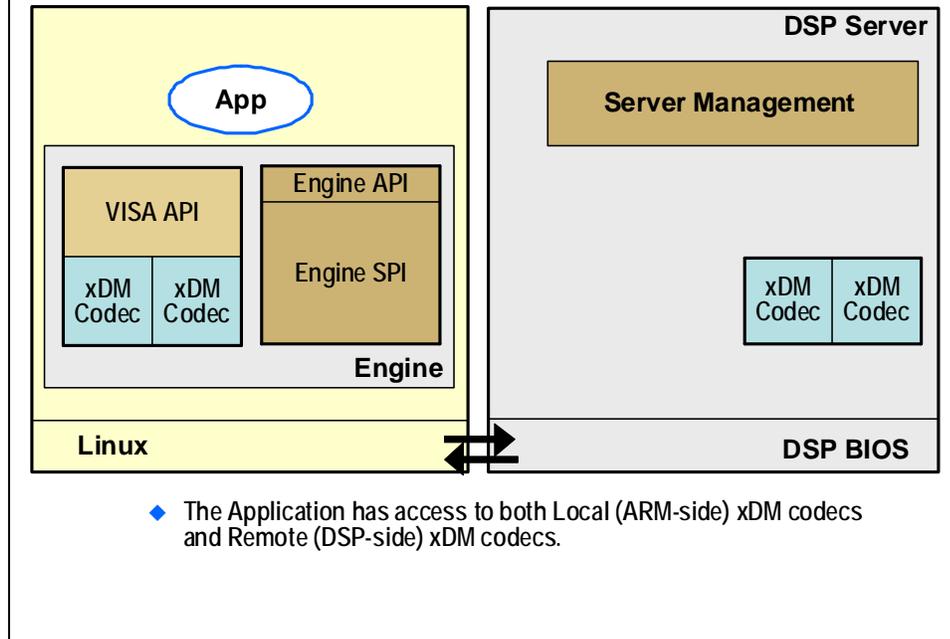
Local Codecs - Using a Given Engine	9-1
<i>VISA (Class) API.....</i>	<i>9-2</i>
<i>Under the Hood of the Codec Engine</i>	<i>9-6</i>
<i>Detailed Look at VISA Functions.....</i>	<i>9-7</i>
<i>Rules for Opening and Using Engines.....</i>	<i>9-13</i>
<i>Using Algorithms with DMAI</i>	<i>9-14</i>
Audio	9-14
Video	9-16
<i>Appendix</i>	<i>9-18</i>
Fancy VIDDEC/Display Buffer Management.....	9-18
DMAI (Digital Media App Interface) Library.....	9-20
Codec Engine Functions Summary	9-22

VISA (Class) API

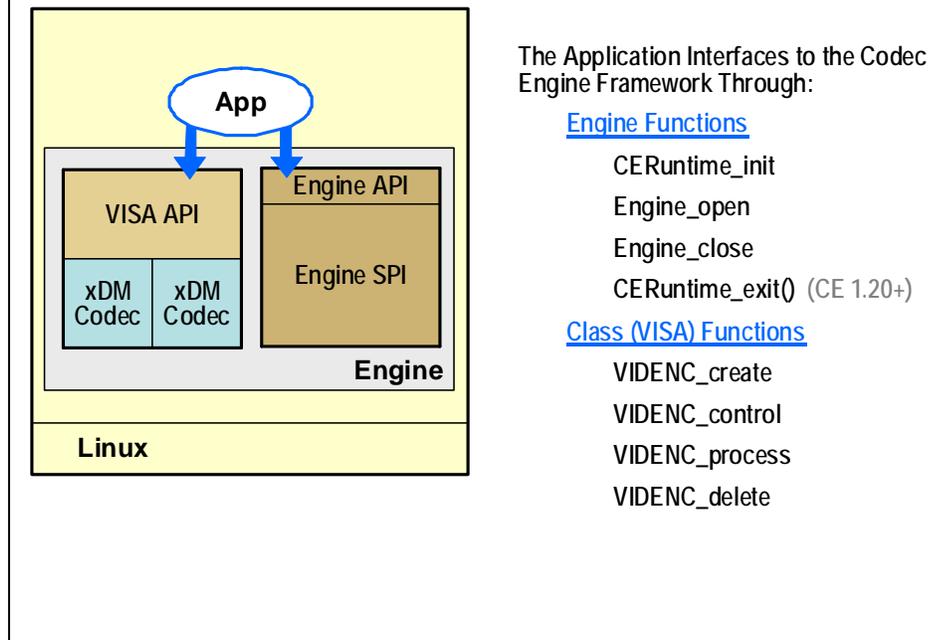




CE Framework -- Conceptual View



Conceptual View – Local Engine



Master Thread Key Activities

```

idevfd = open("/dev/xxx", O_RDONLY);
ofilefd = open("./fname", O_WRONLY);
ioctl(idevfd, CMD, &args);
CERuntime_init();
myCE = Engine_open("vcr", myCEAttrs);
myVE = VIDENC_create(myCE, "videnc", params);
while( doRecordVideo == 1 ) {
    read(idevfd, &rd, sizeof(rd));
    VIDENC_control(myVE, ...);
    VIDENC_process(myVE, ...);
    write(ofilefd, &wd, sizeof(wd));
}
close(idevfd);
close(ofilefd);
VIDENC_delete(myVE);
Engine_close(myCE);
    
```

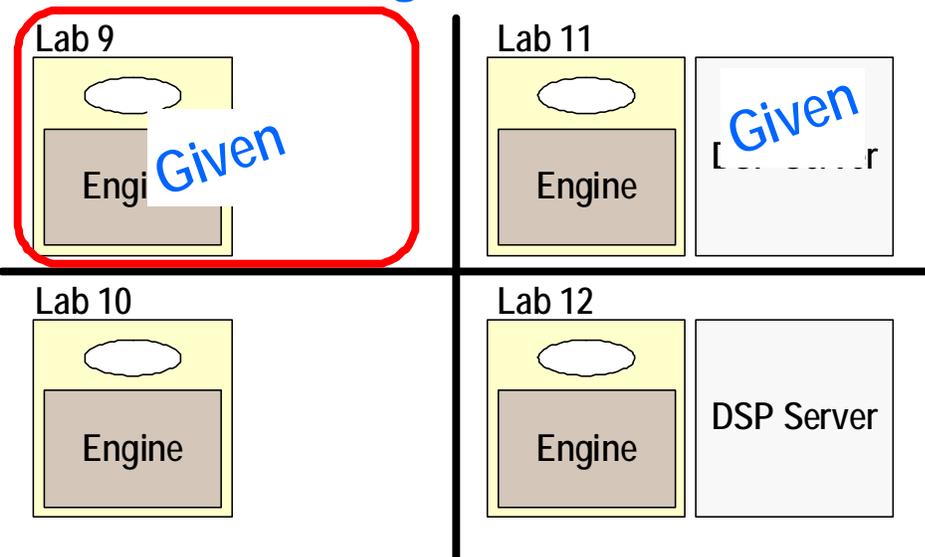
```

// Create Phase
// get input device
// get output device
// initialize IO devices...
// Called once in any CE app
// prepare VISA environment
// prepare to use video encoder
// Execute phase
// read/swap buffer with Input device
// option: perform VISA std algo ctrl
// run algo with new buffer
// pass results to Output device

// Delete phase
// return IO devices back to OS
// algo RAM back to heap
// close VISA framework
    
```

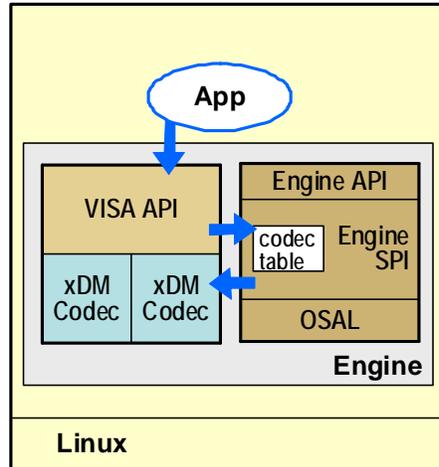
Note: the above pseudo-code does not show double buffering, often essential in RT systems!

Codec Engine - Use Cases



Under the Hood of the Codec Engine

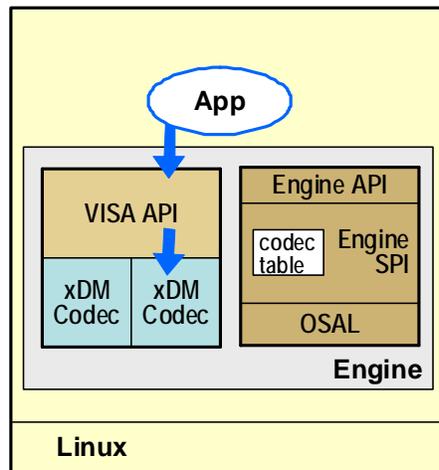
Conceptual View – Local Codec Engine



Create and Delete

- The application creates a local (or remote) video encoder instance through the VIDENC_create API
- The VIDENC_create or VIDENC_delete function passes the request to the Engine, which
 - determines if the requested codec is local via the codec table
 - And, if the codec is local, grants or frees resources such as memory and DMA channels to/from the algorithm
 - These resources ultimately come from the Linux O/S, which the Engine accesses via its O/S Abstraction Layer

VISA Control and Process



Control and Process

- The application accesses a codec instance through VIDENC_control and VIDENC_process API
- The VIDENC_control and VIDENC_process functions call corresponding control or process function from the Codec.
- Control and process calls made via a function pointer in the VIDENC_object
- Reason for this extra mechanism will become more clear when we study remote codecs

Detailed Look at VISA Functions

Calling `Engine_open()` and `_create()`

```

Engine_Handle  myCE;
AUDENC_Handle  myAE;
AUDENC_Params  params;

CERuntime_init();
myCE = Engine_open("myEngine", NULL);

... setup params structure

myAE = AUDENC_create(myCE, "myEnc1", &params);

```

- ◆ Engine and Codec names are declared during the [engine config](#) step of the build process
- ◆ We will explore this in the next chapter

xDM : Creation Parameters

```

typedef struct IAUDDEC_Params { // structure used to initialize the algorithm
    XDAS_Int32  size;           // size of this structure
    XDAS_Int32  maxSampleRate; // max sampling frequency supported in Hz
    XDAS_Int32  maxBitrate;    // max bit-rate supported in bits per secs
    XDAS_Int32  maxNoOfCh;     // max number of channels supported
    XDAS_Int32  dataEndianness; // endianness of input data
} IAUDDEC_Params;

```

```

typedef struct IVIDDEC_Params { // structure used to initialize the algorithm
    XDAS_Int32  size;           // size of this structure
    XDAS_Int32  maxHeight;     // max video height to be supported
    XDAS_Int32  maxWidth;     // max video width to be supported
    XDAS_Int32  maxFrameRate;  // max Framerate * 1000 to be supported
    XDAS_Int32  maxBitRate;    // max Bitrate to be supported in bits per second
    XDAS_Int32  dataEndianness; // endianness of input data; def'd in XDM_DataFormat
    XDAS_Int32  forceChromaFormat; // set to XDM_DEFAULT to avoid re-sampling
} IVIDDEC_Params;

```

Calling `_process()`

```

XDM_BufDesc      inBuf, outBuf;
AUDENC_InArgs    encInArgs;
AUDENC_OutArgs   encOutArgs;
Int              status;

encInArgs.size = sizeof(encInArgs);
encOutArgs.size = sizeof(encOutArgs);

... fill in remaining encInArgs values
... setup the Buffer Descriptors

status = AUDENC_process(myAE, &inBuf, &OutBuf,
                        &encInArgs, &encOutArgs);

if(status !=0) doerror(status);

```

Audio Decoder Process Arguments

```

typedef struct XDM_BufDesc {           // for buffer description (input and output buffers)
    XDAS_Int32  numBufs;                // number of buffers
    XDAS_Int32  *bufSizes;              // array of sizes of each buffer in 8-bit bytes
    XDAS_Int8   **bufs;                 // pointer to vector containing buffer addresses
} XDM_BufDesc;

typedef struct IAUDDEC_InArgs {        // for passing the input parameters for every decoder call
    XDAS_Int32  size;                   // size of this structure
    XDAS_Int32  numBytes;               // size of input data (in bytes) to be processed
} IAUDDEC_InArgs;

typedef struct IAUDDEC_OutArgs {       // relays output status of the decoder after decoding
    XDAS_Int32  size;                   // size of this structure
    XDAS_Int32  extendedError;          // Extended Error code. (see XDM_ErrorBit)
    XDAS_Int32  bytesConsumed;          // Number of bytes consumed during process call
} IAUDDEC_OutArgs;

```

[Looking more closely at the BufDesc ...](#)

Video Decoder Process Arguments

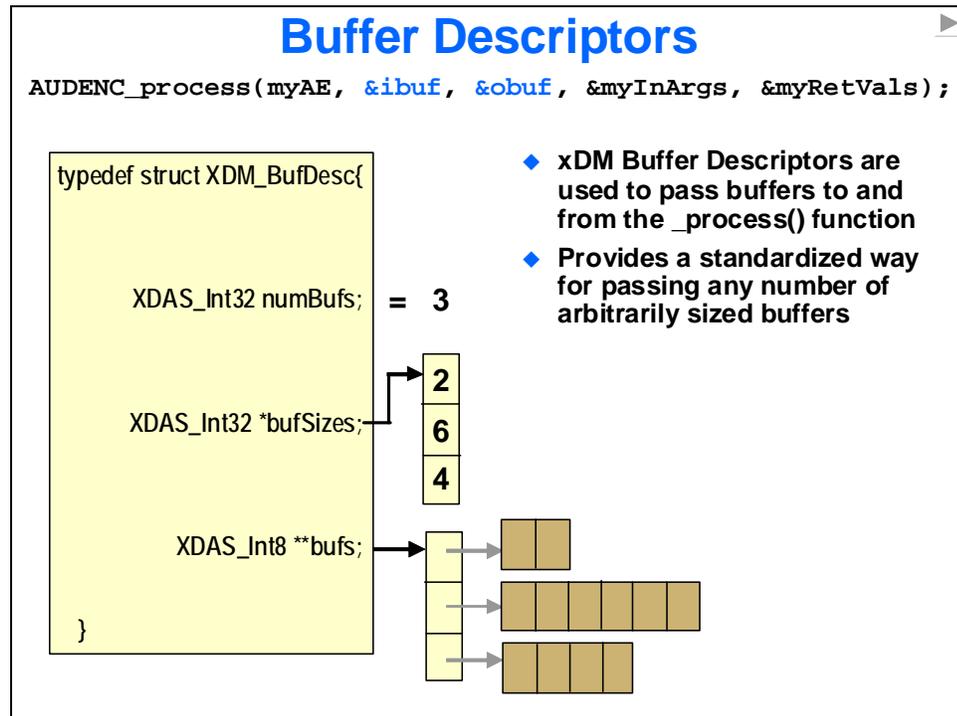
```

typedef struct XDM_BufDesc {           // for buffer description (input and output buffers)
    XDAS_Int32  numBufs;                // number of buffers
    XDAS_Int32  *bufSizes;             // array of sizes of each buffer in 8-bit bytes
    XDAS_Int8   **bufs;                // pointer to vector containing buffer addresses
} XDM_BufDesc;

typedef struct IVIDDEC_InArgs {        // for passing the input parameters for every decoder call
    XDAS_Int32  size;                  // size of this structure
    XDAS_Int32  numBytes;              // Size of valid input data in bytes in input buffer
    XDAS_Int32  inputID;              // algo tags out frames with this (app provided) ID
} IVIDDEC_InArgs;

typedef struct IVIDDEC_OutArgs {       // relays output status of the decoder after decoding
    XDAS_Int32  size;                  // size of this structure
    XDAS_Int32  extendedError;         // extended error code
    XDAS_Int32  bytesConsumed;         // bytes consumed per given call
    XDAS_Int32  decodedFrameType;     // frame type
    XDAS_Int32  outputID;             // output ID: tagged w value from *InArgs:InputId
    IVIDEO_BufDesc displayBufs;        // buffer pointers for current displayable frames.
} IVIDDEC_OutArgs;

```



Example Buffer Descriptor

Allocate Buffers	<pre>XDAS_Int8 buf0[2]; XDAS_Int8 buf1[6]; XDAS_Int8 buf2[4];</pre>
Allocate Buffer Descriptor	<pre>XDM_BufDesc inBufDesc; XDAS_Int32 inBufSizes[3]; XDAS_Int8 *inBufPtrs[3];</pre>
Build Buffer Descriptor	<pre>inBufDesc.numBufs = 3; inBufDesc.bufSizes = inBufSizes; inBufDesc.bufs = inBufPtrs;</pre>
Set size & Pointer Arrays	<pre>inBufSizes[0] = sizeof(buf0); inBufSizes[1] = sizeof(buf1); inBufSizes[2] = sizeof(buf2); inBufPtrs[0] = buf0; inBufPtrs[1] = buf1; inBufPtrs[2] = buf2;</pre>

Buffer Descriptors

```
VIDENC_process(myVE, &ibuf, &obuf, &myInArgs, &myRetVals);
```

```
typedef struct XDM_BufDesc {
  XDAS_Int32  numBufs;
  XDAS_Int32  *bufSizes;
  XDAS_Int8   **bufs;
}
```

◆ xDM Buffer Descriptors are used to pass buffers to and from the `_process()` function

```
VIDENC1_process(myVE, &ibuf, &obuf, &myInArgs, &myRetVals);
```

```
typedef struct XDM1_SingleBufDesc {
  XDAS_Int8   *buf;
  XDAS_Int32  bufSize;
  XDAS_Int32  accessMask;
}
```

```
typedef enum {
  XDM_ACCESSMODE_READ = 0,
  XDM_ACCESSMODE_WRITE = 1
} XDM_AccessMode
```

```
#define XDM_ISACCESSMODE_READ(x)
#define XDM_ISACCESSMODE_WRITE(x)
#define XDM_SETACCESSMODE_READ(x)
#define XDM_SETACCESSMODE_WRITE(x)
#define XDM_CLEARACCESSMODE_READ(x)
#define XDM_CLEARACCESSMODE_WRITE(x)
```

```
typedef struct XDM1_BufDesc {
  XDAS_Int32  numBufs;
  XDM1_SingleBufDesc desc[];
}
```

Calling `_control()` and `_delete()`

```

AUDENC_Status      status;
AUDENC_DynParams   dynParams;
Int                retVal;

retVal = AUDENC_control(myAE, XDM_GETSTATUS,
                        &dynParams, &status);
                        ──────────┬─────────
                        │           │
if(retVal !=0) printf("AUDENC_control Returned
                    extended error %d\n", status.extendedError);

```

Calling `AUDENC_delete()`

```
AUDENC_delete(myAE);
```

xDM control() API

Int (*control) (IAUDEC_Handle handle, IAUDEC_Cmd id, IAUDEC_DynamicParams *params, IAUDEC_Status *status)

handle pointer to instance of the algorithm
cmdId for controlling operation of the control
XDM_GETSTATUS returns status of the last decode call in IAUDEC_Status structure
XDM_SETPARAMS initializes decoder via IAUDEC_DynamicParams structure
XDM_RESET resets the decoder
XDM_SETDEFAULT sets decoder parameters to default set of values
XDM_FLUSH the next process call after this control command will flush the outputs
XDM_GETBUFINFO provides input and output buffer sizes
params structure that allows the parameters to change on the fly of the process call
status status of decoder as of the last decode call is written to IAUDEC_Status structure

```

typedef struct IAUDEC_DynamicParams { // control API argument
    XDAS_Int32 size; // size of this structure
    XDAS_Int32 outputFormat; // sets interleaved/Block format. see IAUDIO_PcmFormat
}IAUDEC_DynamicParams;

```

Audio Decoder Control Arguments

```

typedef struct IAUDDEC_Status { // used in control API call to relay the status of prior decode
    XDAS_Int32 size; // size of this structure
    XDAS_Int32 extendedError; // extended error code. (see XDM_ErrorBit)
    XDAS_Int32 bitRate; // Average bit rate in bits per second
    XDAS_Int32 sampleRate; // sampling frequency (in Hz)
    XDAS_Int32 numChannels; // number of Channels: IAUDIO_ChannelId
    XDAS_Int32 numLFEChannels; // number of LFE channels in the stream
    XDAS_Int32 outputFormat; // output PCM format: IAUDIO_PcmFormat
    XDAS_Int32 autoPosition; // support for random position decoding: 1=yes 0=no
    XDAS_Int32 fastFwdLen; // recommended FF length in case random access in bytes
    XDAS_Int32 frameLen; // frame length in number of samples
    XDAS_Int32 outputBitsPerSample; // no. bits per output sample, eg: 16 bits per PCM sample
    XDM_AlgoBufInfo bufInfo; // input & output buffer information
} IAUDDEC_Status;

typedef struct XDM_AlgoBufInfo { // return the size and number of buffers needed for input & output
    XDAS_Int32 minNumInBufs; // min number of input buffers
    XDAS_Int32 minNumOutBufs; // min number of output buffers
    XDAS_Int32 minInBufSize[XDM_MAX_IO_BUFFERS]; // min bytes req'd for each input buffer
    XDAS_Int32 minOutBufSize[XDM_MAX_IO_BUFFERS]; // min bytes req'd for ea. output buffer
} XDM_AlgoBufInfo ;

```

Video Decoder Control Arguments

```

typedef struct IVIDDEC_Status { // used in control API call to relay the status of prior decode
    XDAS_Int32 size; // size of this structure
    XDAS_Int32 extendedError; // Extended Error code. (see XDM_ErrorBit)
    XDAS_Int32 outputHeight; // Output Height
    XDAS_Int32 outputWidth; // Output Width
    XDAS_Int32 frameRate; // Average frame rate* 1000
    XDAS_Int32 bitRate; // Average Bit rate in bits/second
    XDAS_Int32 contentType; // IVIDEO_PROGRESSIVE or IVIDEO_INTERLACED
    XDAS_Int32 outputChromaFormat; // Chroma output fmt of type IVIDEO_CHROMAFORMAT
    XDM_AlgoBufInfo bufInfo; // Input & output buffer info
} IVIDDEC_Status;

typedef struct XDM_AlgoBufInfo { // return the size and number of buffers needed for input & output
    XDAS_Int32 minNumInBufs; // min number of input buffers
    XDAS_Int32 minNumOutBufs; // min number of output buffers
    XDAS_Int32 minInBufSize[XDM_MAX_IO_BUFFERS]; // min bytes req'd for each input buffer
    XDAS_Int32 minOutBufSize[XDM_MAX_IO_BUFFERS]; // min bytes req'd for ea. output buffer
} XDM_AlgoBufInfo ;

```

Rules for Opening and Using Engines

Engine Rules

- ◆ Only one Engine can be open at a time*
- ◆ The Engine can: Only be accessed from within a single process*
be accessed across multiple threads within a single process
- ◆ All threads must open the same Engine (i.e. Engine's name must match)
- ◆ Each thread should obtain its own Engine handle
 - ◆ Often, main thread creates environment variable – with engine "name" – to pass to each thread which uses CE

```
void *myVideoThread(void *env) {
```

```
Engine_Handle hvEngine;  
VIDENC_Handle hVidenc;
```

```
hvEngine = Engine_open(env->eName);  
hVidenc = VIDENC_create(hvEngine,...);
```

```
VIDENC_process(hVidenc,...);
```

```
void *myAudioThread(void *env) {
```

```
Engine_Handle haEngine;  
AUDENC_Handle hAudenc;
```

```
haEngine = Engine_open(env->eName);  
hAudenc = AUDENC_create(haEngine,...);
```

```
AUDENC_process(hAudenc,...);
```

*Notes: • Codec Engine 2.0 supports use across processes when using LAD (Link Arbitor Daemon).
See http://tiexpressdsp.com/index.php/Multiple_Threads_using_Codec_Engine_Handle

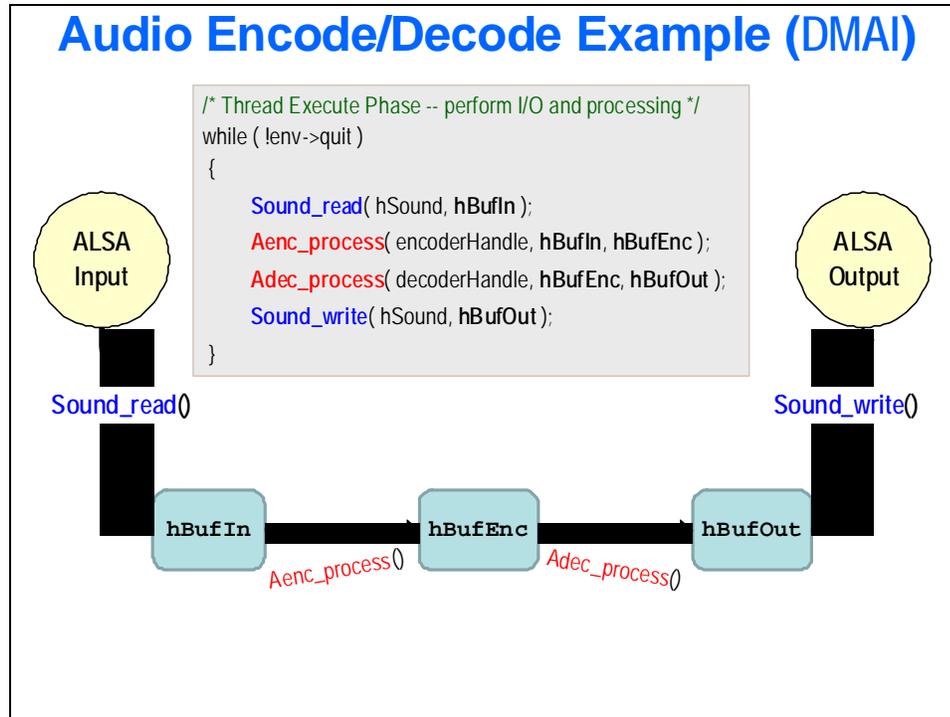
The first rule is that only one Engine can be open at a time. While not strictly accurate, it is a good rule to live by. Technically speaking, while more than one Engine can be opened at a time, only one DSP Server can be opened – and thus loaded into the DSP – at a time. This means that if you try to open multiple Engines, they must either contain only local codecs, or they must reference the same DSP Server. The bottom line is that there is rarely – if ever – a need to have more than one Engine open at a time.

An Engine can only be accessed from within a single process. It is very common, though that an Engine may be accessed by many threads (i.e. pthreads) as needed within that process.

If you use an Engine across multiple threads, each thread should call Engine_open(). In our previous example, we have passed the Engine name to each thread in a variable called *env*. Calling Engine_open() multiple times does not actually re-open the same Engine, rather, it allows the Codec Engine framework to count the number of threads using the Engine. Thus, if later on one thread calls Engine_close(), it won't actually close the Engine until it has been closed by all the threads who earlier called _open().

Using Algorithms with DMAI

Audio



Audio Encoder/Decoder Example : Variables

```

Sound_Handle hSound      = NULL;
Sound_Attrs  sAttrs      = Sound_Attrs_STEREO_DEFAULT;

Buffer_Handle hBufIn     = NULL;
Buffer_Handle hBufOut    = NULL;
Buffer_Handle hBufEnc    = NULL;

Buffer_Attrs bAttrs      = Buffer_Attrs_DEFAULT;

Engine_Handle engineHandle = NULL;

Aenc_Handle  encoderHandle = NULL;
Adec_Handle  decoderHandle = NULL;

AUDENC_Params aeParams     = Aenc_Params_DEFAULT;
AUDEEC_Params adParams     = Adec_Params_DEFAULT;

AUDENC_DynamicParams aeDynParams = Aenc_DynamicParams_DEFAULT;
AUDEEC_DynamicParams adDynParams = Adec_DynamicParams_DEFAULT;

```

Audio Encoder/Decoder Example : Create

```

void *audio_thread_fxn(void *envByRef) {
    audio_thread_env *env = envByRef;
    ...
    hBufIn = Buffer_create( blksize, &bAttrs );
    hBufOut = Buffer_create( blksize, &bAttrs );
    hBufEnc = Buffer_create( blksize, &bAttrs );

    engineHandle = Engine_open( env->engineName, NULL, NULL );
    encoderHandle = Aenc_create( engineHandle, "encoder_name",
                                &aeParams, &aeDynParams );
    decoderHandle = Adec_create( engineHandle, "decoder_name",
                                &adParams, &adDynParams );
}

```

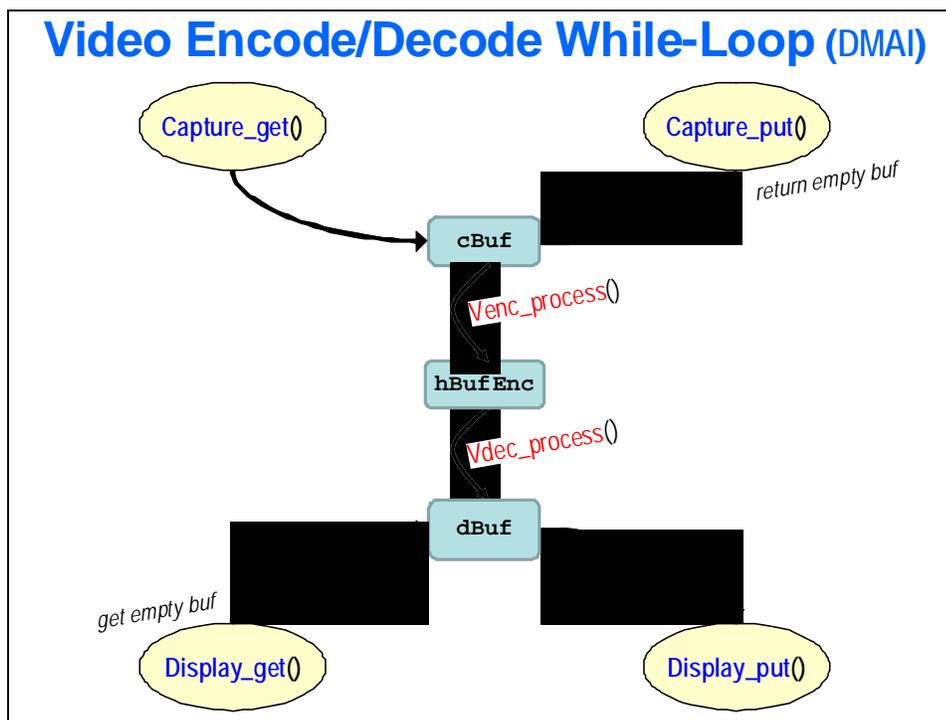
Audio Encoder/Decoder Example : Process

```

/* Thread Execute Phase -- I/O and processing */
while ( !env->quit )
{
    Sound_read( hSound, hBufIn );
    Aenc_process( encoderHandle, hBufIn, hBufEnc );
    Adec_process( decoderHandle, hBufEnc, hBufOut );
    Sound_write( hSound, hBufOut );
}

```

Video



```

/* Thread Execute Phase -- I/O and processing */
while ( !env->quit )
{
    Capture_get( hCap, &cBuf );
    Venc_process( hEnc, cBuf, encBuf );
    Capture_put( hCap, cBuf );

    Display_get( hDisp, &dBuf );
    Vdec_process( hDec, encBuf, dBuf );
    Display_put( hDisp, dBuf );
}

```

Video Encoder Object – Create Example

```

Engine_Handle hEng = NULL;
Venc_Handle hEnc = NULL;
VIDENC_Params sMyEncParams = Venc_Params_DEFAULT;
VIDENC_DynamicParams sMyEncDynParams =
    Venc_DynamicParams_DEFAULT;

void *video_thread_fxn(void *envByRef) {
    audio_thread_env *env = envByRef;
    ...
    hEng = Engine_open( env->engineName, NULL, NULL );
    hEnc = Venc_create( engineHandle ,
        "encoder_name",
        &sMyEncParams ,
        &sMyEncDynParams );
}

```

Video Decoder Object – Create Example

```

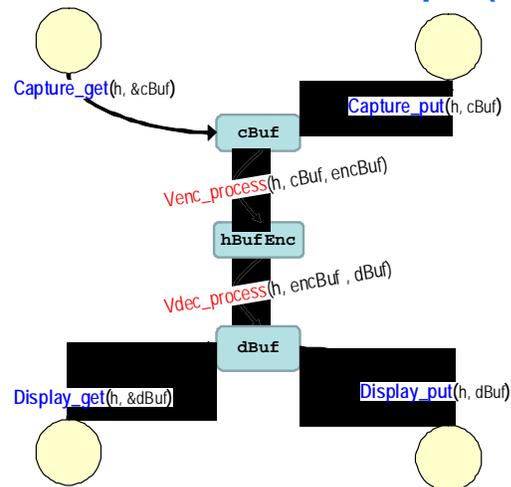
Engine_Handle engineHandle = NULL;
Vdec_Handle decoderHandle = NULL;
VIDDEC_Params sMyDecParams = Vdec_Params_DEFAULT;
VIDDEC_DynamicParams sMyDecDynParams =
    Vdec_DynamicParams_DEFAULT;

engineHandle = Engine_open( env->engineName, NULL, NULL );
decoderHandle = Vdec_create( engineHandle ,
    "decoder name",
    &sMyDecParams ,
    &sMyDecDynParams );

hBufTabDecoder = BufTab_create( NUM_DECODER_BUFS, bufsize,
    BufferGfx_getBufferAttrs(&gfxAttrs) );
Vdec_setBufTab( decoderHandle, hBufTabDecoder );

```

Audio Encode/Decode Example (DMAI)



Appendix

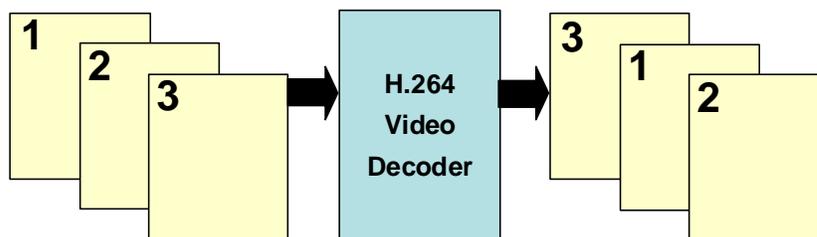
Fancy VIDDEC/Display Buffer Management

How Does v4l2 Reference Buffers

```
typedef struct v4l2_buffer{
    _u32                index;
    enum v4l2_buf_type  type;
    _u32                bytesused;
    _u32                flags;
    enum v4l2_field     field;
    struct timeval      timestamp;
    struct v4l2_timecode  timecode;
    _u32                sequence;
}
```

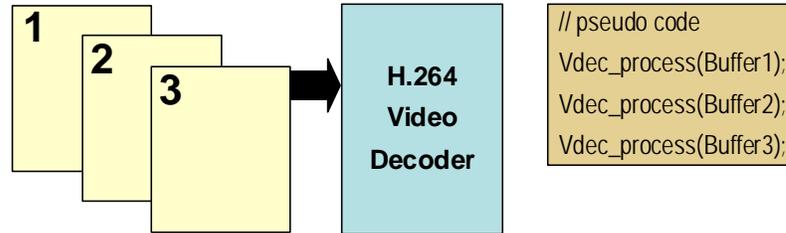
- ◆ V4L2 driver uses/knows a buffer's physical address
- ◆ To determine the application's virtual address requires a cycle-intensive MMU check
- ◆ Instead, V4L2 driver queues and dequeues buffers by [index](#)

Video Decoder Indexing



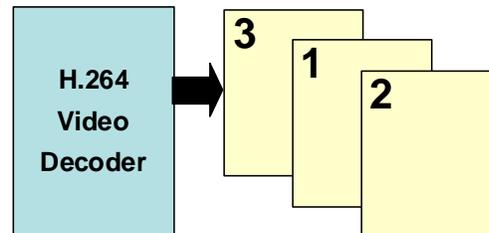
- ◆ Encoded (compressed) video data may contain bi-directional "B" frames which reference data from [previous](#) and [following](#) video frames
- ◆ This means that video decoders may end up decoding video frames [out-of-order](#)

Video Decoder Indexing Details



```
// pseudo code
Vdec_process(Buffer1);
Vdec_process(Buffer2);
Vdec_process(Buffer3);
```

```
// pseudo code
Buffer? = Vdec_getDisplayBuf();
Buffer? = Vdec_getDisplayBuf();
Buffer? = Vdec_getDisplayBuf();
```



BufTab Sharing Example - Initialization

```
/* Open codec engine and create a video decoder instance */
engineHandle = Engine_open( env->engineName, NULL, NULL );
decoderHandle = Vdec_create( engineHandle, VIDEO_DECODER,
                             &dParams, &dDynParams );

/* Create a BufTab to use with decoder and display driver */
hBufTabDecoderDisplay = BufTab_create(
    NUM_DECODER_DISPLAY_BUFS,
    bufSize,
    BufferGfx_getBufferAttrs(&gfxAttrs));

/* Set hBufTabDecoderDisplay for use with video decoder */
Vdec_setBufTab( decoderHandle, hBufTabDecoderDisplay );

/* Initialize Display driver with hBufTabDecoderDisplay */
hDisplay = Display_create( hBufTabDecoderDisplay, &dAttrs );
```

BufTab Sharing Example – Buffer Passing

```

while ( !env->quit ) {
    fread( Buffer_getUserPtr(inBuf), 1      ,
           Buffer_getSize(inBuf)  , hFile );
    Display_get( hDisplay, &dBuf );
    Vdec_process( decoderHandle, inBuf, dBuf );

    dBuf = Vdec_getDisplayBuf( decoderHandle );

    while( dBuf != NULL ) {
        Display_put( hDisplay, dBuf );
        dBuf = Vdec_getDisplayBuf( decoderHandle );
    }
}

```

- ◆ dBuf (BufferGfx_Handle) can be passed between Display and Vdec because both were initialized with *hBufTabDecoderDisplay*
- ◆ Vdec_process() and Vdec_getDisplayBuf() act as "put" and "get" (respectively) for video decoder queue
- ◆ If decoder "get" doesn't return a buffer to display, the display driver queue is not updated on this pass of the loop
- ◆ Conversely, if decoder "get" provides a non-NULL handle for dBuf, then we'll keep que'ing buffers from the decoder to the display until none are left

DMAI (Digital Media App Interface) Library

This shows mapping between our DM6446 lab exercises and DMAI-based labs.

Workshop Example – Lab06b with Audio Decode

```

audio_output_setup()
inBuf = malloc()

fopen("myaudio.dat")
outBuf = malloc()

engine_setup()
audio_decoder_setup()

fread data into inBuf
while()
    decode_audio(inBuf → outBuf)
    write encBuf
    fread data into inBuf

```

audio_thread.c

- initialize OSS device (audio output setup)
- create output buffer
- open audio file
- create input buffer
- open engine
- create auddec algo
- fread first buffer
- ∞ while process (ie. decode) output audio fread audio buffer
- cleanup

- ◆ To simplify workshop labs, we originally created a set of "wrapper" functions to encapsulate driver and visa calls
- ◆ DMAI (digital media applications interface) library includes a more standardized set of functions to implement a similar set of wrapper functions

AAC Audio Decode Example

```

Dmai_init();

/* Module and buffer to manage output to sound per
Sound_Handle hSound = Sound_create(&sAttrs);
Buffer_Handle hOutBuf = Buffer_create(Adec_getOutB
                                     &bAttrs);
-----
/* Module and buffer to manage input from "myfile.
Loader_Handle hLoader = Loader_create("myfile.aac"
Buffer_Handle hInBuf;
-----
/* Module to manage audio decode */
Engine_Handle hEngine = Engine_open("myengine", NU
Adec_Handle hAd = Adec_create(hEngine, "aac"
                             &dynParams);
-----
/* main algorithm */
Loader_prime(hLoader, &hInBuf);
while (1) {
    Adec_process(hAd, hInBuf, hOutBuf);
    Sound_write(hSound, hOutBuf);
    Loader_getFrame(hLoader, hInBuf);
    if (Buffer_getUserPtr(hInBuf) == NULL) break;
}

```

audio_thread.c

- initialize OSS device (audio output setup)
- create output buffer
- open audio file
- create input buffer
- open engine
- create auddec algo
- fread first buffer
- while process (ie. decode) output audio fread audio buffer

Example of DMAI Functions

Function	Description
Loader_create()	Creates a Loader
Loader_prime()	Prime the loader and obtain the first frame
Loader_getFrame()	Load the next frame
Adec_create()	Creates an Audio Decode algorithm instance
Adec_process()	Get the error code of the last failed operation.
Buffer_create()	Creates and allocates a contiguous Buffer
Buffer_delete()	Deletes and frees a contiguous Buffer
Buffer_getSize()	Get the size of a Buffer

- ◆ Library of functions to help ease use of Linux drivers and CE's VISA functions; esp. helpful when utilizing complex-useage algorithms like H.264
- ◆ Easier to port CE applications when using high-level functions like DMAI
- ◆ DMAI does not have anything to do with DMA (direct memory access)
- ◆ DMAI is now an open source project at: <http://qforge.ti.com/qf/project/dmai>

Codec Engine Functions Summary

Core Engine_ Functions	
Function	Description
CERuntime_init	Initialize the Codec Engine
CERuntime_exit	Free CE memory.
Engine_open()	Open an Engine.
Engine_close()	Close an Engine.
Engine_getCpuLoad()	Get Server's CPU usage in percent.
Engine_getLastError()	Get the error code of the last failed operation.
Engine_getUsedMem()	Get Engine memory usage.
Engine_getNumAlgs()	Get the number of algorithms in an Engine.
Engine_getAlgInfo()	Get information about an algorithm.

Server_ Functions (Called by ARM-side Application)	
Function	Description
Engine_getServer()	Get the handle to a Server
Server_getNumMemSegs()	Get the number of heaps in a Server
Server_getMemStat()	Get statistics about a Server heap
Server_redefineHeap()	Set base and size of a Server heap
Server_restoreHeap()	Reset Server heap to default base and size

Memory_ Functions	
Function	Description
Memory_contigAlloc()	Allocate physically contiguous blocks of memory
Memory_contigFree()	Free memory allocated by Memory_contigAlloc()

Local Codecs: Building an Engine

Introduction

In this chapter the steps to build a local engine will be considered.

Learning Outline

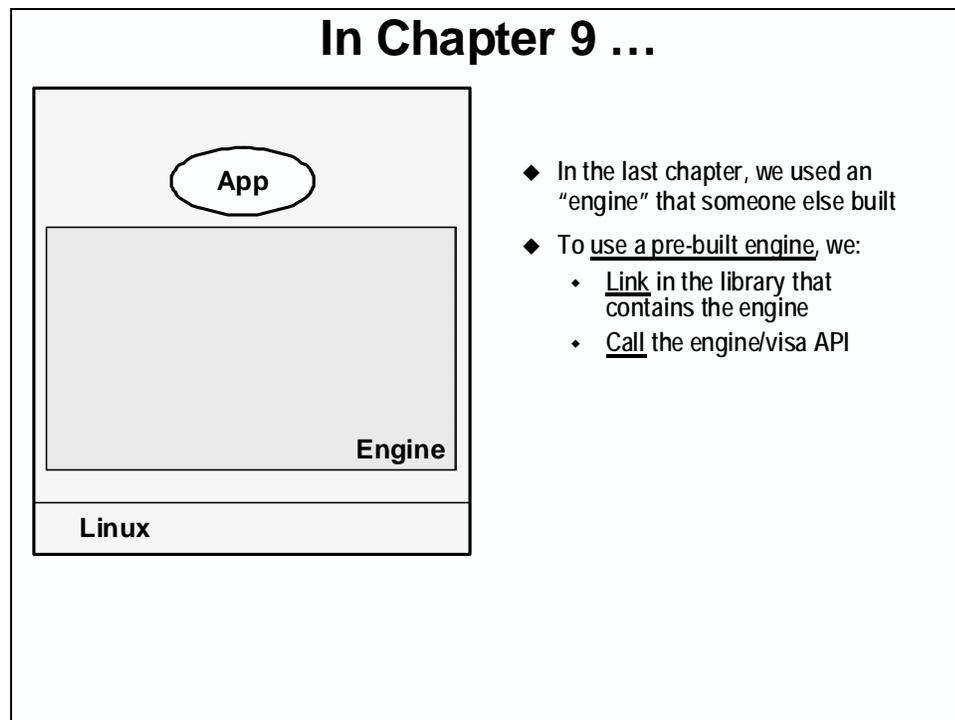
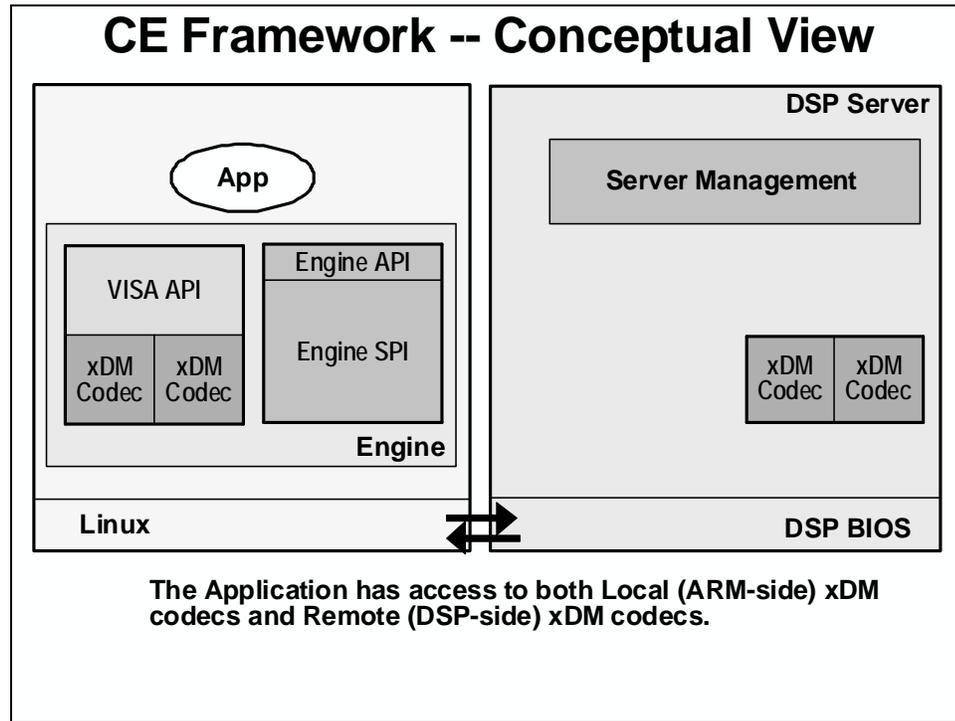
Topics covered in this chapter will include:

- Review of the Codec Engine Framework
- Review the Configuro tool
- Describe “Engine” configuration details
- (Optional) Build of an engine deliverable

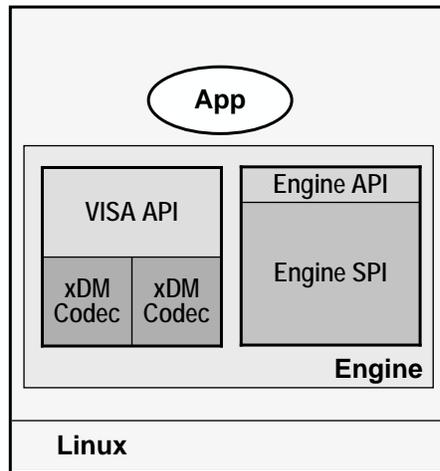
Chapter Topics

Local Codecs: Building an Engine	10-1
<i>CE Framework Review</i>	<i>10-3</i>
<i>Engine Configuration Details</i>	<i>10-6</i>
<i>(Optional) Building an Engine Deliverable</i>	<i>10-9</i>

CE Framework Review



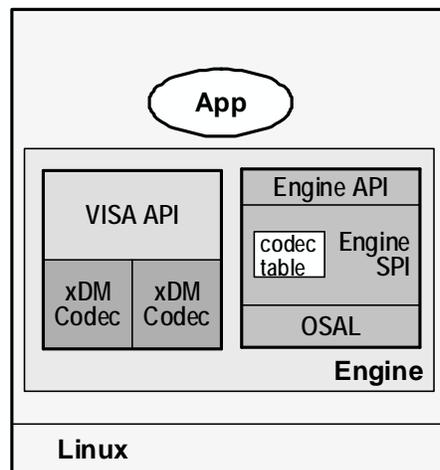
In Chapter 10 ...



Two goals for chapter 10:

1. Build our app and engine together
2. Optionally, build an engine (only) library to provide to others

Local Only Engine for ARM

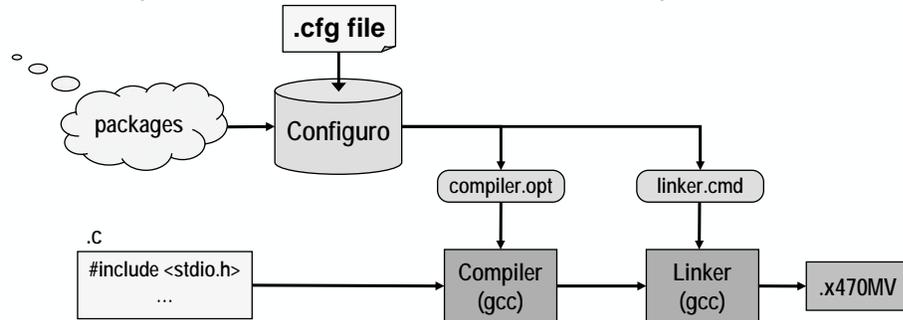


Integrates:

1. Application
2. Codecs
3. Class functions (i.e VISA)
4. Engine functions
5. Codec table
6. O/S Abstraction Layer

Review : Using Configuro

- ◆ *Configuro* is a tool that helps users consume/use packaged content.



