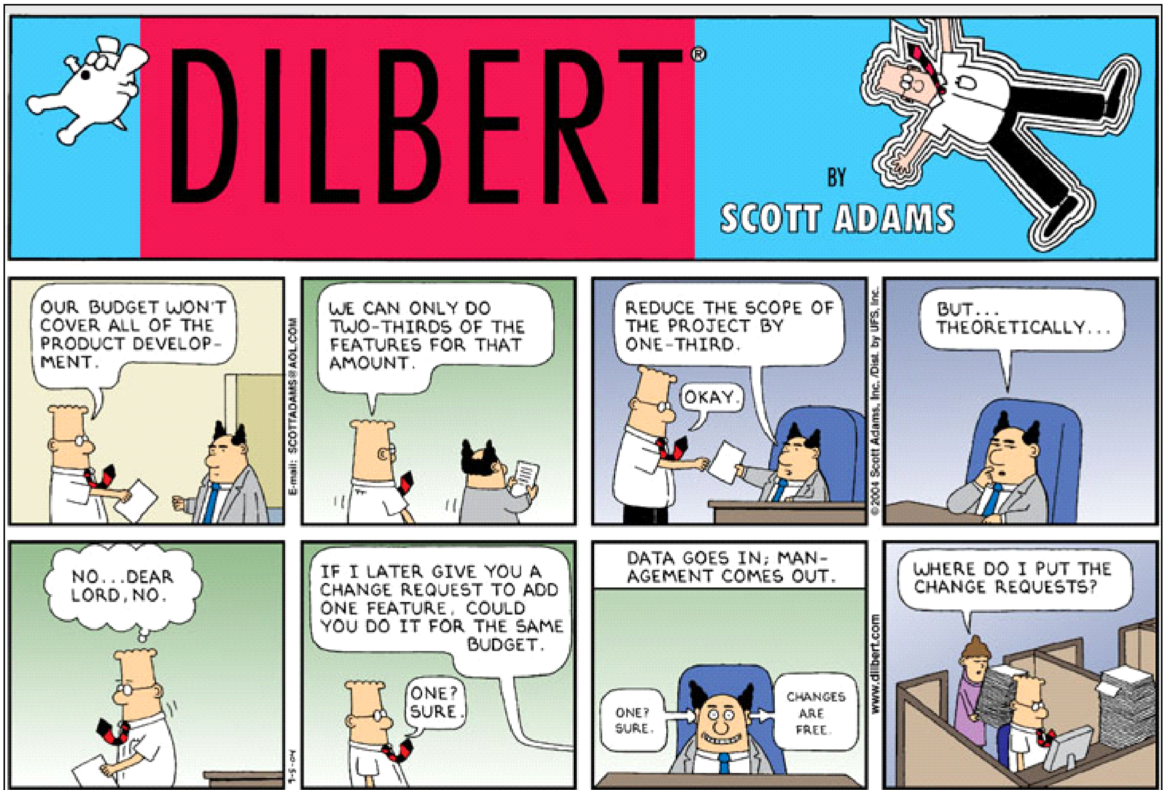
Introduction



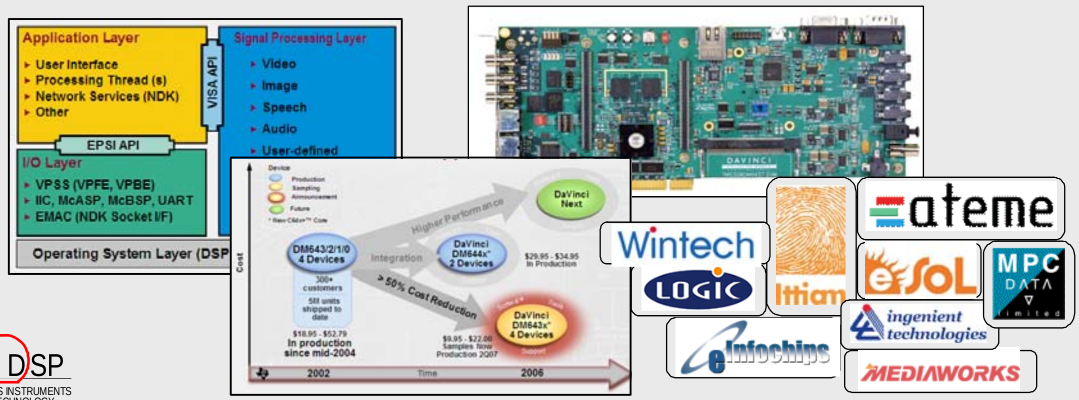
DILBERT: © Scott Adams/Dist. By United Feature Syndicate, Inc.

The electronics industry moves fast and embedded developers are often required to implement feature-rich products with short design cycles. Product requirements often change over the course of development, and, while most product managers are hopefully not as impractical as Dilbert's pointy-haired boss in the above cartoon, embedded developers are often required to react quickly to an ever-changing electronics marketplace.

Module Topics

What is DaVinci™ Technology?