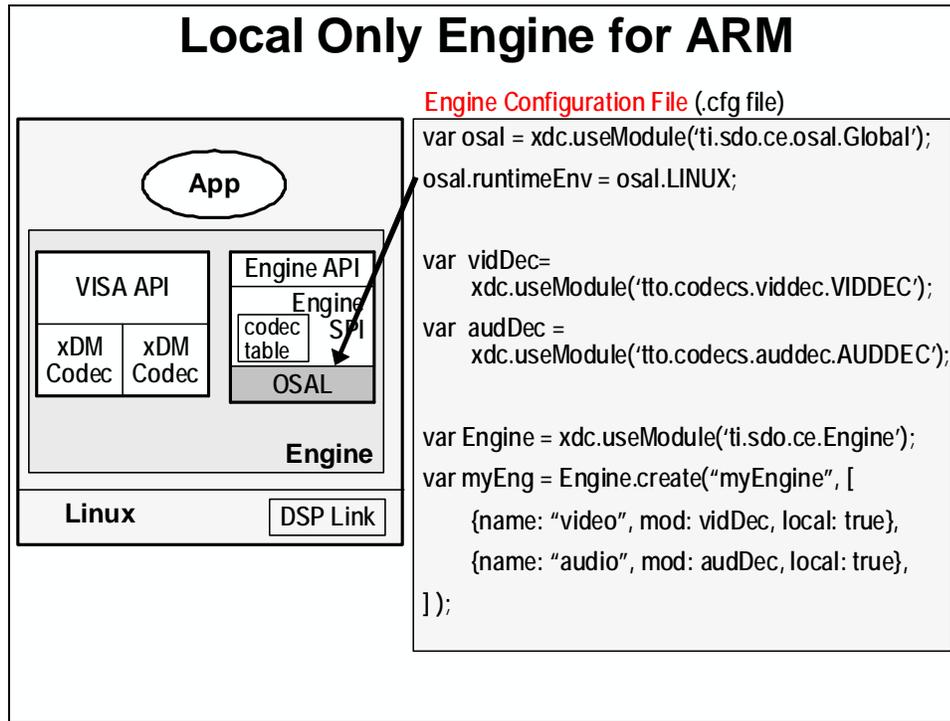
- ◆ Packages needed:

- Codecs, Class functions (i.e VISA)
- Engine functions
- etc...

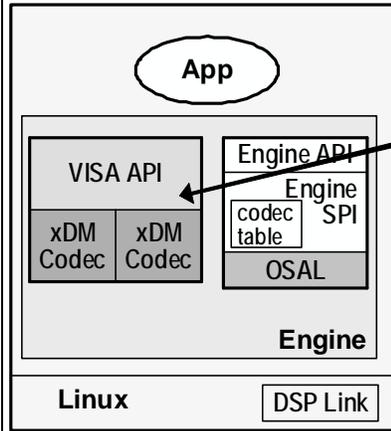
- ◆ Configuro needs four other inputs to help it perform properly:

- XDCPATH – path to Configuro tool
- Platform – e.g. ti.platforms.evmDM6446
- Target – e.g. gnu.targets.MVArm9
- .cfg – indicates which packages to include ← Which packages should we include?

Engine Configuration Details



Local Only Engine for ARM



Example: 'tto.codecs.auddec.AUDDEC'
 Package Name: tto.codecs.auddec
 Codec Module Name: AUDDEC

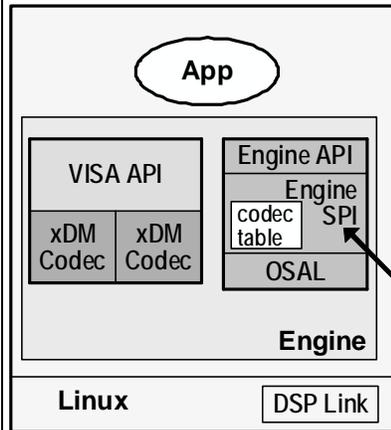
Engine Configuration File (.cfg file)

```
var osal = xdc.useModule('ti.sdo.ce.osal.Global');
osal.runtimeEnv = osal.LINUX;

var vidDec =
    xdc.useModule('tto.codecs.viddec.VIDDEC');
var audDec =
    xdc.useModule('tto.codecs.auddec.AUDDEC');

var Engine = xdc.useModule('ti.sdo.ce.Engine');
var myEng = Engine.create("myEngine", [
    {name: "video", mod: vidDec, local: true},
    {name: "audio", mod: audDec, local: true},
]);
```

Local Only Engine for ARM



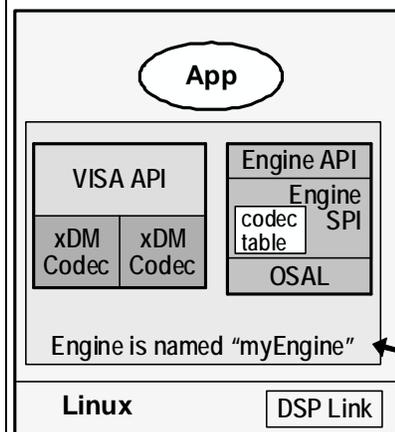
Engine Configuration File (.cfg file)

```
var osal = xdc.useModule('ti.sdo.ce.osal.Global');
osal.runtimeEnv = osal.LINUX;

var vidDec =
    xdc.useModule('tto.codecs.viddec.VIDDEC');
var audDec =
    xdc.useModule('tto.codecs.auddec.AUDDEC');

var Engine = xdc.useModule('ti.sdo.ce.Engine');
var myEng = Engine.create("myEngine", [
    {name: "video", mod: vidDec, local: true},
    {name: "audio", mod: audDec, local: true},
]);
```

Local Only Engine for ARM



Engine Configuration File (.cfg file)

```
var osal = xdc.useModule('ti.sdo.ce.osal.Global');
osal.runtimeEnv = osal.LINUX;

var vidDec =
  xdc.useModule('tto.codecs.viddec.VIDDEC');
var audDec =
  xdc.useModule('tto.codecs.auddec.AUDDEC');

var Engine = xdc.useModule('ti.sdo.ce.Engine');
var myEng = Engine.create("myEngine", [
  {name: "video", mod: vidDec, local: true},
  {name: "audio", mod: audDec, local: true},
]);
```

Engine and Algorithm Names

Engine configuration file (.cfg)

```
var myEng = Engine.create("myEngine", [
  {name: "video", mod: vidDec, local: true},
  {name: "audio", mod: audDec, local: true},
]);
```

Application Source File (app.c)

```
CERuntime_init();
myCE = Engine_open("myEngine", myCEAttrs);
myAE = AUDDEC_create(myCE, "audio", params);

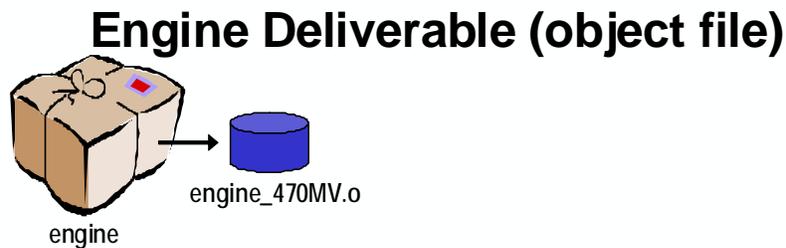
AUDDEC_control(myAE, ...);
AUDDEC_process(myAE, ...);

AUDDEC_delete(myAE);
Engine_close(myCE);
```

(Optional) Building an Engine Deliverable

Here are some rough notes about building a stand-alone engine that was provided for you (and that you used) in the Lab 9 exercises.

Note, we built our engine-only executable using the TI XDC tools. We will discuss how to use these tools in Chapter 12. The optional Lab09b contains the files needed to make the engine-only build using the these tools.



- ◆ Pkg.addExecutable was intended for building executables
- ◆ An ARM executable (e.g. engine_app.x470MV) includes: application code + engine
- ◆ You can create an Engine Deliverable (i.e. without an associated application) by using the linker's `-r` flag

In other words, for an Engine Deliverable, we build in the same way, but using a partial link step, and without linking an application

While stand-alone engine libraries can be built using gMake files, we used the TI XDC tools to create our stand-alone engine library. If you are interested, try looking over the RTSC build description files to see how we accomplished this.

- `_dummy.c` files
- `engine.cfg`
- `package.xdc`
- `config.bld`
- `package.bld`

Build Script for Engine Archive Deliverable

```
/* ===== package.bld =====*/
var appName = "engine_app";
var targ = MVArm9_o;
var sources = [ "audio_decoder_dummy.c",
               "audio_encoder_dummy.c", ... ];

Pkg.addExecutable( appName, targ, "ti.platforms.evmDM6446",
{
    cfgScript: "engine.cfg",
    profile: "debug",
    lopts: "-Wl,-r -nostdlib",
}).addObjects( sources );
```

- ◆ Specify MVArm9_o target to indicate we are building an Arm engine object file
- ◆ Application source files replaced with dummy routines
- ◆ -Wl,r indicates options are to be used by the Linker
 - -r option provides partial link into object file (i.e. builds a relocatable object file)
- ◆ -nostdlib : do not use standard C initialization routines; rather, they will be included by the final end-application build

Dummy Functions

audio_decoder_dummy.c

```
void __audio_decoder_dummy(void) {
    // declarations

    decoderHandle = AUDDEC_create(engineHandle, "invalid", NULL);
    AUDDEC_process(decoderHandle, &inBufDesc, &outBufDesc, &inArgs, &outArgs);
    AUDDEC_control(decoderHandle, XDM_GETSTATUS, &dynParams, &status);
    AUDDEC_delete(decoderHandle);
}
```

- ◆ **Linker will not link functions from libraries unless they are called from within a source file.**
- ◆ **Dummy functions exercise the 4 basic VISA API to ensure all functions are linked into engine deliverable**
- ◆ **Some engine providers will provide diagnostic routines instead of dummy functions for added functionality.**

Remote Codecs: Given a DSP Server

Introduction

In this chapter the steps required to use a given DSP server will be examined

Learning Outline

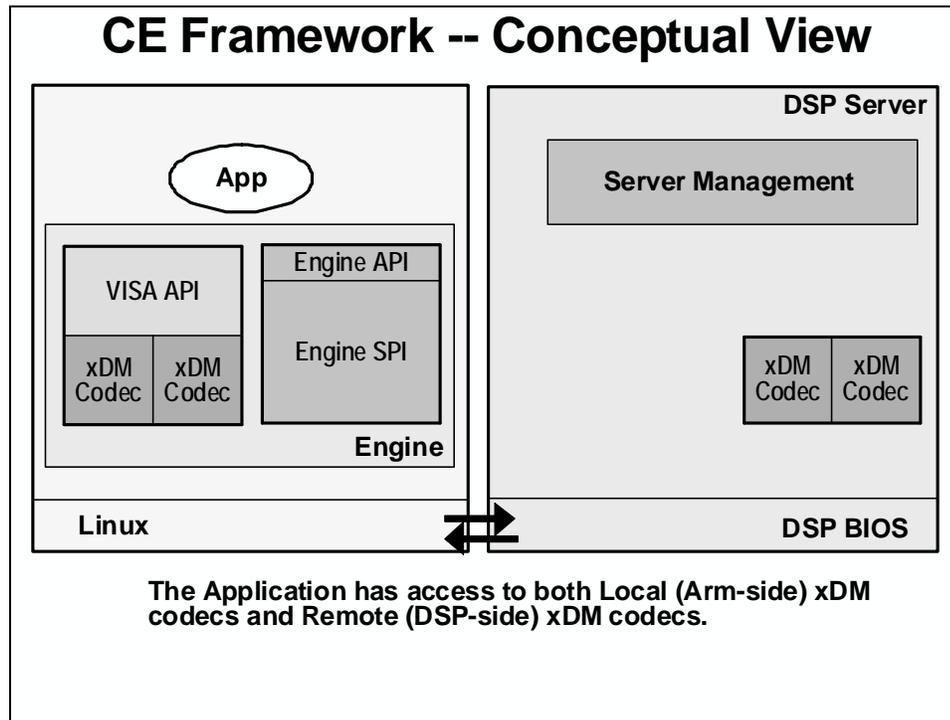
In this chapter the following topics will be presented:

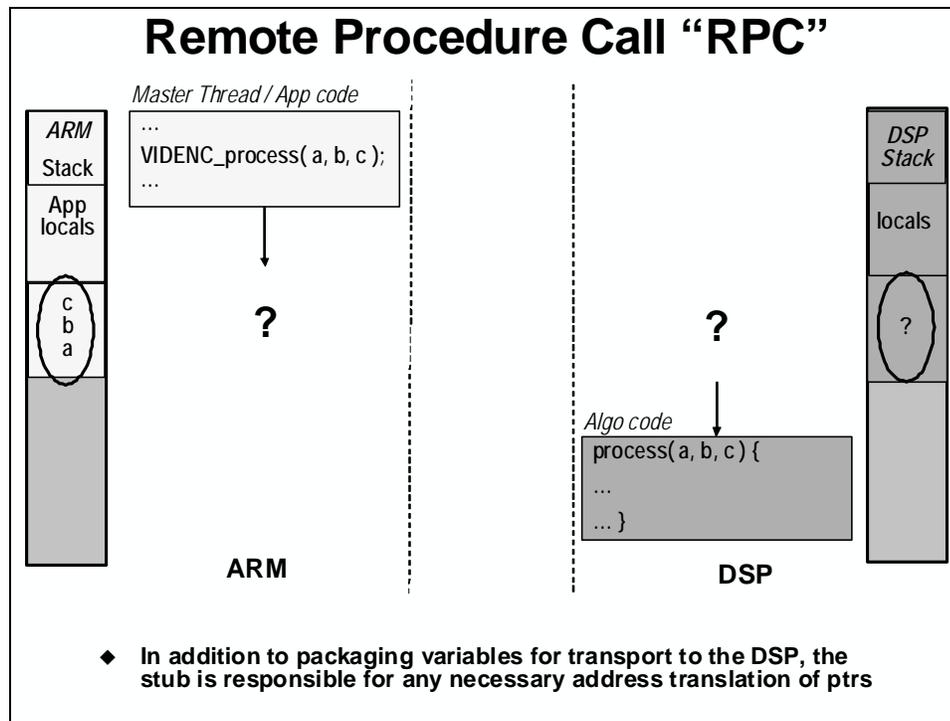
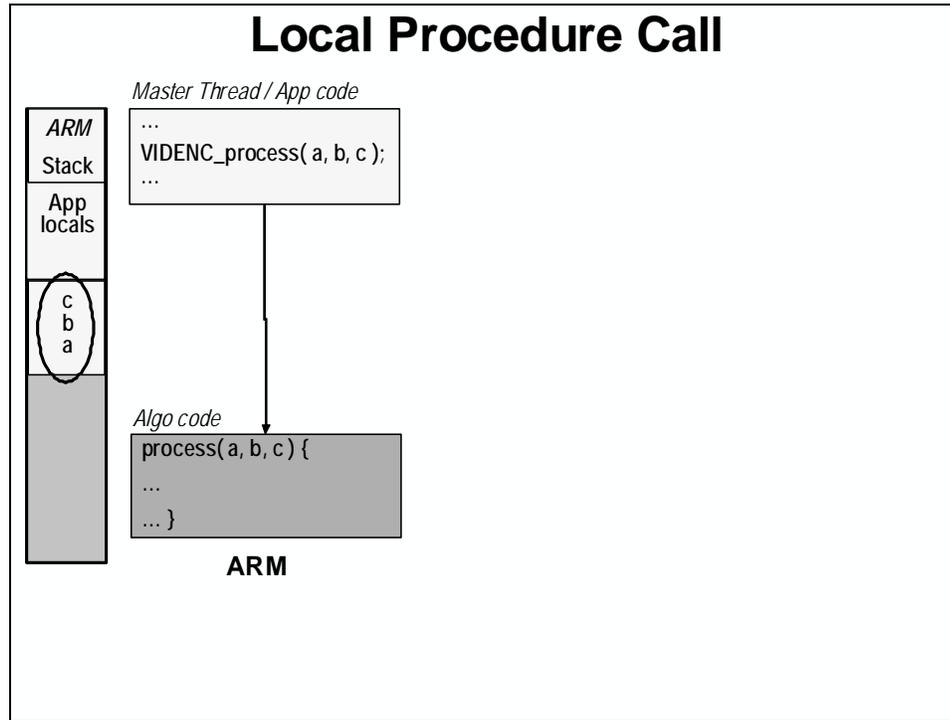
- Remote codec calls
- Server management layer
- Internal mechanisms within the Codec Engine
- Modifying the Engine to use a Remote Server

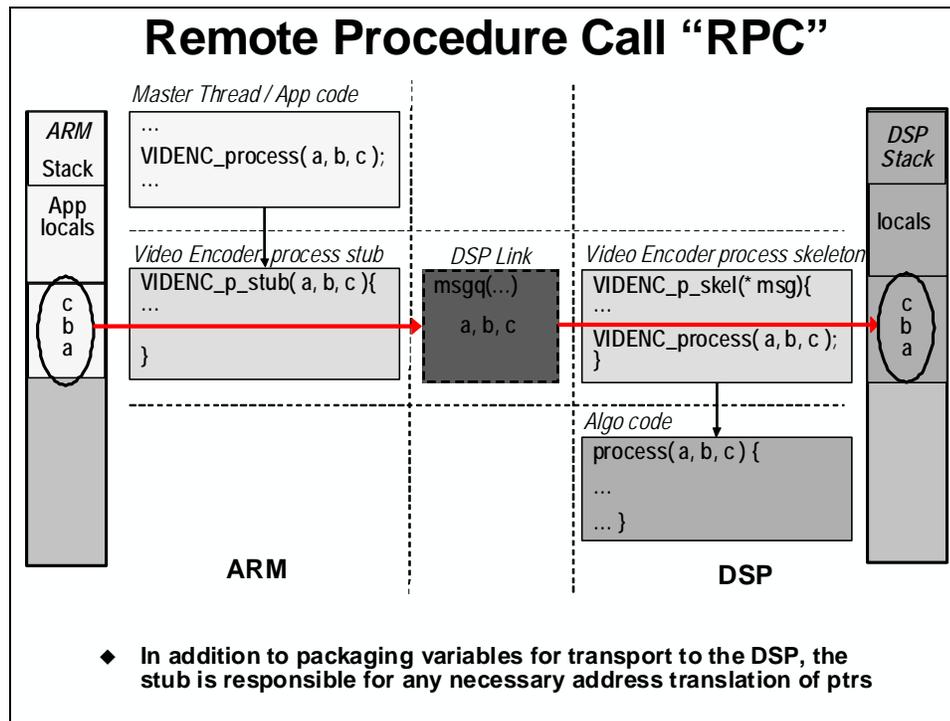
Chapter Topics

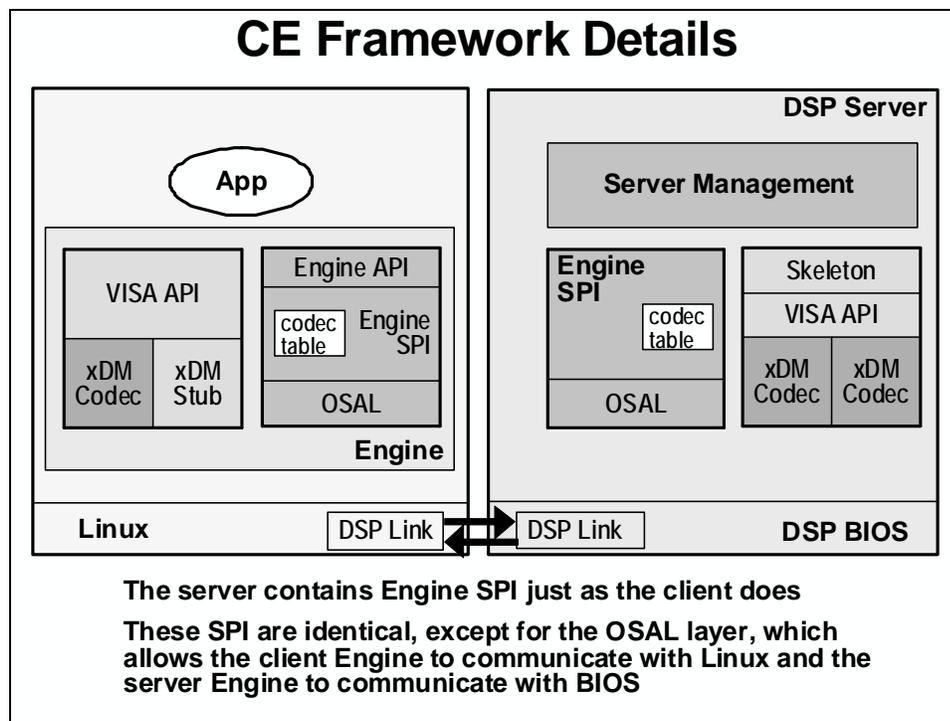
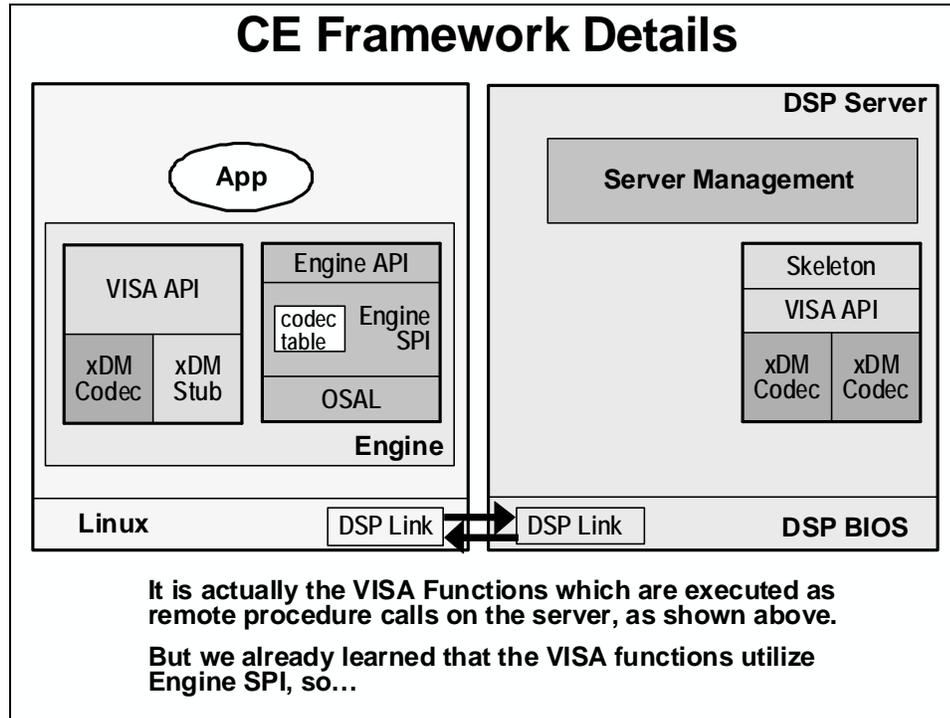
Remote Codecs: Given a DSP Server	11-1
<i>Remote Codec Calls.....</i>	<i>11-2</i>
<i>Server Management Layer (VISA functions).....</i>	<i>11-6</i>
<i>Priorities on the DSP Server.....</i>	<i>11-7</i>
<i>Modifying the Engine to use a Remote Server</i>	<i>11-9</i>
Append Algo's to an Engine created From Server	11-11
<i>Contiguous Memory Functions.....</i>	<i>11-12</i>

Remote Codec Calls

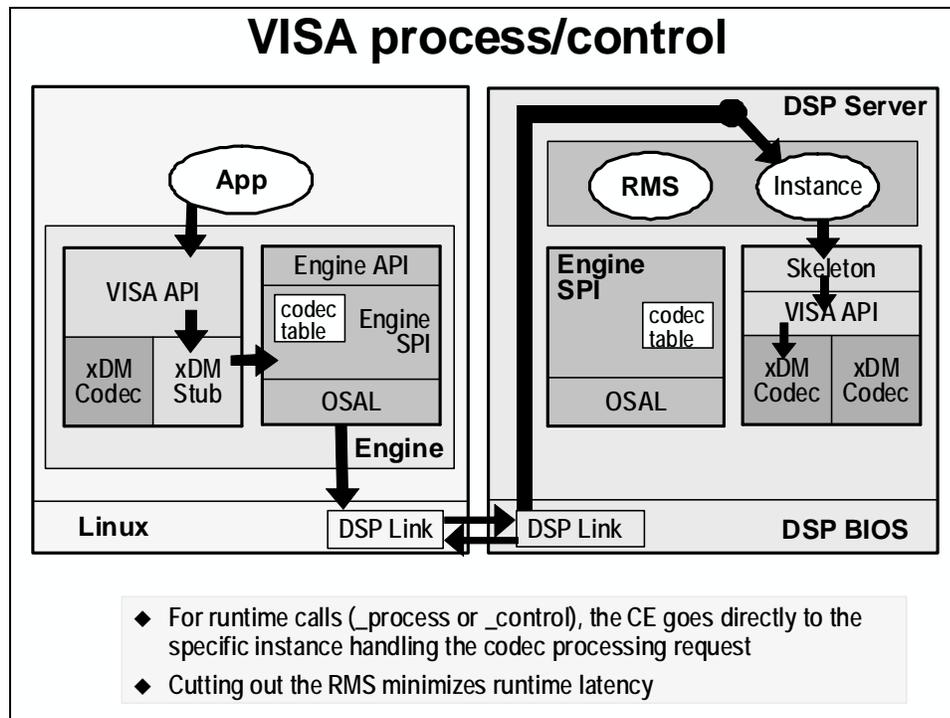
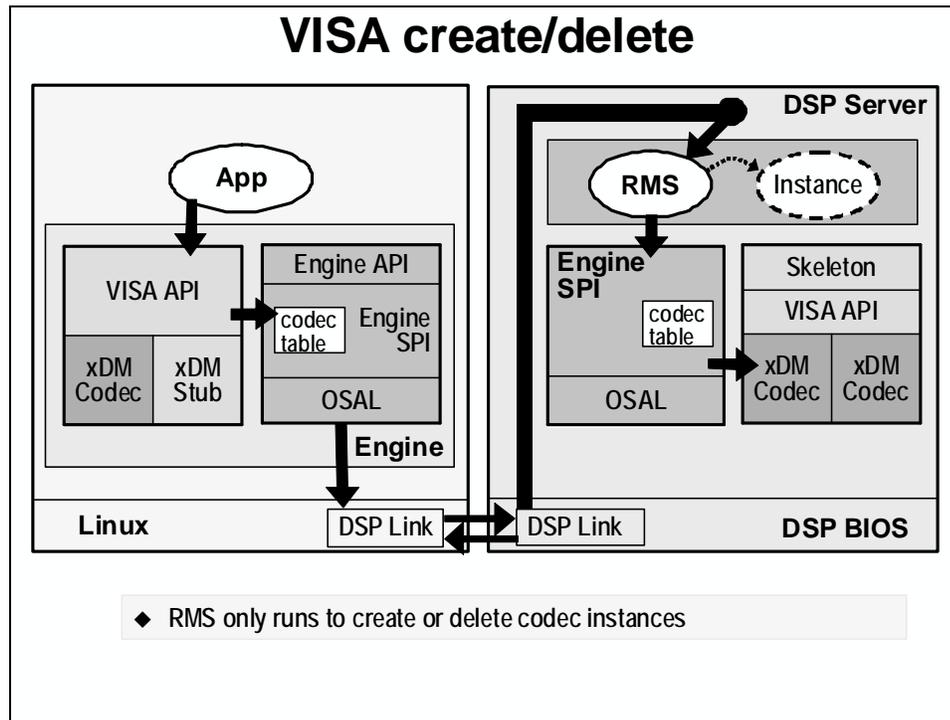




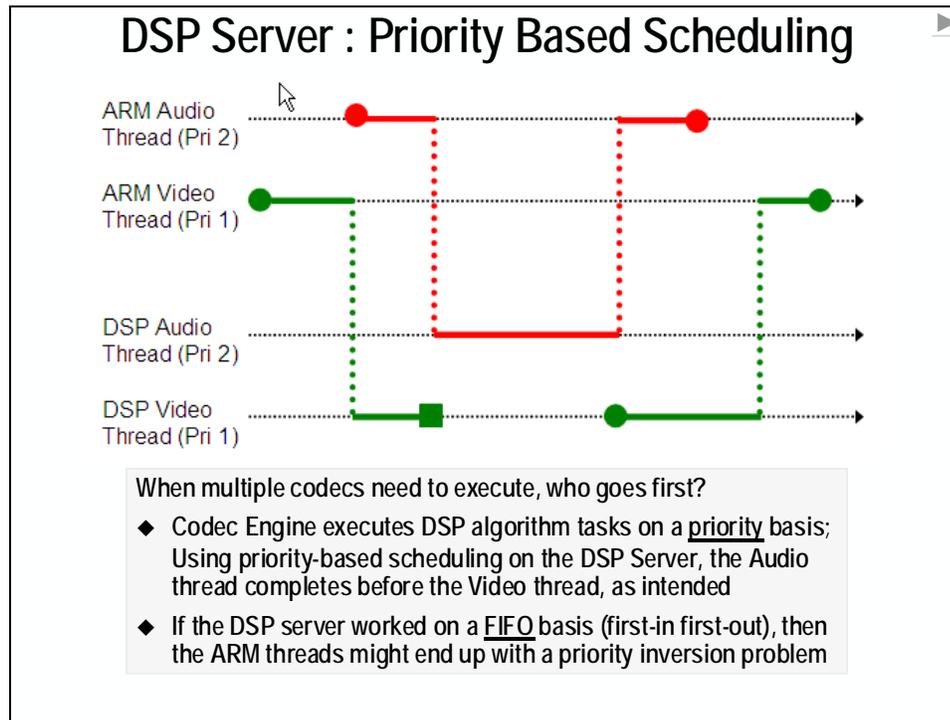




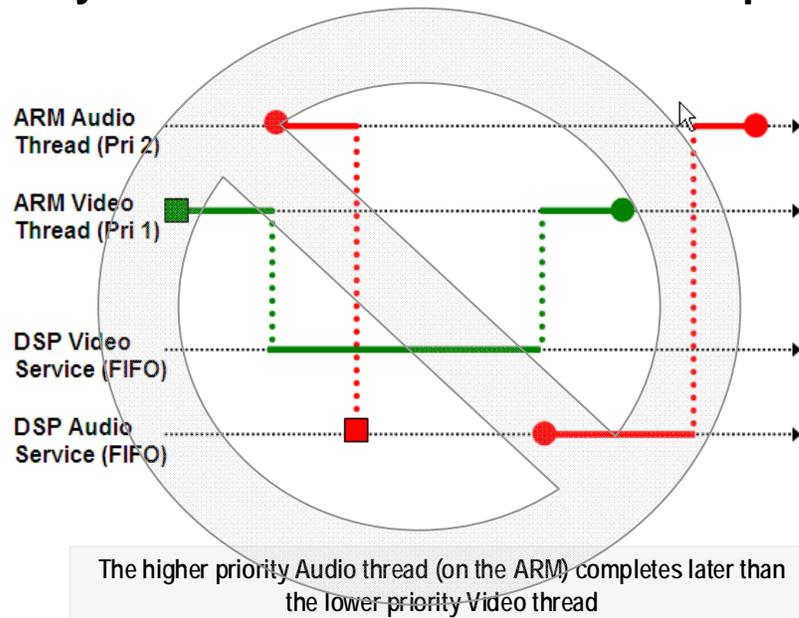
Server Management Layer (VISA functions)

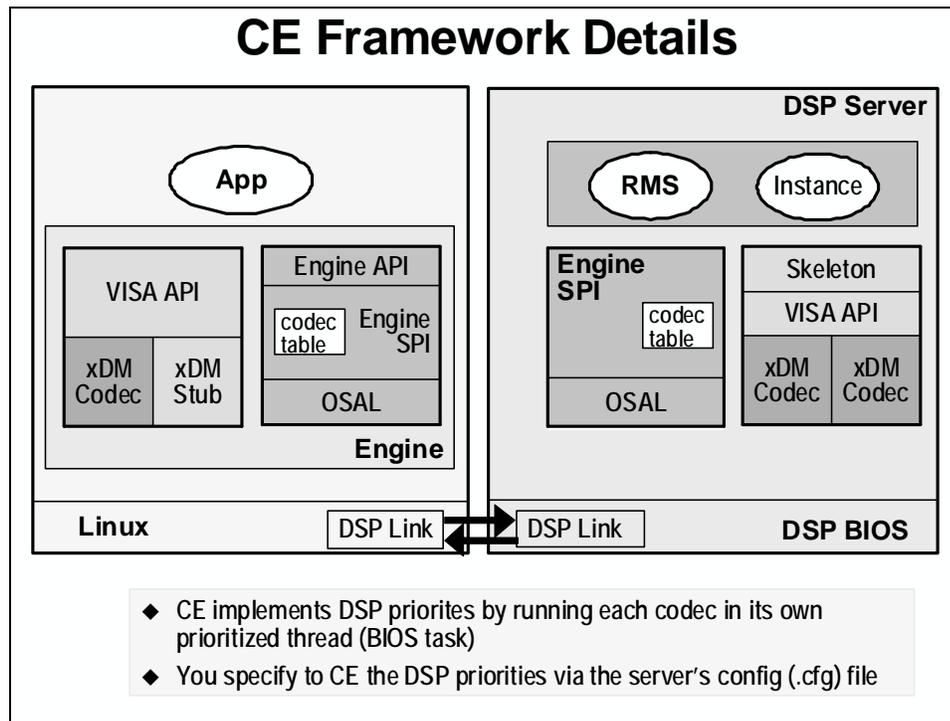


Priorities on the DSP Server



Priority Inversion from FIFO Server Response





Modifying the Engine to use a Remote Server

Changing Engine to use a Server

Engine.cfg Configuration File

```

var osal = xdc.useModule('ti.sdo.ce.osal.Global');
osal.runtimeEnv = osal.DSPLINK_LINUX;

var audEnc1 =
    xdc.useModule('codecs.audenc1.AUDENC1');
var audEnc2 =
    xdc.useModule('codecs.audenc2.AUDENC2');

var Engine = xdc.useModule('ti.sdo.ce.Engine');
var myEng = Engine.create("myEngine", [
    {name: "myEnc1", mod: audEnc1, local: false},
    {name: "myEnc2", mod: audEnc2, local: false},
]);

myEng.server = " ./myServer.x64P";
    
```

- ◆ Set Codec properties to local=false to tell them to remotely on the DSP server.
- ◆ **Warning:** The module and codec names must match between *engine* and *server*!
- ◆ If using remote codecs (local: false), specify the DSP executable file name in the engine config.
- ◆ When Engine_open() is called, the ARM-side engine automatically loads the server image onto the DSP

Better yet ...

Create Engine from Server ▶

Old Method

```

var osal = xdc.useModule('ti.sdo.ce.osal.Global');
osal.runtimeEnv = osal.DSPLINK_LINUX;

var vidDec = xdc.useModule('codecs.viddec.VIDDEC');
var vidEnc = xdc.useModule('codecs.videnc.VIDENC');

var Engine = xdc.useModule('ti.sdo.ce.Engine');
var myEng = Engine.create("myEngine", [
    {name: "myEnc1", mod: vidDec, local: false},
    {name: "myEnc2", mod: vidEnc, local: false},
]);

myEng.server = " ./myServer.x64P";
    
```

New Method is:

- ◆ Shorter (i.e. easier)
- ◆ Less error prone
- ◆ Configuro can extract additional info from package metadata (e.g. memory map)

New Method

```

var osal = xdc.useModule('ti.sdo.ce.osal.Global');
osal.runtimeEnv = osal.DSPLINK_LINUX;

var Engine = xdc.useModule('ti.sdo.ce.Engine');
var myEng = Engine.createFromServer (
    "myEngine",           // Engine name (as referred to in the C app)
    "./myServer.x64P",   // Where to find the .x64P file, inside the server's package
    "to.servers.example" ); // Server's package name
    
```

If you are not planning to put the ARM (.xv5T) and DSP (.x64P) executables into the same directory in the ARM/Linux filesystem, then you should set the `.server` property of your engine. You can see this in the following slide, though we won't need to do this in our upcoming lab exercise – though it doesn't hurt to add it to your configuration file.

Create Engine from Server

Old Method

```
var osal = xdc.useModule('ti.sdo.ce.osal.Global');
osal.runtimeEnv = osal.DSPLINK_LINUX;

var vidDec = xdc.useModule('codecs.viddec.VIDDEC');
var vidEnc = xdc.useModule('codecs.videnc.VIDENC');

var Engine = xdc.useModule('ti.sdo.ce.Engine');
var myEng = Engine.create("myEngine", [
    {name: "myEnc1", mod: vidDec, local: false},
    {name: "myEnc2", mod: vidEnc, local: false},
]);

myEng.server = "./myServer.x64P";
```

New Method is:

- ◆ Shorter (i.e. easier)
- ◆ Less error prone
- ◆ Configuro can extract additional info from package metadata (e.g. memory map)

New Method

```
var osal = xdc.useModule('ti.sdo.ce.osal.Global');
osal.runtimeEnv = osal.DSPLINK_LINUX;

var Engine = xdc.useModule('ti.sdo.ce.Engine');
var myEng = Engine.createFromServer(
    "myEngine", // Engine name
    "./myServer.x64P", // Where to find the .x64P file, inside the server's package
    "tto.servers.example"); // Server's package name

myEng.server = "./myServer.x64P"; // Loc'n of server exe at runtime, relative to ARM program;
// only needed if not found in the same folder as ARM
```

myEng.server is not needed in our upcoming lab, since we'll put both executables (.xv5T, .x64P) into the `/opt/workshop` directory

Append Algo's to an Engine created From Server

Append Local Algo's to Engine.createFromServer()

```

var osal = xdc.useModule( 'ti.sdo.ce.osal.Global' );
osal.runtimeEnv = osal.DSPLINK_LINUX;

var Engine = xdc.useModule( 'ti.sdo.ce.Engine' );
var myEng = Engine.createFromServer (
    "myEngine",           // Engine name (as referred to in the C app)
    "./myServer.x64P",   // Where the .x64P file is, inside the server's pkg
    "tto.servers.example" ); // Server's package name

var audDec = xdc.useModule ( 'codecs.auddec.AUDDEC' );
var audEnc = xdc.useModule ( 'codecs.audenc.AUDENC' );

myEng.algs[ myEng.algs.length++ ] = { name: "auddec", mod: audDec, local: true };
myEng.algs[ myEng.algs.length++ ] = { name: "audenc", mod: audEnc, local: true };

```

Add local algo's, even when using .createFromServer()

- ◆ Reference the codec's package (i.e. xdc.useModule)
- ◆ Append each codec to the Engine's array of algorithms
- ◆ Remember to set "local" to "true"

See the following wiki page for more details:

http://processors.wiki.ti.com/index.php/Configuring_Codec_Engine_in_Arm_apps_with_createFromServer

Contiguous Memory Functions

Allocating Contiguous Memory Buffers	
Function	Description
Memory_contigAlloc()	Allocate physically contiguous blocks of memory ptr Memory_contigAlloc (unsigned int size, unsigned int align)
Memory_contigFree()	Free memory allocated by Memory_contigAlloc() bool Memory_contigFree (void *addr, unsigned int size)

Linux memory allocations may actually be non-contiguous in physical memory – that's the advantage of relying on an MMU (memory mgmt unit). Since most DSP's do not use an MMU, any buffers passed to the DSP must be physically contiguous. The Linux CMEM driver (found in the linuxutils package) performs contiguous buffer allocations

Example / Comparison:

Standard C syntax	Using Memory_contig() functions
<pre>#include <stdlib.h> #define SIZE 128 a = malloc(SIZE); a = {...}; use buffer... free(a);</pre>	<pre>#include <ti/sdo/ce/osal/Memory.h> #define SIZE 128 a = Memory_contigAlloc(SIZE, ALIGN); a = {...}; use buffer... Memory_contigFree(a, SIZE);</pre>

In our upcoming lab exercise we'll need to change to contiguous memory allocations in our ARM/Linux C code so that the buffers we pass to the DSP will be continuous physically, which the DSP requires.

Also, since the buffers will come from the CMEM region of the memory map, they'll also be accessible to ARM+DSP devices that wrap a MMU around the DSP, like the OMAP3530.

We could have easily used the Memory_contigAlloc() functions for our earlier lab exercises, but we wanted to highlight the change in this chapter/lab, since this is where it becomes required.

Remote Codecs: Building a DSP Server

Introduction

In the previous chapter you built an engine that contained a DSP Server. This DSP Server was provided for you, as if you had purchased one from a TI Third Party or obtained it from another team within your company.

In this chapter, we examine the steps required to build a DSP Server. Once built, you will incorporate this into an engine as was done in the previous chapter.

As a final note, why are these codecs referred to as “Remote Codecs”? You might remember, from the application’s perspective, Codecs running on the DSP (i.e. contained in the DSP Server) are considered to be running remote from the Linux/ARM application.

Learning Objectives

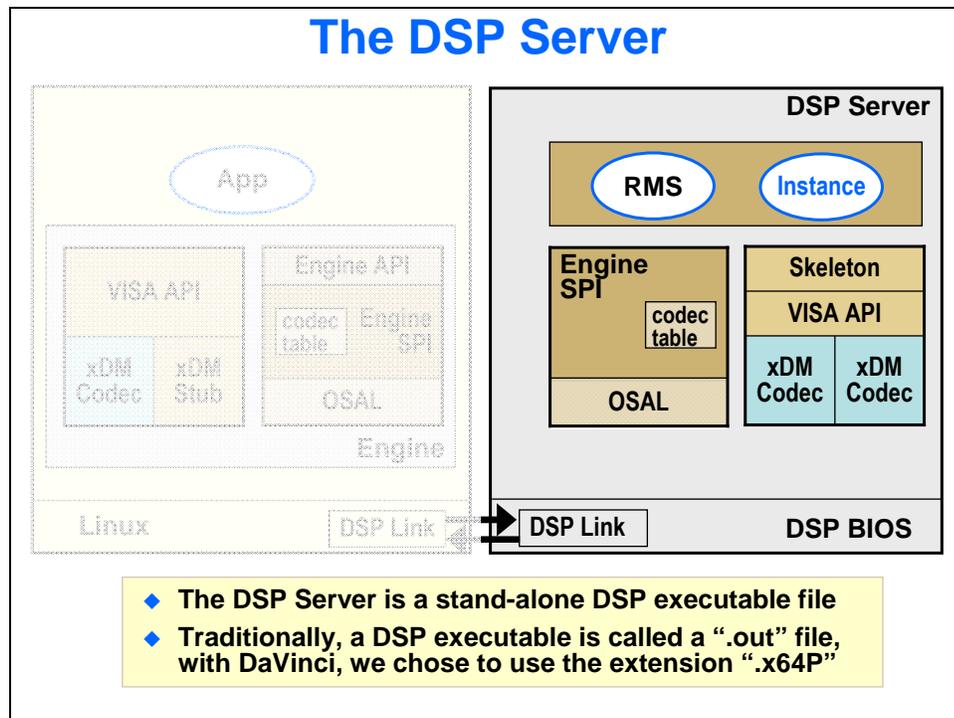
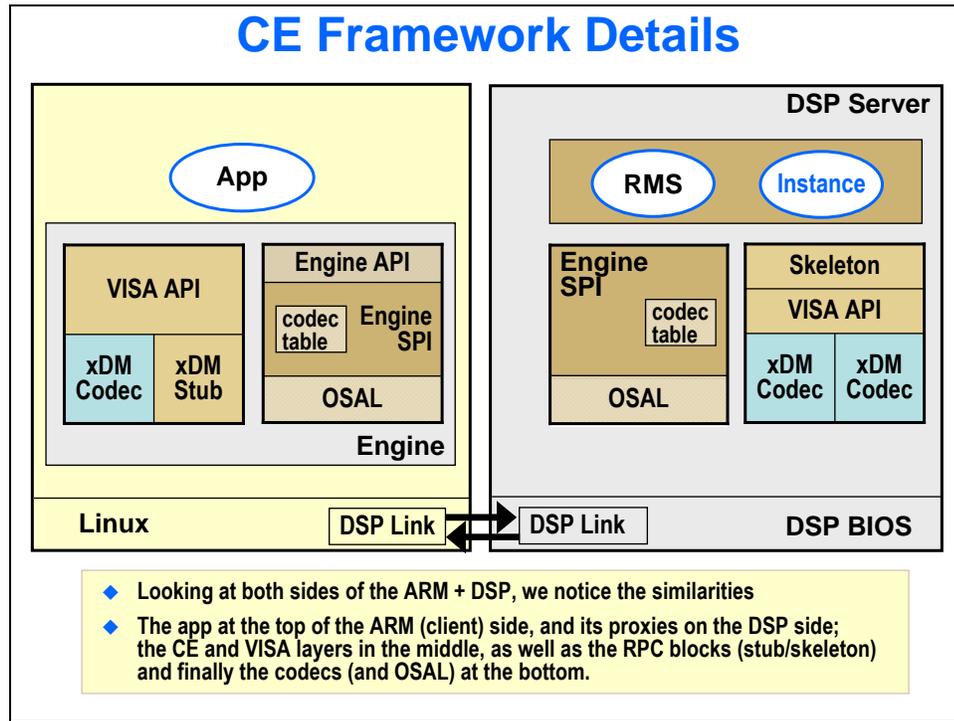
Learning Objectives

- ◆ Review basic codec engine architecture
- ◆ Describe the files required to build a DSP Server
- ◆ Use the Codec Engine’s DSP Server Wizard
- ◆ Describe how CE/DSKT2 manages xDAIS algos

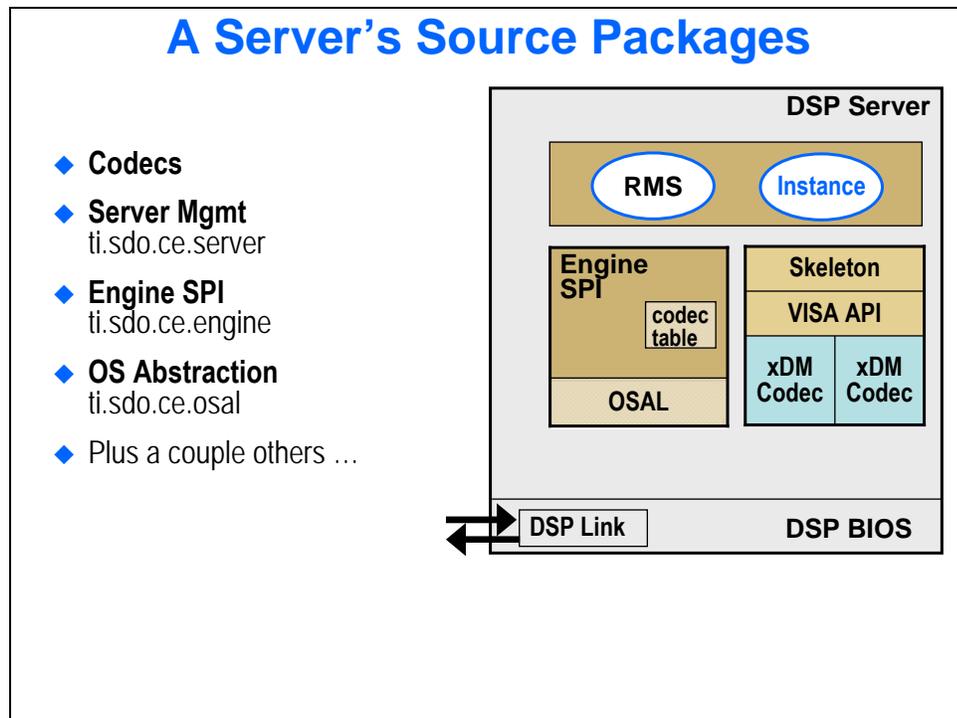
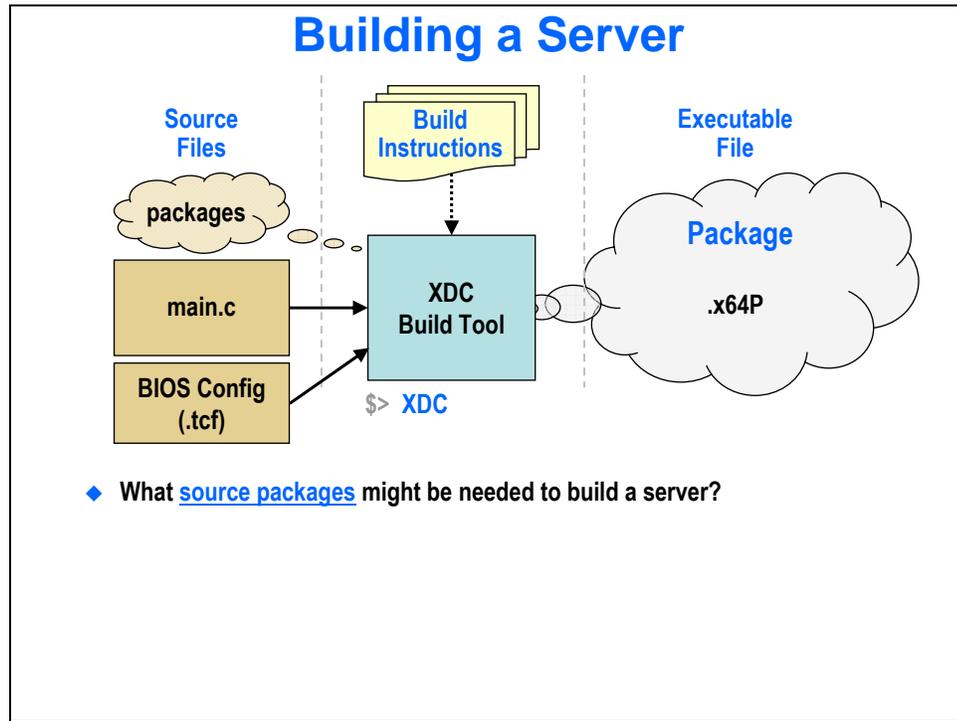
Chapter Topics

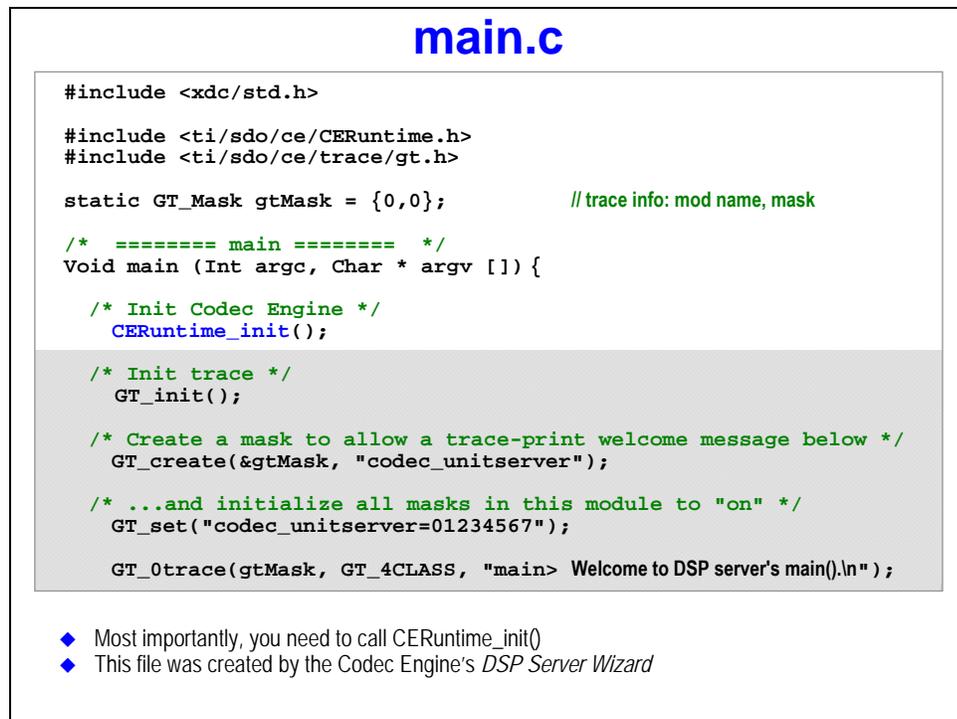
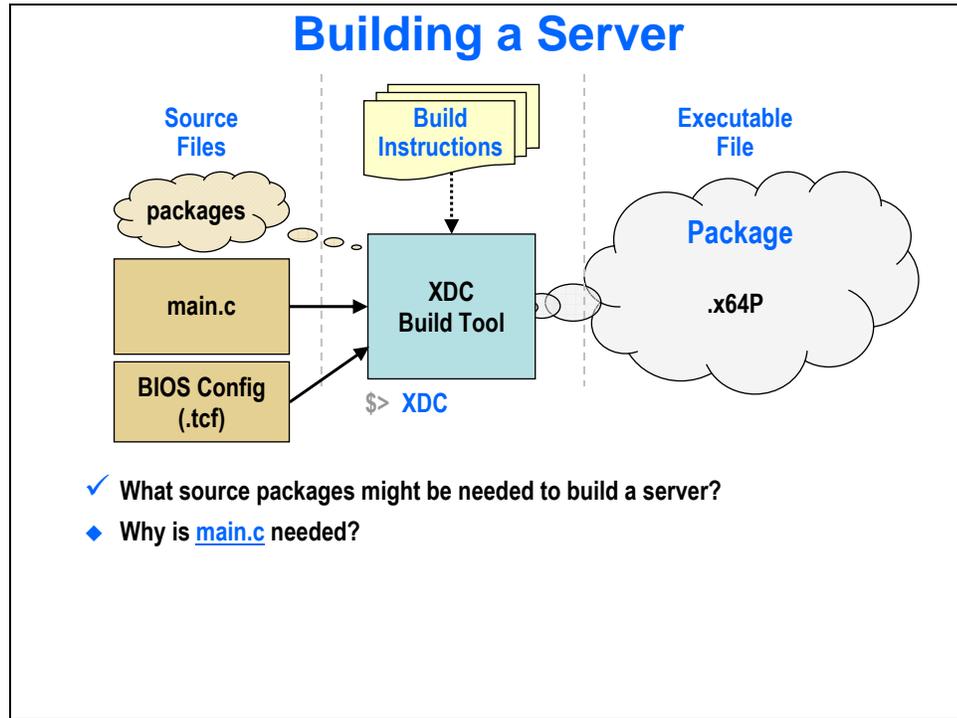
Remote Codecs: Building a DSP Server	12-1
<i>Codec Engine Review</i>	<i>12-3</i>
<i>What Makes Up a DSP Server</i>	<i>12-4</i>
<i>DSP Server Wizard</i>	<i>12-8</i>
<i>Building a Server – Config (.cfg) Files</i>	<i>12-10</i>
server.cfg	12-10
codec.cfg.....	12-11
Memory – Scratch Groups.....	12-13
Back to server.cfg ... and Memory Setup.....	12-14
Sidebar – Minimum Scratch Group Allocations.....	12-18
<i>(Optional) A Short Course in XDC</i>	<i>12-19</i>

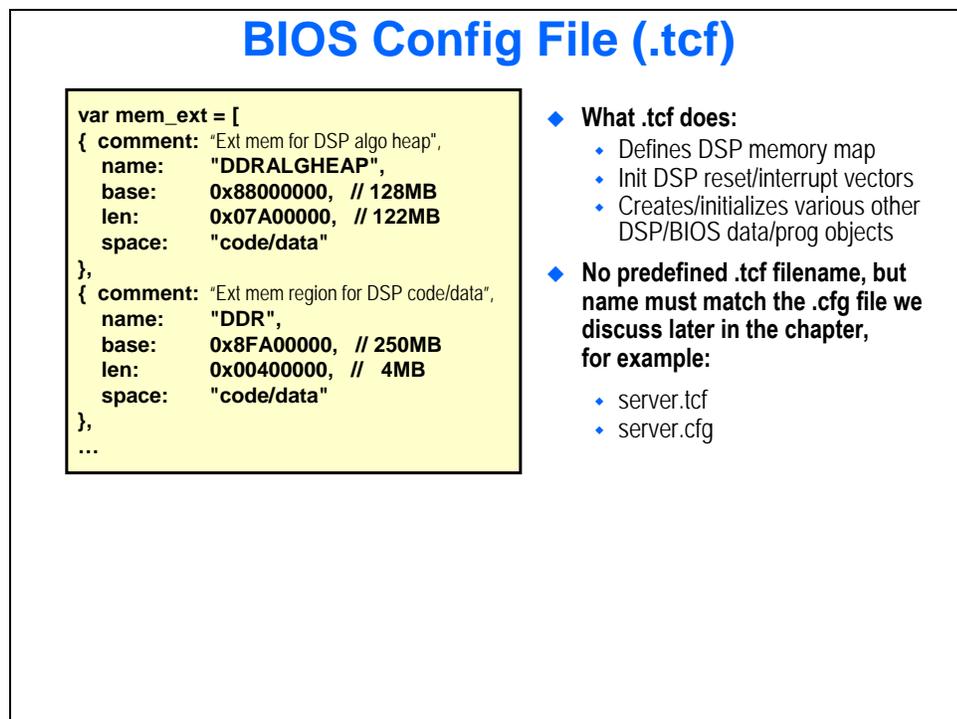
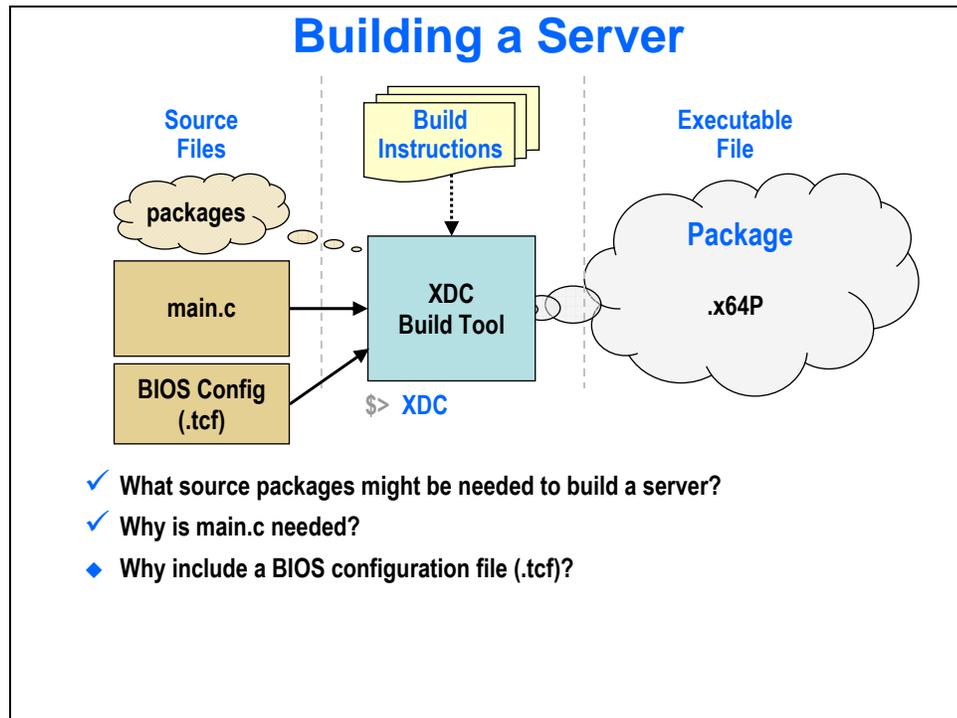
Codec Engine Review

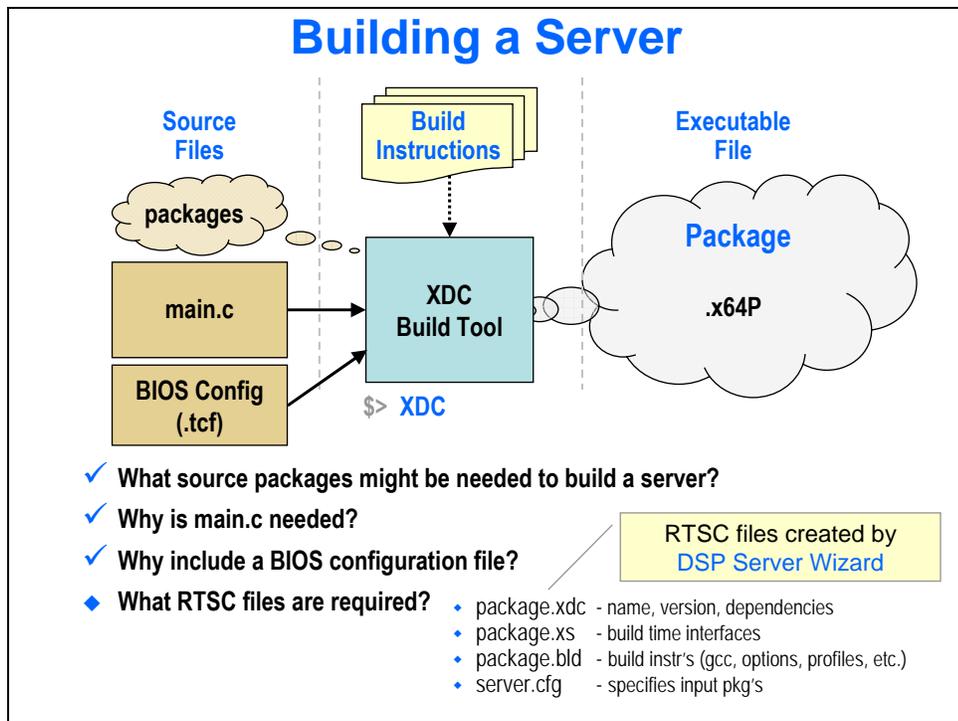
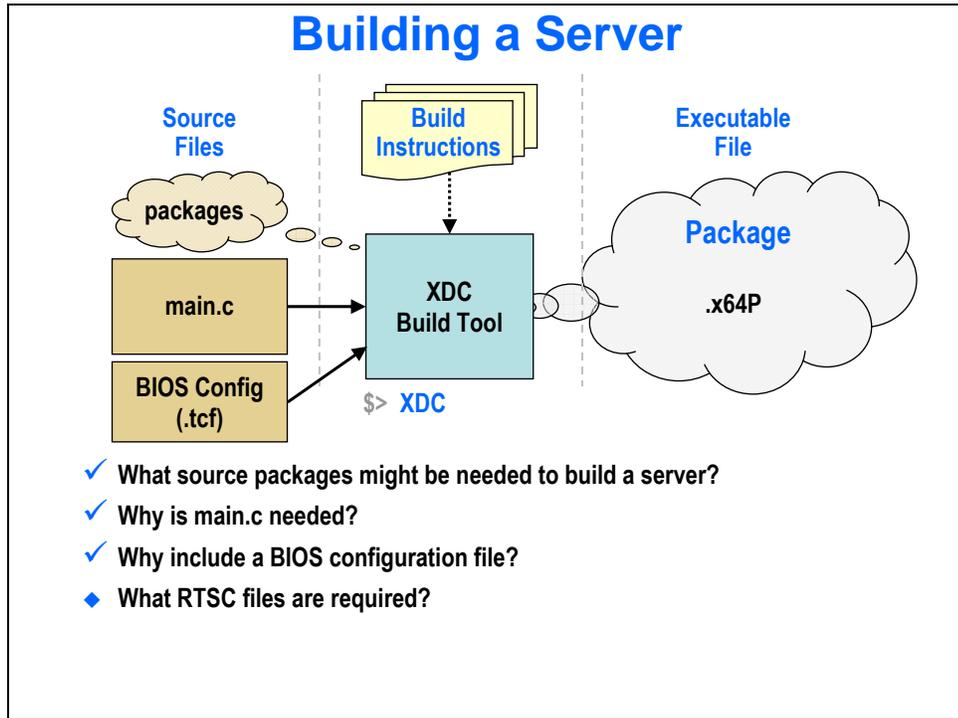


What Makes Up a DSP Server









DSP Server Wizard

Build DSP Server Using Codec Engine Wizard

- ◆ Choose Platform:
- ◆ Package Name
- ◆ Output Location (repository)
- ◆ Algorithms/Codecs in your DSP server

By default, wizard finds all algo's on your XDCPATH

Set Pkg Search Path allows you to vary which folders to search for algorithms

Clicking Next...

Build DSP Server Using Codec Engine Wizard

- ◆ Pick "algo name" used in VIDDEC_create()
- ◆ Choose priority via slider
- ◆ By default, groupings follow priority
- ◆ Override defaults for Stack Location and Stack Size:

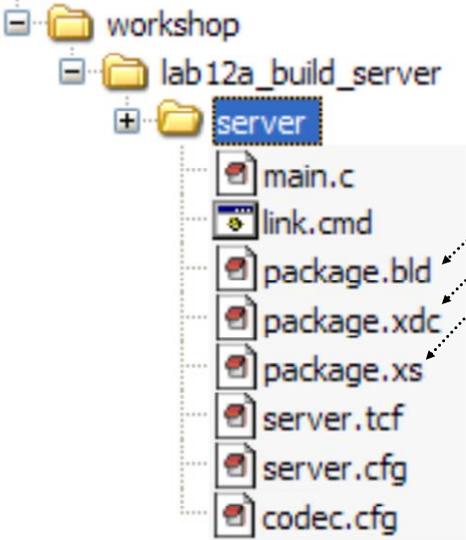
Optional values

StackMemId: 0

StackSize: 2000

What does the wizard create?

Files for DSP Server



- ◆ **Generic main.c**
You probably won't need to change it
- ◆ **Empty link.cmd**
In case you need special linker options
- ◆ **RTSC Package files**
 - ◆ How to build the package
 - ◆ Defines package's contents
 - ◆ Methods XDC tools can call
- ◆ **BIOS config file server.tcf**
Defines memory and other objects as it would for any BIOS program
- ◆ **RTSC config file server.cfg**
 - ◆ Platform default server configuration
 - ◆ Well commented
 - ◆ Name matches .tcf filename
- ◆ **RTSC config file codec.cfg**
 - ◆ Included by server.cfg
 - ◆ Algorithm specific package info

Building a Server – Config (.cfg) Files

RTSC Config File (.cfg)

The RTSC configuration file (.cfg) allows you to specify which packages to include, as well as setting their configuration options

server.cfg

codec.cfg

- 1. General Setup**
 - ◆ Add OSAL module
 - ◆ Add server module
 - ◆ Include codec.cfg
- 2. Memory Setup (DSKT2 module)**
 - ◆ Associate IALG memory types with available target memory
 - ◆ Specify default scratch group memory sizes
- 3. DMA Setup (DMAN3) (discussed in chapter 14)**
 - ◆ Specify number of DMA channels DMAN3 may manage
 - ◆ Specify number of TCCs and Params DMAN3 can offer to algos

4. CODEC Setup:

- ◆ Add module reference for each codec
- ◆ Configure properties for each codec thread:
 - ◆ Thread priority
 - ◆ Stack size
 - ◆ Stack memory location

[Let's look at the server.cfg file...](#)

server.cfg

Setup OSAL (server.cfg)

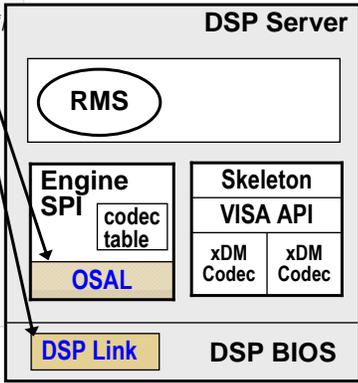
```

/* ===== set up OSAL ===== */
var osalGlobal = xdc.useModule("ti.sdo.ce.osal.Global");
osalGlobal.runtimeEnv = osalGlobal.DSPLINK_BIOS;

/* ===== Server Configuration ===== */
var Server = xdc.useModule("ti.sdo.ce.Server");

/* The server's stackSize. More than we need... but safe. */
/* And, the servers execution priority */
Server.threadAttrs.stackSize = 16384;
Server.threadAttrs.priority = Server.MINPRI;

/* Import the algorithm's configuration */
utils.importFile("codec.cfg");
    
```



Arrows from the code above point to the OSAL and DSP Link components in the diagram.

Server Configuration (server.cfg)

```

/* ===== set up OSAL ===== */
var osalGlobal = xdc.useModule('ti.sdo.ce.osal.Global');
osalGlobal.runtimeEnv = osalGlobal.DSPLINK_BIOS;

/* ===== Server Configuration ===== */
var Server = xdc.useModule('ti.sdo.ce.Server');

/* The server's stackSize. More than we need... but safe. */
/* And, the servers execution priority */
Server.threadAttrs.stackSize = 16384;
Server.threadAttrs.priority = Server.MINPRI;

/* Import the algorithm's configuration */
utils.importFile("codec.cfg");
    
```

Looking more closely at 'codec.cfg'...

codec.cfg

Use Codec Modules (codec.cfg)

```

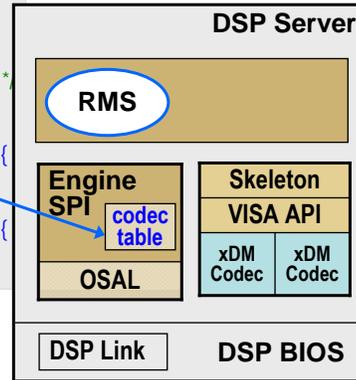
/* ===== Use various codec modules ===== */
var VIDDEC_COPY =
    xdc.useModule('codecs.viddec_copy.VIDDEC_COPY');
var VIDENC_COPY =
    xdc.useModule('codecs.videnc_copy.VIDENC_COPY');
/* ===== Server Configuration ===== */
Server.algs = [
    {name: "viddec_copy", mod: VIDDEC_COPY, threadAttrs: {
        stackSize: 1024, stackMemId: 0, priority: 1}},
    {name: "videnc_copy", mod: VIDENC_COPY, threadAttrs: {
        stackSize: 1024, stackMemId: 0, priority: 1}}, ];
    
```

Configure Server Algo's (codec.cfg)

```

/* ===== Use various codec modules ===== */
var VIDDEC_COPY =
    xdc.useModule('codecs.viddec_copy.VIDDEC_COPY');
var VIDENC_COPY =
    xdc.useModule('codecs.videnc_copy.VIDENC_COPY');
/* ===== Server Configuration ===== */
Server.algs = [
    {name: "viddec_copy", mod: VIDDEC_COPY, threadAttrs: {
        stackSize: 1024, stackMemId: 0, priority: 1}},
    {name: "videnc_copy", mod: VIDENC_COPY, threadAttrs: {
        stackSize: 1024, stackMemId: 0, priority: 1}},
];

```



Another codec config example ...

Codec Config Parameters

```

var V_COPY = xdc.useModule('codecs.viddec_copy.VIDDEC_COPY');
var Server = xdc.useModule('ti.sdo.ce.Server');

Server.algs = [
    {
        name      : "viddec_copy", // C name for the of codec
        mod       : V_COPY,        // Javascript var reference for module
        threadAttrs : {
            stackSize : 1024,      // Size of task's stack
            stackMemId : 0,        // BIOS MEM segment ID for task's stack
            priority  : Server.MINPRI + 1 // Task's priority
        },
        groupid   : 1 // Scratch pool (includes: mem, dma, etc.)
    },
];

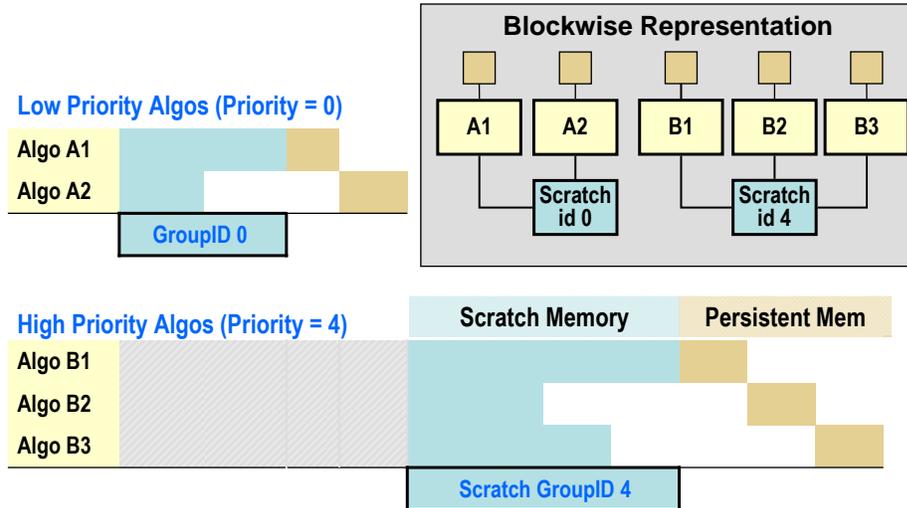
```

- ◆ By default, the Codec Engine automatically matches algorithms scratch memory ID to their thread priority, to help guaranteeing safe operation. (See wiki.davincidsp.com topic for exact details.)

Examining the "groupid"...

Memory – Scratch Groups

Groups of Scratch Memory



- ◆ Scratch memory makes sense – without it, our mem req wouldn't have fit on slide
- ◆ General rule – don't allow differing priorities to share same scratch resources
- ◆ Priority ≠ groupid, though it's a common to assign them that way

Back to server.cfg ... and Memory Setup

RTSC Config File (.cfg)

The RTSC configuration file (.cfg) allows you to specify which packages to include, as well as setting their configuration options

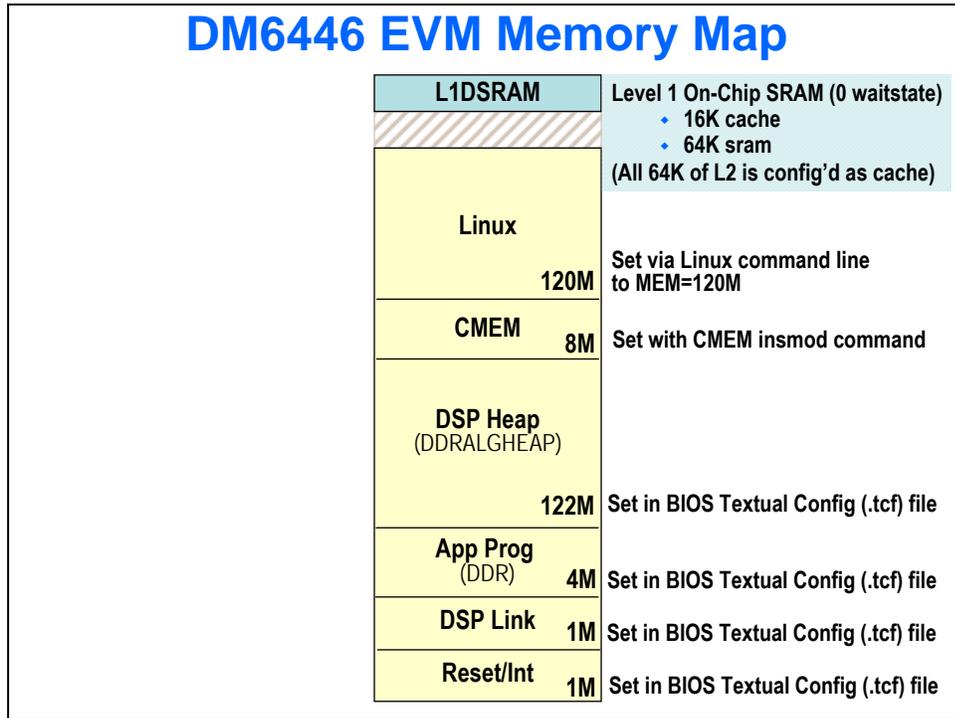
server.cfg	<ol style="list-style-type: none"> <li style="margin-bottom: 10px;"> 1. General Setup <ul style="list-style-type: none"> ◆ Add OSAL module ◆ Add server module ◆ Include codec.cfg <li style="margin-bottom: 10px;"> 2. Memory Setup (DSKT2 module) <ul style="list-style-type: none"> ◆ Associate IALG memory types with available target memory ◆ Specify default scratch group memory sizes 3. DMA Setup (DMAN3) (discussed in chapter 14) <ul style="list-style-type: none"> ◆ Specify number of DMA channels DMAN3 may manage ◆ Specify number of TCCs and Params DMAN3 can offer to algos
codec.cfg	<ol style="list-style-type: none"> <li style="margin-bottom: 10px;"> 4. CODEC Setup: <ul style="list-style-type: none"> ◆ Add module reference for each codec ◆ Configure properties for each codec thread: <ul style="list-style-type: none"> ◆ Thread priority ◆ Stack size ◆ Stack memory location

How was memory defined? (server.tcf)

TCF File : Target Memory Definitions

```

var mem_ext = [
{ comment: "algo heap sent to external memory ",
  name: "DDRALGHEAP",
  base: 0x88000000, // 128MB
  len: 0x07A00000, // 122MB
  space: "code/data"
},
{ comment: " application code and data sent to external memory ",
  name: "DDR",
  base: 0x8FA00000, // 250MB
  len: 0x00400000, // 4MB
  space: "code/data"
},
{ comment: "DSPLINK code and data routed to external memory",
  name: "DSPLINKMEM",
  base: 0x8FE00000, // 254MB
  len: 0x00100000, // 1MB
  space: "code/data"
},
{ comment: "reset vector routed to external memory ",
  name: "RESET_VECTOR",
  base: 0x8FF00000,
  len: 0x00000080,
  space: "code/data"
}
].];
    
```



xDAIS Memory Spaces

- ◆ [xDAIS specifies](#) a variety of [Memory Spaces](#) an algorithm can request
- ◆ These various space names matched the h/w capabilities on older proc's

xDAIS Memory Spaces

“Dual Access” RAMs:
 DARAM0
 DARAM1
 DARAM2

“Single Access” RAMs:
 SARAM0
 SARAM1
 SARAM2

External Data:
 ESDATA

Internal/External Pgm:
 IPROG
 EPROG

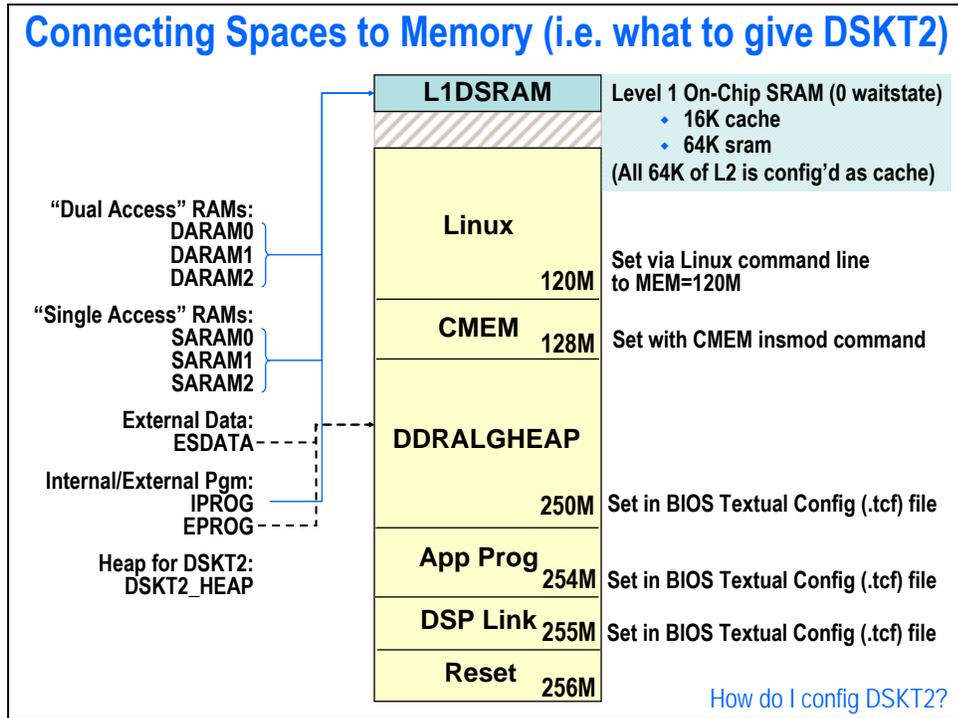
Heap for DSKT2:
 DSKT2_HEAP

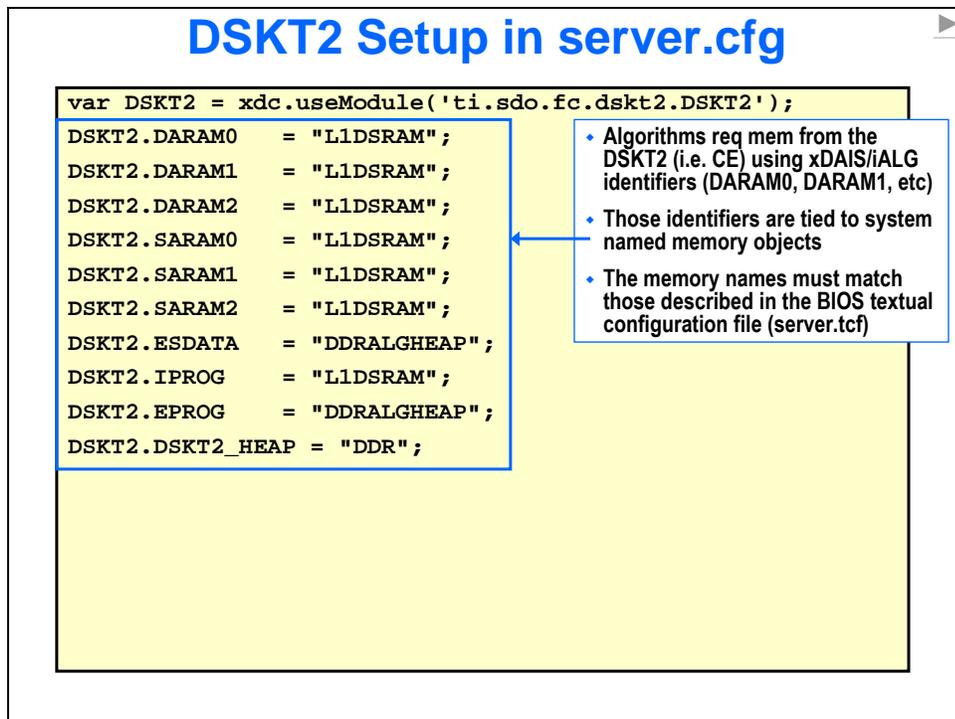
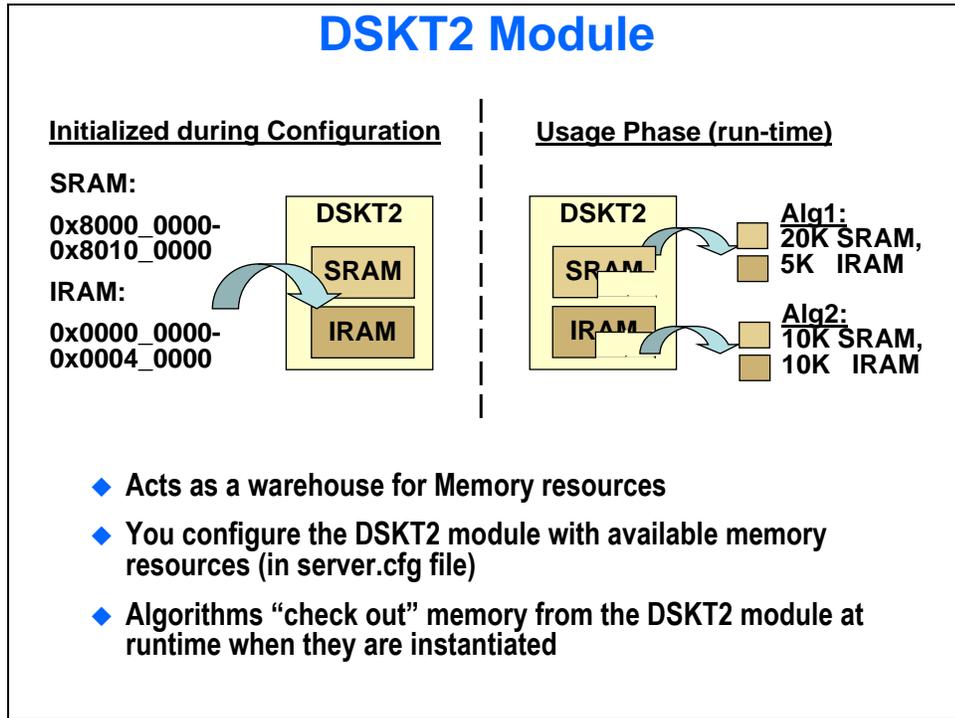
- ◆ With [hindsight](#), it might have been better to name the spaces: ‘Space 1’, ‘Space 2’, etc.
- ◆ Use these different spaces as a generic means of specifying diff parts of memory
- ◆ For example, algo vendors may request a memory space, then indicate in the algo docs how to match the space to real memory:
 - ◆ Algo1 asks for block of memory from DARAM0 and another from SARAM0
 - ◆ In the documentation, they could state:

*Map DARAM0 to L1D SRAM; and,
 Map SARAM0 to L2 (or L1D) SRAM*

The algo must run no matter which memory it is granted, though, to obtain spec'd performance, you must grant the requested memory spaces.

[Connecting Spaces to the Memory Map...](#)





Sidebar – Minimum Scratch Group Allocations

Is Algorithm Creation Order Important?

Scenario 1

Algo	Req.
Algo B1	4K
Algo B2	2K
Algo B3	3K
Total =	4K

Scenario 2

Algo	Req.
Algo B1	3K
Algo B2	2K
Algo B3	4K
Total =	7K

- ◆ Yes, order of algo creation can be important when sharing scratch memory
- ◆ Codec with largest scratch requirements (from a given pool) should be alloc'd 1st
- ◆ Though, this is better solved using another config parameter ... see next slide ...

DSKT2 – Minimum Scratch Group Memory Alloc's

```

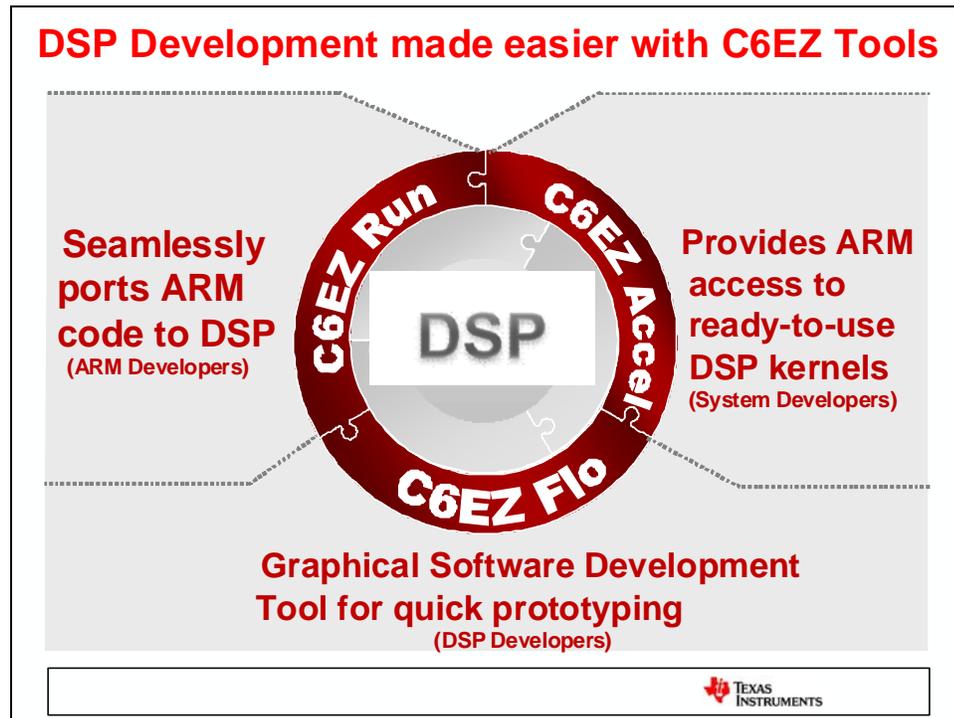
var DSKT2 = xdc.useModule('ti.sdo.fc.dskt2.DSKT2');
DSKT2.DARAM0 = "L1DSRAM";
DSKT2.DARAM1 = "L1DSRAM";
DSKT2.DARAM2 = "L1DSRAM";
DSKT2.SARAM0 = "L1DSRAM";
DSKT2.SARAM1 = "L1DSRAM";
DSKT2.SARAM2 = "L1DSRAM";
DSKT2.ESDATA = "DDRALGHEAP";
DSKT2.IPROG = "L1DSRAM";
DSKT2.EPROG = "DDRALGHEAP";
DSKT2.DSKT2_HEAP = "DDR";

DSKT2.DARAM_SCRATCH_SIZES = [ 65536, 1024, 0,0, /* ... */ 0 ];
DSKT2.SARAM_SCRATCH_SIZES = [ 65536, 1024, 0,0, /* ... */ 0 ];

if( this.prog.build.profile == "debug" )
    DSKT2.debug = true;
    
```

- ◆ Algorithms req mem from the DSKT2 (i.e. CE) using xDAIS/iALG identifiers (DARAM0, DARAM1, etc)
- ◆ Those identifiers are tied to system named memory objects
- ◆ The memory names must match those described in the BIOS textual configuration file (server.tcf)
- ◆ The size for each scratch memory pool is set in an array.
- ◆ The first element is for scratch memory pool id=0, the second for pool id=1, etc.

C6EZ Tools



What can a System Developer do on the DSP?

- ◆ Offload key signal-processing and math intensive algorithms
- ◆ Leverage some DSP features, like floating-point computations
- ◆ Free up ARM MIPS for additional system features
- ◆ Save money by not moving to a more powerful & expensive ARM!

4 Ways to access the DSP from the ARM

1. Call DSP directly over DSPLINK/SYLINK
2. Wrap the algorithm with IUniversal interface to Codec Engine
3. Use C6EZRun to cross compile and redirect code for DSP
4. Use C6EZAccel to call DSP realtime-kernels/functions from the ARM

C6Run

C6EZRun for ARM® Developers

C6EZRun quickly ports ARM code to run on the DSP, making DSP development easy, fast and cost efficient

Compile C code for DSP execution right from the Linux shell

No modification to ARM application code required

Provides a familiar development environment with a GCC-like interface, simplifying the user/development experience

The diagram illustrates two development workflows. The top workflow, 'ARM-only Development', shows source files (module1.c, module2.c, main.c, critical.c) being processed by the 'GCC Tool Chain' to produce an 'ARM Executable'. The bottom workflow, 'ARM+DSP Development with c6runlib', shows the same source files being processed by the 'GCC Tool Chain' to produce an 'ARM Executable' and an 'Embedded DSP Executable'. In this workflow, 'c6runlib tools' and 'critical.lib' are used to generate the DSP executable from the 'critical.c' source file.

C6Accel

C6EZAccel for System Developers

C6EZAccel provides ARM application access to production ready, performance tuned functions on the DSP

The diagram shows a 'DSP + ARM SOC' architecture. On the left, an 'ARM Application' runs on an 'ARM® processor'. An arrow points from the ARM Application to a vertical green bar labeled 'C6EZAccel APIs'. Another arrow points from the APIs to a red box labeled 'C6EZAccel optimized xdais Algorithms for: Audio, Medical, Vision, Power and Control, Image processing, Signal processing', which runs on a 'DSP'.

ARM side API with easy access to 100's of optimized DSP kernels

Asynchronous execution allows ARM/DSP parallel processing

Includes key signal processing, image processing and math functions

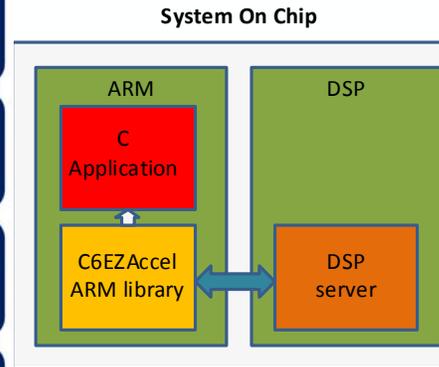
C6EZAccel application

Consists of two parts: DSP server integrating the DSP algorithms and ARM side user interface

DSP server collating DSPLink, CMEM, DSP/BIOS, and DSP libraries

ARM user interface is a static library that abstracts the codec engine setting and ARM side cache management

Function is retargeted to DSP, with error handling managed on ARM/Linux using error codes



47

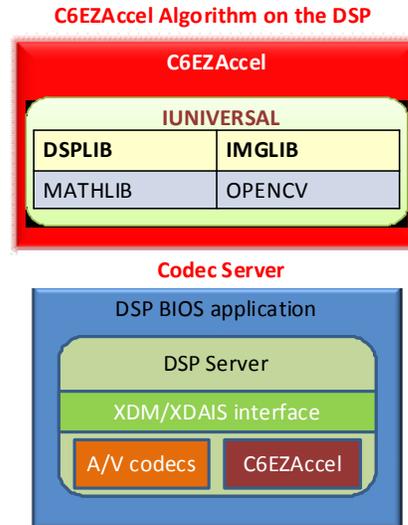
C6EzAccel Processing Blocks

Algorithms	Now	2Q11	4Q11
Foundation Signal Processing Software Algorithms			
Digital Signal Processing	32	45	60
Image Processing	40	50	60
Floating Point Math	30	30	30
Fixed Point Math	32	32	32
Filter Package			128
Application Specific Software Algorithms			
Vision And Analytics Library (VLIB)		52	52
Open Source Computer Vision (OpenCV)		60	100
ProAudio: Audio Processing Library			20
Power and Energy			15
Total Supported Functions			
	134	269	497



C6EZAccel DSP server

- ◆ C6EZAccel algorithm is a collection of DSP optimized libraries combined into a single xDAIS algorithm
- ◆ Implements iUniversal interface to the Codec Engine
- ◆ Server integrates BIOS, Link/Code-Engine, codecs and C6Accel into DSP executable
- ◆ C6Accel includes *Unit Servers* provided in the package
- ◆ Codec Servers with 6EZAccel integrated as of SDK 4.x



C6EZAccel Application Usage

User Experience in ARM Application

Headers and definitions

```
C6accel_Handle hC6accel = NULL
#define ENGINENAME " <platform_name>"
#define ALGNAME "c6accel "
#include "soc/c6accelw/c6accelw "
```

Source code to call fxn1 and fxn2 on DSP

```
CE_Runtime_init();
hC6accel = C6accel_create( );
.....
Status = C6accel_fxn1(hC6accel,parameters1);
Status = C6accel_fxn2(hC6accel,parameters2);
.....
C6accel_delete(hC6accel);
```

- Creates a C6accel handle.
- Calls fxn1 and fxn2 on the DSP.
- Deletes C6Accel handle.

Step 1: Analyze source to determine what to move to the DSP

Step 2: Allocate all buffers to be passed to DSP using CMEM

Step 3: Offload function to the DSP by invoking C6EZAccel APIs or CE APIs

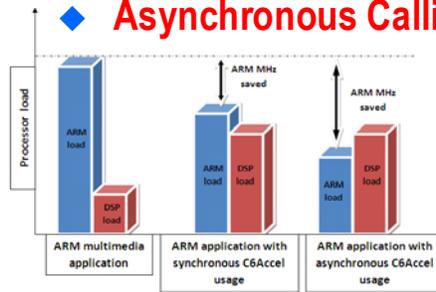
Step 4: Integrate DSP codec server with application using XDC configuration

Step 5: Compile all ARM code with the linker command file generated from Step 4



C6EAccel: Key Features

◆ Asynchronous Calling



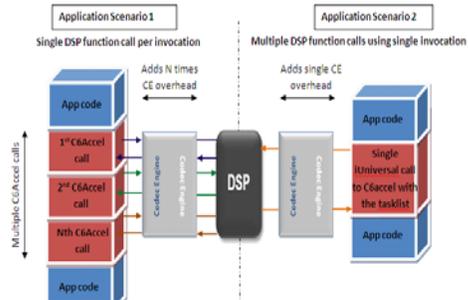
- ▶ Parallel processing on ARM and DSP
- ▶ Frees ARM MIPS, allows feature addition
- ▶ Maximum system level performance

Source to call fxn1 asynchronously on DSP

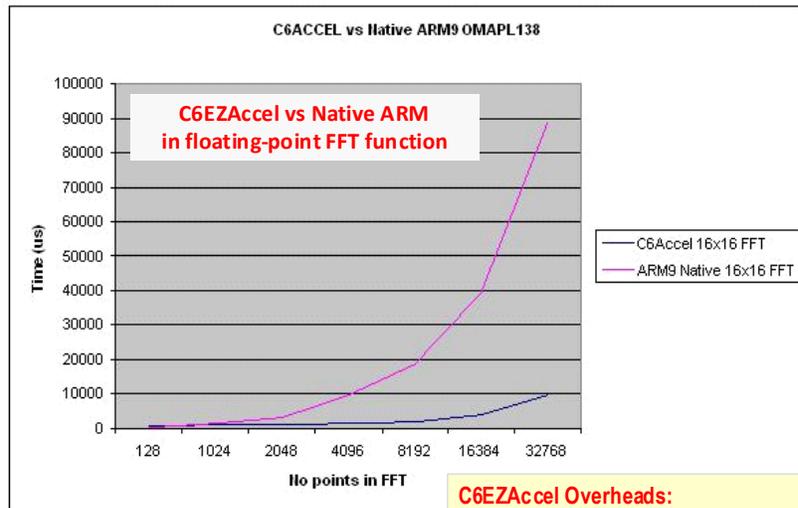


◆ Chaining of APIs

- ▶ Averaging of Inter processor overhead
- ▶ Improved Efficiency to make multiple calls



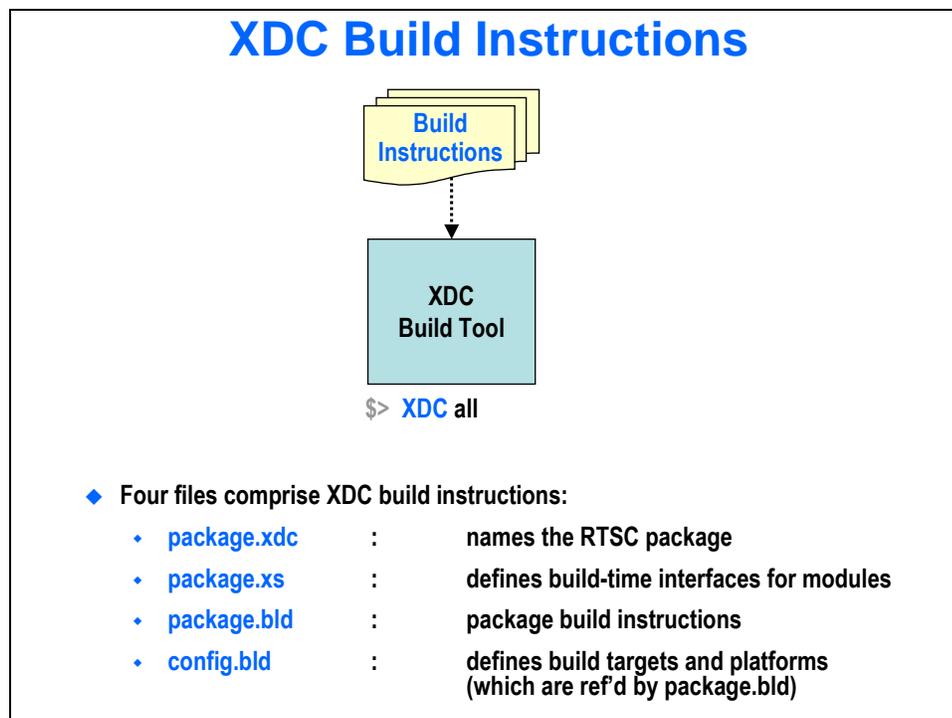
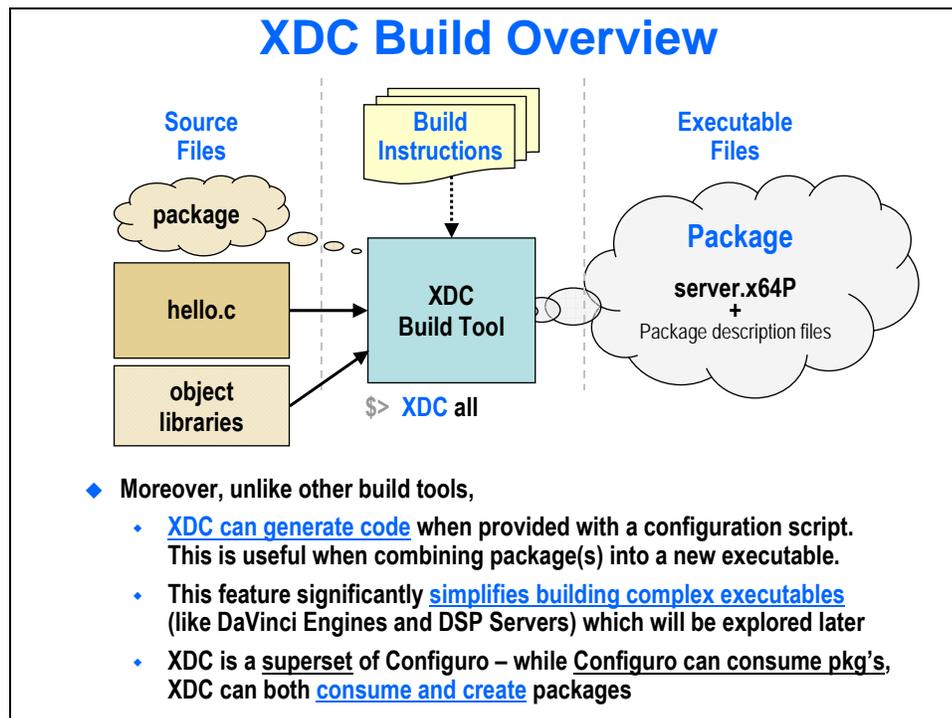
Performance benefits of offloading tasks to the DSP



DSP provides higher gains on larger images/data

- C6EAccel Overheads:**
- ◆ Cache invalidation on ARM and the DSP
 - ◆ Address translation from virtual to/from physical
 - ◆ DSP/SYSLINK overhead.
 - ◆ Activating, processing, deactivating the C6EAccel algorithm on the DSP

(Optional) A Short Course in XDC



package.xdc (2 examples)

Simple

```
// package.xdc for tto workshop app
package app { }
```

Using
more
RTSC
Features

```
requires ti.sdo.ce.video;
// package.xdc for the video_copy example
package servers.video_copy [1, 0, 0] {
}
```

- ◆ The second example shows how to specify dependencies as well as defining versioning (i.e. compatibility keys)
- ◆ The package name imposes a directory structure. In the 2nd case, the video_copy package is located at:
(repository_path)/servers/video_copy/

package.xs

```
function getLibs(prog)
{
    if (prog.build.target.isa == "64P") {
        if ( this.MYALG.Debug == false )
            lib = "lib/myalg_tto_le_release.a64P";
        else
            lib = "lib/myalg_tto_le_debug.a64P";
    }
    return (lib);
}

function getStackSize(prog)
{
    return (64 * 1024);
}
```

- ◆ While this is an .xs file for a codec package, it demonstrates how build-time methods can be defined for a package
 - ◆ The first function returns a library based on the value of a (.cfg) config option
 - ◆ The second tells the wizard (and/or Configuro) how big the stack size should be
- ◆ In most cases, we can leave it up to the GUI wizard tools to create this file for us

package.bld

```
var Pkg = xdc.useModule('xdc.bld.PackageContents');
/*
 * ===== Add Executable =====
 */
print( "Print something to output while building package." );
Pkg.otherFiles = [ "readme.txt", "document.pdf" ];
Pkg.addExecutable(
    "app_debug",           // Name of executable file
    MVArm9,               // Build target (from config.bld)
    MVArm9.platform,      // Platform (from config.bld)
    { cfgScript: "app_cfg.cfg", // App/engine or server cfg file
      profile: "debug", // Build profile to be used
    }
).addObjects( ["main.c"] ); /* JavaScript array of obj files;
                             if source files are passed, xdc will
                             first build them into object files */
```

- ◆ The `addExecutable` method tells XDC what (and how) to build your executable program; there are many other methods, ex: `addLibrary`, `addOtherFiles`, `makeEpilog`
- ◆ This example is a "hard-coded" version and would need to be edited for program; the lab solutions contain a more generic version of `package.bld`

Parts of config.bld:

- ◆ DSP Target
- ◆ ARM Target
- ◆ Linux Host Target
- ◆ Build Targets

- ◆ `config.bld` defines the build targets for your projects
- ◆ Build targets settings include: tool locations, compiler settings, etc.
- ◆ When, in `package.bld`, you specify to build for a particular target, XDC refers to `config.bld` to resolve this info

config.bld

```
// ===== Linux host target =====
var Linux86 = xdc.useModule('gnu.targets.Linux86');
Linux86.rootDir = "/usr";

// ===== Arm target =====
var MVArm9 = xdc.useModule('gnu.targets.MVArm9');
MVArm9.rootDir = "/devkit/arm/v5t_le/bin/arm_v5t_le.";

// ===== DSP target =====
var C64P = xdc.useModule('ti.targets.C64P');
C64P.platform = "ti.platforms.evmDM6446";
C64P.ccOpts.prefix += "-k -a!";

// location of your C6000 codegen tools
C64P.rootDir = java.lang.System.getenv("C6000_CG");

// ===== Build targets =====
// list of targets (ISAs + compilers) to build for
Build.targets = [
    MVArm9,
    Linux86,
    C64P
];
```

Authoring a xDAIS/xDM Algorithm

Introduction

This chapter looks at algorithms from the inside out; how you write a xDAIS algorithm. It begins with the general description of xDAIS and how it is used, then examines the interface standard by focusing on the creation/usage/deletion of an algorithm and how its API deals with memory resource allocations.