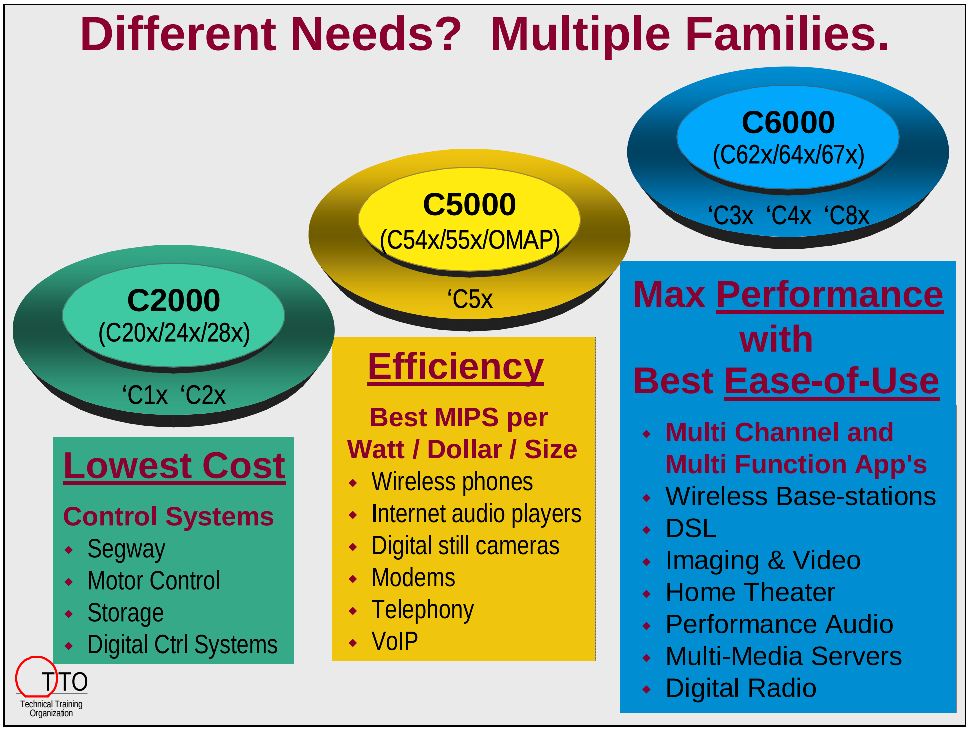
Devices	TMS320DM643x and ‘DM644x
Software	Codec Engine, BIOS, NDK, Codecs
Tools	DM643x DSK, Code Composer Studio
Support	Approved Software Providers (ASP)

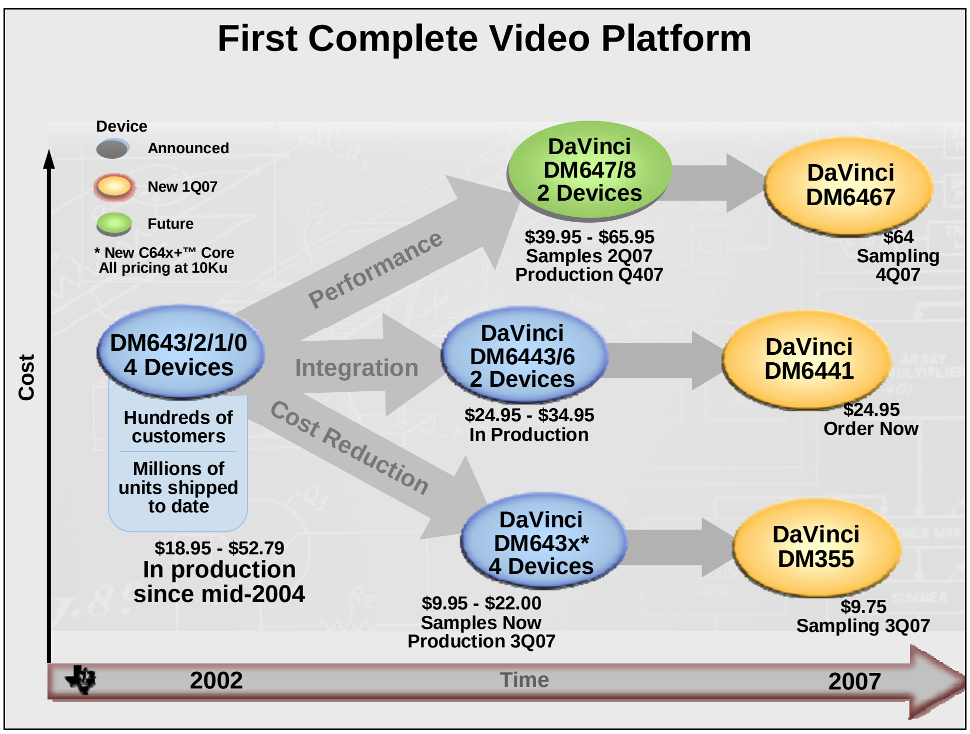


DSP
TEXAS INSTRUMENTS TECHNOLOGY

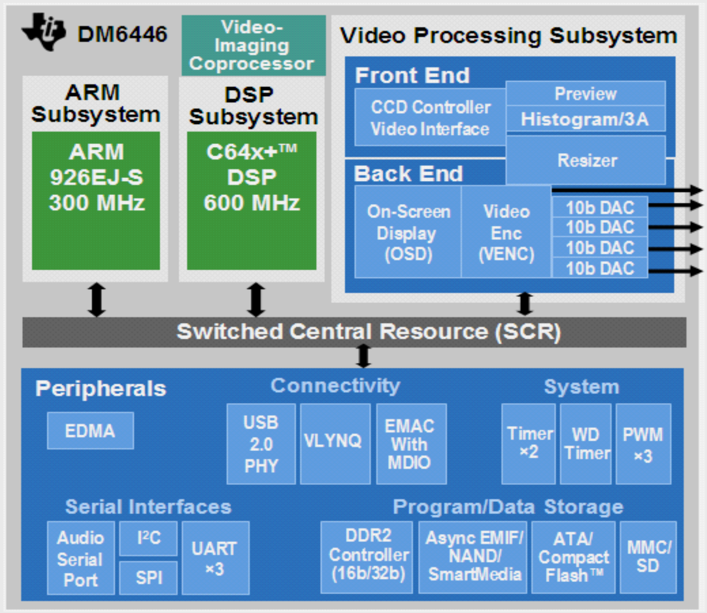
With all of the challenges facing them, embedded software developers cannot afford to reinvent the wheel on every design. A powerful and robust software framework is required to enable rapid prototyping and reliable long-term performance. This is why Texas Instruments’ DaVinci Technology encompasses not only a series of highly integrated multimedia processors but a production quality framework, a number of easily integrated third-party algorithms and a powerful development and debugging toolset.