Learning Objectives

Outline

◆ Introduction

- ◆ What is xDAIS (and VISA)?
- ◆ Software Objects
- ◆ Master Thread Example
- ◆ Intro to Codec Engine Framework (i.e. VISA)

◆ Algorithm Lifecycle

◆ Frameworks

◆ Algorithm Classes

◆ Making An Algorithm

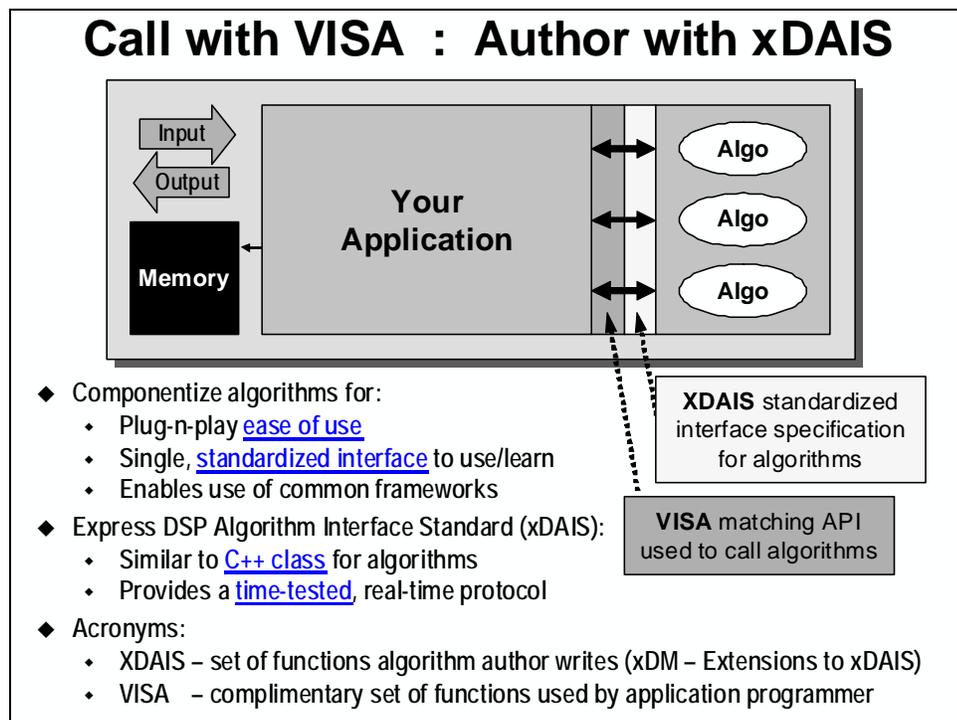
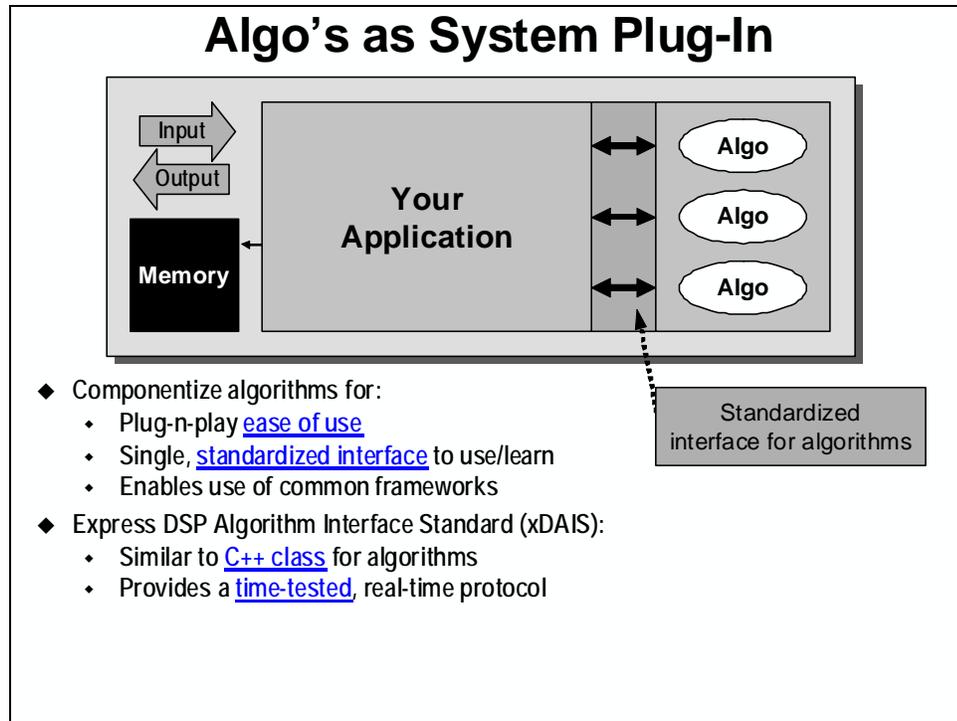
◆ Appendix

Chapter Topics

Authoring a xDAIS/xDM Algorithm	13-1
<i>Introduction</i>	<i>13-3</i>
What is xDAIS (and VISA)?	13-3
Software Objects	13-4
Master Thread Example.....	13-6
Intro to Codec Engine Framework (i.e. VISA).....	13-6
<i>Algorithm Lifecycle.....</i>	<i>13-9</i>
Create.....	13-10
Process.....	13-14
Delete.....	13-16
<i>Frameworks</i>	<i>13-17</i>
Algorithm Classes	13-19
xDM/VISA Classes	13-19
Universal Class	13-20
Extending xDM	13-21
Design Your Own – Extending the Universal Class.....	13-23
<i>Making an Algorithm</i>	<i>13-24</i>
Rules of xDAIS	13-24
Using the Algorithm Wizard	13-26
<i>Appendix</i>	<i>13-30</i>
Reference Info	13-30
Using the Old Algorithm Wizard	13-31
(Optional) xDAIS Data Structures.....	13-39
(Optional) Multi-Instance Ability.....	13-43
(Optional) xDAIS : Static vs Dynamic	13-44

Introduction

What is xDAIS (and VISA)?



Software Objects

Software Components/Objects

- ◆ **Examples of software objects:**
 - C++ classes
 - xDAIS (or xDM) algorithms
- ◆ **What does a software object contain?**
 - Thinking of C++ classes:
 - Attributes:**
 - Class object
 - Creation (i.e. construction) parameters
 - Methods**
 - Constructor
 - Destructor
 - "processing" method(s)

Algo

Algo

Algo

Comparing Objects: C++ / xDAIS

```
class algo{
public:
    // methods
    int method1(int param);
    int method2(int param);
    // attributes
    int attr1;
    int attr2;
}
```

```
typedef struct {
    // methods
    int (*method1) (int param);
    int (*method2) (int param);
    // attributes
    int attr1;
    int attr2;
} algo;
```

- xDAIS (and xDM) provide a C++-like object, implemented in C
- Because C does not support classes, structs are used
- Because structs do not support methods, function pointers are used

Comparing Methods: C++ / xDM

Create Instance: (C++ Constructor)

```
C++    algo::algo(algo_params params)
VISA  VIDENC_create(VIDENC_params params)
```

Process:

```
C++    algo::myMethod(...)
VISA  VIDENC_process(...)
```

Delete Instance: (C++ Destructor)

```
C++    algo::~~algo()
VISA  VIDENC_delete()
```

Note: With VISA, the framework (i.e. Codec Engine library) allocates resources on algorithm creation, as opposed to C++ constructors, which allocate their own resources.

Algorithm Creation

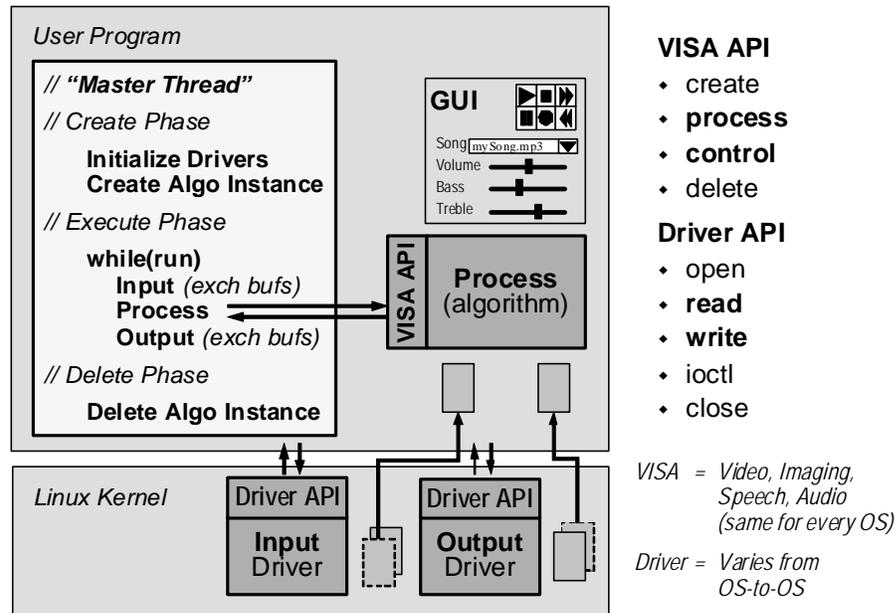
Traditionally algorithms have simply used resources without being granted them by a central source

Benefits of Central Resource Manager:

1. Avoid resource conflict during system integration
2. Facilitates resource sharing (i.e. scratch memory or DMA) between algorithms
3. Consistent error handling when dynamic allocations have insufficient resources

Master Thread Example

Linux System

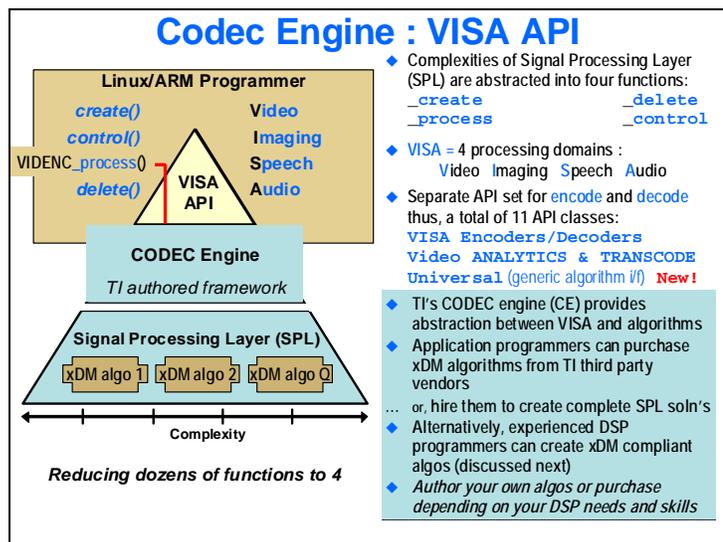
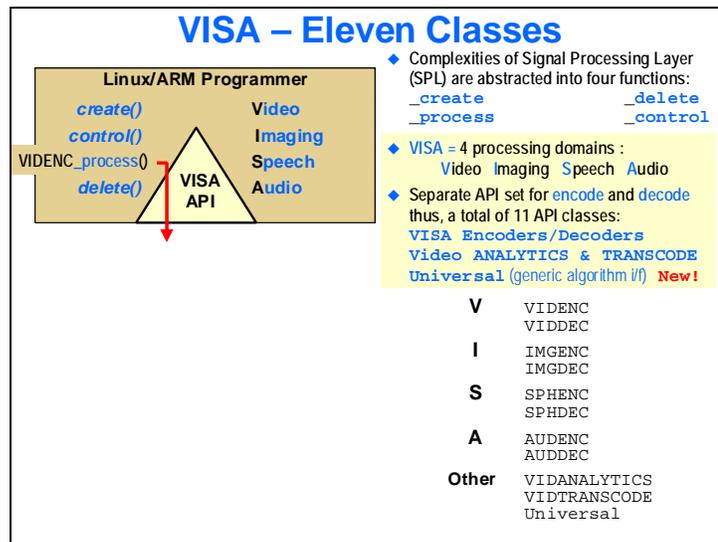
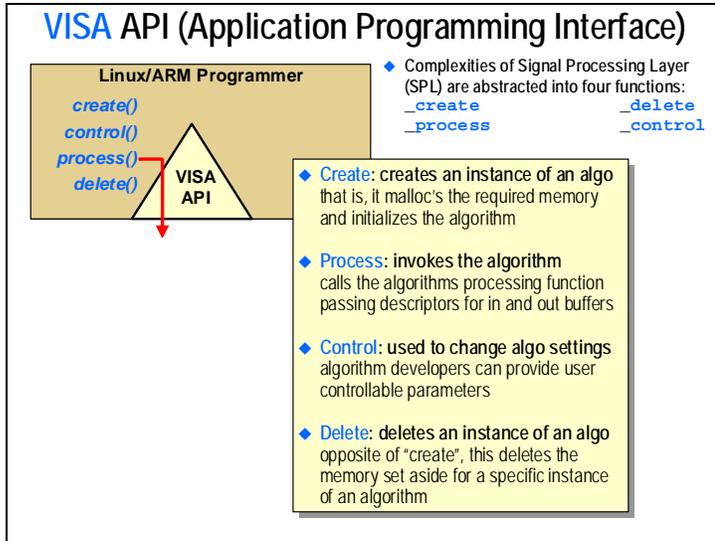


Intro to Codec Engine Framework (i.e. VISA)

Codec Engine Framework Benefits

- ◆ Multiple algorithm channels (instances)
- ◆ **Dynamic (run-time) algorithm instantiation**
- ◆ Plug-and-play for algorithms of the same class (inheritance)
- ◆ Sharing of memory and DMA channel resources
- ◆ Algorithm interoperability with any CE-based Framework
- ◆ Same API, no new learning curve for all algorithm users
- ◆ Provided by TI!

Many of these benefits are a direct result of the object-oriented structure of the codec engine



Master Thread Key Activities

```

idevfd = open("/dev/xxx", O_RDONLY);
ofilefd = open("./fname", O_WRONLY);
ioctl(idev fd, CMD, &args);
myCE = Engine_open("vcr", myCEAttrs);
myVE = VIDENC_create(myCE, "videnc", params);

while( doRecordVideo == 1 ) {
    read(idevfd, &rd, sizeof(rd));
    VIDENC_process(myVE, ...);
    //VIDENC_control(myVE, ...);
    write(ofilefd, &wd, sizeof(wd));
}
close(idevfd);
close(ofilefd);
VIDENC_delete(myVE);
Engine_close(myCE);

```

// Create Phase
// get input device
// get output device
// initialize IO devices...
// prepare VISA environment
// prepare to use video encoder

// Execute phase
// read/swap buffer with Input device
// run algo with new buffer
// optional: perform VISA algo ctrl
// pass results to Output device

// Delete phase
// return IO devices back to OS
// algo RAM back to heap
// close VISA framework

Note: the above pseudo-code does not show double buffering, often essential in Realtime systems!

VISA Function Details

```

Engine_Handle      myCE;
AUDENC_Handle      myAE;
AUDENC_Params      params;
AUDENC_DynParams   dynParams;
AUDENC_Status      status;

CERuntime_init();

myCE = Engine_open("myEngine", NULL);
myAE = AUDENC_create (myCE, "aEncoder", &params);
stat = AUDENC_process(myAE, &inBuf, &OutBuf,
                      &inArgs, &outArgs);

stat = AUDENC_control(myAE, XDM_GETSTATUS,
                      &dynParams, &status);

AUDENC_delete(myAE);
Engine_close (myCE);

```

- ◆ *Engine* and *Codec* string names are declared during the [engine config](#) file
- ◆ The config file (.cfg) specifies which algorithm packages (i.e. libraries) should be built into your application

Pick your algo's using .CFG file

Engine Configuration File (.cfg file)

```

/* Specify your operating system (OS abstraction layer) */
var osal = xdc.useModule('ti.sdo.ce.osal.Global');
osal.runtimeEnv = osal.LINUX;

/* Specify which algo's you want to build into your program */
var vidDec = xdc.useModule('ti.codecs.video.VIDENC');
var audDec = xdc.useModule('ti.codecs.audio.AUDENC');

/* Add the Codec Engine library module to your program */
var Engine = xdc.useModule('ti.sdo.ce.Engine');

/* Create engine named "myEngine" and add these algo's to it */
var myEng = Engine.create("myEngine",
[
    {name: "vEncoder", mod: vidDec, local: true},
    {name: "aEncoder", mod: audDec, local: true},
]);

```

Algorithm Lifecycle

Algorithm Instance Lifecycle

Algorithm Lifecycle	Dynamic
Create ("Constructor")	
Process	<i>doFilter</i>
Delete ("Destructor")	

- ◆ Codec Engine only uses the *Dynamic* features of xDAIS

Algorithm Instance Lifecycle

Algorithm Lifecycle	Dynamic
Create ("Constructor")	<i>algNumAlloc</i> <i>algAlloc</i> <i>algInit</i>
Process	<i>doFilter</i>
Delete ("Destructor")	<i>algFree</i>

- ◆ Codec Engine only uses the *Dynamic* features of xDAIS



iAlg Functions Summary

◆ Create Functions

- **algNumAlloc** - Tells application (i.e. CODEC engine) how many blocks of memory are required; it usually just returns a number
- **algAlloc** - Describes properties of each required block of memory (size, alignment, location, scratch/persistent)
- **algInit** - Algorithm is initialized with specified parameters and memory

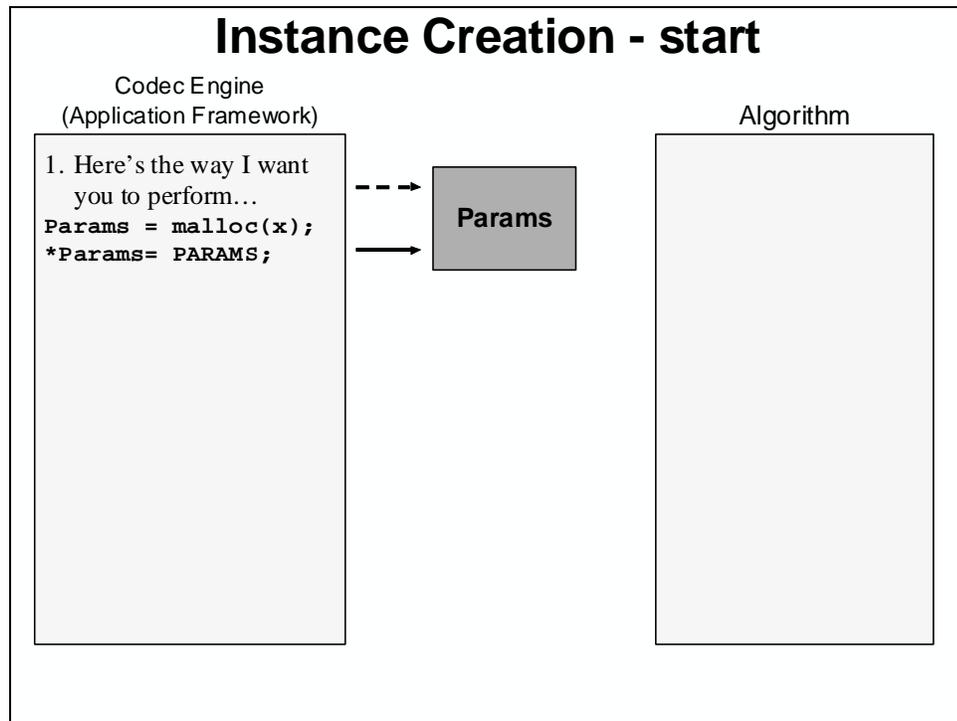
◆ Execute Functions

- **algActivate** - Prepare scratch memory for use; called prior to using algorithm's process function (e.g. prep history for filter algo)
- **algDeactivate** - Store scratch data to persistent memory subsequent to algo's process function
- **algMoved** - Used if application relocates an algorithm's memory

◆ Delete Function

- **algFree** - Algorithm returns descriptions of memory blocks it was given, so that the application can free them

Create



Algorithm Parameters (Params)

- ◆ How can you adapt an algorithm to meet your needs?

Vendor specifies "params" structure to allow user to set creation parameters.

These are commonly used by the algorithm to specify resource needs and/or they are used for initialization.

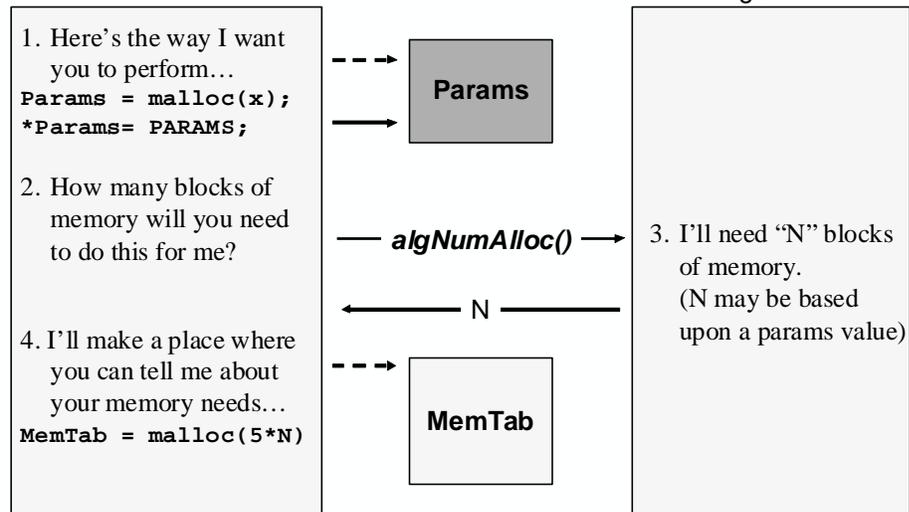
- ◆ For example, what parameters might you need for a FIR filter?

A filter called IFIR might have:

```
typedef struct IFIR_Params {
    Int          size;          // size of params
    XDAS_Int16   firLen;
    XDAS_Int16   blockSize;
} IFIR_Params;
```



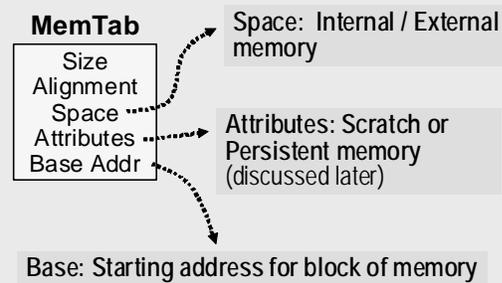
Instance Creation - start



XDAIS Components: Memory Table

- ◆ What prevents an algorithm from “taking” too much (critical) memory?
 - ◆ Algorithms cannot allocate memory.
 - ◆ Each block of memory required by algorithm is detailed in a **Memory Table** (memtab), then allocated by the Application.

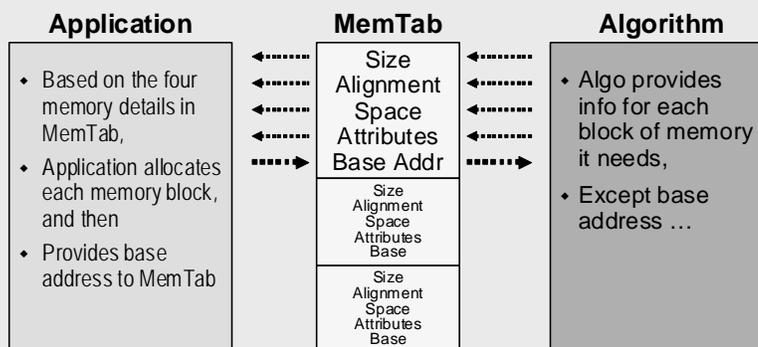
◆ MemTab:

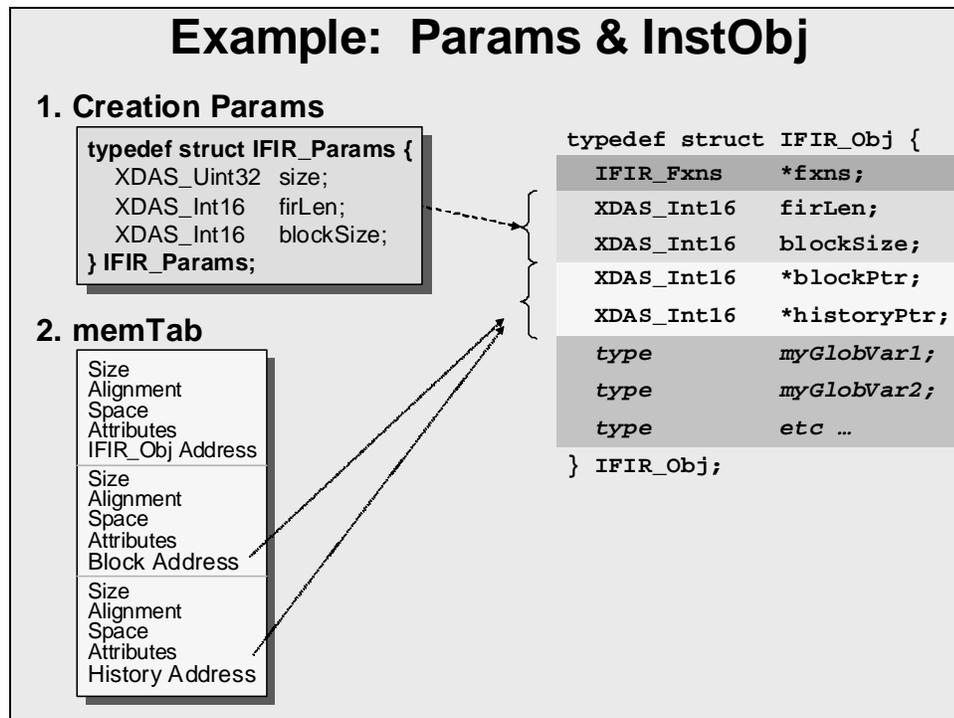
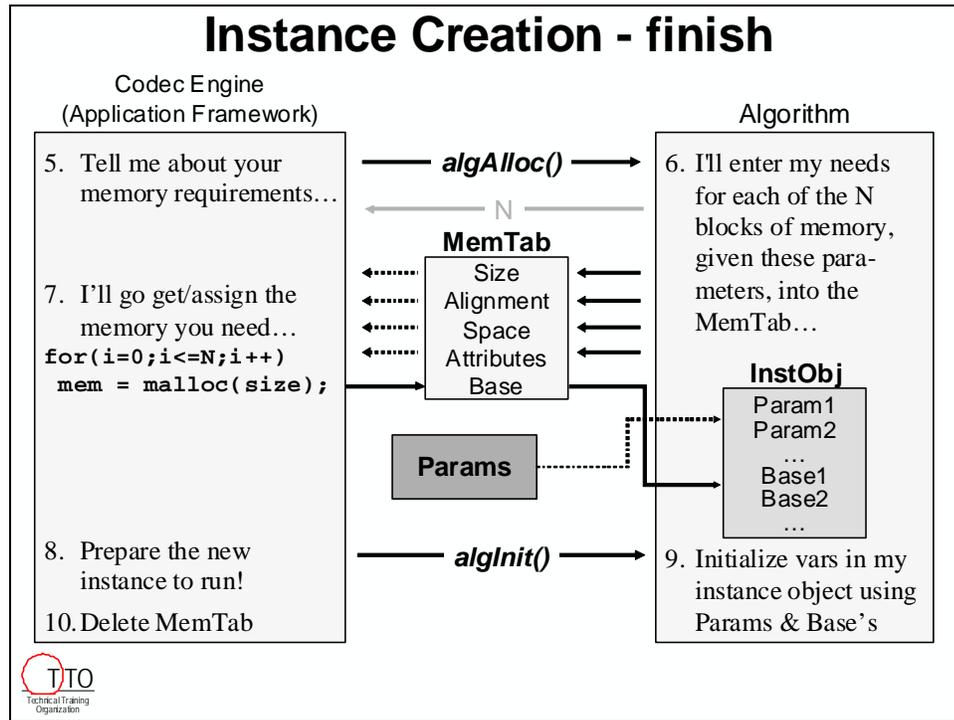


XDAIS Components: Memory Table

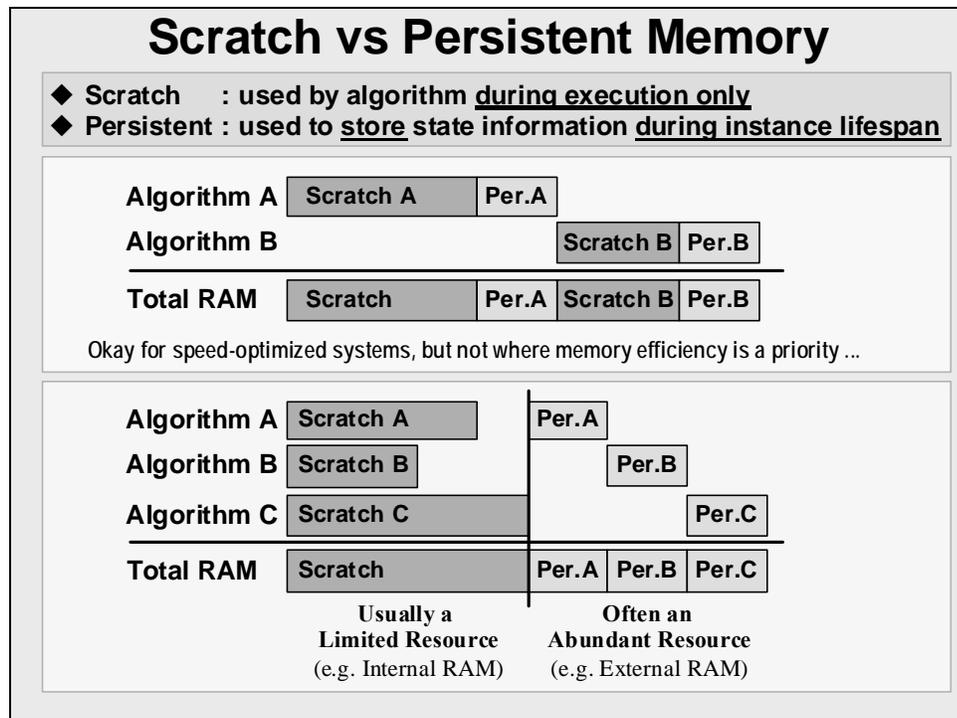
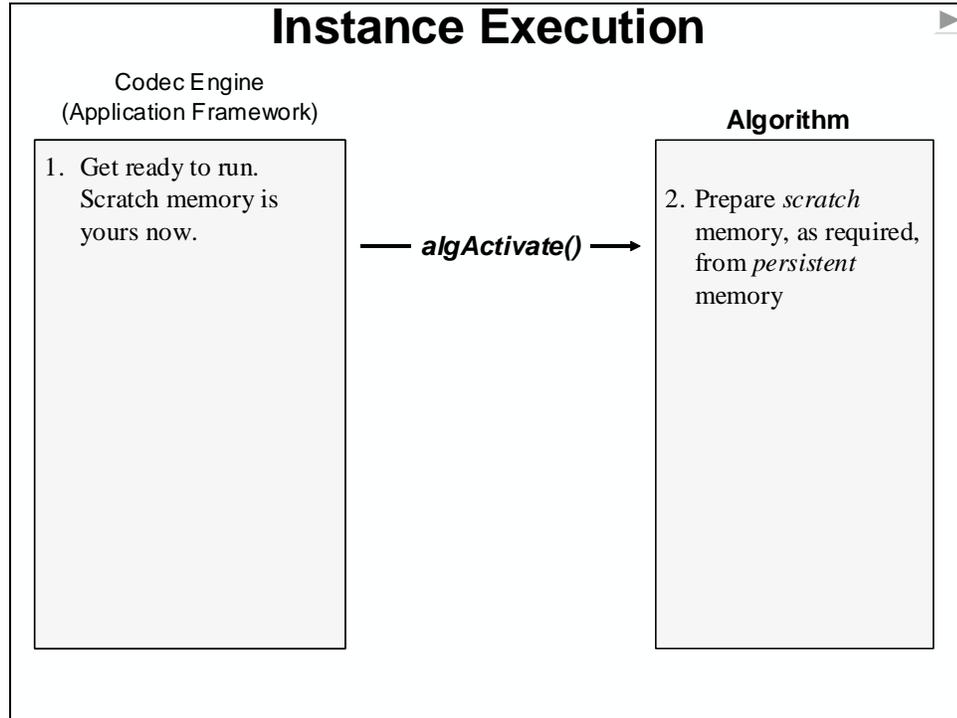
- ◆ What prevents an algorithm from “taking” too much (critical) memory?
 - ◆ Algorithms cannot allocate memory.
 - ◆ Each block of memory required by algorithm is detailed in a **Memory Table** (memtab), then allocated by the Application.

◆ MemTab example:





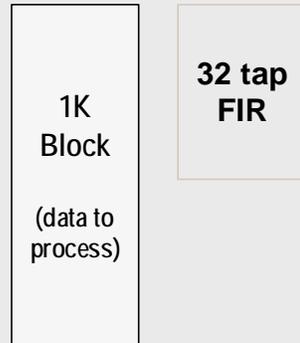
Process



Example of Benefit of Scratch Memory

Example:

- ◆ Let's say we will process 1K block of data at a time
- ◆ For 32-tap filter, 32 samples must be saved from one process call to the next



# Chans	No Overlay / Scratch	Use Scratch
1	1000	1032
2	2000	1064
...		
10	10,000	1320

- ◆ Without using scratch (i.e. overlay) memory, 10 channels of our block filter would take ten times the memory
- ◆ If sharing the block between channels, memory usage drops considerably
Only 32 RAM/channel persistent buffer to hold history vs. 1000 RAM/channel

Instance Execution

Codec Engine (Application Framework)

1. Get ready to run.
Scratch memory is yours now.
3. Run the algorithm ...
5. I need the scratch block back from you now...

— **algActivate()** →

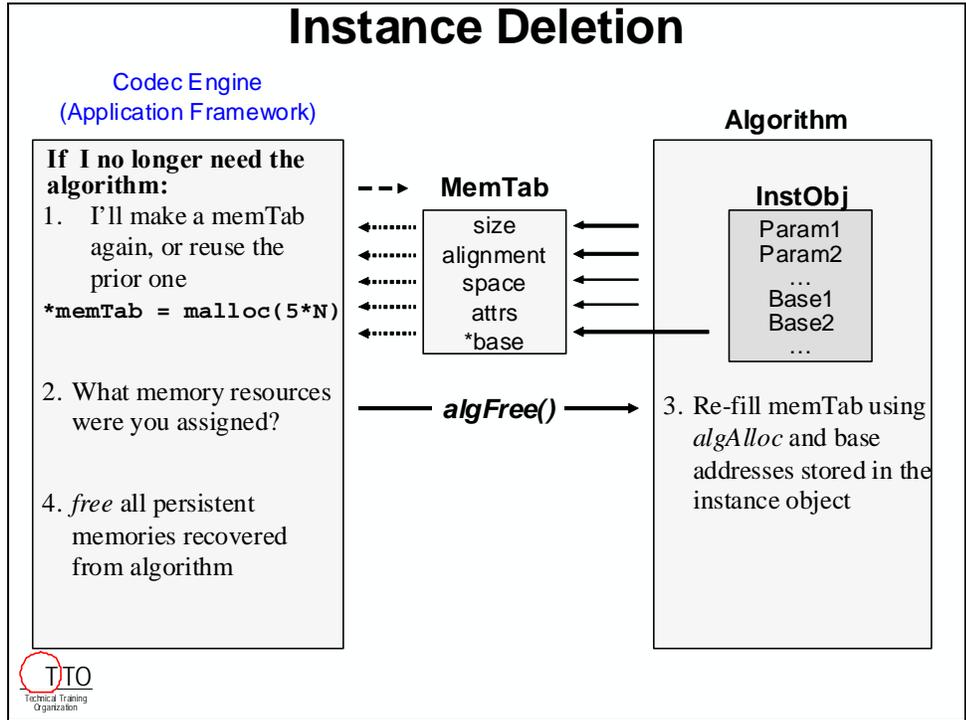
— **process()** →

— **algDeactivate()** →

Algorithm

2. Prepare *scratch* memory, as required, from *persistent* memory
4. Perform algorithm - freely using all memory resources assigned to algo
6. Save scratch elements to persistent memory as desired

Delete




Frameworks

Algorithm Instance Lifecycle

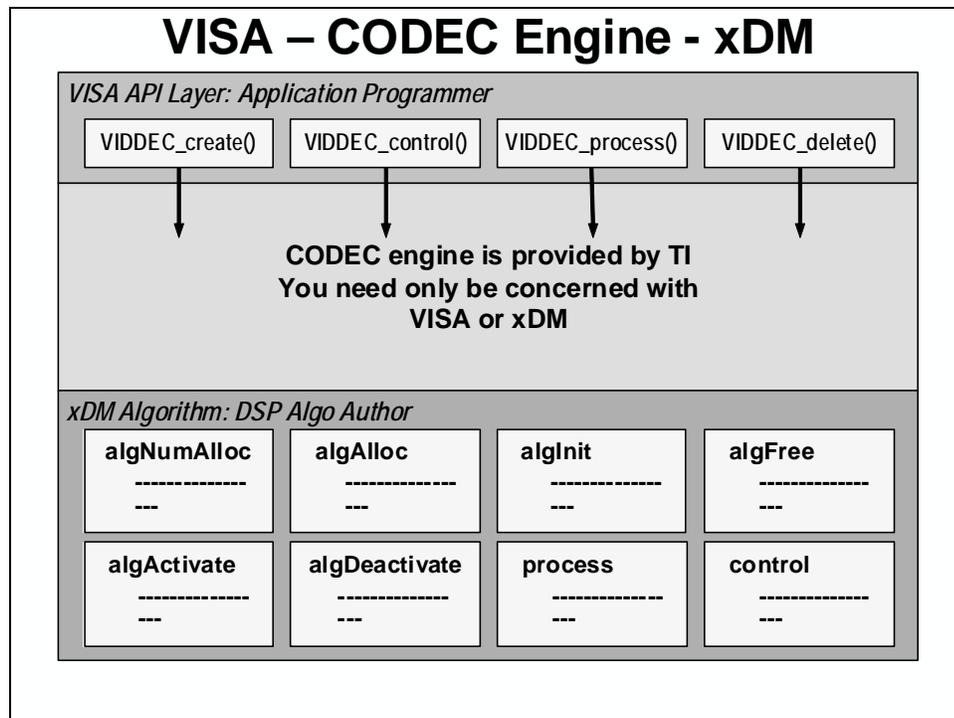
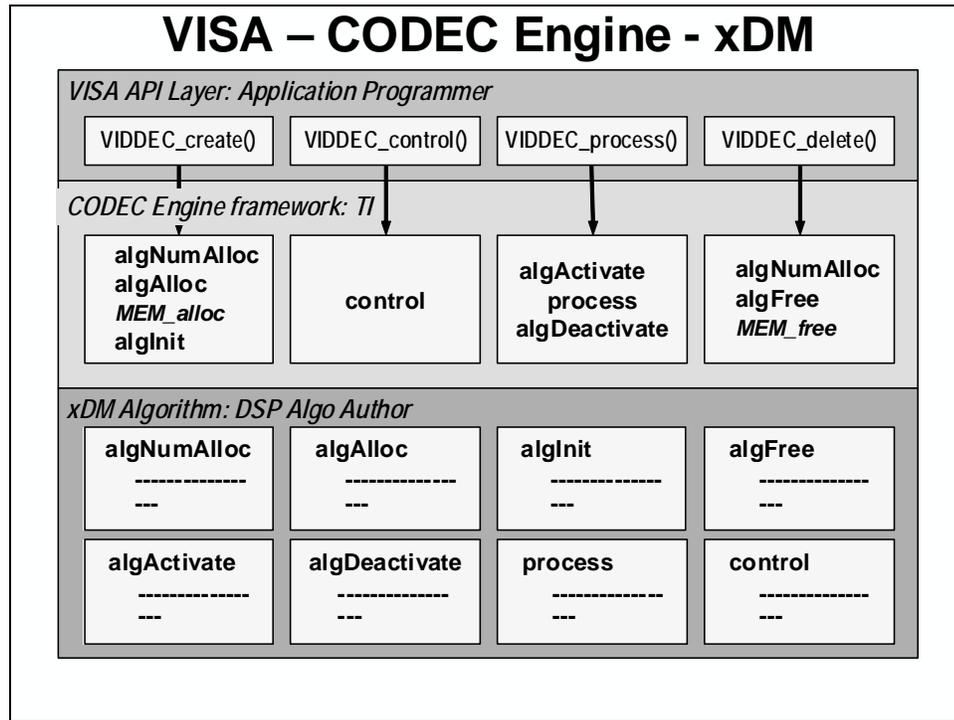
Algorithm Lifecycle	Dynamic
Create	<i>algNumAlloc</i> <i>algAlloc</i> <i>algInit</i>
Process	<i>process</i>
Delete	<i>algFree</i>

◆ If all algorithms must use these 'create' functions, couldn't we simplify our application code?