DaVinci Family Devices





TMD320DM6446: Arm + DSP



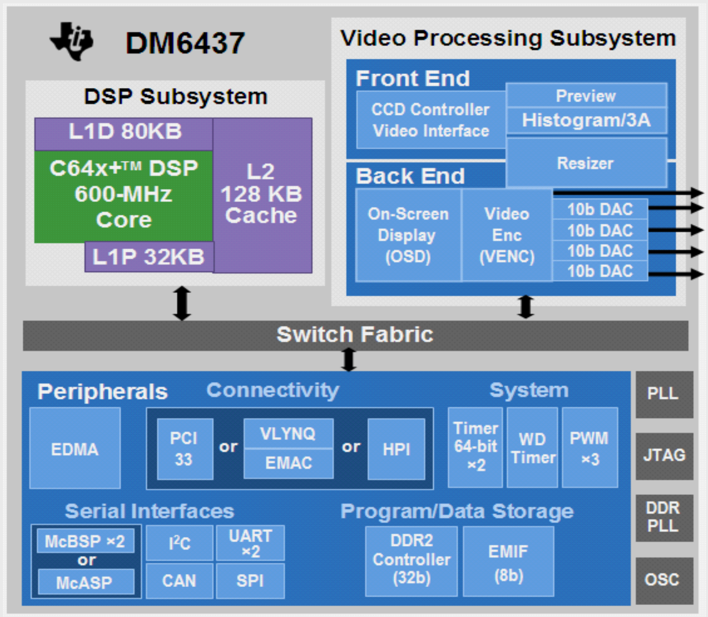
TMS320DM644x in Production Now

Pin for Pin Compatible

	DM6443	DM6446	DM6441
DSP	TMS320C64x+™ 600 MHz	TMS320C64x+™ 600 MHz	TMS320C64x+™ 405/513 MHz
ARM	ARM926EJ-S™ 300 MHz	ARM926EJ-S™ 300 MHz	ARM926EJ-S™ 202/256 MHz
Video Processing Subsystem (VPSS)	**Back-end ONLY**	**Back End AND Front-end**	**Back End AND Front-end**
Video-Imaging Coprocessor (VICP)	**No**	**Yes**	**Yes**
Dual voltage & Clock Scaling	**No** 1.2V core voltage	**No** 1.2V core volta	**Yes** 1.05V / 405MHz 513MHz

>35% Power Reduction

TMS320DM6437: DSP Only



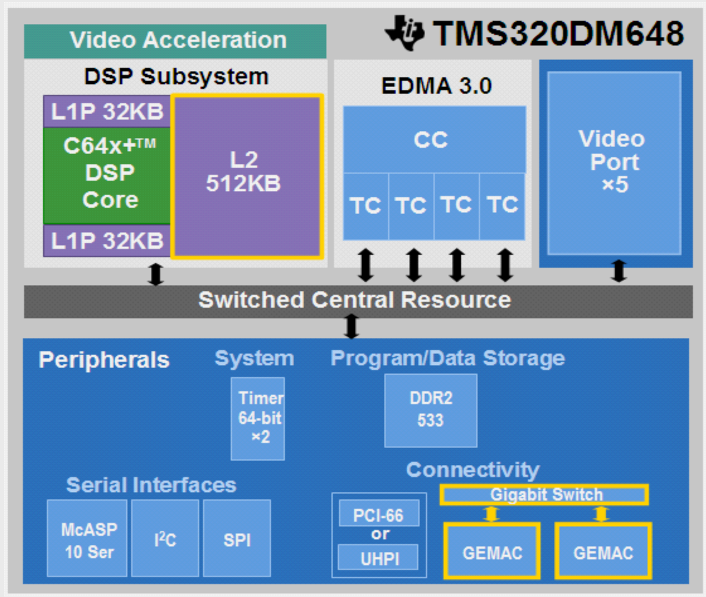
TMS320DM643x Sampling Now

Pin for Pin Compatible

	DM6431	DM6433	DM6435	DM6437
	TMS320C64x+™ 300 MHz	TMS320C64x+™ 600 MHz	TMS320C64x+™ 600 MHz	TMS320C64x+™ 600 MHz
	DDR2 - 266 MHz (16b)	DDR2 - 266 MHz (32b)	DDR2 - 266 MHz (32b)	DDR2 - 266 MHz (32b)
VPSS	1x Video Port (10b)	Back-end ONLY	Front-end ONLY	Back-end AND Front-end
L1/L2	64KB L2 32K L1P / 32K L1D	128KB L2 32K L1P / 80K L1D	128KB L2 32K L1P / 80K L1D	128KB L2 32K L1P / 80K L1D
I/F	EMAC or EMIF McASP, I²C, CAN, SPI, UART	PCI or VLYNQ EMAC, HPI or EMIF McASP, I²C, SPI	VLYNQ AC, HPI or EMIF P, I²C, CAN, SPI, UART (2)	PCI or VLYNQ EMAC, HPI or EMIF McBSP or McASP I²C, CAN, SPI, UART (2)

AEC-Q100 Qualified

TMS320DM647/8: Multiple Video Ports



TMS320DM647/8

	DM647	DM648
	TMS320C64x+™ 900 MHz	**TMS320C64x+™ 900 MHz**
	DDR2 - 533 MHz (32b)	**DDR2 - 533 MHz (32b)**
Video Ports	**5 16-bit Dual Channel**	**5 16-bit Dual Channel**
L1/L2	**256KB L2 32K L1P / 32K L1D**	**512KB L2 32K L1P / 32K L1D**
Dual voltage & Clock Scaling	**PCI or UHPI Gigabit EMAC x2 + Switch McASP, I²C, SPI, Timer (2)**	**PCI or UHPI Gigabit EMAC x2 + Switch McASP, I²C, SPI, Timer (2)**

TMS320DM6467: High Definition Video

Features

- **Core**
  - **ARM926EJ-S™ (MPU) at 300 MHz**
  - **TMS320C64x+™ DSP Core at 600 MHz**
- **Memory**
  - **ARM: 16K I-Cache, 8K D-Cache, 32K TCM RAM, 8K Boot ROM**
  - **DSP: 32K L1 I-Cache, 32K L1 D-Cache, 128K L2 Cache, 64K Boot ROM**
- **HD Coprocessors**
  - **Real-Time HD-HD Transcoding Up to 1080p**
    - **Multi-format (mf) HD to mf HD or mf SD**
    - **Up to 2× real time for HD-to-SD transcode**
    - **Real-time HD-HD transcoding for PVR**
  - **Video Encode and Decode**
    - **HD 720p H.264 BP encode**
    - **HD 1080i/p H.264 HP@L4, decoding; HD 1080i/p VC1/WMV9, decoding; HD 1080i/p MPEG-2 MP@HL, decoding; HD 1080i/p MPEG-4 ASP, decoding; DivX**
    - **Simultaneous SD H.264 BP 30 fps encode and decode**
- **Peripheral Highlights**
  - **Video ports**
    - **Two 8-bit BT.656 or one 16-bit BT 1120 capture**
    - **Two 8-bit BT.656 or one 16-bit BT 1120 display**

◆**Samples Dec. 07; TMS Qual 2Q08**

◆**RTM December 3rd**

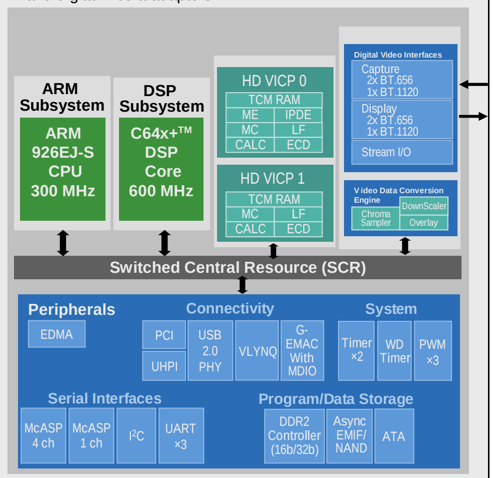
Benefits

◆Scalable video engine building on high-performance C64x+ media DSP, low-cost local controllers, and rich suite of multi-format video accelerators

◆**$64 at 10kU**

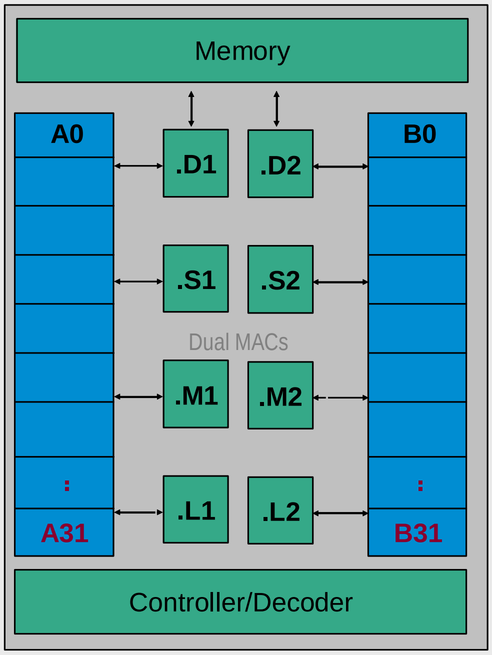
Applications

- Transcoding (HD-HD, HD-SD), HD-video conferencing, HD-IP set-top boxes, video surveillance, video phones and digital media adaptors



DaVinci Architecture

'C6000 CPU Architecture



- ‘C6000 Compiler excels at Natural C
- While dual-MAC speeds math intensive algorithms, flexibility of 8 independent functional units allows the compiler to quickly perform other types of processing
- All ‘C6000 instructions are conditional allowing efficient hardware pipelining
- ‘C6000 CPU can dispatch up to eight parallel instructions each cycle

Challenge: Keeping 4800 MIPS CPU “Fed”

C64x+ CPU can:

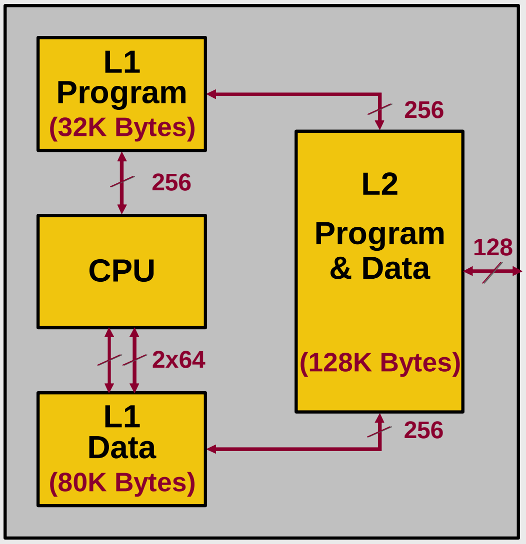
read/write up to 9.6 GBytes per second data

read up to 19.2 GBytes per second program

- Megabytes of fast internal memory is possible but expensive
- Better cost/performance using cache and DMA

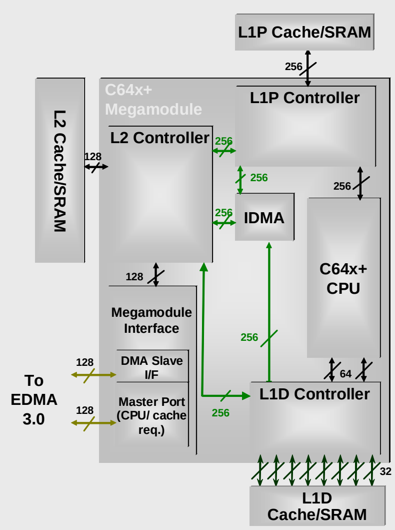
Conclusion: Efficient data routing is crucial to maximizing performance