Framework Create Function

Create Functions	Codec Engine Framework (VISA)
<i>algNumAlloc ()</i> <i>algAlloc ()</i> <i>algInit ()</i>	 <i>VIDENC_create()</i>
<p>◆ <i>These functions are common for <u>all</u> xDAIS/xDM algo's</i></p>	<p>◆ <u>One</u> create function can instantiate <u>any</u> XDM algo</p> <p>◆ History of algorithm creation functions from TI:</p> <ul style="list-style-type: none"> • <u>ALG</u> create is simplistic example create function provided with the xDAIS library • <u>ALGRE</u> library provided in Reference Frameworks • <u>DSKT2</u> library is used by the Codec Engine and Bridge Frameworks • <u>Codec Engine</u> (CE) provides create functions defined in xDM (or xDM-like) algos

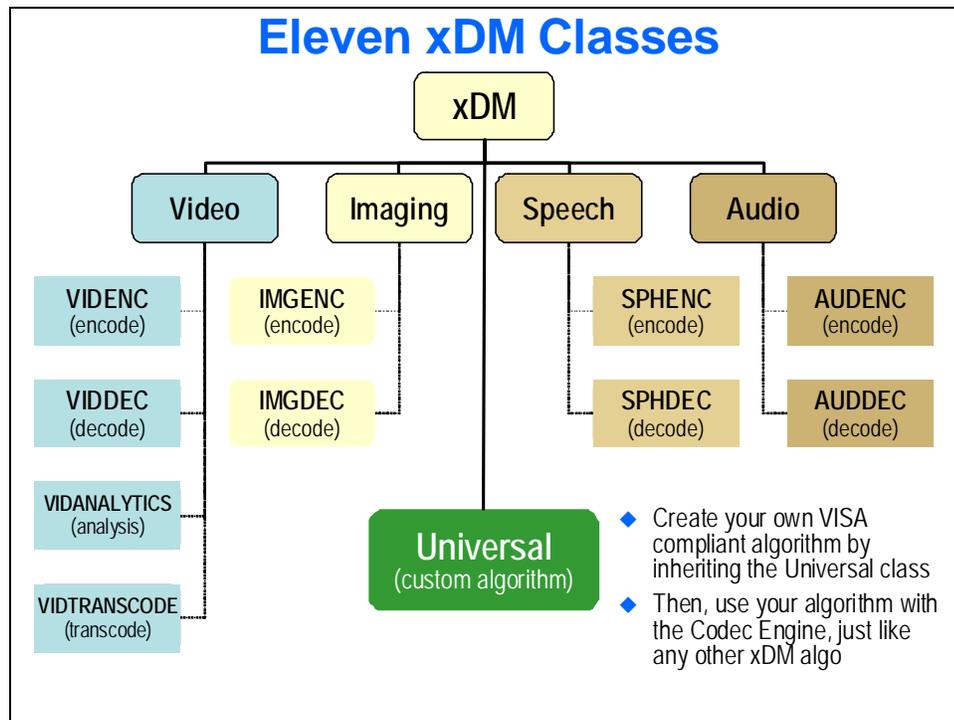


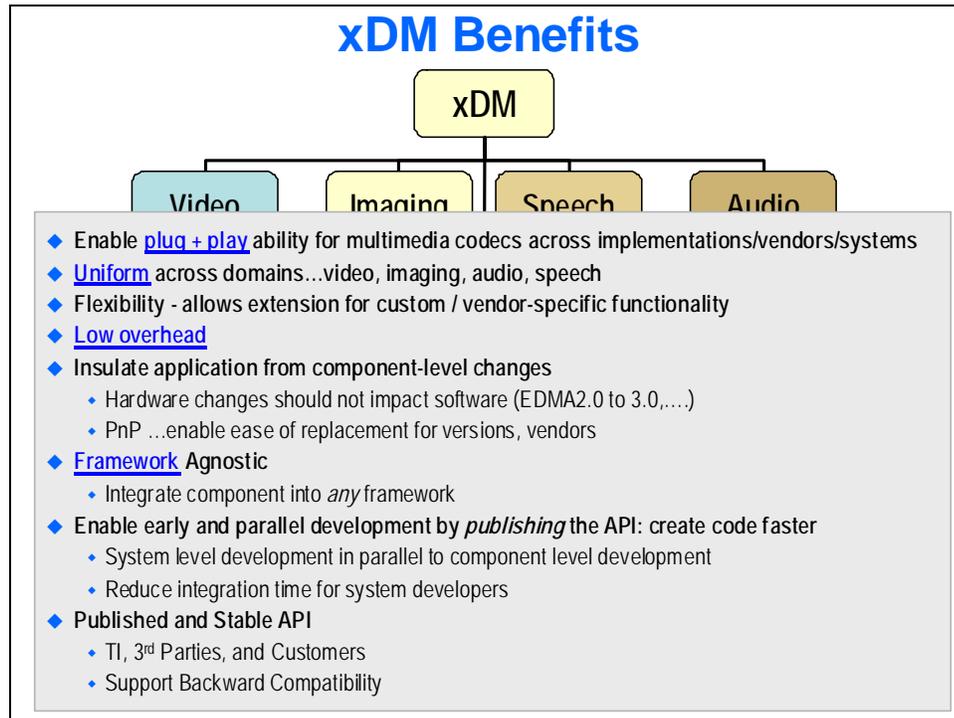
Algorithm Classes

xDAIS Limitations

- ☺ xDAIS defines methods for managing algo \leftrightarrow heap memory : algCreate, algDelete, algMoved
- ☺ xDAIS also defines methods for preparation/preservation of scratch memory : algActivate, algDeactivate
- ☹ Does *not* define the API, args, return type of the processing method
- ☹ Does *not* define the commands or structures of the control method
- ☹ Does *not* define creation or control structures
- ◆ Reason: xDAIS did not want to stifle options of algo author
- ☹ and ☺ Yields *unlimited* number of potential algo interfaces
- ➡ For DaVinci technology, defining the API for key media types would greatly improve
 - Usability
 - Modifiability
 - System design
- ➡ As such, the digital media extensions for xDAIS “xDAIS-DM” or “xDM” has been created to address the above concerns in DaVinci technology
- ➡ Reduces unlimited possibilities to 4 encoder/decoder sets !

xDM/VISA Classes





Universal Class

Universal Algorithm : Methods

```

UNIVERSAL_create ( myCE, "aEncoder", &IUNIVERSAL_Params );
UNIVERSAL_process ( IUNIVERSAL_Handle handle,
  XDM_BufDesc *inBufs,
  XDM_BufDesc *outBufs,
  XDM_BufDesc *inOutBufs,
  IUNIVERSAL_InArgs *inArgs,
  IUNIVERSAL_OutArgs *outArgs );

UNIVERSAL_control ( IUNIVERSAL_Handle handle,
  IUNIVERSAL_Cmd id,
  IUNIVERSAL_DynamicParams *params,
  IUNIVERSAL_Status *status );

UNIVERSAL_delete ( IUNIVERSAL_Handle handle );
  
```

- ◆ **Create each of the required data structures:**
Params, InArgs, OutArgs, DynParams, Status
- ◆ **Structure names must begin with "I" and "my algo's name"**
(which you can read as "interface" to "my algorithm")
- ◆ **Your algo's structures must inherit the IUNIVERSAL datatypes**

Universal Algorithm : Data

```
typedef struct IUNIVERSAL_Params {
    XDAS_Int32    size;
} IUNIVERSAL_Params;

// =====
typedef struct IUNIVERSAL_OutArgs {
    XDAS_Int32    size;
    XDAS_Int32    extendedError;
} IUNIVERSAL_OutArgs;
```

- ◆ Universal interface defined in xDM part of xDAIS spec

/xdais_6_23/packages/ti/xdais/dm/iuniversal.h

Which inherits:

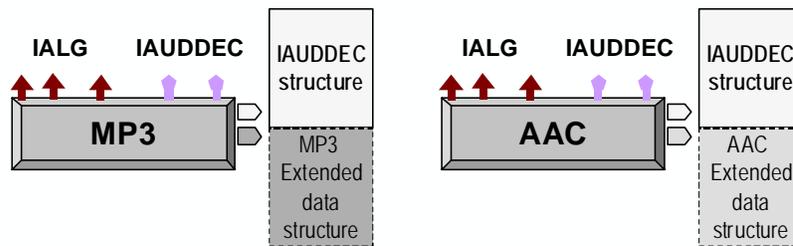
/xdais_6_23/packages/ti/xdais/dm/xdm.h

Which then inherits:

/xdais_6_23/packages/ti/xdais/ialg.h

Extending xDM

Easily Switch xDM Components



- ◆ All audio class decoders (eg: MP3 & AAC) provide the identical API
- ◆ Plug and Play: App using the IAUDDEC_Structures can call all audio decoders
- ◆ Any algorithm specific arguments must be set to default values internally by the vendor (insulating the application from need to specify these parameters)
- ◆ Specific functionality can be invoked by the app using extended data structures
- ◆ To summarize:
 - Most authors can use the default settings of the extended features provided by vendors;
 - "Power users" can (optionally) obtain further tuning via an algos extended structures

Extending xDM – AAC DynamicParams Ex.

```
typedef struct IAUDEC_DynamicParams {
    XDAS_Int32 size;           /* size of this structure */
    XDAS_Int32 outputFormat; /* To set interleaved/Block format. */
} IAUDEC_DynamicParams;

typedef struct IAACDEC_DynamicParams {
    IAUDEC_DynamicParams auddec_dynamicparams;
    Int DownSampleSbr;
} IAACDEC_DynamicParams;
```

AAC Control function code – Using the extended structure

```
XDAS_Int32 AACDEC_TII_control(IAUDEC_Handle AAChandle, IAUDEC_Cmd id,
    IAUDEC_DynamicParams *params, IAUDEC_Status *sPtr)
{
    IAACDEC_DynamicParams *dyparams = (IAACDEC_DynamicParams *)params;
    ...
    case IAACDEC_SETPARAMS:
        if(sizeof(IAACDEC_DynamicParams)==dyparams->auddec_dynamicparams.size)
            handle->downsamplerSBR=dyparams->DownSampleSbr;
        else
            handle->downsamplerSBR=0;
```

AAC and MP3 Extended Data Structures

<pre>typedef struct IAACDEC_Params { IAUDEC_Params auddec_params; } IAACDEC_Params;</pre>	<pre>typedef struct IMP3DEC_Params { IAUDEC_Params auddec_params; } IMP3DEC_Params;</pre>
<pre>typedef struct IAACDEC_DynamicParams { IAUDEC_DynamicParams auddec_dynamicparams; Int DownSampleSbr; // AAC specific } IAACDEC_DynamicParams;</pre>	<pre>typedef struct IMP3DEC_DynamicParams { IAUDEC_DynamicParams auddec_dynamicparam } IMP3DEC_DynamicParams;</pre>
<pre>typedef struct IAACDEC_InArgs { IAUDEC_InArgs auddec_inArgs; } IAACDEC_InArgs;</pre>	<pre>typedef struct IMP3DEC_InArgs { IAUDEC_InArgs auddec_inArgs; XDAS_Int32 offset; } IMP3DEC_InArgs;</pre>
<pre>typedef struct IAACDEC_OutArgs{ IAUDEC_OutArgs auddec_outArgs; }IAACDEC_OutArgs;</pre>	<pre>typedef struct IMP3DEC_OutArgs{ IAUDEC_OutArgs auddec_outArgs; XDAS_Int32 layer; // MP3 specific layer info XDAS_Int32 crcErrCnt; }IMP3DEC_OutArgs;</pre>

Design Your Own – Extending the Universal Class

Creating a Universal Algorithm : Data

```
typedef struct IMYALG_Params {
    IUNIVERSAL_Params base;           // IUNIVERSAL_Params.size
    XDAS_Int32         param1;
    XDAS_Int32         param2;
} IMYALG_Params;

// =====
typedef struct IMYALG_OutArgs {
    IUNIVERSAL_OutArgs base;           // IUNIVERSAL_OutArgs.size
                                        // IUNIVERSAL_OutArgs.extendedError
    XDAS_Int32         outArgs1;
} IMYALG_OutArgs;
```

- ◆ Create each of the required data structures:
Params, InArgs, OutArgs, DynParams, Status
- ◆ Structure names must begin with “I” and “my algo’s name”
(which you can read as “interface” to “my algorithm”)
- ◆ Your algo’s structures must inherit the IUNIVERSAL datatypes

Creating a Universal Algorithm : Methods

```
FIR_create (    myCE, "aEncoder", &IFIR_Params );
FIR_process (  IUNIVERSAL_Handle handle,
               XDM_BufDesc *inBufs,
               XDM_BufDesc *outBufs,
               XDM_BufDesc *inOutBufs,
               IFIR_InArgs *inArgs,
               IFIR_OutArgs *outArgs );
FIR_control (  IUNIVERSAL_Handle handle,
               IFIR_Cmd id,
               IFIR_DynamicParams *params,
               IFIR_Status *status );
FIR_delete (   IUNIVERSAL_Handle handle );
```

Making an Algorithm

Rules of xDAIS

Application / Component Advantages

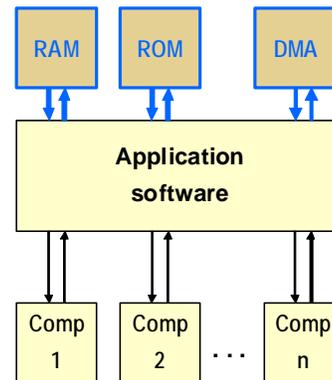
Dividing software between components and system integration provides optimal reuse partitioning, allowing:

- ◆ *System Integrator (SI):* to have full control of [system resources](#)
- ◆ *Algorithm Author:* to write components that can be used in any kind of system

What are “system resources”?

- ◆ CPU Cycles
- ◆ RAM (internal, external) : Data Space
- ◆ DMA hardware
 - Physical channels
 - PaRAMs
 - TCCs

How does the system integrator manage the usage of these resources?



Resource Management : CPU Loading

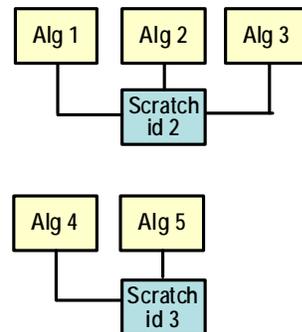
- ◆ All xDAIS algorithms [run only when called](#), so no cycles are taken by algos without being first called by SI (application) code
- ◆ Algos [do not define their own priority](#), thus SI's can give each algo any priority desired – usually by calling it from a BIOS task (TSK)
- ◆ xDAIS algos are required to [publish their cycle loading](#) in their documentation, so SI's know the load to expect from them
- ◆ Algo documentation also must define the worst case latency the algo might impose on the system

Resource Management : RAM Allocation

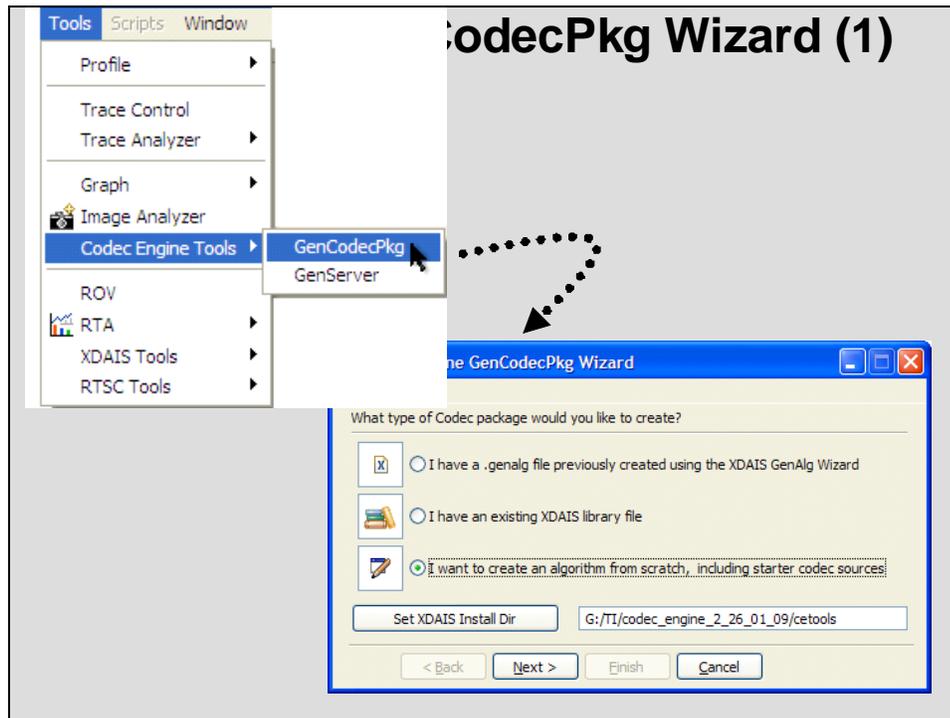
- ◆ **Algos never 'take' memory directly**
 - ◆ Algos tell system its needs (*algNumAlloc()*, *algAlloc()*)
 - ◆ SI determines what memory to give/lend to algo (*MEM_alloc()*)
 - ◆ SI tells algo what memories it may use (*algInit()*)
- ◆ **Algos may request internal or external RAM, but must function with either**
 - ◆ Allows SI more control of system resources
 - ◆ SI should note algo cycle performance can/will be affected
- ◆ **Algo authors can request memory as 'scratch' or 'persistent'**
 - ◆ **Persistent** : ownership of resource must persist during life of algo
 - ◆ **Scratch** : ownership of resource required only when algo is running

Resource Management : Scratch Memory

- ◆ SI can assign a permanent resource to a Scratch request
 - ◆ Easy - requires no management of sharing of temporary/scratch resources
 - ◆ Requires more memory in total to satisfy numerous concurrent algos
- ◆ SI must assure that each scratch is only lent to one algo at a time (*algActivate()*, *algDeactivate()*)
- ◆ No **preemption** amongst algos sharing a common scratch is permitted
 - ◆ Best: share scratch only between equal priority threads - preemption is implicitly impossible
 - ◆ *Tip*: limit number of thread priorities used to save on number of scratch pools required
 - ◆ Other scratch sharing methods possible, but this is method used by C/E
- ◆ Scratch management can yield great benefits
 - ◆ More usage of highly prized internal RAM
 - ◆ Smaller total RAM budget
 - ◆ Reduced cost, size, and power when less RAM is specified



Using the Algorithm Wizard



Files Created by genCodecPkg

Name	Size	Type	Date Modified
.settings		Folder	10/29/2010 1:42 PM
.cdtproject	1 KB	CDTPROJECT File	10/29/2010 1:42 PM
.project	3 KB	PROJECT File	10/29/2010 1:42 PM
config.bld	2 KB	EditPlus JavaScript ...	10/29/2010 1:42 PM
makefile	1 KB	File	10/29/2010 1:42 PM
mixer.c	11 KB	EditPlus C/C++ (.c)	10/29/2010 1:42 PM
MIXER.xdc	1 KB	EditPlus JavaScript ...	10/29/2010 1:42 PM
MIXER.xs	2 KB	EditPlus JavaScript ...	10/29/2010 1:42 PM
mixer_tto.h	3 KB	EditPlus C/C++ (.h)	10/29/2010 1:42 PM
mixer_tto_priv.h	2 KB	EditPlus C/C++ (.h)	10/29/2010 1:42 PM
package.bld	3 KB	EditPlus JavaScript ...	10/29/2010 1:42 PM
package.xdc	1 KB	EditPlus JavaScript ...	10/29/2010 1:42 PM
package.xs	2 KB	EditPlus JavaScript ...	10/29/2010 1:42 PM
tto_codecs_mixer_wizard.gencodecpkg	2 KB	GENCODECPKG File	10/29/2010 1:42 PM

Code Created: Functions

```

/*
 * ===== MIXER_TTO_IMIXER =====
 * This structure defines TTO's implementation of the IUNIVERSAL
 * interface for the MIXER_TTO module.
 */
IUNIVERSAL_Fxns MIXER_TTO_IMIXER = {
    {IALGFXNS},
    MIXER_TTO_process,
    MIXER_TTO_control,
};

```

- ◆ The required iAlg functions (as discussed) plus functions defined by iUniversal class
- ◆ `_process()`
- ◆ `_control()`

Code Created : algAlloc()

```

Int MIXER_TTO_alloc( const IALG_Params *algParams,
                    IALG_Fxns **pf, IALG_MemRec memTab[])
{
    /* Request memory for my object */
    memTab[0].size      = sizeof(MIXER_TTO_Obj);
    memTab[0].alignment = 0;
    memTab[0].space     = IALG_EXTERNAL;
    memTab[0].attrs     = IALG_PERSIST;

    return (1);
}

```

Code Created : algInit

```

Int MIXER_TTO_initObj( IALG_Handle handle, const IALG_MemRec memTab[],
                     IALG_Handle parent, const IALG_Params *algParams )
{
    const IMIXER_Params *params = (IMIXER_Params *)algParams;

    /*
     * Typically, your algorithm will store instance-specific details
     * in the object handle. If you want to do this, uncomment the
     * following line and the 'obj' var will point at your instance object.
     */
    // MIXER_TTO_Obj *obj = (MIXER_TTO_Obj *)handle;

    /*
     * If no creation params were provided, use our algorithm-specific ones.
     * Note that these default values _should_ be documented in your algorithm
     * documentation so users know what to expect.
     */
    if ( params == NULL ) {
        params = &IMIXER_PARAMS;
    }

    /* Store any instance-specific details here, using the 'obj' var above */

    return (IALG_EOK);
}

```

Code Created : process()

```

XDAS_Int32 MIXER_TTO_process ( IUNIVERSAL_Handle h,
                               XDM1_BufDesc *inBufs, XDM1_BufDesc *outBufs,
                               XDM1_BufDesc *inOutBufs,
                               IUNIVERSAL_InArgs *universalInArgs,
                               IUNIVERSAL_OutArgs *universalOutArgs)
{
    XDAS_Int32 numInBytes, i;
    XDAS_Int16 *pIn0, *pIn1, *pOut, gain0, gain1;

    /* Local casted variables to ease operating on our extended
    IMIXER_InArgs *inArgs = (IMIXER_InArgs *)universalInArgs;
    IMIXER_OutArgs *outArgs = (IMIXER_OutArgs *)universalOutArgs;

    /*
    * Note that the rest of this function will be algorithm-spec
    * the initial generated implementation, this process() funct
    * copies the first inBuf to the first outBuf. But you should
    * this to suit your algorithm's needs.
    */

    /* report how we accessed the input buffers */

    /* report how we accessed the output buffer */

    return (IUNIVERSAL_EOK);
}

```

Code Created : process()

```

XDAS_Int32 MIXER_TTO_process ( IUNIVERSAL_Handle h,
                               XDM1_BufDesc *inBufs, XDM1_BufDesc *outBufs,
                               XDM1_BufDesc *inOutBufs,
                               IUNIVERSAL_InArgs *universalInArgs,
                               IUNIVERSAL_OutArgs *universalOutArgs)
{
    XDAS_Int32 numInBytes, i;
    XDAS_Int16 *pIn0, *pIn1, *pOut, gain0, gain1;

    /* Local casted variables to ease operating on our extended
    IMIXER_InArgs *inArgs = (IMIXER_InArgs *)universalInArgs;
    IMIXER_OutArgs *outArgs = (IMIXER_OutArgs *)universalOutArgs;

    /*
    * Note that the rest of this function will be algorithm-spec
    * the initial generated implementation, this process() funct
    * copies the first inBuf to the first outBuf. But you should
    * this to suit your algorithm's needs.
    */

    /* report how we accessed the input buffers */

    /* report how we accessed the output buffer */

    return (IUNIVERSAL_EOK);
}

```

Appendix

Reference Info

References

- ◆ **Codec Engine Algorithm Creator User's Guide**
SPRUED6 Texas Instruments
- ◆ **Codec Engine Server Integrator's Guide**
SPRUED5 Texas Instruments
- ◆ **xdctools_1_21/doc directory in DVEVM 1.1 software**
Documentation on XDC Configuration Kit and BOM
- ◆ **Using adapters to run xDAIS algorithms in the Codec Engine**
SPRAAE7 Texas Instruments

Glossary

API	Application Programming Interface
Codec Engine	DaVinci framework for instantiating and using remote or local codecs
DMAN	Dma MANager module. Manages DMA resource allocation
DSKT2	Dsp SocKeT module, rev. 2. Manages DSP memory allocation
DSP Link	Physical Transport Layer for Inter-processor Communication
Engine	CE framework layer for managing local and remote function calls
EPSI API	Easy Peripheral Software Interface API. Interface to system drivers.
OSAL	Operating System Abstraction Layer
RPC	Remote Procedure Call
Server	Remote Thread that Services Create/Delete RPC's from the Engine
Skeleton	Remote Thread that Services Process/Control RPC's for Codecs
Stub	Function that Marshalls RPC Arguments for Transport over DSP Link
VISA API	Functions to interface to xDM-compliant codecs using CE framework
xDAIS	eXpress DSP Algorithm Interface Standard. Used to instantiate algos
xDM	Interface that extends xDAIS, adding process and control functionality

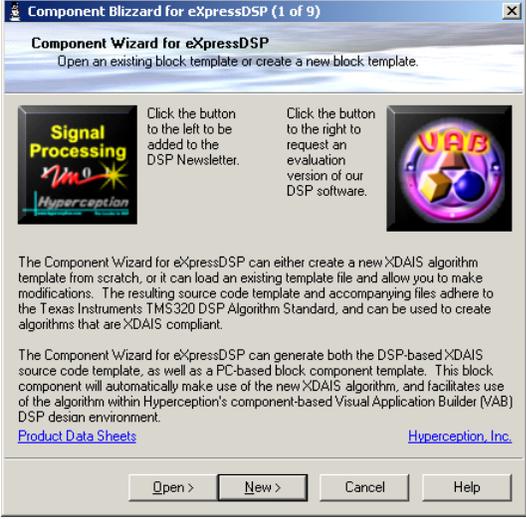
Using the Old Algorithm Wizard

Creating alg Interfaces: Component Wizard



This chapter documents the older wizard available in xDAIS 4.0.

Find updated Algorithm Wizard available for purchase from NEXTXEN at:
<http://www.xdaiswizard.com>



The Component Wizard for eXpressDSP can either create a new xDAIS algorithm template from scratch, or it can load an existing template file and allow you to make modifications. The resulting source code template and accompanying files adhere to the Texas Instruments TMS320 DSP Algorithm Standard, and can be used to create algorithms that are xDAIS compliant.

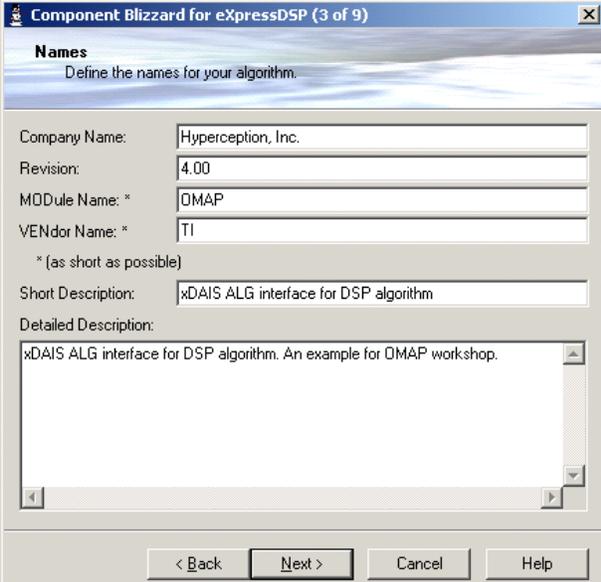
The Component Wizard for eXpressDSP can generate both the DSP-based xDAIS source code template, as well as a PC-based block component template. This block component will automatically make use of the new xDAIS algorithm, and facilitates use of the algorithm within Hyperception's component-based Visual Application Builder (VAB) DSP design environment.

[Product Data Sheets](#) [Hyperception, Inc.](#)



Skip Wizard

Information About the Component



Names
Define the names for your algorithm.

Company Name:

Revision:

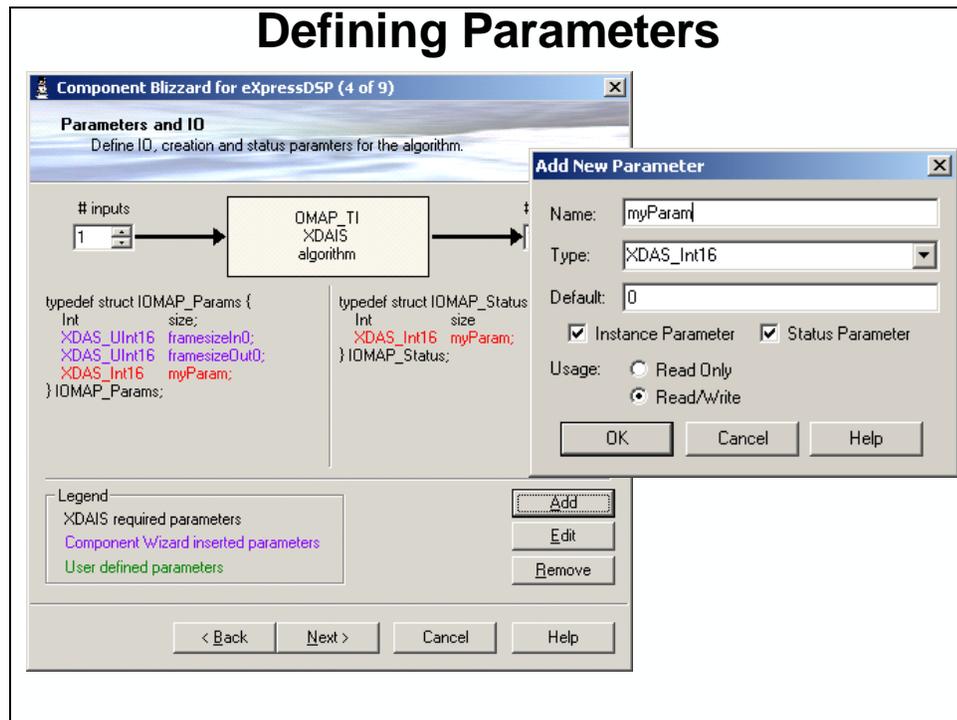
Module Name: *

Vendor Name: *

* (as short as possible)

Short Description:

Detailed Description:

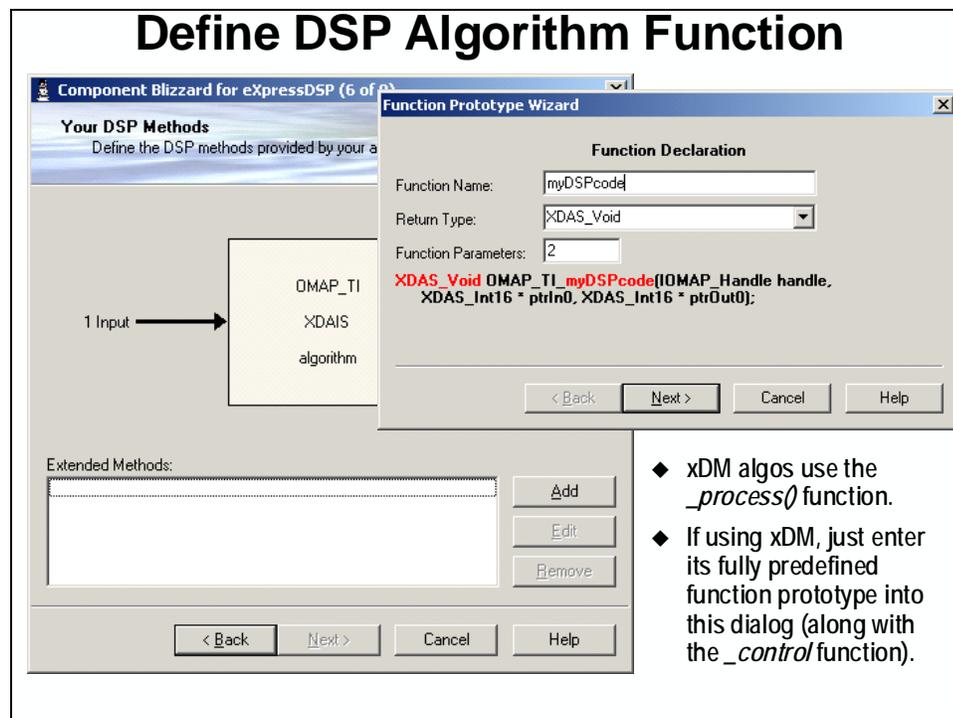
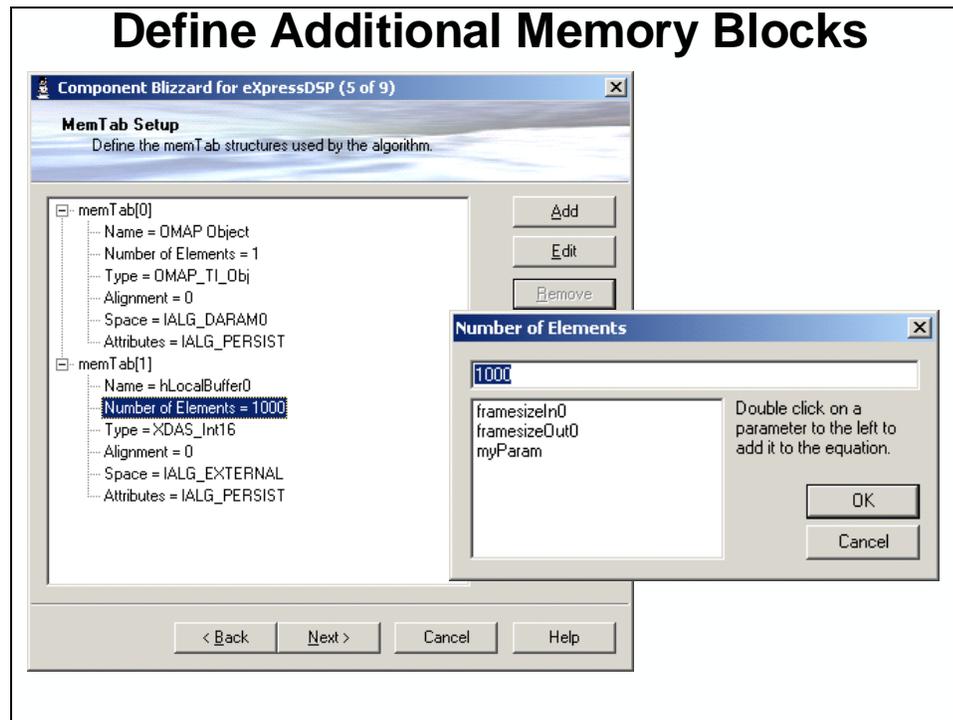


xDM : “Pre-defined” Params

IALG Functions – Common across all codec types

```
Void (*algActivate) (IALG_Handle);
Int (*algAlloc) (const IALG_Params *, struct IALG_Fxns **, IALG_MemRec *);
Void (*algDeactivate) (IALG_Handle)
Int (*algFree) (IALG_Handle, IALG_MemRec *)
Int (*algInit) (IALG_Handle, const IALG_MemRec *, IALG_Handle, const IALG_Params *);
Void (*algMoved) (IALG_Handle, const IALG_MemRec *, IALG_Handle, const IALG_Params *)
Int (*algNumAlloc) (Void)
```

```
typedef struct IAUDDC_Params { // structure used to initialize the algorithm
    XDAS_Int32 size; // size of this structure
    XDAS_Int32 maxSampleRate; // max sampling frequency supported in Hz
    XDAS_Int32 maxBitrate; // max bit-rate supported in bits per secs
    XDAS_Int32 maxNoOfCh; // max number of channels supported
    XDAS_Int32 dataEndianness; // endianness of input data
}IAUDDC_Params;
```



Define Algorithm Parameters... Done

The screenshot shows the 'Define Algorithm Parameters... Done' dialog box. The main window displays 'Your DSP Methods' with a box for 'OMAP_TI XDAIS algorithm' and '1 Input' pointing to it. Below is 'Extended Methods' with a list containing 'XDAS_Void OMAP_TI_myDSPcode(IOMAP_Handle handle, XDAS_Int16 * myFirstParam, XDAS_Int16 * mySecondParam);'. A 'Function Prototype Wizard' dialog is open over this, showing 'Parameter Name: mySecondParam', 'Parameter Type: XDAS_Int16 *', and 'Parameter Number: 2 of 2'. The wizard has 'Back', 'Finish', 'Cancel', and 'Help' buttons.

- ◆ The component wizard has a dialog for creating the function's arguments
- ◆ Since CW hasn't been updated lately, xDM datatypes aren't in the dropdown boxes – you'll enter these manually

Example Audio Decoder Data Structures

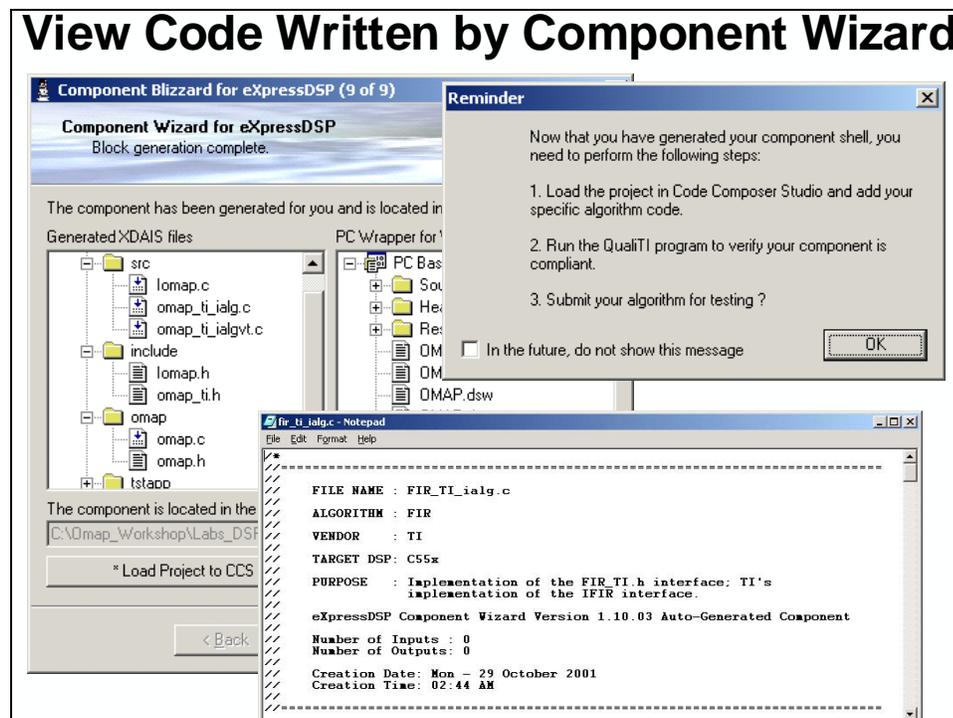
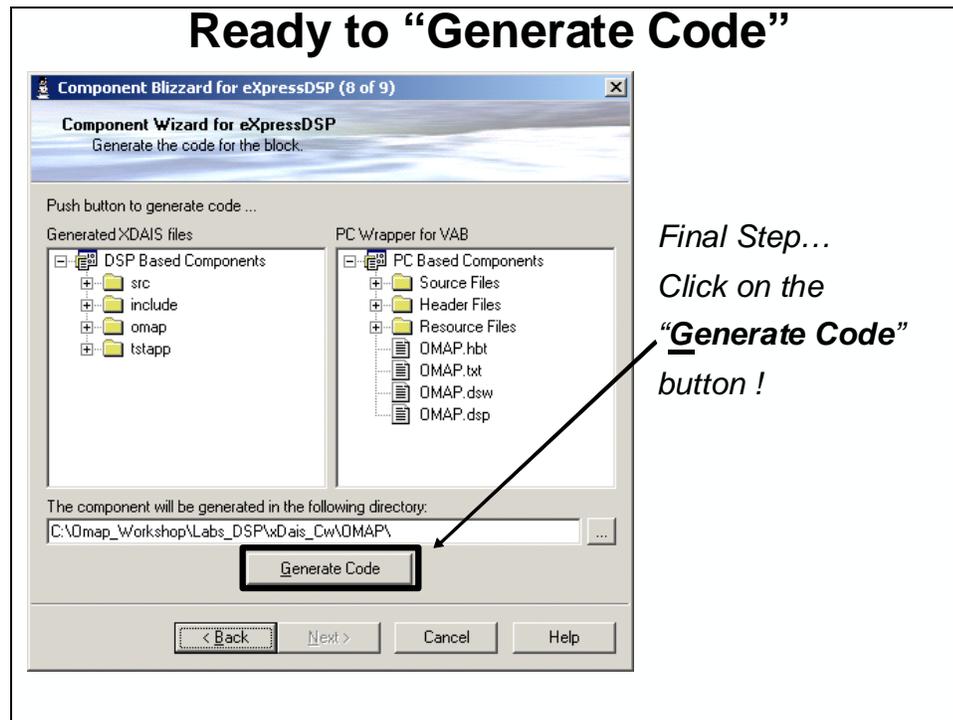
```

typedef struct XDM_BufDesc {           // for buffer description (input and output buffers)
    XDAS_Int32    numBufs;             // number of buffers
    XDAS_Int32    *bufSizes;          // array of sizes of each buffer in 8-bit bytes
    XDAS_Int8     **bufs;             // pointer to vector containing buffer addresses
} XDM_BufDesc;

typedef struct IAUDDEC_InArgs {        // for passing the input parameters for every decoder call
    XDAS_Int32    size;               // size of this structure
    XDAS_Int32    numBytes;          // size of input data (in bytes) to be processed
} IAUDDEC_InArgs;

typedef struct IAUDDEC_OutArgs {       // relays output status of the decoder after decoding
    XDAS_Int32    size;               // size of this structure
    XDAS_Int32    extendedError;      // Extended Error code. (see XDM_ErrorBit)
    XDAS_Int32    bytesConsumed;      // Number of bytes consumed during process call
} IAUDDEC_OutArgs;

```



Component Wizard Made Instance Object

```

/*
//=====
// FIR_TI_Obj
*/
typedef struct FIR_TI_Obj {
    IALG_Obj    alg; /* MUST be first field of all FIR objs */
    XDAS_Int16  firLen;
    XDAS_Int16  blockSize;
    XDAS_Int16 * coeffPtr;
    XDAS_Int16 * workBuffer;
    XDAS_Int16 * historyBuffer;

    /* TODO: add custom fields here */
} FIR_TI_Obj;

```

Component Wizard Made algAlloc()

```

Int FIR_TI_alloc(const IALG_Params *FIRParams, IALG_Fxns **fxns, IALG_MemRec memTab[])
{
    const IFIR_Params *params = (Void *)FIRParams;

    if (params == NULL) {
        params = &IFIR_PARAMS; /* set default parameters */
    }

    memTab[0].size = sizeof(FIR_TI_Obj);
    memTab[0].alignment = (4 * 8) / CHAR_BIT;
    memTab[0].space = IALG_SARAM0;
    memTab[0].attrs = IALG_PERSIST;

    memTab[WORKBUFFER].size = (params->firLen+params->blockSize-1) * sizeof(XDAS_Int16);
    memTab[WORKBUFFER].alignment = (2 * 8) / CHAR_BIT;
    memTab[WORKBUFFER].space = IALG_SARAM0;
    memTab[WORKBUFFER].attrs = IALG_SCRATCH;

    memTab[HISTORYBUFFER].size = (params->firLen-1) * sizeof(XDAS_Int16);
    memTab[HISTORYBUFFER].alignment = (2 * 8) / CHAR_BIT;
    memTab[HISTORYBUFFER].space = IALG_EXTERNAL;
    memTab[HISTORYBUFFER].attrs = IALG_PERSIST;

    return (MTAB_NRECS);
}

```

Component Wizard Made algFree()

```

Int FIR_TI_free(IALG_Handle handle, IALG_MemRec memTab[ ])
{
    Int n;
    FIR_TI_Obj *FIR = (Void *)handle;

    n = FIR_TI_alloc(NULL, NULL, memTab);

    memTab[WORKBUFFER].base = FIR->workBuffer;
    memTab[WORKBUFFER].size = (FIR->firLen+FIR->blockSize-1) * sizeof(XDAS_Int16);
    memTab[HISTORYBUFFER].base = FIR->historyBuffer;
    memTab[HISTORYBUFFER].size = (FIR->firLen-1) * sizeof(XDAS_Int16);

    return (n);
}

```

Component Wizard Made algInit()

```

Int FIR_TI_initObj(IALG_Handle handle, const IALG_MemRec memTab[ ],
                 IALG_Handle p, const IALG_Params *FIRParams)
{
    FIR_TI_Obj *FIR = (Void *)handle;
    const IFIR_Params *params = (Void *)FIRParams;

    if(params == NULL){
        params = &IFIR_PARAMS; /* set default parameters */
    }

    FIR->firLen = params->firLen;
    FIR->blockSize = params->blockSize;
    FIR->coeffPtr = params->coeffPtr;
    FIR->workBuffer = memTab[WORKBUFFER].base;
    FIR->historyBuffer = memTab[HISTORYBUFFER].base;

    /* TODO: Implement any additional algInit desired */

    return (IALG_EOK);
}

```

algActivate & algDeactivate Incomplete...

```

Void FIR_TI_activate(IALG_Handle handle)
{
    FIR_TI_Obj *FIR = (Void *)handle;

    // TODO: implement algActivate
    // TODO: Initialize any important scratch memory values to FIR->workBuffer
}

```

```

Void FIR_TI_deactivate(IALG_Handle handle)
{
    FIR_TI_Obj *FIR = (Void *)handle;

    // TODO: implement algDeactivate
    // TODO: Save any important scratch memory values from FIR->workBuffer
    //   to persistent memory.
}

```

algActivate / algDeactivate Completed ▶

```

Void FIR_TI_activate(IALG_Handle handle)
{
    FIR_TI_Obj *FIR = (Void *)handle;

    memcpy((Void *)FIR->workBuffer, (Void *)FIR->historyBuffer,
           (FIR->firLen-1) * sizeof(Short));
}

```

```

Void FIR_TI_deactivate(IALG_Handle handle)
{
    FIR_TI_Obj *FIR = (Void *)handle;

    memcpy((Void *)FIR->historyBuffer, (Void *)FIR->workBuffer +
           FIR->blockSize, (FIR->firLen-1) * sizeof(Short));
}

```

(Optional) xDAIS Data Structures

The Param Structure

Purpose : To allow the application to specify to the algorithm the desired modes for any options the algorithm allows, eg: size of arrays, length of buffers, Q of filter, etc...

<code>sizeof()</code>	Defined by : <i>Algorithm</i>
<code>filterType</code>	<i>(in header file)</i>
<code>filterOrder</code>	Allocated by : <i>Application</i>
<code>bufferSize</code>	
<code>...</code>	Written to by : <i>Application</i>
<code>...</code>	
	Read from by : <i>Algorithm</i>

Param Structures Defined in IMOD.H

// IFIR_Params - structure defines instance creation parameters

```
typedef struct IFIR_Params {
    Int size;                /* 1st field of all params structures */
    XDAS_Int16    firLen;
    XDAS_Int16    blockSize;
    XDAS_Int16 *  coeffPtr;
} IFIR_Params;
```

// IFIR_Status - structure defines R/W params on instance

```
typedef struct IFIR_Status {
    Int    size;                /* 1st field of all status structures */
    XDAS_Int16    blockSize;
    XDAS_Int16 *  coeffPtr;
} IFIR_Status;
```

IFIR_Params : IFIR.C

```
#include <std.h>
#include "ifir.h"
```

```
IFIR_Params IFIR_PARAMS = {
    sizeof(IFIR_Params),
    32,
    1024,
    0,
};
```

Defines Parameter Defaults
Length of Structure
Filter Length
Block Size
Coefficient Pointer

- ◆ User may replace provided IFIR.C defaults with their preferred defaults
- ◆ After defaults are set, Params can be modified for instance specific behavior

```
#include "ifir.h"
```

```
IFIR_Params IFIR_params ;
IFIR_params = IFIR_PARAMS;
IFIR_params.firLen = 64;
IFIR_params.blockSize = 1000;
```

Create a Param structure
Put defaults in Param structure
Override length parameter
Override block size parameter

The MemTab Structure

Purpose : Interface where the algorithm can define its memory needs and the application can specify the base addresses of each block of memory granted to the algorithm

```
size
alignment
space
attrs
*base
```

Defined by : IALG Spec & Algorithm
(*rtn value of algNumAlloc*)

Allocated by : Application
5*algNumAlloc()

```
size
alignment
space
attrs
*base
```

Written to by : Algorithm (4/5) &
Application (base addr)

```
...
...
```

Read from by : Application (4/5) &
Algorithm (base addr)

The Instance Object Structure

Purpose : To allow the application to specify to the algorithm the desired modes for any options the algorithm allows, eg: size of arrays, length of buffers, Q of filter, etc...

```
*fxns
```

Defined by : Algorithm

```
filterLen
```

```
blockSize
```

Allocated by : Application

```
*coeffs
```

via memRec(0) description

```
*workBuf
```

Written to by : Algorithm

```
...
```

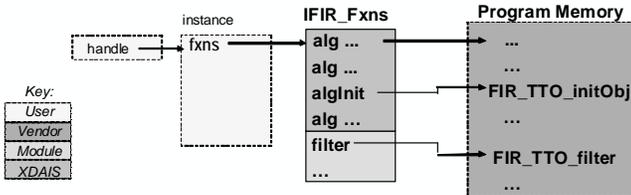
```
...
```

Read from by : Algorithm

(private structure!)

The vTab Concept and Usage

```
#include <ialg.h>
typedef struct IFIR_Fxns {
    IALG_Fxns ialg; /* IFIR extends IALG */
    Void (*filter)(IFIR_Handle handle, XDAS_Int8 in[],XDAS_Int8 out[]);
} IFIR_Fxns;
```



```
hFir->fxns=&FIR_TTO_IFIR;
hFir->fxns->alg.algInit((IALG_Handle)hFir, memTab,NULL,(IALG_Params *)&firParams);
hFir->fxns->filter(hFir,processSrc,processDst);
```

vTab Structure

```
typedef struct IALG_Fxns {
    Void *implementationId;
    Void (*algActivate) (...);
    Int (*algAlloc) (...);
    Int (*algControl) (...);
    Void (*algDeactivate) (...);
    Int (*algFree) (...);
    Int (*algInit) (...);
    Void (*algMoved) (...);
    Int (*algNumAlloc) (...);
} IALG_Fxns;
```

Pragmas - For Linker Control of Code Sections

```
#pragma CODE_SECTION(FIR_TTO_activate, ".text:algActivate")
#pragma CODE_SECTION(FIR_TTO_alloc, ".text:algAlloc")
#pragma CODE_SECTION(FIR_TTO_control, ".text:algControl")
#pragma CODE_SECTION(FIR_TTO_deactivate, ".text:algDeactivate")
#pragma CODE_SECTION(FIR_TTO_free, ".text:algFree")
#pragma CODE_SECTION(FIR_TTO_initObj, ".text:algInit")
#pragma CODE_SECTION(FIR_TTO_moved, ".text:algMoved")
#pragma CODE_SECTION(FIR_TTO_numAlloc, ".text:algNumAlloc")
#pragma CODE_SECTION(FIR_TTO_filter, ".text:filter")
```

Linker Control of Code Sections

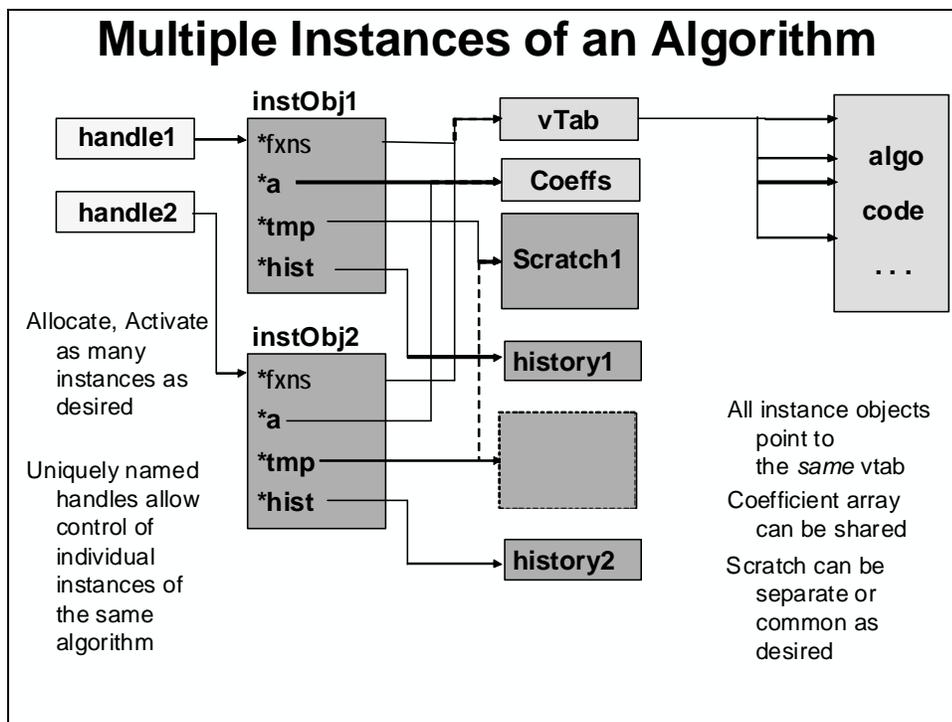
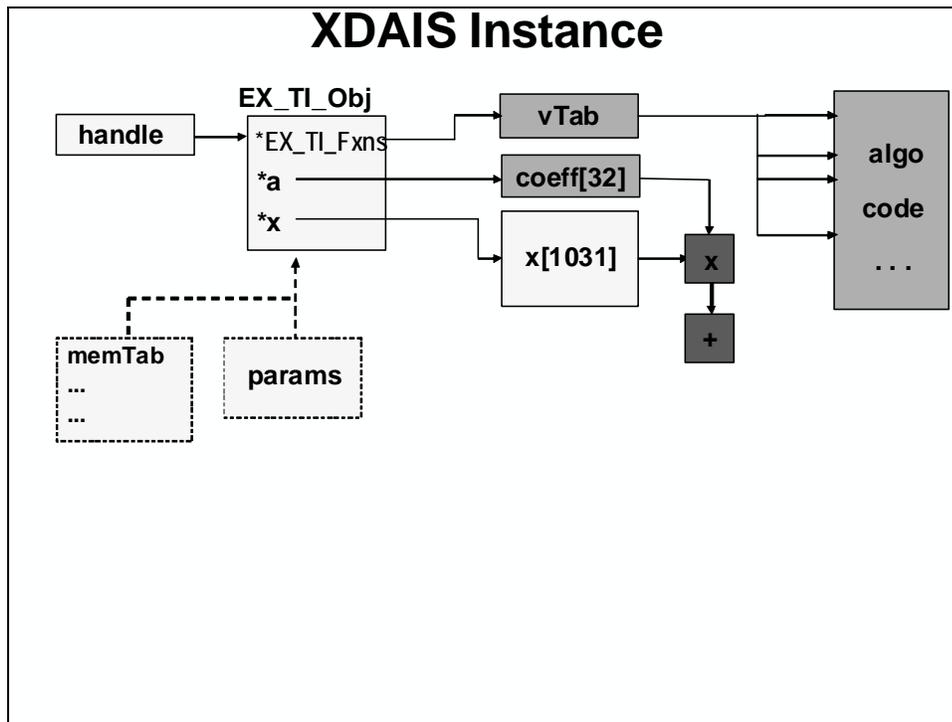
- ◆ Users can define, with any degree of specificity, where particular algo components will be placed in memory

```
.text:algActivate > IRAM
.text:algDeactivate > IRAM
.text:filter > IRAM
.text > SDRAM
```

- ◆ Components not used may be discarded via the “NOLOAD” option

```
.text:algActivate > IRAM
.text:algDeactivate > IRAM
.text:filter > IRAM
.text:algAlloc > SDRAM, type = NOLOAD
.text:algControl > SDRAM, type = NOLOAD
.text:algFree > SDRAM, type = NOLOAD
.text:algMoved > SDRAM, type = NOLOAD
.text:algNumAlloc > SDRAM, type = NOLOAD
.text > SDRAM
```

(Optional) Multi-Instance Ability



(Optional) xDAIS : Static vs Dynamic**xDAIS : Static vs. Dynamic**

Algorithm Lifecycle	Static	Dynamic
Create	<i>SINE_init</i>	<i>algNumAlloc</i> <i>algAlloc</i> <i>algInit (aka sineInit)</i>
Process	<i>SINE_value</i> <i>SINE_blockFill</i>	<i>SINE_value</i> <i>SINE_blockFill</i>
Delete	- none -	<i>algFree</i>

- ◆ Static usage requires programmer to read algo datasheet and assign memory manually.
- ◆ Codec Engine only uses the *Dynamic* features of xDAIS.