DM643x Internal Memory



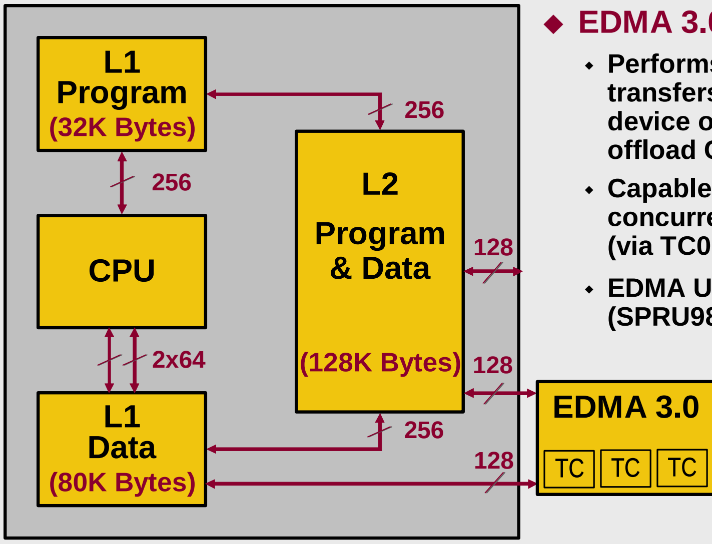
- **Level 1 Caches**
  - Single-cycle access
  - Always enabled
  - L2 accessed on miss
  - Configurable as cache or addressable RAM or combination
- **Level 2 Memory**
  - Unified: Prog or Data
  - Configure L2 as cache or addressable RAM or combination

C64x+ Improved Bandwidth



- Buses Internal to Megamodule
  - Use of 256-bit buses
  - Run at ½ CPU rate to save on power
- CPU/DMA concurrency in L1D
  - 256-bits available every cycles
    - 8 x 32-bits
  - 128-bits maximum requested by CPU
    - 2 x 64-bit Paths to CPU
  - 128 bits left over for DMA
  - Each 32-bit bank arbitrated independently
- CPU/DMA concurrency in L2
  - CPU Access to L2 is infrequent over time
  - In some chips, two distinct 128-bit pages arbitrated separately
- More DMA System Memory Connection compared to C64x
  - 2 x busses. Dedicated buses for each
    - Cache requests
    - DMA's into internal memory
  - Upto 2 x width
    - 128 bits wide

DM643x EDMA 3.0

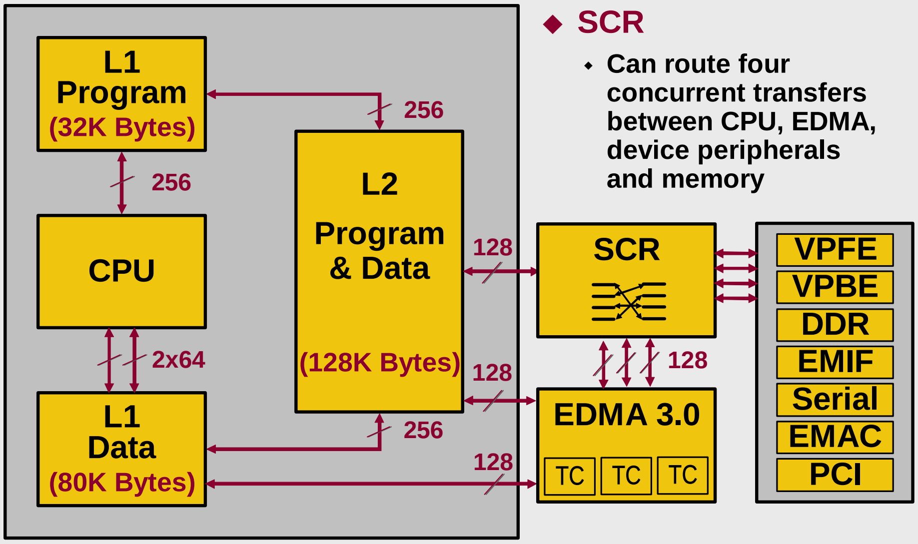


- **EDMA 3.0**
  - Performs data transfers with external device or memory to offload CPU
  - Capable of three concurrent transfers (via TC0, TC1, TC2)
  - EDMA User's Guide (SPRU987)

Enhanced Direct Memory Access Controller 3.0 (EDMA 3.0)

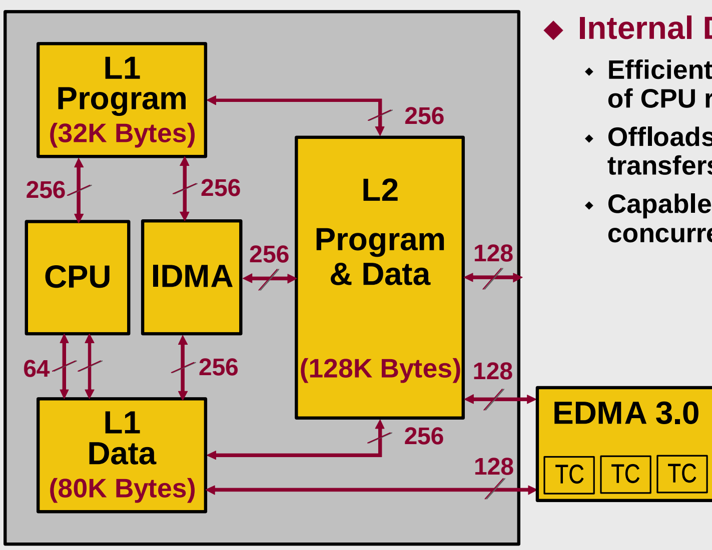
- **Performs Data Transfers to offload DSP**
- **Transfer Controller (TC) Improvements over DM644x**
  - 3 Transfer Controllers (TC0, TC1, TC2)
    - TC0: Short burst transfers with stringent deadlines (e.g. audio data)
    - TC1: High throughput bulk transfers
    - TC2: PCI or miscellaneous transfers
  - Programmable Default Burst Size (DBS) for each TC
    - System Module register EDMATCCFG
    - Recommendation: Use default DBS
- **Reference**
  - EDMA User's Guide (SPRU987)
  - Device-Specific Data Manual for details on EDMATCCFG register

DM643x Switched Central Resource



- **SCR**
  - Can route four concurrent transfers between CPU, EDMA, device peripherals and memory

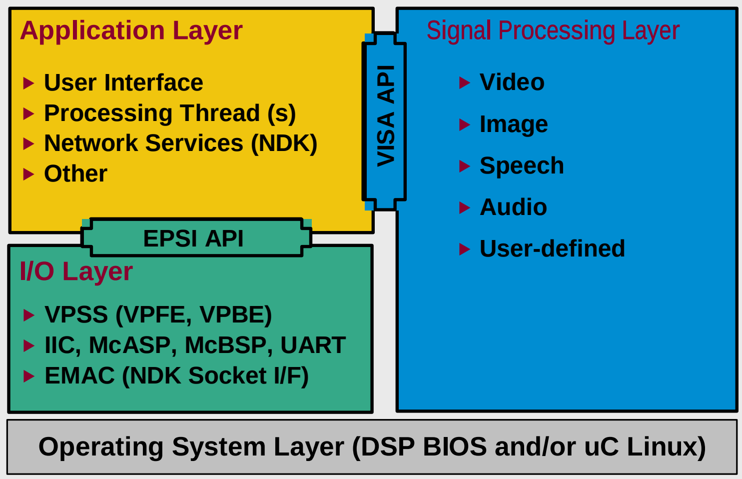
DM643x IDMA



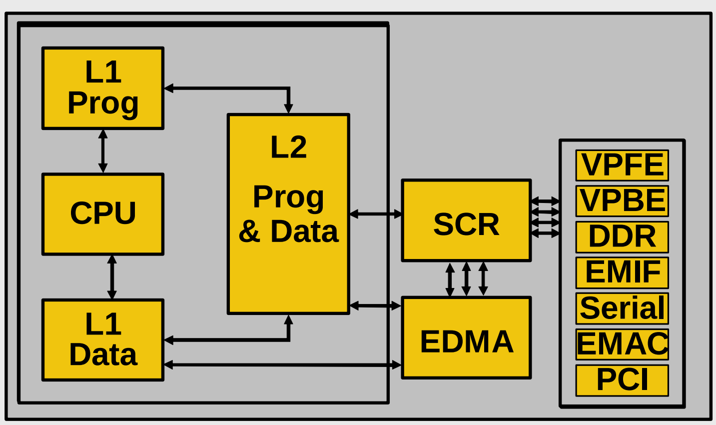
- **Internal DMA**
  - Efficient programming of CPU registers
  - Offloads internal cache transfers from EDMA
  - Capable of two concurrent transfers

DaVinci Family Software

TMS320DM643x Software Overview

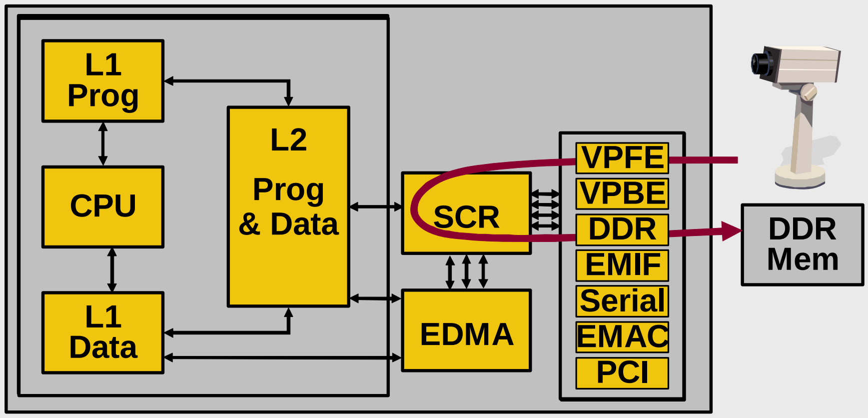


Hardware Abstraction: EPSI and VISA



It's great to have this extensive memory and data bus support, but is it difficult to program?

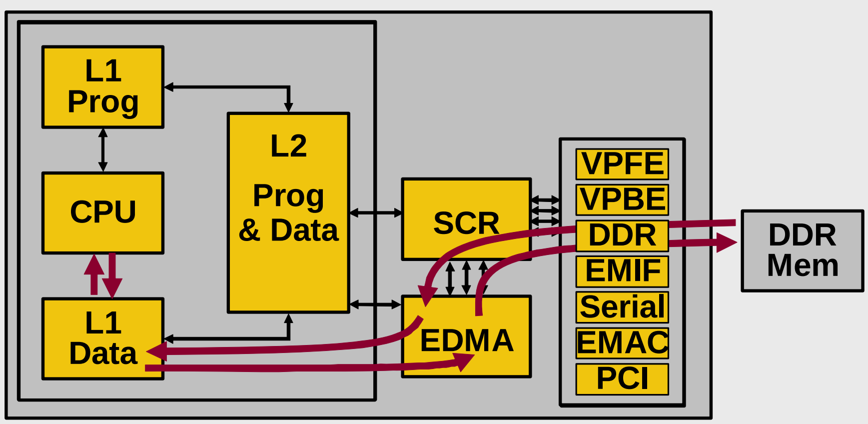
Application Programming: EPSI API



Step 1: **Read video data through VPFE, use EDMA to route through SCR into external DDR memory**