**Dynamic (top) vs Static (bottom)**

```

1 n = fxns->ialg.algNumAlloc(); //Determine number of buffers required
    memTab = (IALG_MemRec *)malloc (n*sizeof(IALG_MemRec) ); //Build the memTab
    n = fxns->ialg.algAlloc((IALG_Params *)params,&fxnsPtr,memTab); //Inquire buffer needs from alg

2 for (i = 0; i < n; i++) { //Allocate memory for algo
        memTab[i].base = (Void *)memalign(memTab[i].alignment, memTab[i].size); }

3 alg = (IALG_Handle)memTab[0].base; //Set up handle and *fxns pointer
    alg->fxns = &fxns->ialg;

4 fxns->ialg.algInit(alg, memTab, NULL, (IALG_Params *)params); // initialize instance object

1 IALG_MemRec memTab[1]; // Create table of memory requirements
    int buffer0[5]; // Reserve memory for instance object

2 memTab[0].base = buffer0; // with 1st element pointing to object itself

3 ISINE_Handle sineHandle; // Create handle to InstObj
    sineHandle = memTab[0].base; // Setup handle to InstObj
    sineHandle->fxns = &SINE_TTO_ISINE; // Set pointer to algo functions

4 sineHandle->fxns->ialg.algInit((IALG_Handle)sineHandle,memTab,NULL,(IALG_Params *)&sineParams);

```

Using EDMA3 and ACPY3

Introduction

In this chapter DMA use by algorithms/codecs will be considered.

Learning Objectives

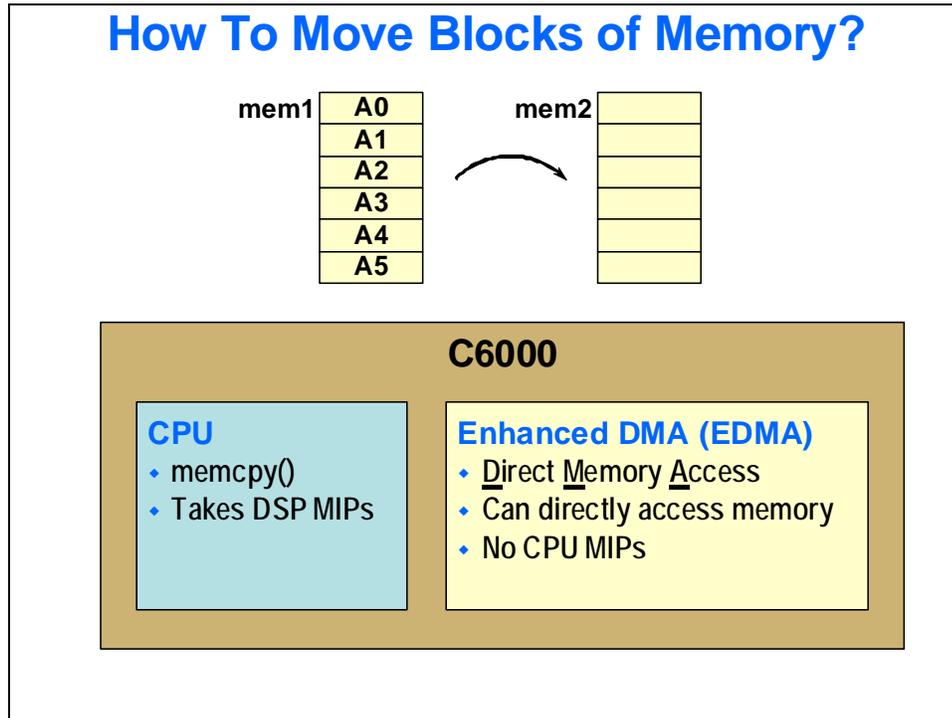
At the conclusion of this chapter, you should be able to:

- ◆ Describe the range of operations possible with the DMA
- ◆ Demonstrate how to use ACPY3 API to perform DMA operations
- ◆ Describe how iDMA defines the needs of a given algorithm
- ◆ Describe how DMAN3 manages DMA hardware
- ◆ Show how to control the behavior of DMAN3 via CFG file entries

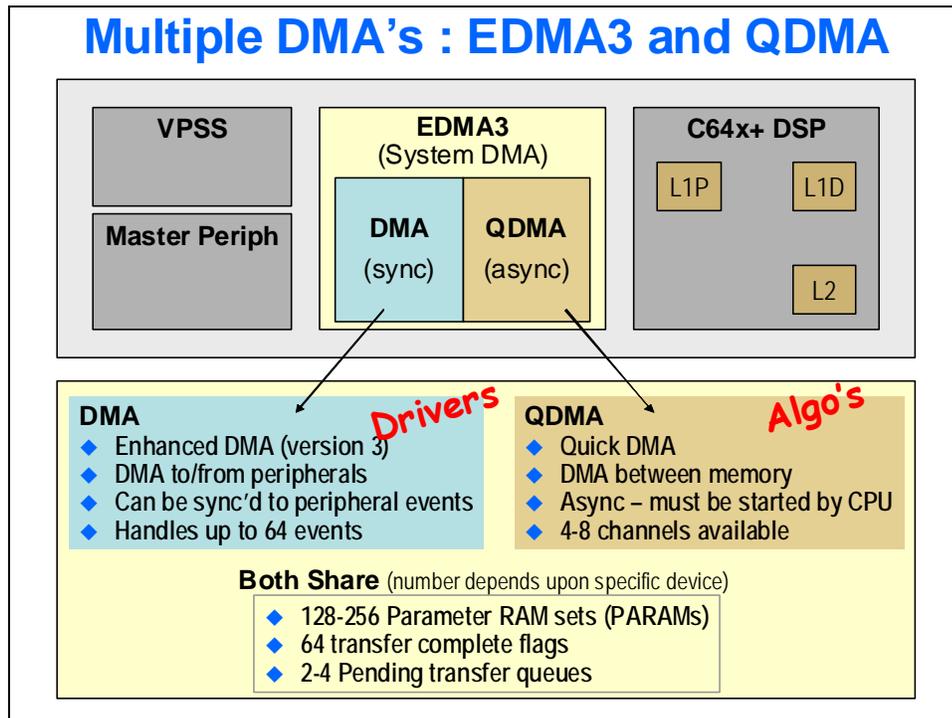
Chapter Topics

Using EDMA3 and ACPY3	14-1
<i>What is a DMA? (Hardware memcpy).....</i>	<i>14-3</i>
<i>EDMA3 Overview</i>	<i>14-3</i>
<i>DMA : Basics</i>	<i>14-4</i>
Basics.....	14-4
DMA Examples	14-5
Basic ACPY3 (memcpy on steroids).....	14-8
<i>DMA : Advanced Basics</i>	<i>14-12</i>
Event, Transfer, Action (ETA)	14-12
Advanced ACPY3	14-13
<i>Working with Fixed Resources</i>	<i>14-14</i>
Create, Process (transfer), Delete	14-14
Writing iDMA3 Functions (Algo Author).....	14-16
Configuring DMAN3 (System Integrator ... i.e. Algo User)	14-17

What is a DMA? (Hardware memcopy)



EDMA3 Overview



DMA : Basics

Basics

DMA : Direct Memory Access

Goal : ♦ Copy from memory to memory

```

graph LR
    A[Original Data Block] --> B[DMA]
    B --> C[Copied Data Block]
    
```

Examples : ♦ Import raw data from off-chip to on-chip before processing
 ♦ Export results from on-chip to off-chip afterward

Controlled by : ♦ Transfer Configuration (i.e. Parameter Set - aka PaRAM or PSET)
 ♦ Transfer configuration primarily includes 8 control registers

Source
Length
Destination

}

Transfer Configuration

How Much to Move?

Block

Frame 1
Frame 2
⋮
Frame M

“C” Count

Frame

Elem 1
Elem 2
⋮
Elem N

“B” Count

Element

--	--	--	--

“A” Count
(# of contig bytes)

Transfer Configuration

Options	
Source	
B	A
Destination	
Index	
Cnt Reload	Link Addr
Index	Index
Rsvd	C

B Count (# Elements)		A Count (Element Size)	
31	16	15	0

C Count (# Frames)	
31	16 15 0

Let's look at a simple example...

EDMA Example : Indexing (Vertical Line)

Goal:

Transfer 4 vertical elements from loc_8 to a port

loc_8 (bytes)					
1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

myDest:

8
14
20
26

←8 bits→

- ◆ ACNT is again defined as element size : 1 byte
- ◆ Therefore, BCNT is still framesize : 4 bytes
- ◆ SRCBIDX now will be 6 – skipping to next column
- ◆ DSTBIDX now will be 2

Source		= &loc_8
4 = BCNT	ACNT	= 1
Destination		= &myDest
2 = DSTBIDX	SRCBIDX	= 6
DSTCIDX		= 0
SRCCIDX		= 0
CCNT		= 1

EDMA Example : Block Transfer (less efficient)

Goal:

Transfer a 5x4 subset from loc_8 to myDest

Frame

16-bit Pixels					
1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

myDest:

8
9
10
11
14
15
...

←16 bits→

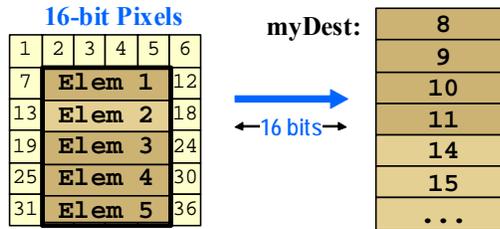
- ◆ ACNT is defined here as 'short' element size : 2 bytes
- ◆ BCNT is again framesize : 4 elements
- ◆ CCNT now will be 5 – as there are 5 frames
- ◆ SRCCIDX skips to the next frame

Source		= &loc_8
4 = BCNT	ACNT	= 2
Destination		= &myDest
2 = DSTBIDX	SRCBIDX	= 2 (2 bytes going from block 8 to 9)
2 = DSTCIDX	SRCCIDX	= 6 (3 elements from block 11 to 14)
CCNT		= 5

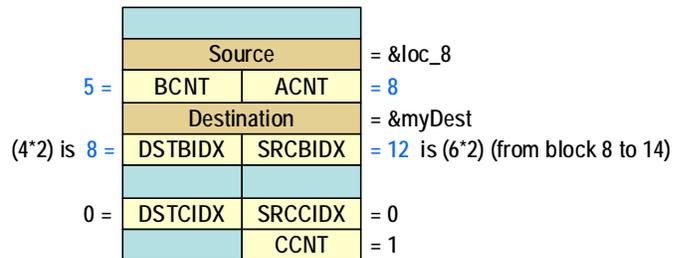
EDMA Example : Block Transfer (more efficient)

Goal:

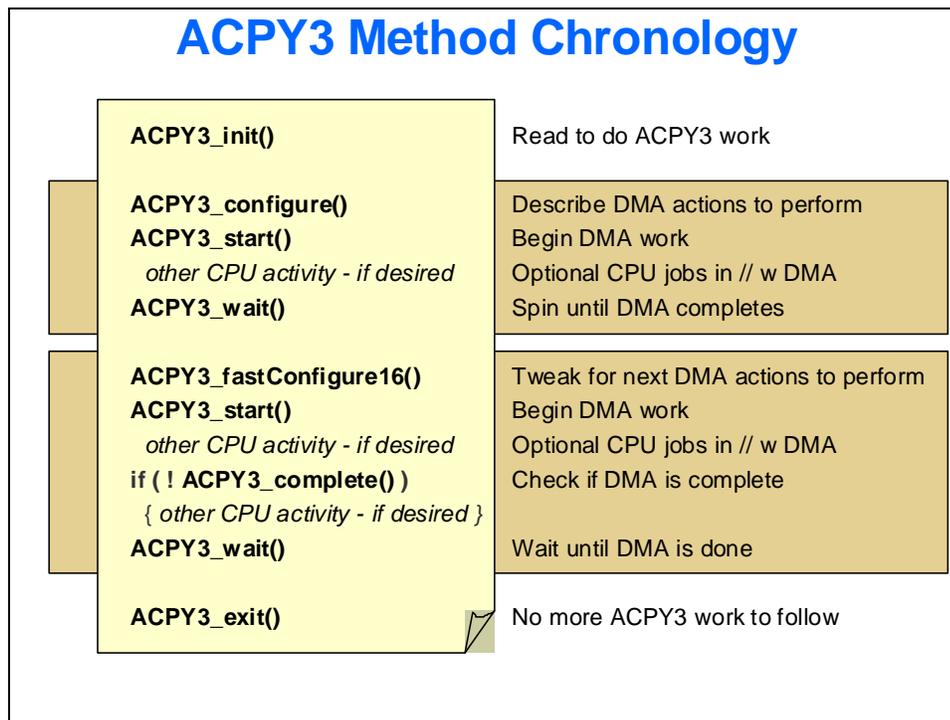
Transfer a 5x4 subset
from loc_8 to myDest



- ◆ ACNT is defined here as the entire frame : 4 * 2 bytes
- ◆ BCNT is the number of frames : 5
- ◆ CCNT now will be 1
- ◆ SRCBIDX skips to the next frame



Basic ACPY3 (memcpy on steroids)



ACPY3_configure

```
extern void ACPY3_configure (IDMA3_Handle hdl
                             ACPY3_PaRam *PaRam, short transferNo);
```

ACPY3_configure must be called at least once for each individual transfer in a logical channel prior to starting the DMA transfer using ACPY3_start()...

ACPY3 PaRam:

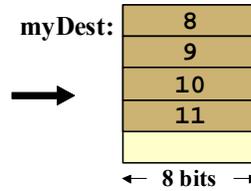
Options		ACPY3_TransferType	transferType	1D1D, 1D2D, 2D1D or 2D2D
Source		Void *	srcAddr	
BCNT	ACNT	Void *	dstAddr	
Destination		MdUns	elementSize	ACNT
DSTBIDX	SRCBIDX	MdUns	numElements	BCNT
BCNTRLD	LINK	MdUns	numFrames	CCNT
DSTCIDX	SRCCIDX	MdInt	srcElementIndex	SRCBIDX
- rsvd -	CCNT	MdInt	dstElementIndex	DSTBIDX
		MdInt	srcFrameIndex	SRCCIDX
		MdInt	dstFrameIndex	DSTCIDX
		MdInt	waitId	-1 (<i>discussed later</i>)

ACPY3_configure Example

Goal:
Transfer 4 elements
from loc_8 to myDest

loc_8 (bytes)

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30



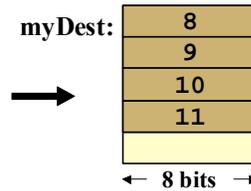
	ACPY3_TransferType	transferType	ACPY3_1D1D
	Void *	srcAddr	(IDMA3_AdrPtr) loc_8
	Void *	dstAddr	(IDMA3_AdrPtr) myDest
Source = &loc_8	MdUns	elementSize	4
BCNT = 1 ACNT = 4	MdUns	numElements	1
Destination = &myDest	MdUns	numFrames	1
DSTBIDX = 0 SRCBIDX = 0	MdInt	srcElementIndex	0
	MdInt	dstElementIndex	0
DSTCIDX = 0 SRCCIDX = 0	MdInt	srcFrameIndex	0
	MdInt	dstFrameIndex	0
CCNT = 1	MdInt	waitId	-1 (discussed later)

ACPY3_configure Example Code

Goal:
Transfer 4 elements
from loc_8 to myDest

loc_8 (bytes)

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30



	ACPY3_PaRam PaRam;
Source = &loc_8	PaRam.srcAddr = (IDMA3_AdrPtr)loc_8;
BCNT = 1 ACNT = 4	PaRam.dstAddr = (IDMA3_AdrPtr)myDest;
Destination = &myDest	PaRam.transferType = IDMA3_1D1D;
DSTBIDX = 0 SRCBIDX = 0	PaRam.elemSize = 4;
	PaRam.numElements = 1;
DSTCIDX = 0 SRCCIDX = 0	PaRam.numFrames = 1;
CCNT = 1	ACPY3_configure(hMyDma, &PaRam);
	ACPY3_start(hMyDma);

TransferType : 1D / 2D

	Source	Destination	ACPY3_PaRam Fields Used
1D1D	[-----]	[-----]	elementSize
1D2D	[---][---][---]...	[---] [---] [---] ...	elementSize numElements dstElementIndex
2D1D	[---] [---] [---] ...	[---][---][---]...	elementSize numElements srcElementIndex
2D2D	[---] [---] [---] ...	[---] [---] [---] ...	elementSize numElements srcElementIndex dstElementIndex

- ◆ Where [-----] represents an "element" of *elementSize*
- ◆ Obviously, all transfers require *srcAddr*, *dstAddr*

N

ACPY3 Interface

ACPY3 Functions	Description
ACPY3_init	Initialize the ACPY3 module
ACPY3_activate	Activate individual DMA channel before using
ACPY3_configure	Configure a logical channel
ACPY3_fastConfigure16b	Modify a single (16-bit) PaRameter of the logical DMA channel
ACPY3_fastConfigure32b	Modify a single (32-bit) PaRameter of the logical DMA channel
ACPY3_start	Submit dma transfer request using current channel settings
ACPY3_wait	Wait for all transfers to complete on a specific logical channel
ACPY3_waitLinked	Wait for an individual transfer to complete on logical channel
ACPY3_complete	Check if the transfers on a specific logical channel have completed
ACPY3_completeLinked	Check if specified transfer on a specific logical channel have completed
ACPY3_setFinal	Specified transfer will be the last in a sequence of linked transfers
ACPY3_deactivate	Deactivate individual DMA channel when done using
ACPY3_exit	Free resources used by the ACPY3 module

ACPY3_fastConfigure32b, 16b

```
void ACPY3_fastConfigure32b ( IDMA3_Handle handle, ACPY3_PaRamField32b fieldId,
                             unsigned int value, short transferNo);
```

```
void ACPY3_fastConfigure16b ( IDMA3_Handle handle, ACPY3_PaRamField16b fieldId,
                              unsigned short value, short transferNo);
```

List of field Id's:

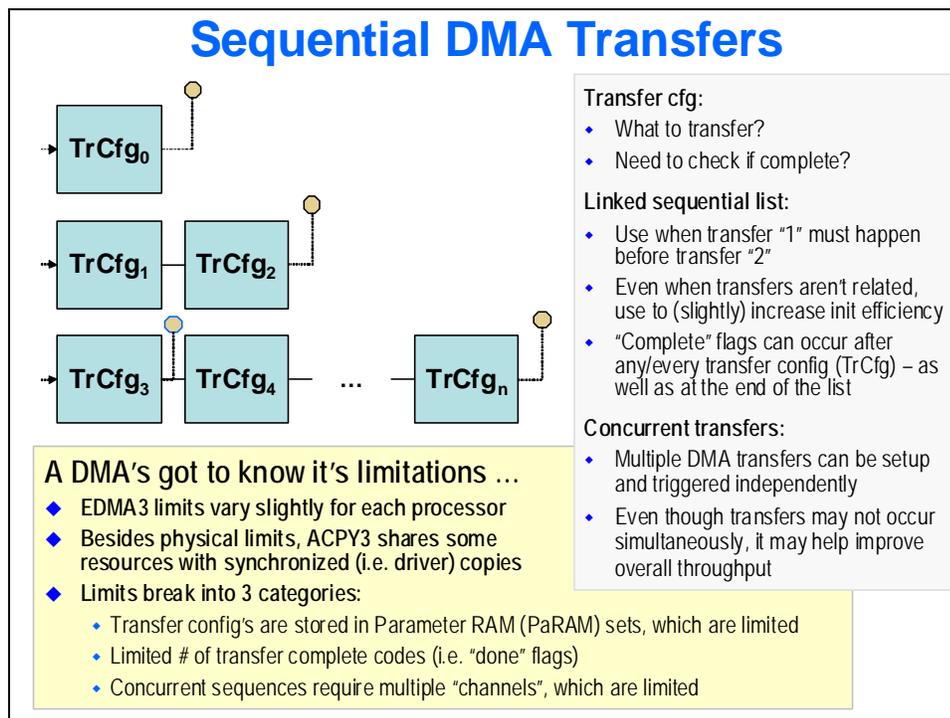
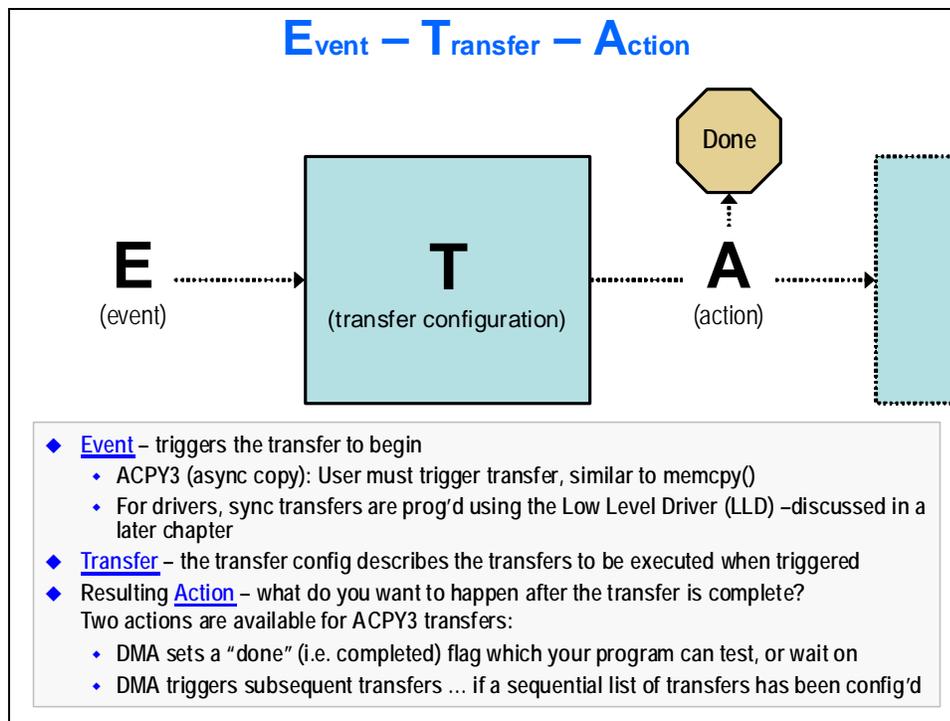
- ◆ *This is a fast configuration function for modifying existing channel settings.*
- ◆ *Exactly one 16 (32) -bit channel transfer property, corresponding to the specified ACPY3_PaRam field, can be modified.*
- ◆ *Remaining settings of the channels configuration are unchanged.*

```
typedef enum ACPY3_PaRamField32b {
    ACPY3_PaRamFIELD_SRCADDR           = 4,
    ACPY3_PaRamFIELD_DSTADDR           = 12,
    ACPY3_PaRamFIELD_ELEMENTINDEXES    = 16,
    ACPY3_PaRamFIELD_FRAMEINDEXES     = 24
} ACPY3_PaRamField32b;
```

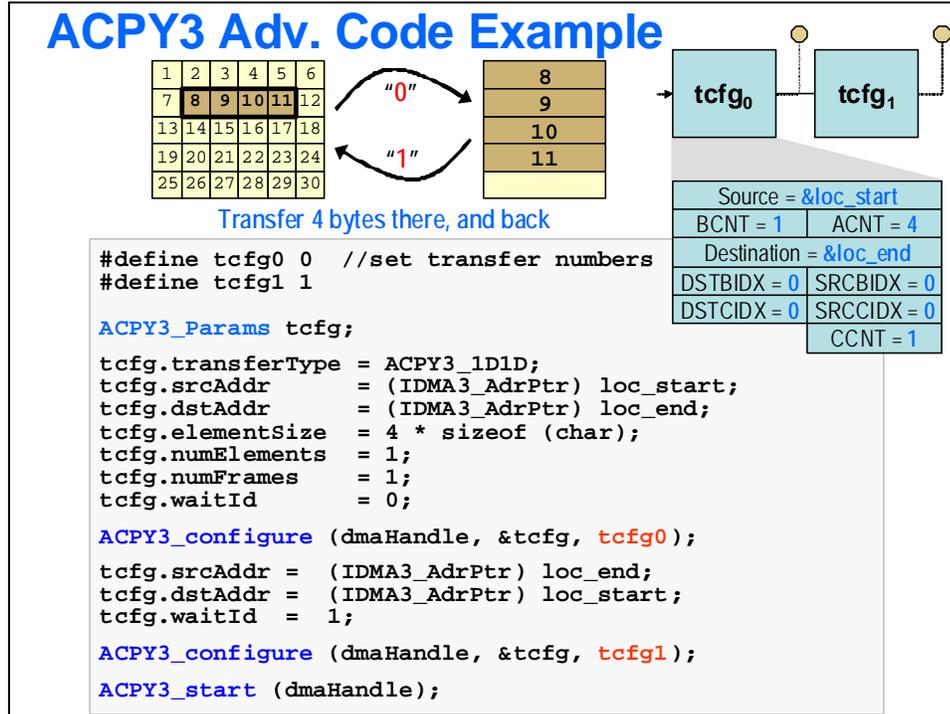
```
typedef enum ACPY3_PaRamField16b {
    ACPY3_PaRamFIELD_ELEMENTSIZE       = 8,
    ACPY3_PaRamFIELD_NUMELEMENTS       = 10,
    ACPY3_PaRamFIELD_ELEMENTINDEX_SRC  = 16,
    ACPY3_PaRamFIELD_ELEMENTINDEX_DST  = 18,
    ACPY3_PaRamFIELD_FRAMEINDEX_SRC    = 24,
    ACPY3_PaRamFIELD_FRAMEINDEX_DST    = 26,
    ACPY3_PaRamFIELD_NUMFRAMES         = 28
} ACPY3_PaRamField16b;
```

DMA : Advanced Basics

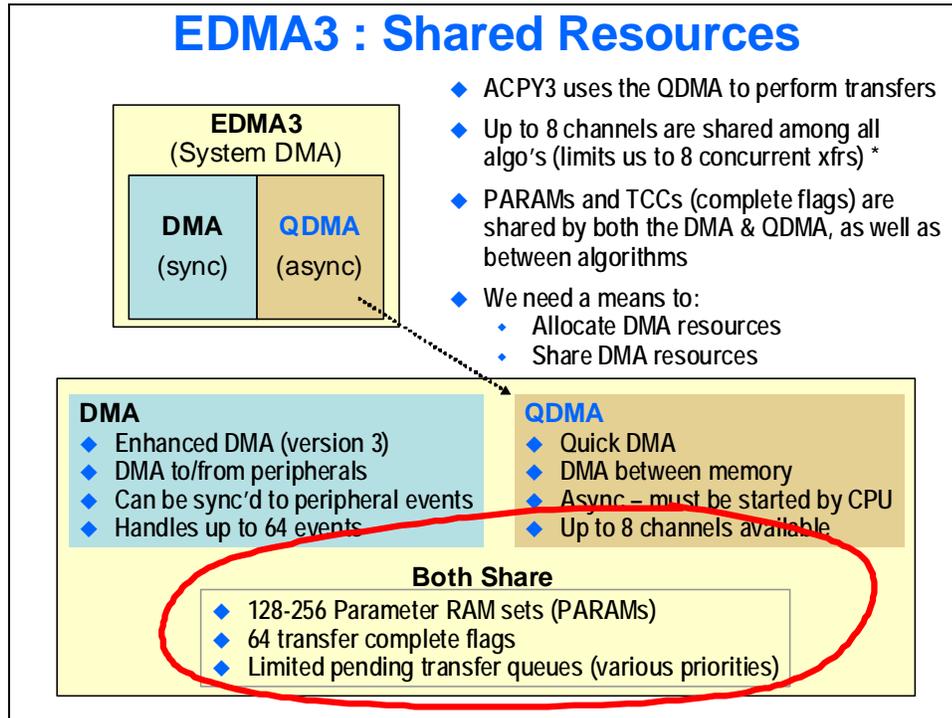
Event, Transfer, Action (ETA)



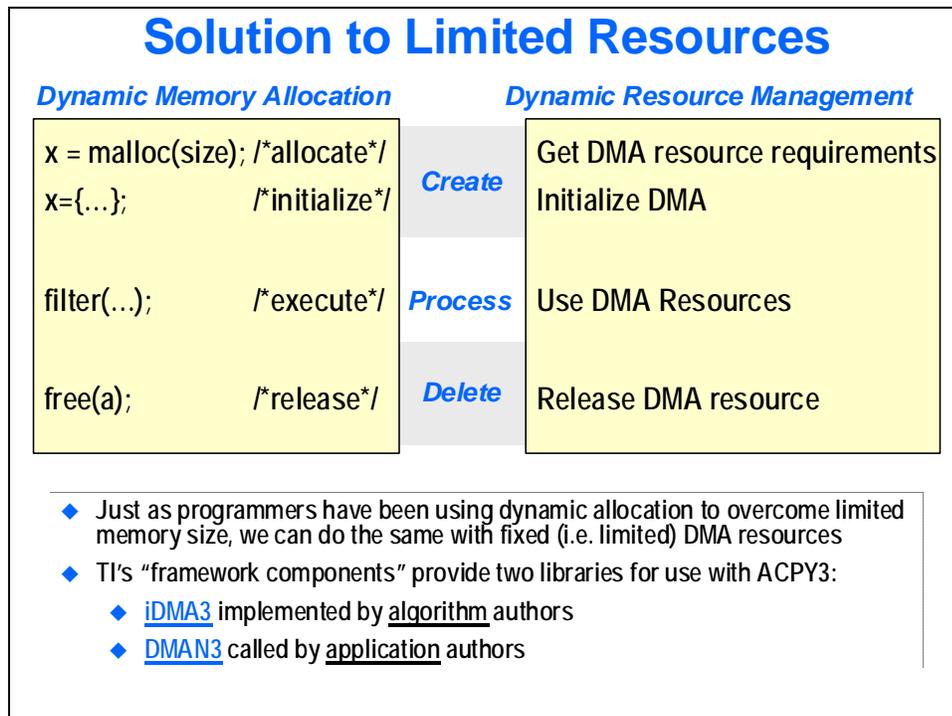
Advanced ACPY3

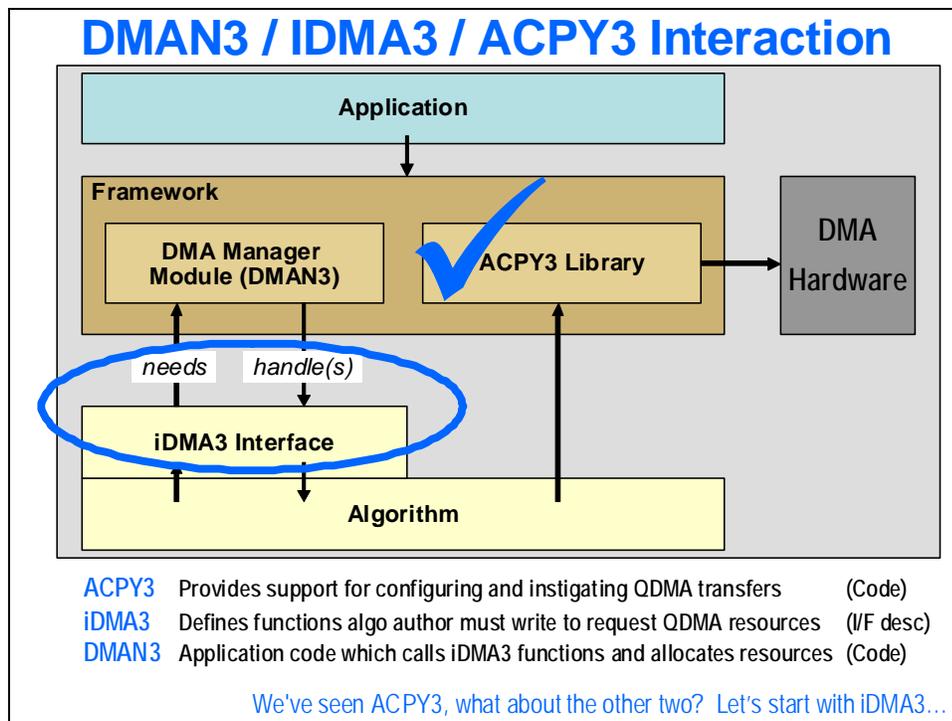
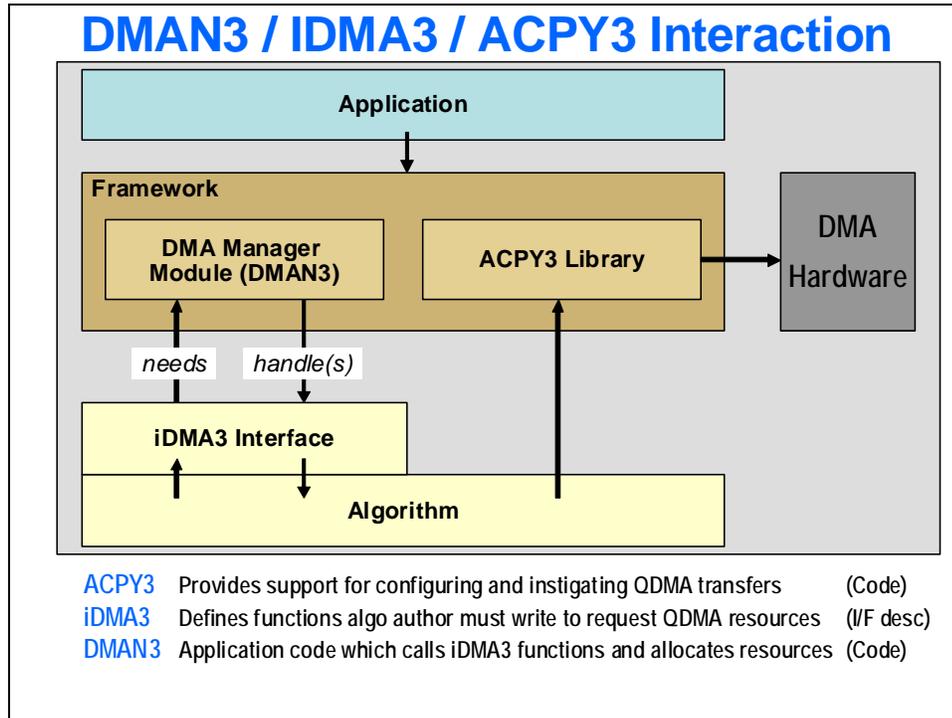


Working with Fixed Resources



Create, Process (transfer), Delete





iDMA3 : Revisiting the Lifecycle of an Algo Instance

Algorithm Lifecycle	Memory (iALG)	DMA (iDMA)
Create ("Constructor")	<i>algNumAlloc</i> <i>algAlloc</i> <i>algInit</i>	<i>dmaGetChannelCnt</i> <i>dmaGetChannels</i> <i>dmaInit</i>
Process	<i>algActivate</i> <i>doDSP</i> (i.e. <i>VIDDEC_process</i>) <i>algDeactivate</i>	<i>ACPY3_activate</i> <i>ACPY3_config</i> <i>ACPY3_start</i> <i>ACPY3_deactivate</i>
Delete ("Destructor")	<i>algFree</i>	<i>dmaGetChannels</i>

- ◆ Great similarity between allocating Memory and DMA resources
- ◆ Like iALG, iDMA supports scratch sharing of resources between algo's

Writing iDMA3 Functions (Algo Author)

Coding iDMA3 Functions (Algo Author)

```

/* How many sets of DMA resources do I need? */
Uns FIR_TTO_dmaGetChannelCnt (Void) {
    return(1); } // I want ONE

/* Fill in the DMA description table for each set of DMA resources you requested */
Uns FIR_TTO_dmaGetChannels (IALG_Handle handle, IDMA3_ChannelRec dmaTab[]) {
    FIR_TTO_Object *hObject = (Void *)handle;

    dmaTab[0].handle = hObject->dmaHandle;

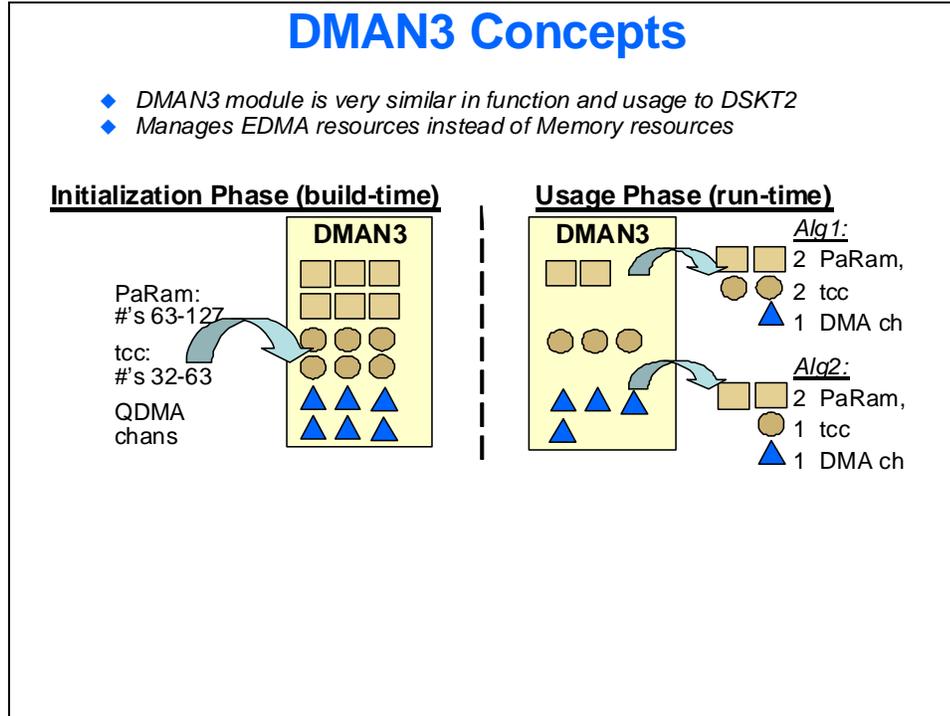
    dmaTab[0].numTransfers = 2; // matches our previous ACPY3 example
    dmaTab[0].numWaits = 2;
    dmaTab[0].priority = IDMA3_PRIORITY_LOW;
    dmaTab[0].protocol = &ACPY3_PROTOCOL; // Using ACPY3 or your own code?
    dmaTab[0].persistent = FALSE;

    return (1); }

/* Save handle to 'your' DMA resources ret'd by DMAN3 into the instance (i.e. class) object */
Int FIR_TTO_dmaInit (IALG_Handle handle, IDMA3_ChannelRec dmaTab[]) {
    FIR_TTO_Object *hObject = (Void *)handle;

    hObject->dmaHandle = dmaTab[0].handle;
    return (retval);
}
    
```

Configuring DMAN3 (System Integrator ... i.e. Algo User)



DMAN3 Setup : server.cfg

```

var DMAN3 = xdc.useModule("ti.sdo.fc.dman3.DMAN3");

// set dman to use all external memory because video codecs take it all
// note: scratch size of 60000 worked for mpeg and then leave internal
DMAN3.idma3Internal = false;
DMAN3.heapInternal = "L1DSRAM";
DMAN3.heapExternal = "DDR";
DMAN3.PaRamBaseIndex = 78;
DMAN3.numPaRamEntries = 48;
DMAN3.numQdmaChannels = 8;
DMAN3.qdmaChannels = [0,1,2,3,4,5,6,7];
DMAN3.numPaRamGroup[0] = 48;
DMAN3.tccAllocationMaskL = 0;
DMAN3.tccAllocationMaskH = 0xff;
DMAN3.numTccGroup[0] = 8;

if( this.prog.build.profile == "debug")
    DMAN3.debug = true;
    
```

In addition to physical DMA resources, module needs some memory for storing PaRam shadows and other channel configuration states.

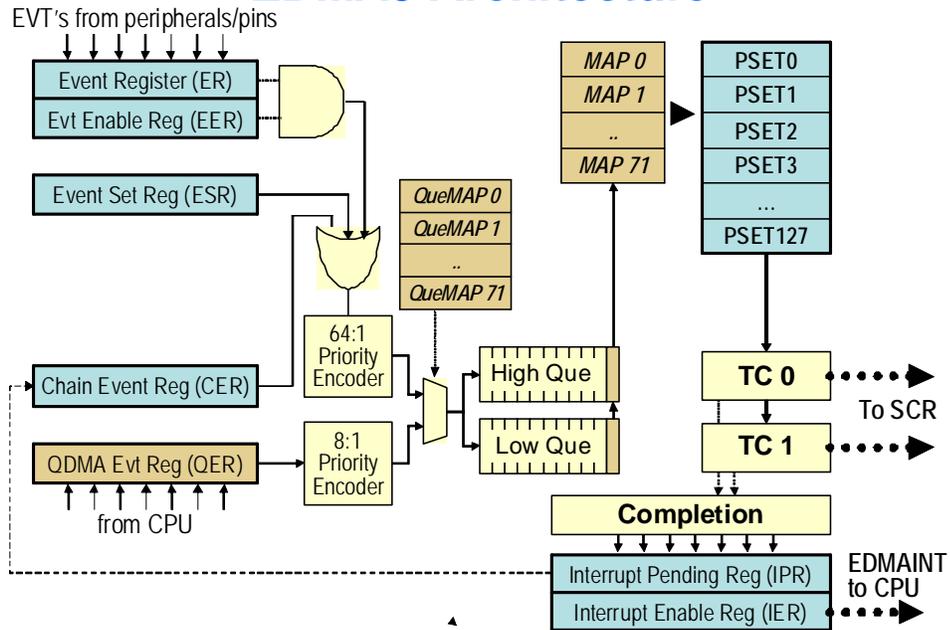
PaRam granted to the DMAN3 module by base index and number

Up to 8 QDMA channels available on 644x

tcc's are granted by bit mask

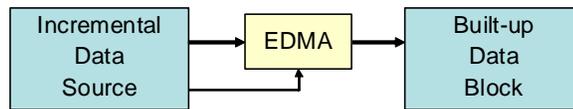
Appendix (More on EDMA3 Architecture)

EDMA3 Architecture



EDMA : Adding Synchronization to DMA

Goal : ◆ Synchronized copy from port to memory



Examples : ◆ Accumulate new data from port to buffer before processing
 ◆ Export results from buffer to port a word at a time afterward

Synchronization Events	Event Enable
0-1 : reserved	0
2 : ASP Rcv	1
3 : ASP Xmt	1
4-7 : VPSS	1
8-11 : VICP	0
12-15 : res'd	0
16-17 : SPI	1
18-23 : UART	1
...	0
52-54 : PWM	1
55-64 : res'd	0

Event occurs → [Table] → DMA job is placed on queue if EE=1

- ◆ Events can act as a trigger for A or B transfers (transfer all 3D with one event using DMA chaining)
- ◆ 1 event queues one transfer
- ◆ Allows multiple ports to be serviced concurrently by the EDMA engine

(Optional) Introduction to DSP/BIOS Link

Introduction

This optional chapter provides a brief introduction to the DSP/BIOS Link protocol which provides a retargetable/portable interface between different processors. While this lower-level layer of target software is used by TI's Codec Engine framework, it can also be used stand-alone.

Covered here are the basic architecture and the commonly used modules that make up the BIOS Link software, as well as a discussion of how the Codec Engine implements and extends the use of the Link protocol. Finally, a set of guidelines – along with a series of different use-cases – are provided to help you determine when it's best to use the Codec Engine, or when it's you may want to only implement the lower-level Link service.

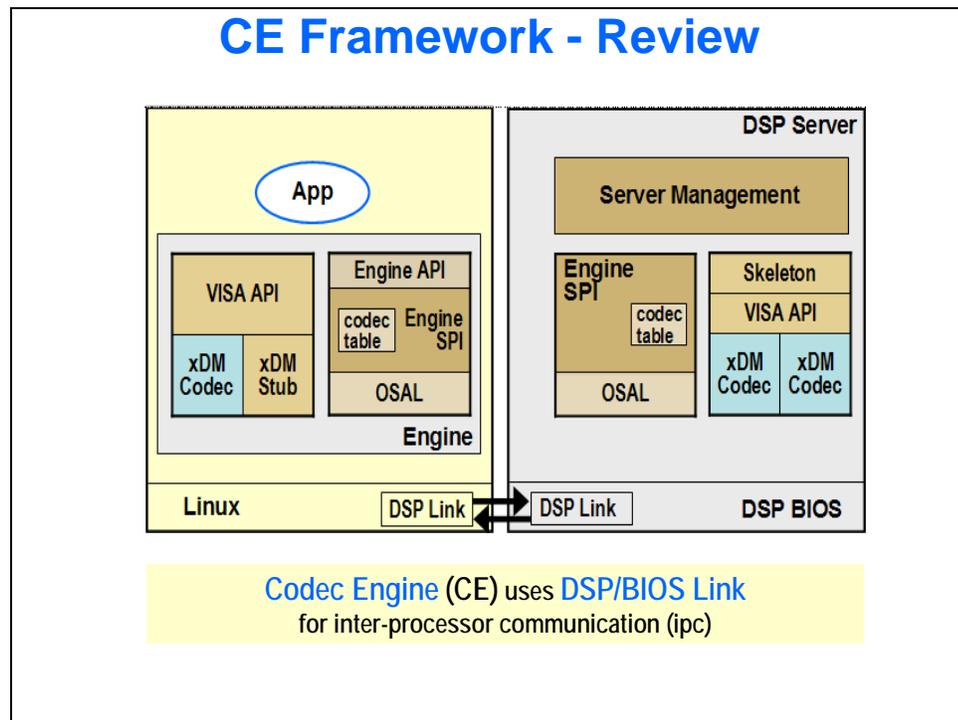
Chapter Topics

(Optional) Introduction to DSP/BIOS Link.....	15-1
<i>Introduction</i>	<i>15-3</i>
Where Have We Seen DSP/BIOS Link Before?	15-3
What is DSP/BIOS Link?	15-4
Codec Engine Advantages.....	15-5
<i>Which Should I Use – Codec Engine vs DSP/BIOS Link.....</i>	<i>15-6</i>
Guidelines.....	15-6
Use Cases	15-7
<i>DSP/BIOS Link Architecture</i>	<i>15-13</i>
<i>DSP/BIOS Link Modules.....</i>	<i>15-14</i>
PROC.....	15-15
MSGQ	15-17
POOL.....	15-20
CHNL	15-22
(Optional) RINGIO.....	15-24
<i>What's Next.....</i>	<i>15-26</i>
<i>How DSPLink API's are used by Codec Engine.....</i>	<i>15-27</i>
<i>For More Information.....</i>	<i>15-28</i>

Introduction

Where Have We Seen DSP/BIOS Link Before?

Looking at the diagram that we've examined throughout most of this workshop, we can see that the DSP Link layer is a driver-level service that provides the intercommunication between both CPU's.



What is DSP/BIOS Link?

Reviewing the description and features on this page, you may notice how similar Link is to the Codec Engine. Obviously, the Codec Engine does a good job using many of the various services provided by Link.

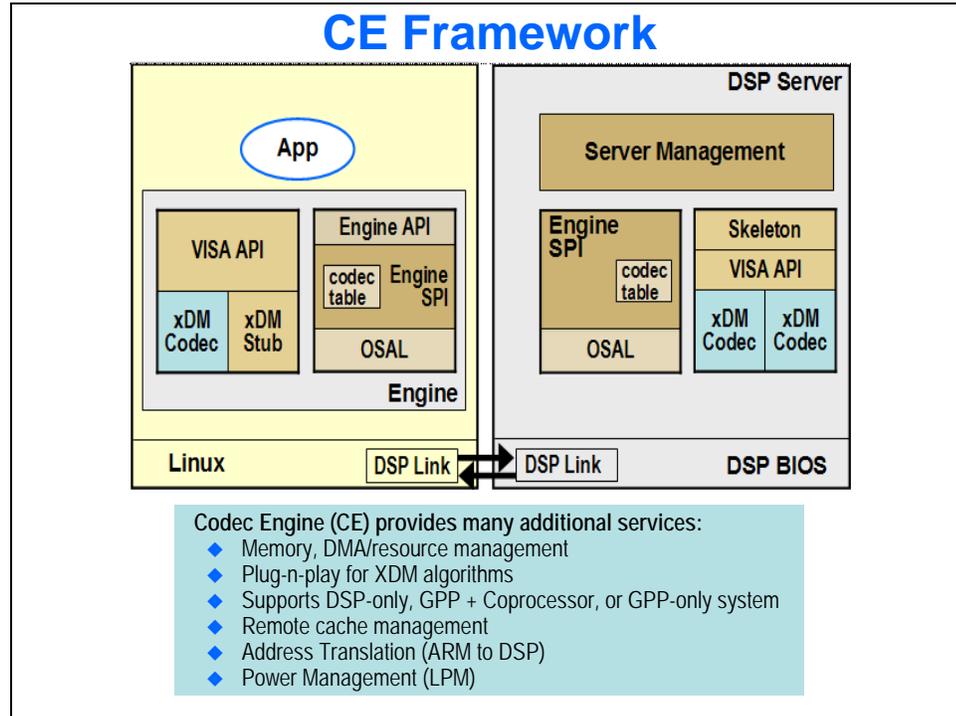
What is DSP/BIOS™ LINK?

- ◆ [Lower level](#) inter processor communication link
- ◆ Allows master processor to [control execution](#) of slave processor
 - ◆ Boot, Load, Start, Stop the DSP
- ◆ Provides [peer-to-peer protocols](#) for communication between the multiple processors in the system
 - ◆ Includes [complete protocols](#) such as MSGQ, CHNL, RingIO
 - ◆ Includes [building blocks](#) such as POOL, NOTIFY, MPCS, MPLIST, PROC_read/PROC_write which can be used to develop different kinds of frameworks above DSPLink
- ◆ Provides [generic APIs](#) to applications
 - ◆ Platform and OS independent
- ◆ Provides a [scalable](#) architecture, allowing system integrator to choose the optimum configuration for footprint and performance

DSPLink Features

- ◆ [Hides platform/hardware](#) specific details from applications
- ◆ [Hides GPP operating system](#) specific details from applications, otherwise needed for talking to the hardware (e.g. interrupt services)
- ◆ Applications written on DSPLink for one platform can directly work on other platforms/OS combinations requiring [no or minor changes](#) in application code
- ◆ Makes applications [portable](#)
- ◆ Allows [flexibility](#) to applications of choosing and using the most appropriate high/low level protocol

Codec Engine Advantages



Bottom line

- When using the DSP to execute algorithms called from the ARM CPU, the Codec Engine provides a greater ease of use due to its simple (but powerful) API, as well as the many additional services it provides.
- Accessing DSP/BIOS Link directly is more appropriate in those cases where additional flexibility is needed; for example, when you want to run two independent programs on the two CPU's and just send some minor communications between them.