FVID_exchange(vpfeChan, bufp);

Application Programming: VISA API



Step 2: **Transfer raw video data from DDR memory into L1 Data memory using EDMA. Compress in blocks directly from L1 memory and store result back to DDR using EDMA.**

VIDENC_process(videncChan, inbuf, outbuf, inarg, outarg);

TMS320DM643x Software Overview

Module 1

Module 3

Module 2

Module 4

- **We'll examine these concepts in more detail in the modules to come**

DM643x Family Development Tools

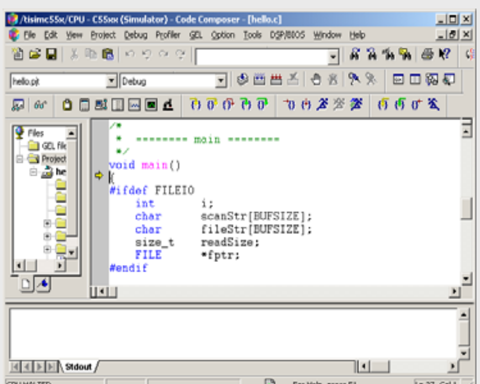
CCStudio v3.3 Merges All Platforms into One Easy-to-Use IDE

C6000™ C5000™ C2000™ OMAP™ DaVinci™

Now supports C64x+™ and DaVinci processors

- Graphical User Interface
- Integrated Compiler, Assembler and Linker
- DSP/BIOS operating system (no per-unit license)
- Stop-based and Real-time JTAG Debugging
- Open interface for Plug-ins (Such as Mathworks' Matlab™)
- Consistent IDE across 5 major DSP Platforms

Programming with DSP/BIOS



- C- and ASM-callable functions
- Interactive configuration tool
- Kernel-aware debug support
- On-the-fly program analysis

EMULATION

HOST DEVELOPMENT COMPUTER

RTDX

JTAG

target application program

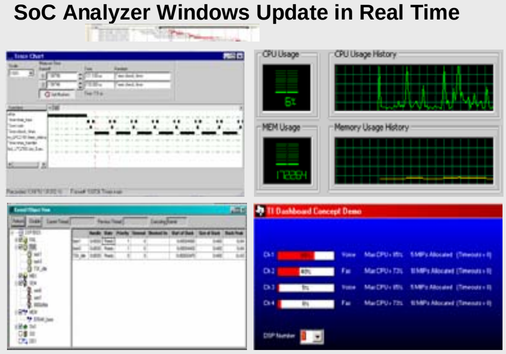
DSP/BIOS Kernel Interface

real-time capture	multiple threads	hardware abstraction

TARGET TMS320 DSP HARDWARE

SoC Analyzer

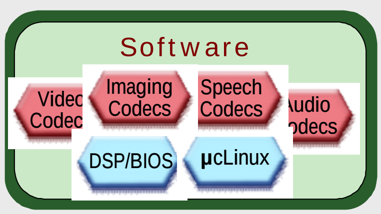
SoC Analyzer Windows Update in Real Time



- Sits beside any IDE (TI or 3rd party)
- Can be easily made into an Eclipse plug-in (ideal for CCE)
- Primary use - visualizing SoC streaming data, not post-mortem static data analysis
- It is not an IDE … will not provide basic debugging, project management or editor
- Based on TI's multi-client target server
- Will support multiple host OS's
- Can be extended (by many – not just TI) to provide multiple solutions
- Primarily for high-level system tuning

Collection Type	Extraction Protocol	Visualization Format
Sequencer logs	BIOS logs over RTDX	Time sequence of execution profile
Application level profiling time stamps	TCP/IP through ARM serial port	Bar chart or Pareto rank
Link messages	MSGQ (SW)	Time stamped message sequence

DVDP Kit Features



DVDP Software
- **Codec Engine**
- **CSL/Drivers**
- **Demos**
- **Codecs**
- **Docs**

A/V Files

CCS
- **CSP**
- **DSP/BIOS**

SDI Software
- **Flashburn**
- **EVM Docs**
- **Diagnostics**
- **PCI Host Driver**

SoCrates System Analyzer

Virtual Logix Linux

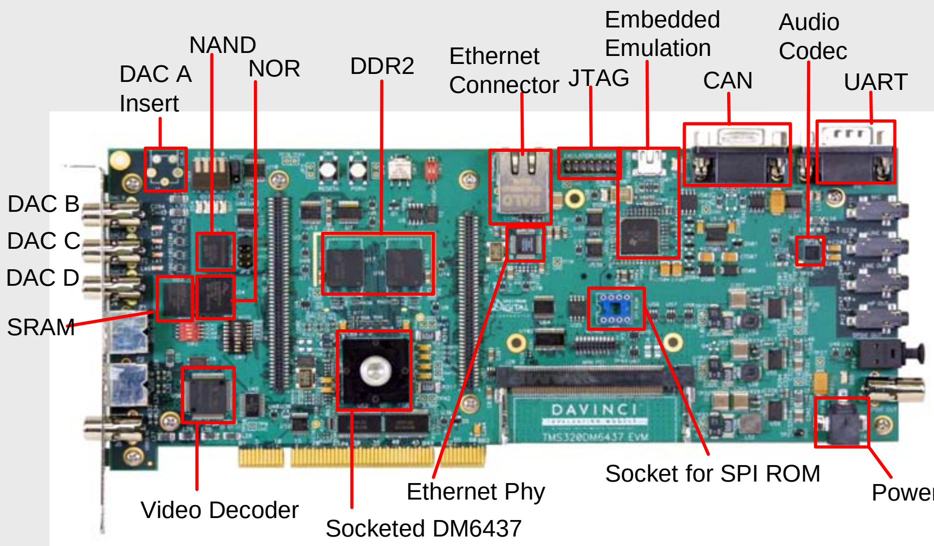
Quick Start Guide(s)