Which Should I Use – Codec Engine vs DSP/BIOS Link

Guidelines

Guidelines for Choosing IPC

◆ Codec Engine

- When using XDAIS based Algorithms
- Using multiple algorithms (or instances) on the DSP
- Using the DSP as a the “ultimate” programmable H/W accelerator
- If migration from one platform to another is needed
- You prefer a structured, modular approach to software and want to leverage as much TI software as possible
- When application runs algorithms locally

◆ DSPLink

- When running a stand-alone DSP program which needs to communicate with other processors (i.e. ARM) – often the case when using DSP-side I/O (i.e. DSP-based drivers)
- When communicating between two discrete processors over PCI

Note

While it is possible to use both the Codec Engine and DSPLink simultaneously, this use-case is rarely required. Further it requires knowledge of Codec Engine, DSPLink, and LAD.

Use Cases

Use Case #1

Request:

I'm using DVSDK 2.0 on DM6446 - I want to add another audio algorithm.

Suggestion:

Codec Engine

Guidelines:

- Using xDAIS based algorithm (xDM audio)
- Also, using DSP as algorithm accelerator to the ARM/Linux program

Notes:

- DVSDK includes video, speech, audio, image and universal interfaces
- Details on how to integrate new codecs into DVSDK:
http://wiki.davincidsp.com/index.php?title=How_do_I_Integrate_new_codecs_into_DVSDK

Use Case #2

Request:

I'm using OMAP35x - I want to add a bar-code scanner algo on the DSP

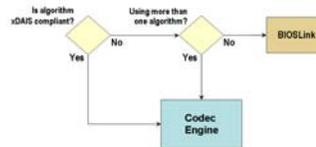
Suggestion:

Codec Engine – if algorithm is xDAIS compliant or using multiple ones

DSPLink – if using single, non-compliant algorithm

Guidelines:

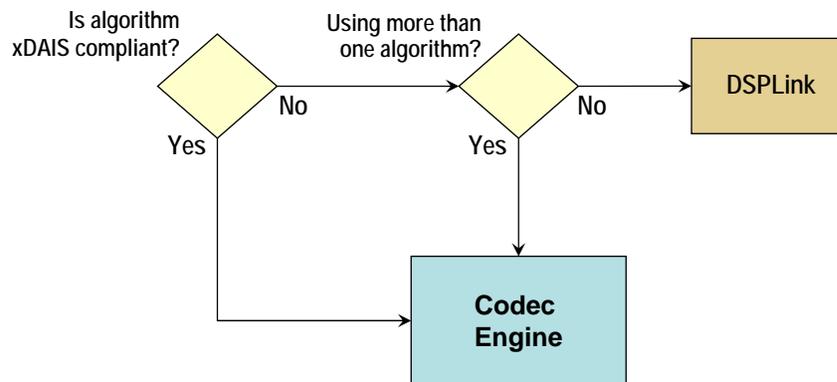
- Multiple – see provided flowchart (next slide)



Notes:

- Using Codec Engine eases burden for ARM/Linux programmer, but requires algorithm author to package DSP algo into a xDAIS/xDM class
- DSPLink provides lower-level interface (simpler architecture), but does not manage resources which makes sharing between algorithms more difficult.

Use Case #2 – Guideline Flowchart



Use Case #3

Request:

I'm using an OMAP-L138 - I want to build my own DSP-side application and use DSP side I/O

Suggestion:

DSPLink

Guidelines:

- Running a stand-alone DSP program which needs to communicate with other processors (i.e. ARM)

Notes:

- ARM is not using DSP as an algorithm accelerator
- Example:
 - Industrial application where ultra-low latency I/O drivers and processing is critical
 - Only req's a few "control cmds" from the ARM-side to influence the DSP processing
- Since this use-case does not require additional services of Codec Engine, the less-complicated DSPLink architecture may be preferred
- If multiple DSP algo's are needed, but being called from the DSP, you might choose to use a "local" Codec Engine
- An example application showing this is part of OMAP-L138 SDK release

Use Case #4

Request:

I'm on DM6446 but I plan to migrate to DM365 later - I want to keep the same APIs.

Suggestion:

Codec Engine !

Guidelines:

- Using xDAIS based algorithm (xDM video/imaging/speech/audio)
- If migrating from one platform to another

Notes:

- Codec Engine is exactly the right fit here as CE has the same APIs regardless of whether you are on an ARM-only, DSP-only, or ARM+DSP
- CE provides needed resource management services to share memory, DMA, co-processors effectively

Use Case #5

Request:

I have an audio/video system - I want to add another non-VISA codec.

Suggestion:

Codec Engine – add algorithm using VISA's iUNIVERSAL API

Guidelines:

- Using multiple algorithms (or instances) on the DSP

Notes:

- Codec Engine framework can be extended to support add'l algorithms
http://wiki.davincidsp.com/index.php?title=Porting_GPP_code_to_DSP_and_Codec_Engine
http://wiki.davincidsp.com/index.php?title=Getting_started_with_IUNIVERSAL
- Example: Bit-blit algorithm has been supported using IUNIVERSAL APIs
<https://gforge.ti.com/gf/project/bitblit/>

Use Case #6

Request:

I am using ARM/x86 (GPP) and multiple DM6437 (DSP) devices in my system connected over PCI. How can I pass information between these discrete processors?

Suggestion:

DSPLink

Guidelines:

- When communicating between discrete processors over PCI
- Running a stand-alone DSP program

Notes:

- Currently, CE only supports ARM+DSP SOC devices (using shared-memory) since shared memory allows for fast data-sharing (i.e. pointer passing) between CPU's.
- DSP and ARM each manage their own I/O; usually IPC is only needed to pass control/command information, as opposed to streaming data
- It's likely that you will use CE on each processor to easily implement 'local' algorithms; but DSPLink would be used for IPC

Use Case #7

Request:

I am using DM355 processor and plan to run audio algorithm on ARM.

Suggestion:

Codec Engine

Guidelines:

- ◆ When application run algorithms locally

Notes:

- ◆ Codec Engine supports running algorithms locally on GPP only (DM355, DM365) or DSP only devices (DM6437, DM648)
- ◆ DSP Link is used for communication between discrete processors and not useful for GPP only or DSP only implementation.

Use Case #8

Request:

I am using OMAP-L138 processor for calling audio algorithm and in addition, I want to get messages from DSP task that is doing I/O to ARM.

Suggestion:

Codec Engine + DSPLink + Link Arbiter Daemon (LAD)

Guidelines:

- ◆ When using XDAIS based Algorithms
- ◆ When running a stand-alone DSP program which needs to communicate with other processors (i.e. ARM) – often the case when using DSP-side I/O (i.e. DSP-based drivers)

Notes:

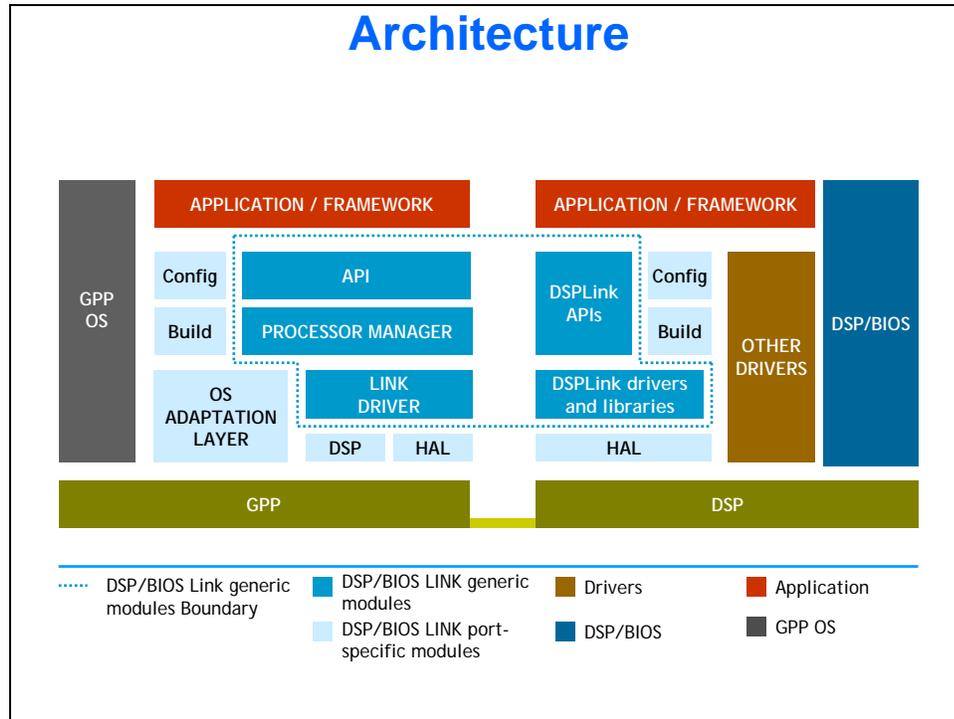
- ◆ Codec Engine is used to call XDAIS audio algorithm.
- ◆ DSP Link is used to exchange messages with DSP task that is doing I/O separately from the DSP Server and codecs combinations.
- ◆ Link Arbiter Daemon (LAD) is needed to support running Codec Engine and DSPLink simultaneously

http://tiexpressdsp.com/index.php/Link_Arbiter_Daemon

IPC Use Case Summary

#	Use Case	CE / Link	Device / Family	Advantage
1	Add xDM algo	dvsdk	DM6446	use ce
2	Add Bar code algo	Algorithm not part of DVSDK	OMAP35x	Use IUNIVERSAL APIs
3	Separate ARM/Linux and DSP programs	DSPLink	OMAP-L1x	For control commands & messages
4	Want to migrate (i.e. port) later on	Codec Engine	dm355 going to omap35x	CE provides migration from GPP only to GPP+DSP
5	A/V system and want to add non-VISA algo	Codec Engine	DM6446, OMAP35x	Use IUNIVERSAL APIs
6	DSP-side I/O	DSPLink	OMAP-L1x	Simpler architecture
7	Multiple algorithms	Codec Engine	DM644x/DM3xx/OMAP3	Multiple XDAIS algorithms
8	xDM algos and own bios program doing DSP I/O	CE + DSPLink + LAD	OMAP-L1x	Preferred APIs

DSP/BIOS Link Architecture



DSP/BIOS Link Modules

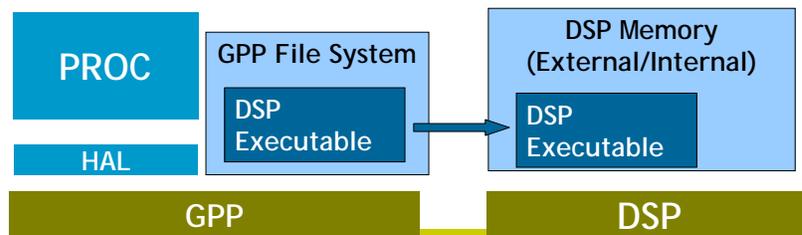
Modules - Overview

- ◆ Device Manager for DSP control
 - [PROC](#): Boot-load, start, stop the DSP
- ◆ Inter-Processor Communication protocols
 - Complete protocols for different types of data transfer between the processors
 - Each protocol meets a different data transfer need
 - [MSGQ](#): Message Queue
 - [CHNL](#): SIO/streaming based on issue-reclaim model
 - [RingIO](#): Circular ring buffer
- ◆ Inter-Processor Communication building blocks
 - Low-level building blocks used by the protocols
 - Each building block is also exposed as APIs to allow framework writers to define their own application-specific protocols
 - [POOL](#): Memory Manager - shared/non-shared
 - [NOTIFY](#): Interrupt abstraction and de-multiplexing
 - [MPCS](#): Multi-processor critical section
 - [MPLIST](#): Multi-processor doubly linked circular list
 - [PROC_read & PROC_write](#): Read from or write to DSP memory

PROC

PROC: DSP Boot-loading

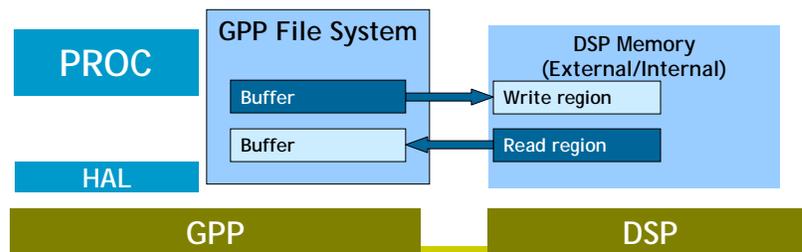
- ◆ DSP executable is present in the GPP file system
- ◆ The specified executable is loaded into DSP memory (internal/external)
- ◆ The DSP execution is started at its entry point
- ◆ Boot-loading using: Shared memory, PCI, etc.



PROC: Write/Read

- ◆ [PROC_write](#)
Write contents of buffer into specified DSP address
- ◆ [PROC_read](#)
Read from specified DSP address into given buffer
- ◆ Can be used for very simple data transfer in static systems
- ◆ Application Example on using PROC_read and PROC_write

http://wiki.davincidsp.com/index.php/Writing_DSPLink_Application_using_PROC_read_and_write_APIs



PROC APIs

PROC_setup	Setup PROC sub-component
PROC_attach	Attach to specific DSP and initialize it
PROC_load	Load the specific base image on target DSP
PROC_start	Starts the execution of loaded code on DSP
PROC_control	Provides a hook to perform device dependent control operations
PROC_debug	Prints the current status of this component for debugging purposes
PROC_loadSection	Load particular section from base image
PROC_destroy	Destroys PROC sub-component
PROC_detach	Detach the DSP
PROC_stop	Stops the execution on the target DSP

PROC APIs (Contd..)

PROC_getState	Get current state of the target DSP
PROC_read	Reads from specific addresses of the target DSP
PROC_write	Writes to specific addresses of the target DSP
PROC_instrument	Gets instrumentation data associated with PMGR_PROC sub-component
PROC_isAttached	Checks if the target DSP is attached

MSGQ

MSGQ: Overview

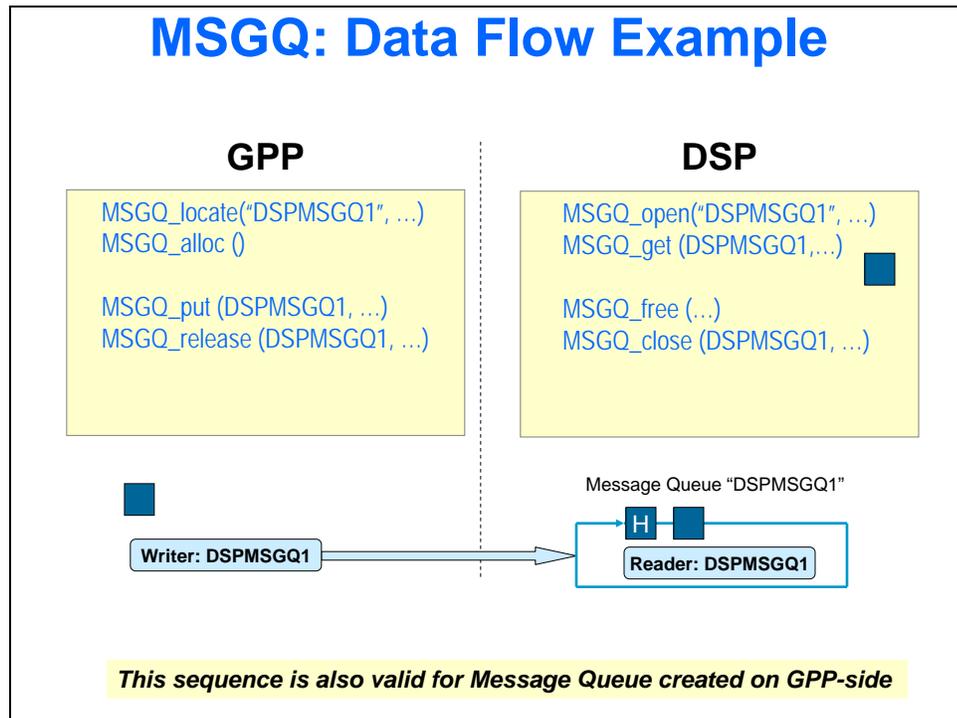
- ◆ Messaging protocol allows clients to send messages to a named Message Queue located on any processor on the system
- ◆ Message Queue can have: single reader, multiple writers



MSGQ: Features

- ◆ Messaging provides logical connectivity between GPP and DSP clients
- ◆ Messages are sent at a higher priority than CHNL data buffers by default (Configurable)
- ◆ Messages can be variable sized
- ◆ Messages can be sent to a named Message Queue
- ◆ Message Queues have unique system-wide names. Senders locate the Message Queue using this name to send messages to it.
- ◆ Client can send messages to Message Queues that are created on any processor connected to the local processor using a transport

MSGQ: Data Flow Example



MSGQ APIs

MSGQ_open	Opens message queue to be used for receiving messages
MSGQ_close	Closes the message queue
MSGQ_locate	Synchronously locates the message queue identified by specified MSGQ name
MSGQ_release	Releases the message queue that was located earlier
MSGQ_alloc	Allocates a message
MSGQ_free	Frees a message
MSGQ_put	Sends a message to specified Message queue
MSGQ_get	Receives a message on specified message queue
MSGQ_setErrorHandler	Allows the user to designate MSGQ as an error handler MSGQ to receive asynchronous error messages from the transports
MSGQ_count	Returns count of number of messages in a local message queue

MSGQ APIs (Contd..)

MSGQ_transportOpen	Initializes transport associated with the specified processor
MSGQ_getSrcQueue	Gets source message queue of a message to be used for replying to the message
MSGQ_debug	Prints current status of MSGQ sub-component
MSGQ_locateAsync	Asynchronously locates the message queue
MSGQ_transportClose	Closes the transport associated with the specified processor
MSGQ_instrument	Gets instrumentation information related to specified message queue

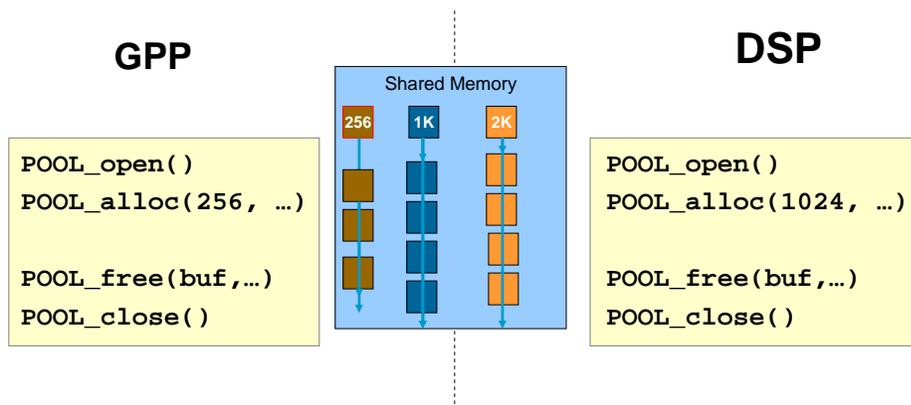
POOL

POOL: Features

- ◆ Allows configuration of shared memory regions as buffer pools
- ◆ These buffers are [used by other modules](#) from DSPLink for providing inter-processor communication functionality.
- ◆ The specific services provided by this module are:
 - [Configure](#) shared memory region through open & close calls.
 - [Allocate](#) and [free](#) buffers from the shared memory region.
 - [Translate address](#) of a buffer allocated to different address spaces (e.g. GPP to DSP)
 - [Synchronize contents of memory](#) as seen by the different CPU cores.
- ◆ Provides a [uniform view](#) of different memory pool implementations, which may be specific to the hardware architecture or OS on which DSPLink is ported.
- ◆ This component is based on the [POOL interface](#) in DSP/BIOS™.
- ◆ APIs for SMA POOL in DSPLink are callable from TSK/SWI context on DSP-side. They must not be called from ISR context.

POOL: Overview

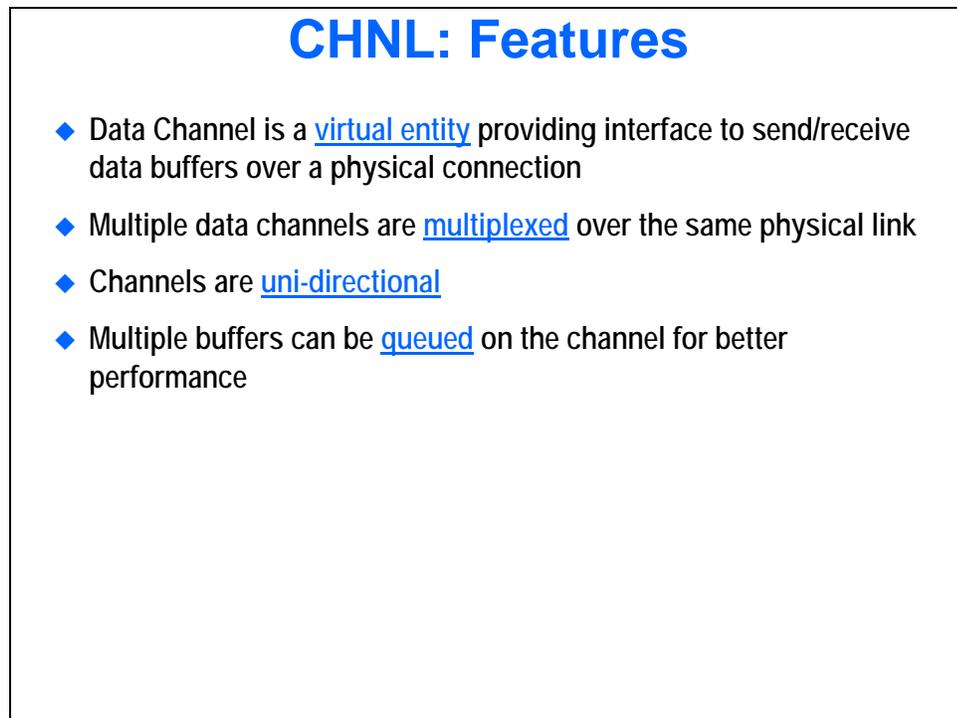
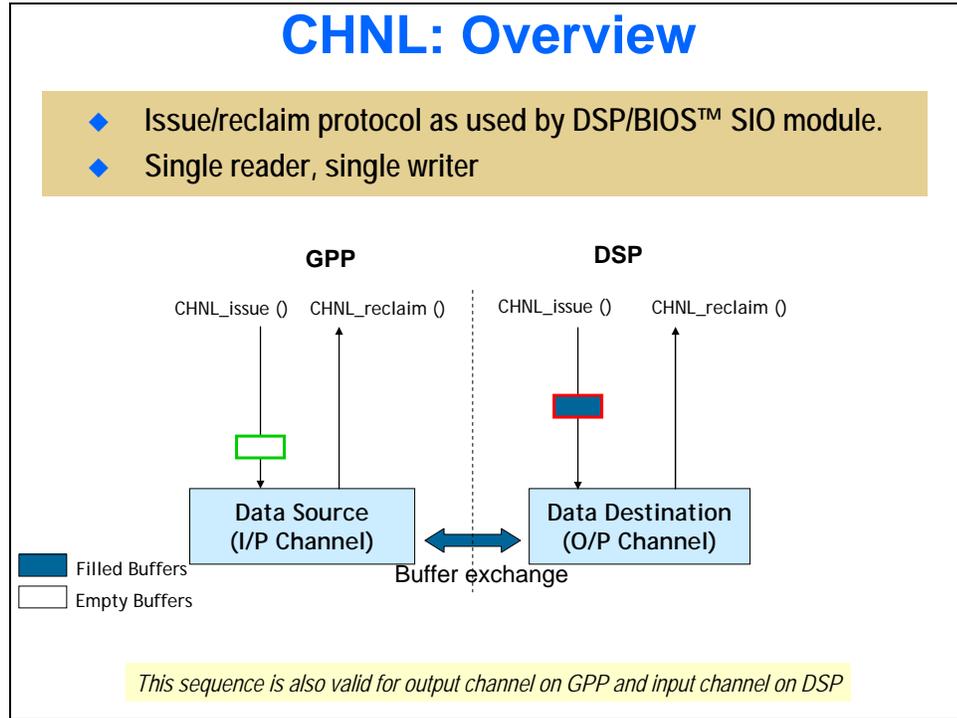
- ◆ Configures and manages shared memory regions across processors
- ◆ Supports multiple clients on GPP and DSP



POOL APIs

<code>POOL_open</code>	Opens a specified pool referenced by the pool ID
<code>POOL_close</code>	Closes a specific pool
<code>POOL_alloc</code>	Allocates a buffer of specified size from a pool
<code>POOL_free</code>	Frees a buffer into the specified pool
<code>POOL_translateAddr</code>	Translates addresses between two address spaces for a buffer that was allocated from the pool
<code>POOL_writeback</code>	Writes the content of GPP buffer into DSP buffer (with offset in sync)
<code>POOL_invalidate</code>	Invalidates the contents of the buffer

CHNL



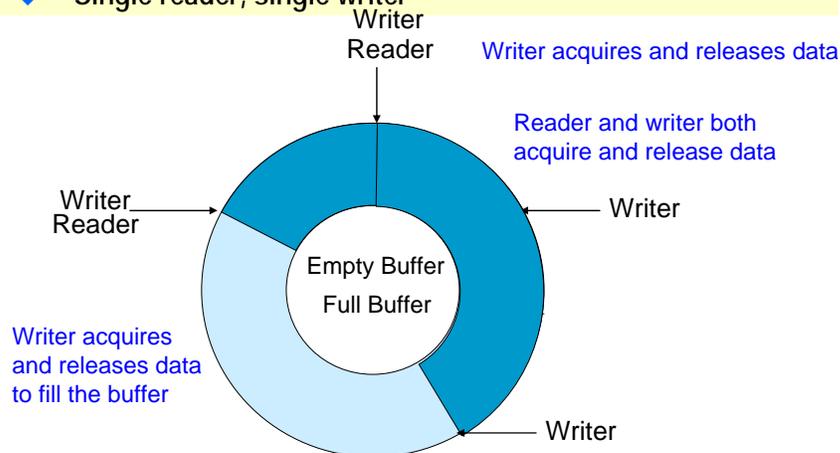
CHNL APIs

CHNL_create	Create resources used for transferring data between GPP and DSP
CHNL_delete	Release channel resources used for transferring data between GPP and DSP
CHNL_allocateBuffer	Allocates an array of buffers of specified size
CHNL_freeBuffer	Frees buffer that were previously allocated
CHNL_issue	Issues an input or output request on a specified channel
CHNL_reclaim	Gets the buffer back that has been issued to this channel
CHNL_idle	Resets the input stream channel. Waits/idles the output stream channel for as long as it takes to transfer currently queued buffers
CHNL_flush	Discards all requested buffers pending for transfer in both input and output channels
CHNL_control	Provides a hook to perform device dependent control operations
CHNL_debug	Prints current status of PMGR_CHNL sub-component

(Optional) RINGIO

RINGIO: Overview

- ◆ Circular Ring Buffer based data streaming, optimized for audio/video processing
- ◆ Single reader, single writer



RingIO: Features

Feature	Benefit
Provides true circular buffer view to the application	Internally handles wraparound
Data and attributes/messages associated with data can be sent	In-band attributes are supported
Reader and writer can work on different data sizes	Use case: Writer can be unaware of reader buffer size needs
Reader and writer can operate completely independent of each other	Synchronization only needed when acquire fails due to insufficient available size
Capability to minimize interrupts by choosing specific notification type	Notification configurable based on threshold and type (ONCE/ALWAYS)
Ability to cancel unused (acquired but unreleased) data	Use case: Simultaneously work on two frames: previous and new
Ability to flush out released data	Use case: stop/fast forward
On DSP-side, APIs can be called from TSK or SWI context. (Though, they must not be called from ISR context.)	Flexibility in DSP-side choice of thread type

RINGIO APIs

RingIO_getAcquiredOffset	Returns the current acquire offset for the client
RingIO_getAcquiredSize	Returns size of buffer current acquired by the client
RingIO_getWatermark	Returns the current watermark level specified by client
RingIO_create	Creates a RingIO instance in shared memory
RingIO_delete	Deletes a RingIO instance in shared memory
RingIO_open	Opens a RingIO instance handle
RingIO_close	Closes an already open RingIO reader/writer
RingIO_acquire	Acquires a data buffer from RingIO for reading or writing
RingIO_release	Releases a previously acquired buffer
RingIO_cancel	Cancel any data buffer acquired by reader or writer

RINGIO APIs contd..

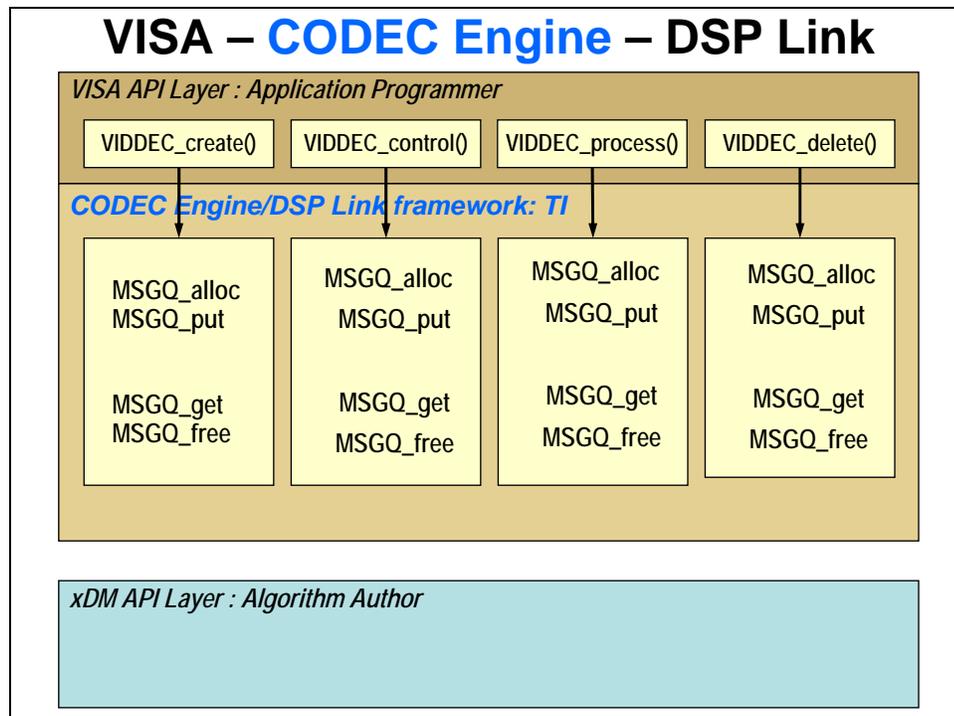
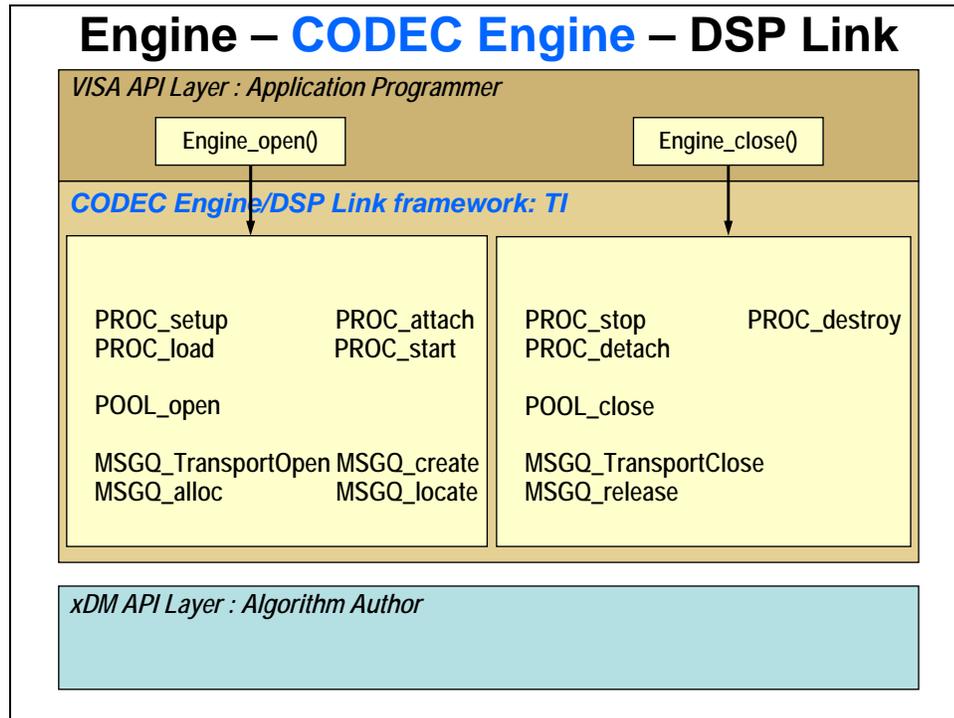
RingIO_getAttribute	Gets a fixed size attribute from the attribute buffer
RingIO_setAttribute	Sets a fixed size attribute at the offset provided in acquired data buffer
RingIO_getvAttribute	Gets an attribute with a variable sized payload from the attribute buffer
RingIO_setvAttribute	Sets an attribute with a variable sized payload at the offset provided in acquired data buffer
RingIO_flush	Flush the data from the RingIO
RingIO_setNotifier	Sets Notification parameters for the RingIO client
RingIO_sendNotify	Sends a notification to the other client with an associated message value
RingIO_getValidSize	Returns valid size in the RingIO
RingIO_getValidAttrSize	Returns the valid attributes size in the RingIO
RingIO_getEmptySize	Returns the current empty data buffer size
RingIO_getEmptyAttrSize	Returns the current empty attribute buffer size

What's Next

What's Next

- ◆ Support for [Grid topology](#) (any to any connection)
 - ◆ Currently supports only star topology (one master, many slaves)
- ◆ Portable architecture to support completely [different types of physical links](#) like EMAC/USB
 - ◆ Currently, DSPLink is difficult to port to non-shared memory architectures except in PCI & VLYNQ .
- ◆ Available on main-line Linux kernel tree for Davinci/OMAP
- ◆ [Dynamic linking/loading](#); Dynamic memory-mapping
- ◆ Add [Power management](#) support

How DSPLink API's are used by Codec Engine



For More Information

For More Information

- ◆ *DSP Link User's Guide*
(Part of DSPLink installation under docs folder)
- ◆ *DSP Link Programmer's Guide*
(Part of DSPLink installation under docs folder)
- ◆ *DSPLink Wiki*
<http://wiki.davincidsp.com/index.php/Category:BIOSLink>

Appendix A: Questions & Answers

Table of Contents

Note, since not all lab exercises ask you written questions, there will be gaps between labs in this appendix.

Appendix A: Questions & Answers	1-1
<i>Lab 5</i>	<i>1-2</i>
<i>Lab 6</i>	<i>1-3</i>
<i>Lab 7</i>	<i>1-4</i>
<i>Lab 8</i>	<i>1-5</i>
<i>Lab 9</i>	<i>1-6</i>
<i>Lab 10</i>	<i>1-7</i>
<i>Lab 11</i>	<i>1-7</i>
<i>Lab 12</i>	<i>1-8</i>

Lab 5

Lab Questions 5.1

1. Which file in the lab05a_hello_world/app directory specifies that one or more executable files are to be built?
`package.bld`
2. Which function call within this file actually scripts the building of a new executable file?
`Pkg.addExecutable(...)`
3. Examination of the config.bld file (at the top level of the workshop directory) shows that we have configured a third target, named C64P. This target is configured to build for the C64x+ DSP core that is available on the TMS320C6446. How could you modify the available files to build a Hello World executable that would run on the C64x+ core?
in package.bld change:

```
var targs = [MVArm9, Linux86]  
to:  
var targs = [MVArm9, Linux86, C64P]
```
4. (Advanced) Explain in your own words how the C source files used to build this application are specified in the package.bld build script of lab05b_extended_features.
The `java.io.File('.')` `list()` I/O method is used to get a listing of all files in the current directory. Then the list of files is iterated through and the `String().match()` method is used to filter out those files which end in ".c" using the regular expression `/.*\c$/`

Lab Questions 5.2

5. When compiling code for the C64x+ DSP, what compiler options are used by in this lab exercise.

Hint: look at options specified in these files:

`config.bld`: `--no_compress --mem_model: data_far --disable:splloop "+ remarks"`
(xdc parameter)

`ti.targets.C64P`: `-eo.o64P -ea.s64P -mv64p`
(specified in config.bld)

`ti.platforms.evmDM6446`: `none`
(specified in config.bld)

`package.bld`: `none`
(implicit xdc file)

Lab 6

Lab Questions 6.1

1. Which ioctl command is called in the `init_mixer()` function of `audio_input_output.c` in order to specify whether the microphone or line input will provide the audio source for the driver?

`ioctl(mixerFd, SOUND_MIXER_WRITE_RECSRC, &recmask)`

2. Which ioctl commands set the number of channels and sample rate for the OSS driver (two different ioctl commands are needed), and in which function of `audio_input_output.c` are they found?

`ioctl(fd, SNDCTL_DSP_CHANNELS, &numchannels)`

`ioctl(fd, SNDCTL_DSP_SPEED, &samplerate)`

and they're found in `init_sound_device()`

3. For the `while()` loop in the `audio_thread_fxn` of either the audio recorder or audio playback application: There is an `(f)read` and an `(f)write` function in this loop. For the file read or write function, a FILE pointer is the first parameter passed. For the driver read or write function, a file descriptor (int) is the first parameter passed. What is the purpose of the FILE pointer and file descriptor, and where do these come from? (In other words, what function is used to generate valid file descriptors and FILE pointers from which read and write operations can be made?)

The purpose of a FILE pointer or a file descriptor is a reference to a given file or device which is used to read from or write to the given resource. It is set during the `(f)open` call

Lab Questions 6.2

4. (Advanced) The `audio_input_setup` and `audio_output_setup` functions take as their first parameter not an integer file descriptor, but rather a pointer to an integer. The name of the variable in these function prototypes is `fdByRef`, indicating that it is a variable passed by reference. Why is a pointer to integer used here when file descriptors are integers?

Because a pointer to the file descriptor is passed, i.e. the file descriptor is passed by reference, its value can be modified by the function. Variables that are not passed by reference are passed by copy and any changes made by a function are made to the copy and not the original.

5. (Advanced) When the `audio_input_setup` function is called from the `audio_thread_fxn` of `lab6a_audio_recorder`, the `inputFd` variable is passed preceded by an ampersand. Why? (See question 4, which is related.)

Because the file descriptor is passed by reference, its address must be passed instead of its value. The ampersand operator indicates the variable's address.

6. What changes were made to the package.bld XDC build script between `lab04_hello_world` and `lab06a_audio_recorder` in order to generate two executables: one that is a debug version and one that is a release (optimized) version?

`var profiles = ["release"];` → `var profiles = ["release", "debug"];`

Lab 7

Lab 7.1

1. How would you modify the lab07a_osd_setup application to make the banner you created semi-transparent instead of solid?
in the video_osd_place function, change the value of trans to 0x44
2. How would you modify the lab07a_osd_setup application to place your banner at the top of the screen instead of the bottom?
in the video_osd_place function, change the value of y_offset to 0
3. Why is it necessary to run the bmpToRgb16.x86U utility on the bitmapped image that you created with gimp before it could be displayed in the DM6446 on-screen-display window?
The Gimp program uses 24-bit color (8 red, 8 green, 8 blue) per pixel bitmaps. The OSD window is 16-bit color (5 red, 6 green, 5 blue) per pixel.
4. Which ioctl command is called in setup_video_input function of video_input.c in order to specify NTSC (as opposed to PAL) as the video input standard?
ioctl(captureFd, VIDIOC_S_STD, &std)
5. Which ioctl command is used to set the pixel width and height for the V4L2 input driver in setup_video_input?
ioctl(captureFd, VIDIOC_S_FMT, &fmt)

Lab 7.2

6. video_input_setup uses the VIDIOC_REQBUFS ioctl to request the number of video input buffers desired be allocated by the driver. The driver then uses VIDIOC_QUERYBUFS in order to determine the length and offset of each buffer. Which function call then uses this length and offset? What is the purpose of this function call?
The mmap function uses this information to map the buffers into the application's memory space so that the application can access them directly.
7. What is the underlying (Linux) function utilized by wait_for_frame in video_input.c to block the application's execution until the next frame of video data is available on the V4L2 driver?
select()
8. Which ioctl command is used by the flip_display_buffers function to exchange a working video output buffer for the displayed video output buffer?
ioctl(displayFd, FBIO_PAN_DISPLAY, &vinfo)
9. In addition to closing the video output device, what other cleanup is handled by the video_output_cleanup function?
munmap function is used to unmap the driver buffers from the application's memory space.

Lab 8

Lab 8

1. In lab08b_audio_video_rtime, main.c, what priority is the audio thread set to? (You may use an expression here.)
`sched_get_priority_max(SCHED_RR)`
2. When running the debug version of the application (./app_debug.x470MV), what does the debug output indicate the audio thread priority is set to? (Numerical value)
99
3. What priority is the video thread set to? (You may use an expression here.)
`sched_get_priority_max(SCHED_OTHER)`
4. When running the debug version of the application (./app_debug.x470MV), what does the debug output indicate the video thread priority is set to? (Numerical value)
0



Lab 9

Lab 9.1

1. Which VISA api function call is made in `video_encoder_setup`?
`VIDENC_create()`
2. Why is the `encoderHandleByRef` parameter that is passed to the `video_encoder_setup` function a pointer to a `VIDENC_Handle` instead of just a `VIDENC_Handle`?
So that the handle's value can be modified in order to return the handle to the newly created video encoder.
3. In the `inBufDesc` buffer descriptor used in the `encode_video` function, explain why the `.bufs` parameter is a pointer to a pointer to `XDAS_Int8` (i.e. a pointer to a pointer to char).
The pointer to char value represents the address of a buffer. This is a pointer to pointer to char because it is an array of pointer to char, i.e. an array of buffer addresses.
4. Both the `audio_thread_fxn` and `video_thread_fxn` open the Codec Engine, using the Engine name passed from `main.c` to ensure that they open the same engine. Why is this done? (as opposed to opening the Codec Engine once in `main.c` and passing the handle to the audio and video threads)
Each thread must obtain its own engine handle using the `Engine_open()` function call to ensure conflicts do not arise between multiple threads attempting to access the Engine concurrently.

Lab 9.1

5. (Advanced) Show how you would modify the `inBufDesc` buffer descriptor building code of the `encode_audio` function to handle two input buffers instead of one as it is set up now (say for instance, a left and right channel). Use `char *inputBufferL` and `char *inputBufferR` as your two input buffers.

```
inBufDesc.numbufs = 2;
inBufDesc.bufSizes = inBufSizeArray;
inBufDesc.bufs = inBufArray;
inBufSizeArray[0] = inBufSizeArray[1] = inputSize;
inBufArray[0] = inputBufferL;
inBufArray[1] = inputBufferR;
```
6. (Advanced) Describe in your own words how the makefile incorporates the code contained in the published engine into the final application build.
The published engine is contained in an object file named `engine_debug_x470MV.o` or `engine_release_x470MV.o` depending on whether the release or debug version is required. The makefile simply links this object file in with the application object files using the `gcc` tool.

Lab 10

Lab 10 Questions

1. In the engine.cfg file, all codecs are specified with a local parameter of true. What does this imply regarding the need for a DSP server?
That no DSP server is needed.
2. What differences are there between the runxdc.sh script used to build the application in lab10b_all_rtsc and the runxdc.sh script used to build the application in lab08b_audio_video_rtime?
The XDCPATH environment variable has been modified to add the repository paths of those packages referenced in engine.cfg
3. What differences are there between the package.bld script used to build the application in lab10b_all_rtsc and the package.bld script used to build the application in lab08b_audio_video_rtime?
The addExecutable function call has been modified to specify the engine.cfg configuration file in the parameters.
4. Not counting the VISA commands that need to be inserted into the application source files, what steps would be necessary to migrate the solution in lab08b_audio_video_rtime to the solution in lab10_all_rtsc?
The above changes to runxdc.sh and package.bld in addition to creating an engine.cfg configuration file to configure the local engine.

Lab 11

Lab 11 Questions

1. How could engine.cfg be modified to use a DSP server image that is placed in a subdirectory of the directory into which the application executable is installed? (Say if the app is run from /opt/workshop, and the server is stored in /opt/workshop/servers.) If you wanted to use an absolute path, would this be an absolute path on the Host computer's linux filesystem, or an absolute path on the DaVinci board's linux filesystem?
demoEngine.server = "./server_debug.x64P";
Changed to:
demoEngine.server = "./servers/server_debug.x64P";
or
demoEngine.server = "/opt/workshop/servers/server_debug.x64P";
2. What would be the problem with the audio thread and the video thread using two different engines that use two different servers?
When the Engine_open() call is made to an engine that uses a remote server, the server image is loaded onto the DSP and the DSP is taken out of reset. Since servers in the current Codec Engine Framework implementation are complete images, only one can run on the DSP at a time.

Lab 12

Lab 12 – Questions (1)

1. Which Operating System Abstraction Layer (OSAL) is specified in the server configuration file `server.cfg`? Which OSAL is specified in the engine configuration file? Why are these different?
`engine.cfg` uses `osalGlobal.DSPLINK_LINUX` because it runs on Linux
`server.cfg` uses `osalGlobal.DSPLINK_BIOS` because it runs on DSP/BIOS
2. What attributes are specified for the video and audio encoders and decoders when they are added into the server in the server configuration file? Which attributes are specified for the same codecs when added into the engine in the engine configuration file? Explain for each non-matching attribute why it makes sense for a codec added to an engine or server, but not vice-versa.
`engine: local, groupID`
(codecs on server are always local, groupID set by priority)
`server: stacksize, memID, priority`
(codecs on server run within their own threads, so these thread attributes are needed. Codecs on engine run within the thread context that the VISA call is made from)

Lab 12 – Questions (2)

3. All EDMA resources are shared between the Arm and DSP cores and must be allocated between them. For the `lab12_build_server` solution, how many QDMA channels are allocated for use by the DSP server based on the `server.cfg` file? How many param (parameter ram tables) are available for these QDMA channels to use? How many transfer complete codes have been allocated for use by the server?
48 PARAMS, 8 QDMA channels and 8 tccs
4. (Advanced) We also see that the DMAN3 module (covered in more detail during the next chapter) has been configured in the `server.cfg` file. We can see from the configuration that when an algorithm requests memory of type DARAM0, this memory will come from the "L1DSRAM" memory segment. Where is this memory segment defined?
The L1DSRAM memory segment is allocated in the `server.tcf` file, the BIOS configuration file.

Acronyms

API	Application Process Interface – <i>defined protocol to interact with software components</i>
APL	Application Layer – <i>uppermost “master” layer in a DaVinci software system</i>
ASP	Approved Software Provider – <i>third party recognized by TI to provide DaVinci software support</i>
AV	Audio/Video – <i>common multi-data grouping used in most video systems (sound and picture)</i>
BIOS	<i>or “DSP/BIOS”- RTOS for TI DSPs</i>
BSL	Board Support Library – <i>extension of CSL for a given target board</i>
BSP	Board Support Package – <i>see LSP</i>
CCS	Code Composer Studio – <i>IDE for developing DSP software</i>
CE	Codec Engine – <i>IDE for developing DSP software</i>
CGT	Code Generation Tools –
CSL	Chip Support Library – <i>interface to peripherals via predefined structures and API</i>
CXD	Create / Execute / Delete - <i>three phases of the life cycle of a dynamic application</i>
DIO	Device IO – <i>interface between an IOM and SIO</i>
DMA	Direct Memory Access – <i>sub-processor that manages data copy from one place in memory to another</i>
DMAN	Direct Memory Access Manager – <i>TI authored framework to abstractly manage DMA channels</i>
DSK	DSP Starter Kit – <i>low-cost evaluation tool for developing DSP solutions</i>
DSKT2	DSP Socket, rev 2 – <i>TI authored framework to manage DSP memory allocations</i>
DSP	Digital Signal Processor - <i>processor with enhanced numerical abilities</i>
DVEVM	Digital Video Evaluation Module – <i>hardware platform allowing system test and application development</i>
DVSDK	Digital Video Software Development Kit – <i>adds DSP side development ability to DVEVM</i>
EMIF	External Memory Interface – <i>TI DSP sub-component that manages flow of data on/off chip</i>
EPSI	Easy Peripheral Software Intevace – <i>DaVinci API for communication with IOL (drivers)</i>
EVM	Evaluation Module – <i>hardware test and debug platform</i>
GHS	Green Hills Software – <i>makers of Multi IDE and Integrity OS; TI ASP</i>
GPP	General Purpose Processor – <i>standard micro processor, as opposed to a special purpose processor</i>
HLL	High Level Language – <i>eg: C, C++, Java</i>
HW	Hardware – <i>physical components</i>
HWAL	Hardware Adaptation Layer – <i>TI software adapting DaVinci software framework to a given HW system</i>
ICE	In Circuit Emulator – <i>hardware debug tool</i>
IDE	Integrated Development Environment –
IOM	Input Output Mini-driver – <i>device driver standard in DSP/BIOS</i>
IPC	Interprocessor Communication – <i>interface between processors, eg: DSP/BIOS Link</i>
IPO	Input / Process / Output – <i>flow of data in a DSP system</i>
JTAG	Joint Test Action Group – <i>standard for interface to debug host</i>
LSP	Linux Support Package – <i>device/board specific kernel and driver software</i>
OSAL	Operating System Adaptation Layer – <i>TI software porting Codec Engine to a given GP OS</i>
MAC	Multiply Accumulate – <i>core activity in many DSP algorithms</i>
McBSP	Multi Channel Buffered Serial Port – <i>serial port that interfaces to numerous data formats</i>
MIPS	Millions of Instructions Per Second – <i>basic measure of DSP performance</i>
MP3	MPEG 4 level 3 – <i>video standard audio encoding methodology</i>
MPEG	Motion Picture Experts Group – <i>video compression standard</i>
MPU	Micro Processor Unit – <i>processor core</i>
MV	MonteVista – <i>Leading support provider for embedded Linux</i>
MVL	MonteVista Linux – <i>Version of Linux supported by TI for their DaVinci DM644x devices</i>
OEM	Original Equipment Manufacturer – <i>maker of a given hardware solution</i>
OS	Operating System – <i>software that provides sophisticated services to software authors</i>
RAM	Random Access Memory – <i>memory that can be read from or written to</i>

RISC	Reduced Instruction Set Computer – <i>processor with small, fast instruction set</i>
RPC	Remote Procedure Call – <i>calls that may invoke local or trans-processor functions</i>
RTA	Real-Time Analysis – <i>ability to observe activity on a processor without halting the device</i>
RTSC	Real-Time System Component – <i>new TI methodology for packaging software components</i>
RTOS	Real-Time Operating System – <i>software kernel that is tuned to deterministic temporal behavior</i>
SARAM	Single Access RAM – <i>RAM internal to the DSP cell that allows a transaction each cycle</i>
SDRAM	Synchronous DRAM – <i>clock driven (fast) Dynamic Random Access Memory</i>
SPL	Signal Processing Layer – <i>DaVinci software partition holding VISA Codecs and other DSP elements</i>
SW	Software – <i>code run on hardware</i>
TI	Texas Instruments – <i>semiconductor manufacturer specializing in DSP and analog</i>
TTO	Texas Instruments Training Organization – <i>group within TI chartered to provide DSP training</i>
UART	Universal Asynchronous Receiver Transmitter – <i>serial port with clock embedded in data</i>
USB	Universal Serial Bus – <i>modern interface between PCs, handhelds, and numerous peripherals</i>
VISA	Video, Imaging, Speech, Audio – <i>SPL API on DaVinci</i>
xDAIS	eXpress DSP Algorithm Interface Standard – <i>rules that define how to author DSP components</i>
XDC	
xDM	xDAIS for Digital Media – <i>xDAIS extended to directly support VISA</i>

Terms

3rd Party Network	TI approved list of vendors supporting DaVinci based system development
Codec	This term can be used in three ways. (1) Software engineers use this term to describe Compression/Decompression algorithms commonly used in video, imaging, speech, and audio applications. Examples include: MPEG2, H.264, JPEG, G.711, MP3, AAC. (2) In the world of DaVinci software, the term "codec" is often used to imply any signal processing algorithm. It is common to hear algorithms that adhere to TI's xDM or xDAIS software interface standard referred to as "codecs". (3) From a hardware engineers point of view, the term codec refers to a single integrated circuit device which contains both an analogue-to-digital converter (ADC) as well as a digital-to-analogue converter (DAC). Note, most A/V systems contain both types of codecs, which can be confusing when during conversations involving both software and hardware engineers.
Codec Engine	Software infrastructure developed by TI for use on DaVinci processors. Allows application code to leverage DSP algorithms without need to know the low level details
DevRocket	MontaVista's Linux IDE
DSL Link	Physical transport layer for inter-processor communication
Engine	CE framework layer for managing local and remote function calls
Multi	GreenHills IDE
Server	Remote thread that services create/delete RPC's from the engine
Skeleton	Function that unmarshals RPC arguments sent by stub
Stub	Function that marshalls RPC arguments for transport over DSP Link

Building Programs with the XDC Build Tool

Introduction

DaVinci software is built and packaged using the Express DSP Component (XDC) command line tool. This tool can create (and consume) Real Time Software Component (RTSC) packages. Similar, in many ways, to other make/build tools, XDC can build executable files from source files and libraries. Unlike other make/build tools, it can automatically perform dependency and version checking.

This chapter introduces the XDC tool and describes its basic features and requirements.

Learning Objectives

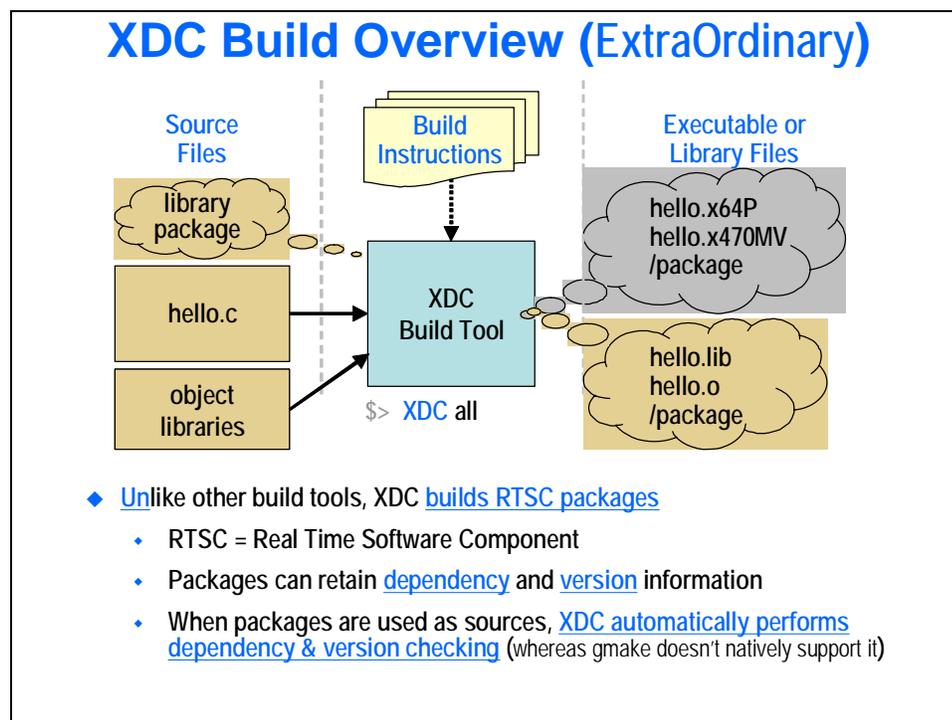
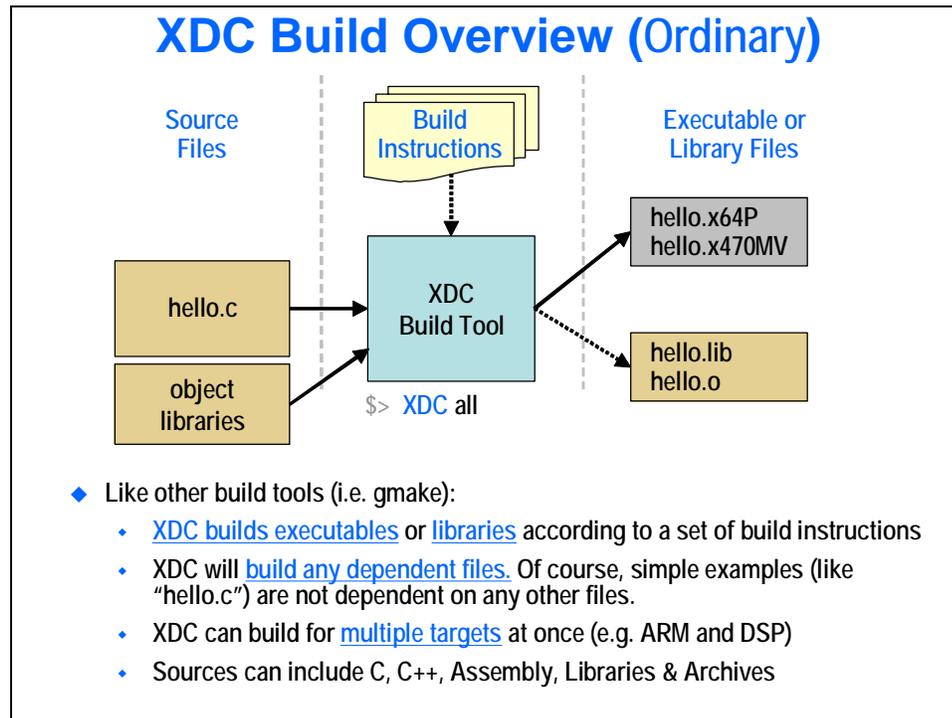
Objectives

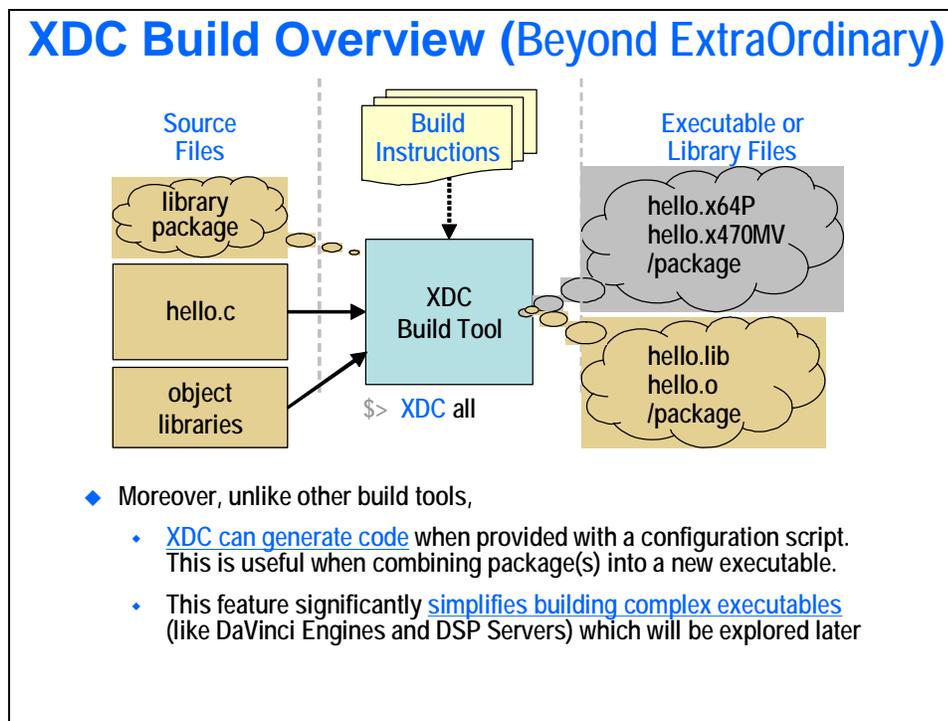
- ◆ **List the features of XDC and describe its benefits over other make tools**
- ◆ **Describe the three basic files required by XDC (package.xdc, package.bld, config.bld)**
- ◆ **Chart out the flow XDC follows when building a package and executables**
- ◆ **Build an application package using XDC**

Chapter Topics

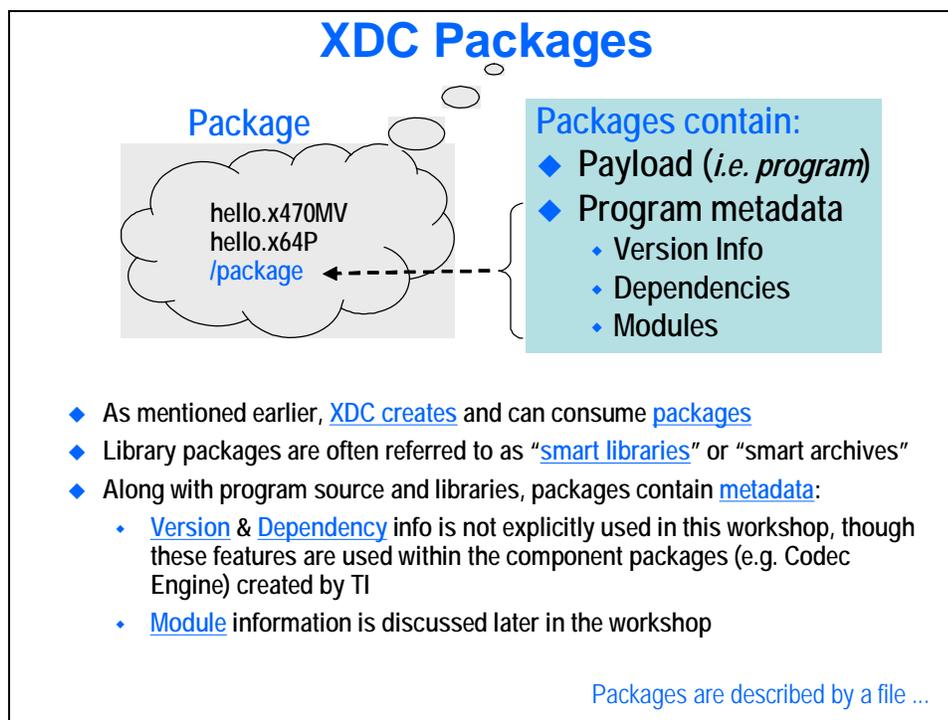
Building Programs with the XDC Build Tool.....	99-1
<i>XDC Overview</i>	99-3
<i>Packages (and Repositories)</i>	99-4
package.xdc	99-5
Repositories	99-6
<i>Invoking XDC</i>	99-8
<i>XDC Build Flow</i>	99-10
config.bld.....	99-10
package.bld.....	99-12
package.mak	99-15
<i>XDC Summary</i>	99-16
<i>Lab 99 – Using XDC</i>	99-17
Lab Objectives.....	99-17
Lab99a_hello_world.....	99-17
Lab99b_extended_features	99-22
Questions	99-27
Challenges	99-28

XDC Overview





Packages (and Repositories)



package.xdc

package.xdc

Parts of package.xdc:

- ◆ Package Name
- ◆ Version Info
- ◆ Dependencies
- ◆ Modules

```

/*
 * ===== application =====
 *   vendor:   ti.tto
 *   app name: app
 *   pkg name: ti.tto.app
 */
package ti.tto.app {
}

```

- ◆ “package.xdc” describes a package:
Name, Dependencies, Version, lists its Modules
- ◆ Package name must be [unique](#)
- ◆ It’s often common to name the package along these lines:
vendor . project name . name of what you’re building (but don’t use spaces)
- ◆ Packages are usually [delivered as an archive](#): ti.tto.app.tar

[More package.xdc examples...](#)

More “package.xdc” Examples

Parts of package.xdc:

- ◆ Package Name
- ◆ Version Info
- ◆ Dependencies
- ◆ Modules

```

/*
 * ===== codec =====
 */
requires ti.sdo.ce.audio [ver]

package codecs.auddec_copy [version] {
  module AUDDDEC_COPY;
}

```

Parts of package.xdc:

- ◆ Package Name
- ◆ Version Info
- ◆ Dependencies
- ◆ Modules

```

package app { }

```

[Where are packages located?](#)

Repositories

Repositories

- Directories containing packages are called [repositories](#)
- For example, we could have two packages in the workshop/lab repository

```

graph TD
    workshop --> lab
    lab --> app1[ti.tto.app1.tar]
    lab --> app2[ti.tto.app2.tar]
    
```

- In our workshop, we chose to use the same package name over and over: **app**
 - Thus, we effectively chose to have a separate repository for each lab.
 - While not necessarily common, this seemed the easiest way to handle a series of lab exercises and solutions.

Three repo/pkg examples ...

Package and Repository (Example 1)

Repository: /home/user/workshop/lab
 Package name: app

```

graph TD
    workshop --> lab
    lab --> app
    app --> package_dir[package]
    package_dir --> package_xdc[package.xdc]
    package_dir --> package_bld[package.bld]
    package_dir --> hello_c[hello.c]
    package_dir --> hello_x470MV[hello.x470MV]
    package_dir --> hello_x64P[hello.x64P]
    
```

Misc. package files (auto-generated by XDC)

Package description files (you'll supply these)

- XDC creates a number of files and directories as part of a compiled package. While you must keep them, you will not need to work with them.
- Note, package.bld will be discussed later in this chapter

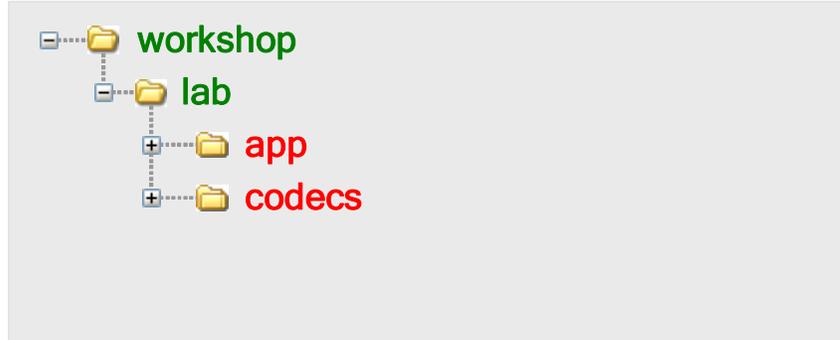
Package and Repository (Example 2)

Example 2

Repository: `/home/user/workshop/lab`

Package names: `app`, `codecs`

Delivered as: `app.tar`, `codecs.tar`



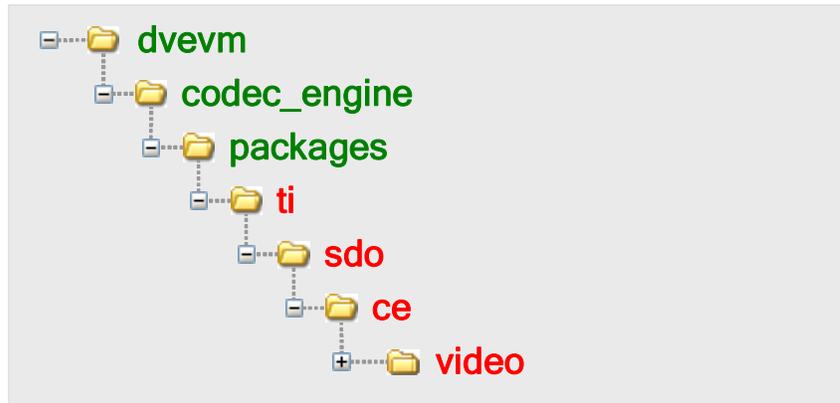
Package and Repository (Example 3)

Example 3

Repository: `/home/user/dvevm/codec_engine/packages`

Package name: `ti.sdo.ce.video`

Deliver: `ti.sdo.ce.video.tar.gz`



Invoking XDC

XDC Invocation (Explicit Arguments)

XDC `<target files>` `<XDCPATH >` `<XDCBUILDCFG >`

- ◆ Builds one or more files, as specified
- ◆ Files can also be specified via implicit commands (next slide)
- ◆ XDCPATH specifies the directories to be searched during build
- ◆ Specifies "config.bld" file which contains your platform's build instructions

Typing in all the search directories each time is tedious, so
Simplify XDC with scripts...

Invoking XDC with Shell Scripts (.SH)

XDC `<target files>` `<XDCPATH >` `<XDCBUILDCFG >`

```
runxdc.sh
#!/bin/sh
# Import installation paths
../setpaths.sh
# Define search paths
export XDCPATH="$CE_INSTALL_DIR/packages;
$DVEVM_INSTALL_DIR;$SDK_INSTALL_DIR;
$BIOS_INSTALL_DIR"
# Define options for execution
export XDCBUILDCFG="$(pwd)/../config.bld"

xdc
```

```
setpaths.sh
#!/bin/sh
## Absolute Paths must be set per system ###
# DVEVM software directory
export DVEVM_INSTALL_DIR=
"home/user/dvevm_1_10"
# Where Codec Engine package is installed
export CE_INSTALL_DIR=
"$DVEVM_INSTALL_DIR/codec_engine_1_02"
# Installation directory of the software
# development kit (may be same as above)
export SDK_INSTALL_DIR="$DVEVM_INSTALL_DIR"
# where the BIOS tools are installed
export BIOS_INSTALL_DIR=
"$DVEVM_INSTALL_DIR/bios_5_30"
...
export PATH="$XDC_INSTALL_DIR:$PATH"
```

XDC also has implicit args...

XDC Invocation (Implicit Arguments)

Explicit Arguments

```
runxdc.sh
#!/bin/sh
# Import installation paths
.../..setpaths.sh
# Define search paths
export XDCPATH="$CE_INSTALL_DIR/packages;
$DVEVM_INSTALL_DIR;$SDK_INSTALL_DIR;
$BIOS_INSTALL_DIR"
# Define options for execution
export XDCBUILDCFG="$(pwd)/../config.bld"

xdc
```

Implicit Arguments

Along with the explicit arguments XDC has three implicit arguments it looks for:

1. `package.xdc`
Names the package to be built and defines its contents
2. `package.bld`
Provides package specific build instruct's

XDC allows for various targets...

Specifying XDC Targets

Explicit Arguments

```
runxdc.sh
#!/bin/sh
# Import installation paths
.../..setpaths.sh
# Define search paths
export XDCPATH="$CE_INSTALL_DIR/packages;
$DVEVM_INSTALL_DIR;$SDK_INSTALL_DIR;
$BIOS_INSTALL_DIR"
# Define options for execution
export XDCBUILDCFG="$(pwd)/../config.bld"

xdc $@ -PR .
```

Pass all runxdc.sh args

build all pkgs recursively

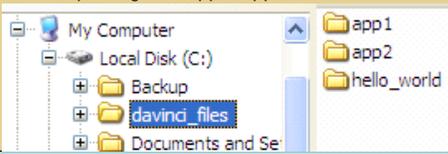
Some Target Options

- Build a single executable:
- xdc `hello_release.x64P`
 - xdc `myapp_release.MVArm9`
- Build all exe's in xdc invocation directory:
- xdc `all` -or-
 - xdc
- Clean (i.e.delete) previously built files:
- xdc `clean`

Build all packages within this heirarchy:

• xdc `-PR .`

If XDC is invoked in "davinci_files", it will build packages in app1, app2, and hello...



XDC Build Flow

config.bld

XDC Flow : config.bld

Invoke XDC	XDC <files> <XDCPATH> <XDCBUILDCFG>
Platform Def's	↳ config.bld (path specified by xdcbuildcfg)

- ◆ [config.bld](#) defines platform-wide definitions
- ◆ Its [path is defined by XDCBUILDCFG](#) environment variable
- ◆ Usually copied from TI examples and edited once per platform
- ◆ This script is run prior to all build scripts. It sets host-system-independent values for targets and platforms, then it attempts to find the host-system-specific user.bld script that sets rootDirs.

Looking at config.bld...

Let's start at the end of the file. It shows an array of build targets – each of which are defined at the top of the file.

config.bld

Parts of config.bld:

- ◆ DSP Target
- ◆ ARM Target
- ◆ Linux Host Target
- ◆ Build Targets

```

/*
 * ===== Build.targets =====
 * list of targets (ISAs & compilers)
 * you want to build for
 */
Build.targets = [
    C64P,
    MVArm9,
    Linux86
];

```

- ◆ First "platform" definition in config.bld is for the various targets which can be built when you run XDC
- ◆ You can add to this list, but probably won't need to
- ◆ Notice that each target is defined at the top of the file (shown on the next 3 slides)

config.bld

Parts of config.bld:

- ◆ DSP Target
- ◆ ARM Target
- ◆ Linux Host Target
- ◆ Build Targets

```
// ===== DSP target =====
/* Create Javascript var "remarks" - set equal to string of C6x
   C options to: enable remarks; warn if func proto not found */
var remarks = " -pden" + " -pdsW=225";

// Inherit target specified in TI provided target module package
var C64P
    = xdc.useModule("ti.targets.C64P");

// Modify elements of build target data structure
C64P.platform = "ti.platforms.evmDM6446";
C64P.ccOpts.prefix += " -k";
C64P.ccOpts.prefix += " --mem_model:data=far";
C64P.ccOpts.prefix += remarks;

// C64P codegen tool location ... C6000_CG set in setpaths.sh
C64P.rootDir = java.lang.System.getenv("C6000_CG");;
```

- ◆ TI provides a default build models for each target (i.e. [DSP](#), [ARM](#), [x86](#))
- ◆ `useModule()` is the XDC method to pull-in a RTSC package; `ti.targets.C64P` is a TI provided package
- ◆ Target's can be found at:
`/home/user/dvevm_1_10/xdctools_1_21/packages/ti/targets`
- ◆ The defaults are inherited and overridden using `config.bld` (as shown above)

config.bld

Parts of config.bld:

- ◆ DSP Target
- ◆ ARM Target
- ◆ Linux Host Target
- ◆ Build Targets

```
// ===== ARM target =====
var MVArm9 =
    xdc.useModule('gnu.targets.MVArm9');

MVArm9.ccOpts.prefix += " "

MVArm9.platform =
    "ti.platforms.evmDM6446";

/* add pthreads */
MVArm9.lnkOpts.suffix = "-lpthread " +
    MVArm9.lnkOpts.suffix;

/* MontaVista ARM9 tools path */
MVArm9.rootDir =
    java.lang.System.getenv("MONTAVISTA_DEVKIT") +
    "/arm/v5t_le/armv5tl-montavista-linuxeabi";
```


Build Package with Executable Program

package.bld

```
var Pkg = xdc.useModule('xdc.bld.PackageContents');
// Build package ... and Add an Executable to it
Pkg.addExecutable("hello_debug", MVArm9, ...);
```

Before
Running XDC



After
Running XDC



[Looking more closely at addExecutable...](#)

Pkg.addExecutable

```
var Pkg = xdc.useModule('xdc.bld.PackageContents');
/*
 * ===== Add Executable =====
 */
Pkg.addExecutable(
    "app_debug",           // Name of executable file
    MVArm9,               // Build target (from config.bld)
    MVArm9.platform,     // Platform (from config.bld)
    {
        cfgScript: null, // Will use this in later chapters
        profile: "debug", // Build profile to be used
    }
).addObjects( ["main.c"] ); /* JavaScript array of obj files;
                             if source files are passed, xdc will
                             first build them into object files */
```

- ◆ The `addExecutable` method tells XDC what (and how) to build your executable program.
- ◆ This example is a "hard-coded" version. It needs to be edited for each lab.

[Can we make package.bld more flexible?](#)

Generic package.bld

```

var basename = "myApp";

// Define array of targets to build
var targs = [MVArm9, Linux86];

// Define array of profiles to build
var profiles = ["debug","release"];

// ===== Add Executable =====
/* Cycles through the arrays of build targets and profiles
and create an executable for each combination */
for (var i = 0; i < targs.length; i++) {
  for (var j = 0; j < profiles.length; j++){
    Pkg.addExecutable (
      basename + "_" + profiles[j],
      targs[i],
      targs[i].platform,
      {
        cfgScript: null,
        profile: profiles[j]
      }
    ).addObjects ( ["main.c"]);
  }
}

```

- ◆ Use JavaScript to create a generic build routine
- ◆ Quickly change what's built by simply changing arrays
- ◆ Almost "One size fits all"

- ◆ Example on first pass: MVArm9 with debug profile

```

Pkg.addExecutable (
  "myApp_debug",
  MVArm9,
  MVArm9.platform,
  {
    cfgScript: null,
    profile: "debug"
  }
).addObjects ( ["main.c"]);

```

Generic Source Array

```

// ===== Source Array =====
// Generate an array of all source files in the directory
var sources = java.io.File('.').list();
var csources = [];

for (var i = 0; i < sources.length; i++){
  if(String(sources[i]).match(/.\.c$/))
    csources.push(sources[i]);

// ===== Add Executable =====
for (var i = 0; i < targs.length; i++) {
  for (var j = 0; j < profiles.length; j++){
    Pkg.addExecutable (
      ...
    ).addObjects ( csources );
  }
}

```

← Creates a list of all files in the folder "."
Create an empty array

← Go thru entire "sources" array,
if you find a file ending in ".c"
add it to the "csources" array

- ◆ Fancy Javascript way to gather together all the source files found in the current directory (i.e. directory where package.bld resides)
- ◆ Simulates a common practice used in makefiles
- ◆ With our "Generic package.bld" and "Generic Source Array" we can [use the same package.bld for all lab exercises](#)

package.bld

Parts of package.bld:

- ◆ Define target array
- ◆ Define build profiles
- ◆ Source array
- ◆ Create executable(s)
- ◆ Epilogue

```

/*
 * ===== Epilogue =====
 */
// Force recognition of new source file added to directory
Pkg.makeEpilogue = "include ../custom.mak\n\n";

// Handle "install" rule, if added to XDC command line
Pkg.makeEpilogue += "install: ";
...
                
```

- ◆ Epilogue is not required
- ◆ Add additional *makefile* rules to the end of the file

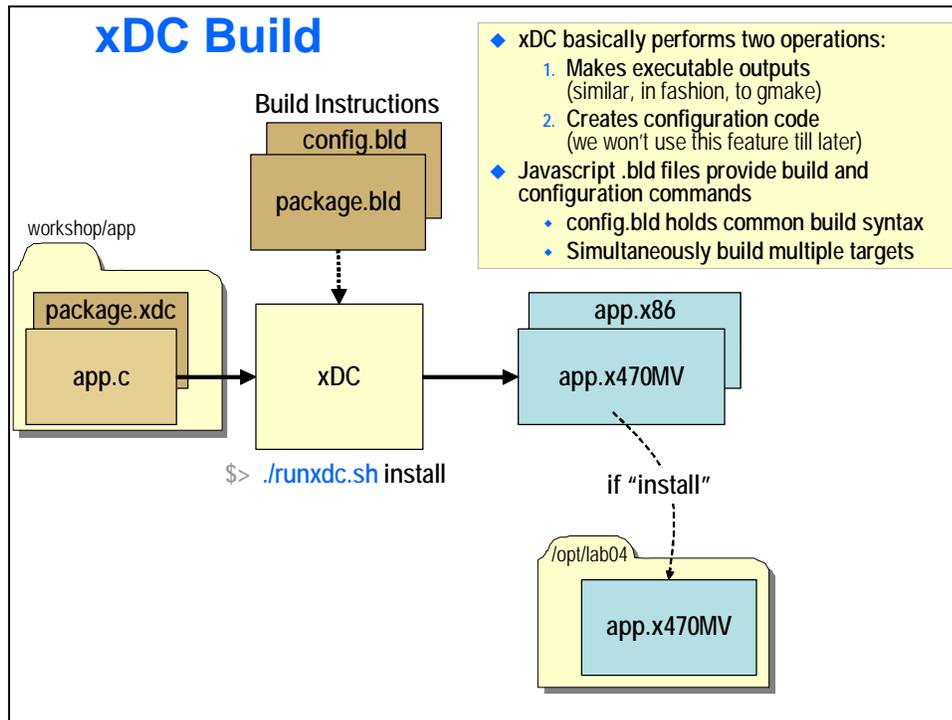
package.mak

XDC Flow

Invoke XDC	XDC <files> <XDCPATH> <XDCBUILDCFG>
Platform Def's	→ config.bld (path specified by xdcbuildcfg)
Package Def's	→ package.xdc (current directory)
Executable/Library Build definitions	→ package.bld (current directory) addExecutable ...
Generated by XDC	→ package.mak (internal XDC script to create)

- ◆ XDC generates and runs a file called package.mak
- ◆ You can open and examine package.mak, but you should not edit it (since it will be re-generated by XDC)

XDC Summary



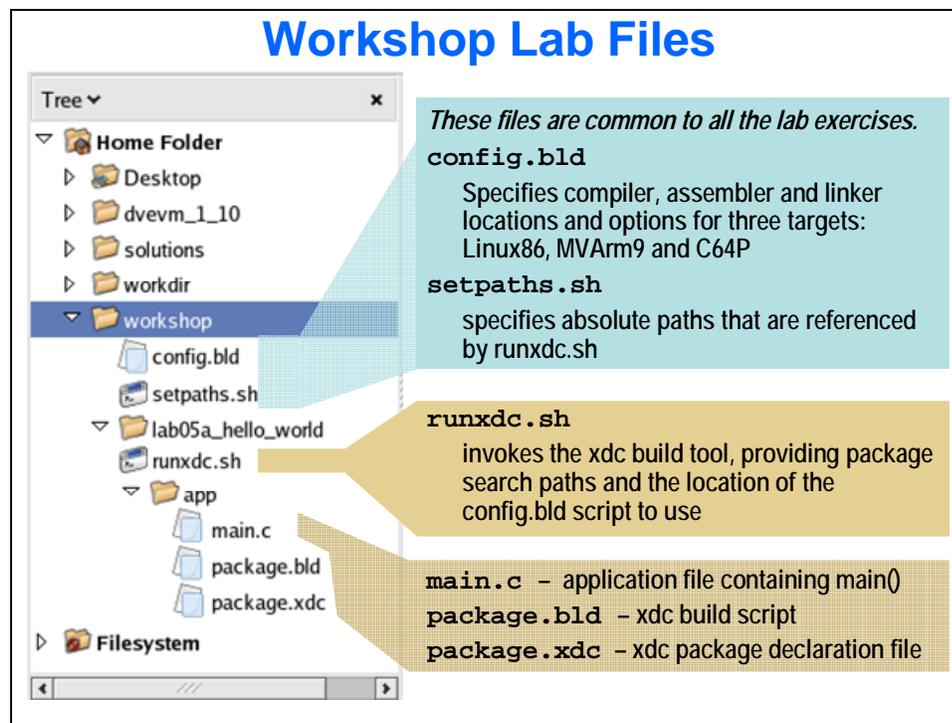
Lab 99 – Using XDC

Welcome to the compulsory “Hello World!” lab, where we will begin our exploration of the DaVinci Evaluation Module (DVEVM) and the software programming tools available to you in your development.

Lab Objectives

In this lab, you will:

- Build a simple application using the XDC build utility in conjunction with the shell scripting capability of a Linux host machine.
- Execute the “Hello world!” application on both the x86-based Linux host system and the Arm-926-based DaVinci target system using Linux terminals.



Lab99a_hello_world

1. Begin by logging into the Linux Host (i.e. desktop) Computer and opening a terminal by right clicking on the desktop and selecting “Open Terminal”

You will begin in the /home/user directory (the home directory of the user named “user”), also represented by the tilde (~) symbol.

2. **Descend into to the `/home/user/workshop` directory using the “`cd`” command. (“`cd`” is short for “change directory”).**
3. **You may use the “`ls`” command to survey the contents of this directory. (Lower case “`LS`” is short for “list”)**

The workshop directory is the working directory for the lab exercises – and contains all the starter files needed for this workshop. (Note, the `/home/user/solutions` folder contains solutions for all of the workshop labs.) There are two additional files: `config.bld` and `setpaths.sh`. The `config.bld` file provides configuration information that is used by the XDC build tool. The `setpaths.sh` file contains absolute path listings for the locations of various tools and packages that will be used to build projects throughout the workshop.

The `setpaths.sh` file contains the only absolute paths used in all of the workshop labs and solutions. The `runxdc.sh` scripts used in each lab to build the solutions reference the absolute paths set in `setpaths.sh`. As such, the workshop files can be placed in any directory within the Linux filesystem, and/or the various required packages and tools placed in any directory and still be made to build properly by changing only the paths set in `setpaths.sh`.

For the DaVinci 4-day workshop, the proper filepaths have already been configured for the setup we are using. However, when you take your labs and solutions home to work on them further, you may need to modify the `setpaths.sh` in order to build correctly on your system.

4. Examine into the `lab99a_hello_world` directory.

```
# cd ~/workshop/lab99a_hello_world
```

The `lab99a_hello_world` project has only one directory, “app.” (As our lab exercises become more complex, some projects will have multiple directories at this level.) The files which are used to build the `lab99a_hello_world` application are as follows (note that you may see additional files as the XDC tool generates files as well):

Workshop Files (common to all lab exercises)

`config.bld` – specifies compiler, assembler and linker locations and options for three targets: Linux86, MVArm9 and C64P

`setpaths.sh` – specifies absolute paths that are referenced by `runxdc.sh`

`lab99a_hello_world`

`runxdc.sh` – invokes the XDC build tool, providing package search paths and the location of the `config.bld` script to use

app (found in `lab99a_hello_world` app directory)

`main.c` – application file containing the `main()` entry point

`package.bld` – XDC build script

`package.xdc` – XDC package declaration file

5. Examine the C source file `app/main.c` which prints the string “hello world” to standard output.

```
# gedit app/main.c &
```

Note: You don’t need to type the pound sign “#”, this is shown to indicate the terminal prompt.

For help on using the `printf` command, you can access the `man(ual)` utility from within a Linux terminal by typing:

```
# man printf
```

In this case, there is a `printf` terminal utility as well as a `printf.c` library command, so to get the page you want, you will need to type:

```
# man -a printf
```

which will pull up all manual pages matching `printf`. Type “q” (no return key needed) to quit the current page and go to the next entry. The second page, `printf(3)`, is the page that you want. Note that the man page also shows the header file required to use this command.

6. Examine `app/package.xdc`.

This is a very simple file that does nothing more than to declare the name of the package used to build the application. This name must match the directory that the `package.xdc` file resides in, so in this case the package name used is “app” (It could also have been “lab99a_hello_world.app” or even “workshop.lab99a_hello_world.app”).

We will discuss packages in more detail in Chapter 10. For now we will simply state that the XDC tool requires a declared package in order to perform a build.

Examine `app/package.bld`

`package.bld` is the build script that specifies what executable(s) are to be built by the XDC tool and which files are used to build that executable. It is similar in function to a makefile script, just as the XDC tool itself is similar in function and purpose to the make utility.

7. **package.bld begins by declaring two arrays, one specifying all targets to be built, and one specifying all the profiles to be built. The script then declares a variable to store the base name of the executable you will generate.**

While not a requirement, it is a good idea to build arrays that specify all targets you are building for and all profiles to build under so that you can quickly and easily add or remove a given target or profile. One great advantage to the XDC build tool is its capability to build for a number of targets (such as x86 host computer and arm-based DaVinci processor) and a number of profiles (such as debug and release) all simultaneously.

Similarly the `basename` variable can be quickly modified. By placing these three variables at the top of the build script, most of the changes that will ever need to be made to this file are localized at the top.

8. **Next, `package.bld` iterates through the targets and profiles arrays described in step 7 in nested loops, and for every combination of target and profile specified, uses the `Pkg.addExecutable` method to create an executable. The `Executable.addObjects()` method, is used to add the “`csources`” array of C source files into the executable build.**

You can refer to /home/user/dvSDK_1_30_00_40/doc/index.html for html-browsable links to XDC documentation, i.e.:

```
# mozilla file: /home/user/dvSDK_1_30_00_40/xdctools_3_10/doc/index.html
```

And from this page, select the XDC Object Model reference.

Within the XDC Object Model reference, the `Pkg.addExecutable` method can be found under `xdc.bld.PackageContents.addExecutable()` (`Pkg` is an alias to `xdc.bld.PackageContents`), and the `Executable.addObjects()` method can be found under `xdc.bld.Executable.addObjects()`

Examine runxdc.sh

This is the shell script used to invoke the XDC tool. The shell script is used to set environment variables used by the tool, such as XDCPATH, the listing of search paths in which the tool will look for included packages, and XDCBLDCONFIG, the location of the config.bld script used to configure the tool.

9. The script begins by executing the setpaths.sh script from the workshop top level.

The workshop places all required absolute path references into environment variables set in a single file called setpaths.sh. Each runxdc.sh script for each of the labs then references these environment variables. This allows the package path references for all of the labs to be updated by changing a single script (for instance if migrating to a new version of the Codec Engine or when being installed onto a new computer where various components have been installed in different locations.) While not required, this is often a helpful practice to implement.

You should begin the runxdc.sh script for this lab by running the setpaths.sh script at the workshop top level. Recall that this command should be preceded by a single period indicating that the script will run in the current environment. Otherwise all of the environment variables set by the setpaths.sh script will be discarded when the script is exited. Using a relative path to the setpaths.sh script is preferable as it allows relocation of the workshop folder.

10. Next, the script defines the XDCBLDCONFIG and XDCPATH environment variables using the export shell command.

(For instructions on using the export command, type “man export” in the Linux terminal)

XDCBLDCONFIG and XDCPATH are two environment variables referenced by the XDC tool. For more information, refer to xdc.pdf, which is found at

```
# mozilla file: /home/user/dvSDK_1_30_00_40/xdctools_3_10/doc/index.html
```

This lab needs only to specify one XDC package path, “\$CE_INSTALL_DIR/packages” (Where CE_INSTALL_DIR is the installation directory for the Codec Engine as specified in setpaths.sh) This is the path to the ti.platforms.evmDM6446 platform package module, which is the default platform for both the C64P and MVArm9 targets as set in config.bld. (We’re not using the Codec Engine in this lab, it’s just that the platform description files referenced by config.bld are found in the same directory.)

The XDC build configuration file that the script specifies is config.bld at the top level of the workshop directory. It is best to use an absolute path here, so the script uses the \$PWD environment variable to reference the directory that the runxdc.sh script is located within and then references config.bld relative to this location.

Note: PWD is an environment variable automatically maintained by Linux which contains the absolute path to the current directory. PWD stands for “print working directory”.

11. Finally, the script adds a line to execute the XDC command to make all packages in the current directory.

The script passes the `$@` special variable as the first argument to XDC. `$@` is a shell script symbol that references the array of parameters that were passed to the build script. By passing these directly to the XDC tool, we can run the script with different build goals such as “all”, “clean” and “install” Also, though it is not necessary for this simple single-package build, let’s go ahead and create a general script that will build all packages in the current directory by using the `-P XDC` option. You can use the asterisk (wildcard) character “*” in conjunction with the `-P` option to specify all subdirectories.

For more information on supported XDC environment variables and flags for the XDC tool, refer to `xdc.pdf`, which can be referenced at

```
# mozilla file: /home/user/dvSDK_1_30_00_40/xdctools_3_10/doc/index.html
```

Build and Test the “Hello World” Application

12. Build our *Hello World* application.

Run the `runxdc . sh` script in the Host RedHat Linux terminal.

```
# cd ~/workshop/lab99a_hello_world
# ./runxdc.sh
```

13. Run the x86-compiled version of the *Hello World* application.

The method for executing this file is the same as was used to execute the `runxdc . sh` script in step 12 of this procedure.

Lab99b_extended_features

The “hello world” project that we have will build arm and x86 executables just as desired, but we have not fully taken advantage of the features the JavaScript language provides for us.

First, it would be nice to build into the script a simple means of installing the DM6446 executable to a location where the DVEVM could run it (via a shared NFS folder).

Second, it would be nice to have our build script determine all source files in the current directory and automatically include them into the executable. This way, as we add files in the following labs, they will be included without needing to modify the script.

14. Begin by copying the files from `lab99a_hello_world` into the `lab99b_extended_features` folder.

```
# cp -R lab99a_hello_world/* lab99b_extended_features
```

The `-R` option indicates *recursive* copying. Lookup the `cp` man page for more info.

Note: There is an extra file, `app/install.rule`, already included in `lab99b_extended_features`, so be sure not to overwrite it.

Adding an install rule to the XDC build

As discussed in the lecture portion of this chapter, the XDC tool parses the `config.bld`, `package.xdc` and `package.bld` files associated with the project and uses them to create the file `package.mak`.

`package.mak` is a standard gnu makefile that is utilized to perform the actual build of any files specified in `package.bld`. Additionally, the XDC tool provides a mechanism for appending to the generated `package.mak` file using the **makeEpilogue** element of the Pkg object, for instance:

```
Pkg.makeEpilogue = "This exact string will be added to the end of package.mak";
```

15. Cut and paste the code provided in the `lab99b_extended_features/app/install.rule` file to the end of `package.bld`.

Creating a string with the above text and appropriate variables substituted in is not difficult, but is mainly an exercise in string manipulation and tab and newline characters. For this reason, we have provided the code for you in a file named `install.rule` in the `lab99b_extended_features/app` directory.

What this code that you are cut and pasting into `package.bld` does is to append the following to `package.mak` (provided as pseudo-code below):

```
install: <basename>_<profile1>.x<suffix1> ...
$(CP) $^ <exec_dir>

clean::
$(RM) <exec_dir>/<basename>*
```

Where `<basename>`, `<profile>`, `<suffix>` are variables determined by the `basename` string, `profiles` array and `targets` array that are specified at the beginning of `package.bld`. By iterating through the `profile` and `targets` arrays, we can build all combinations of target and profile into the install rule's dependencies and then copy these to `<exec_dir>`, which is a variable set in `config.bld` that specifies the directory (relative to the host's filesystem) to copy executable files to so that they can be executed on the DVEVM. This is typically an NFS shared directory between the DVEVM and the host.

You can erase the `install.rule` file after you have copied it, if you like.

Hint: If you accidentally overwrote `install.rule`, just cut and paste this section from the `package.bld` of the solution file in `solutions/lab99b_extended_features/app`

16. Test that the install rule is correctly functioning by changing directory to the top level of `lab99b_extended_features` and executing the `runxdc.sh` script with "install" as an argument

```
# ./runxdc.sh install
```

17. Log a Linux terminal into the DVEVM if one is not already available.

The suggested terminal is via the serial connection. Within the Linux environment (i.e. within the VMware virtual machine), the minicom utility is available. In the windows environment, the terra-term utility is available. Finally, for those more experienced with Linux, a telnet terminal can be used to connect to the board over Ethernet. All three of these methods have been configured in your setup.

18. Execute `app.x470MV` on the DaVinci EVM via the terminal opened in step 17.

The “install” phase of the build places this executable into the `/opt/workshop` directory on the board. (This corresponds to the `/home/user/workdir/filesys/opt/workshop` directory on the host as the board is booted using an NFS root mount that shares the `/home/user/workdir/filesys` directory from the host computer as its root directory.)

Adding a source file search to package.bld

Instead of manually specifying main.c as the source file for this build, we can take advantage of JavaScript’s `java.io.File(<path>)` object, which contains all of the files located at the given path. You can specify a path of `‘.’` to indicate the current path and then use the `list()` method of this object to create an array of all files in the current path. Store this in an intermediate array such as “sources.”

Recall that the `java.io.File(<path>)` object contains all of the files located at a given path. You may then iterate through the “sources” array and use the `String.match()` standard JavaScript method of the `String` class to test each file in the directory against the following regular expression:

```
/.*\.c$/
```

For those files in the directory which match the regular expression, use the `Array.push()` method to place the matched file onto a new array called “csources”

Hint: Regular Expressions

Hint: While it is beyond this course to discuss regular expressions in detail, we will explain the above regular expression, hopefully providing a basis for those unfamiliar with regular expressions to potentially modify if needed.

Hint: The forward slashes (/) bracketing the expression are required to indicate that the text inside is a regular expression.

Hint: The period (.) indicates any character except for a newline.

Hint: The asterisk (*) indicates that the preceding set can be repeated any number of times, so a period followed by an asterisk (.*) indicates any series of characters of any length.

Hint: The backslash (\) is an escape character, meaning that the character to follow it will be interpreted literally instead of with any special meaning assigned to that character. In this case, a backslash and period together (\.) indicates simply the period as a character, instead of as a special character the way it was used previously.

Hint: The c is simply a character to be matched

Hint: Finally, the dollar sign (\$) is a special character called an anchor and indicates that the preceding set must be matched at the end of the entry. In this case, it means “myFile.c” will be matched, but “myFile.cpp” will not be matched.

Hint: The expression above (/ . * \ . c \$ /), then, indicates <anything>.c

19. Instead of directly specifying the C source files for your build in an array called `csources`, use the `java.io.File().list()` method to search for all files in the current directory and the `String.match()` method to sort those files which have a `.c` extension

This code should look like the following:

```
var sources = java.io.File('.').list();
var csources = [];

for (var i = 0; i < sources.length; i++){
    if(String(sources[i]).match(/.*\.c$/))
        csources.push(sources[i]);
}
```

Be careful of typos – it’s common to miss one of the “)”.

20. Finally, there is one last `package.mak` rule that we must add in using `Pkg.makeEpilogue` to `include ../../custom.mak` (see discussion below).

Put the following statement at the end of `package.bld`.

```
Pkg.makeEpilogue += "include ../../custom.mak\n\n"
```

Recall that, as discussed in the lecture, the XDC tool parses the `config.bld`, `package.xdc` and `package.bld` project files and uses them to build `package.mak`, a gnu makefile that specifies how all files for the project are to be built.

As much as makefiles keep track of dependencies for files they build, the XDC tool keeps track of the dependencies for this `package.mak` file, which are `config.bld`, `package.xdc` and `package.bld`. As such, `package.mak` is only regenerated if one of these three files changes, which saves time in the build process.

However, since we are using JavaScript to build the `csources` array based on the contents of the current directory, the array of source files could change if a new source file is added into the directory or one is taken away. We therefore need to manually place a rule in `package.mak` that will rebuild itself if the contents of the current directory change. This is exactly what the code in `custom.mak` (the file we are including) does. Feel free to look through the file if you like, or simply use it as example code and include in any `package.bld` scripts that use the `java.io.File('.')` object.

21. Try building and running the project using the new `csources` code.

Questions

1. Which file in the `lab99a_hello_world/app` directory specifies that one or more executable files are to be built?

2. Which function call within this file actually scripts the building of a new executable file?

3. Examination of the `config.bld` file (at the top level of the workshop directory) shows that we have configured a third target, named C64P. This target is configured to build for the C64x+ DSP core that is available on the TMS320C6446. How could you modify the available files to build a Hello World executable that would run on the C64x+ core?

4. (Extended lab) Explain in your own words how the C source files used to build this application are specified in the `package.bld` build script of `lab99b_extended_features`.

5. (Extra credit) When compiling code for the C64x+ DSP, what compiler options are used for this lab exercise.

Hint: It's a combination of the C64x+ compiler strings from these files.

`config.bld:` _____

`ti.targets.C64P:` _____

`package.bld:` _____

Challenges

Note: Before attempting any of the following challenges, please copy the `lab99b_extended_features` folder into a new working folder, such as `lab99_challenge1`, `lab99_challenge2`, etc. In order to copy the entire folder, you will need to use the “-R” (recursive) flag with the “cp” command:

```
# cd /home/user/workshop
# mkdir lab99_challenge1
# cp -R lab99b_extended_features/* lab99_challenge1
```

Hint: Hint: The following man pages may come in handy

```
# man stdio.h
# man string.h
# man ctype.h
```

1. Modify the hello world application to ask the user’s name and use it in the greeting:

```
# ./app_release.x470MV
```

```
What is your name?
Steve
Hello, Steve!
```

2. Modify the hello world application to take the user’s name as a command line argument and use it in the greeting:

```
# ./app_release.x470MV Steve
```

```
Hello, Steve!
```

3. Modify the hello world application to determine the user’s name by either of the methods above. If the user’s name is “Steve”, greet as normal. Otherwise, display the message, “You’re not Steve!”:

```
# ./app.x470MV Scott
```

```
You’re not Steve!
```

4. (advanced) Modify the hello world application to determine the user’s name by either of the methods above, then convert it into an all-caps version in the greeting:

```
# ./app.x470MV Steve
```

```
Hello, STEVE!
```