Setup Guide

USB/Ethernet Cables

Power Supply

DM6437 EVM Pinout



DM6437 EVM Block Diagram

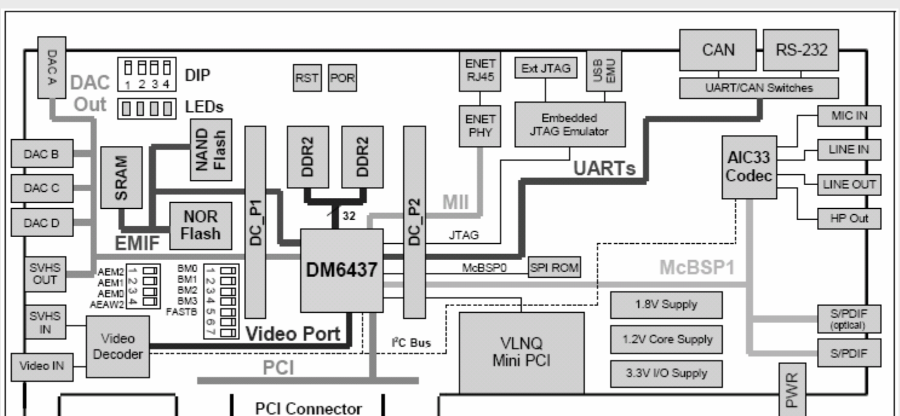


Figure 1-1, Block Diagram DM6437 EVM

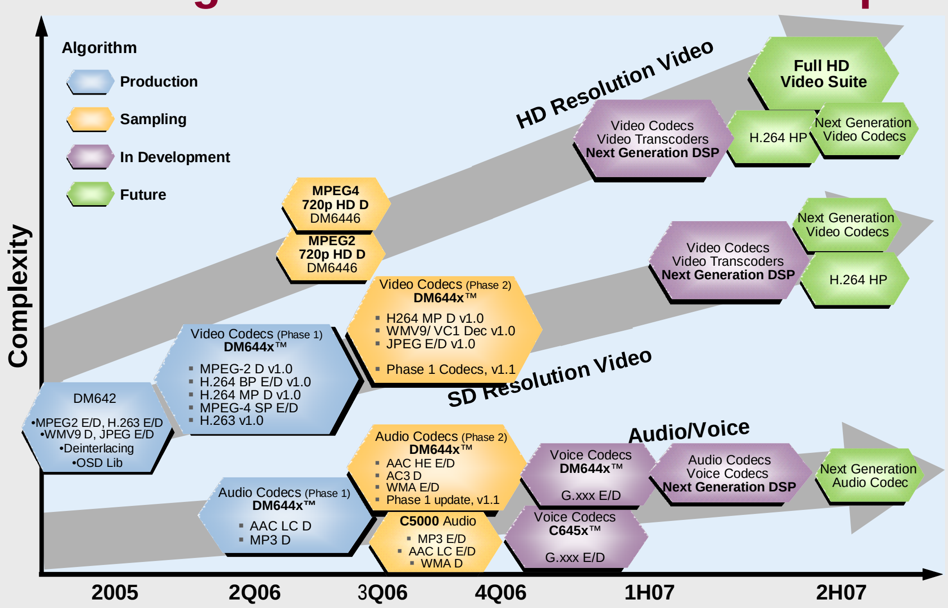
Authorized Software Providers

Customized Support from Authorized Software Providers

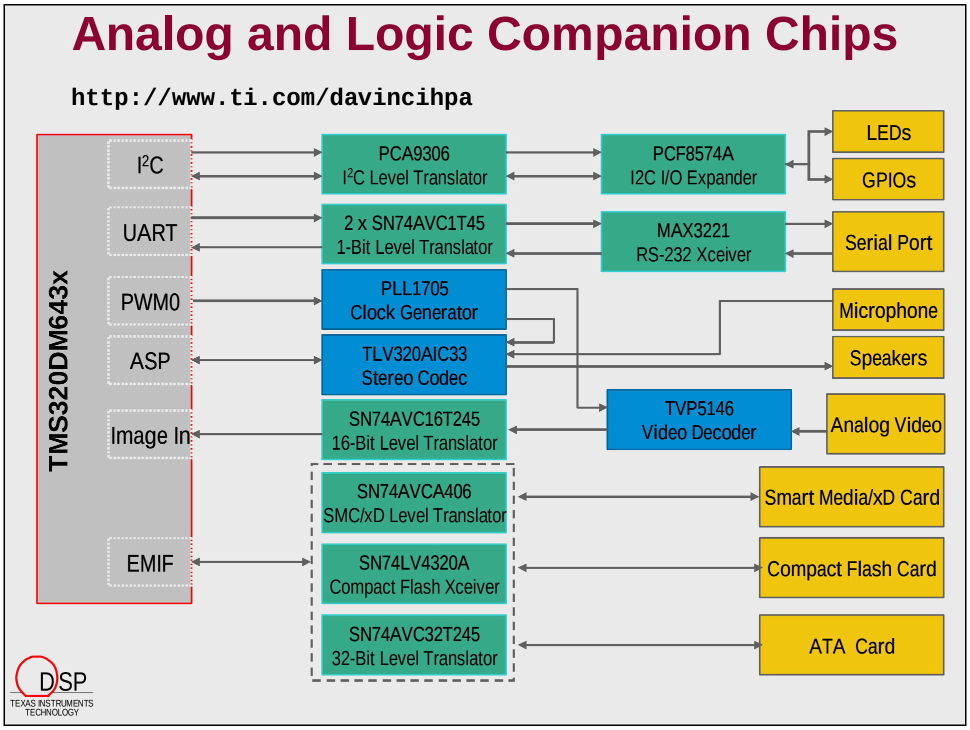
- Free 60 day evaluation with up to four hours evaluation support
- Up to 40 hours of support per production license

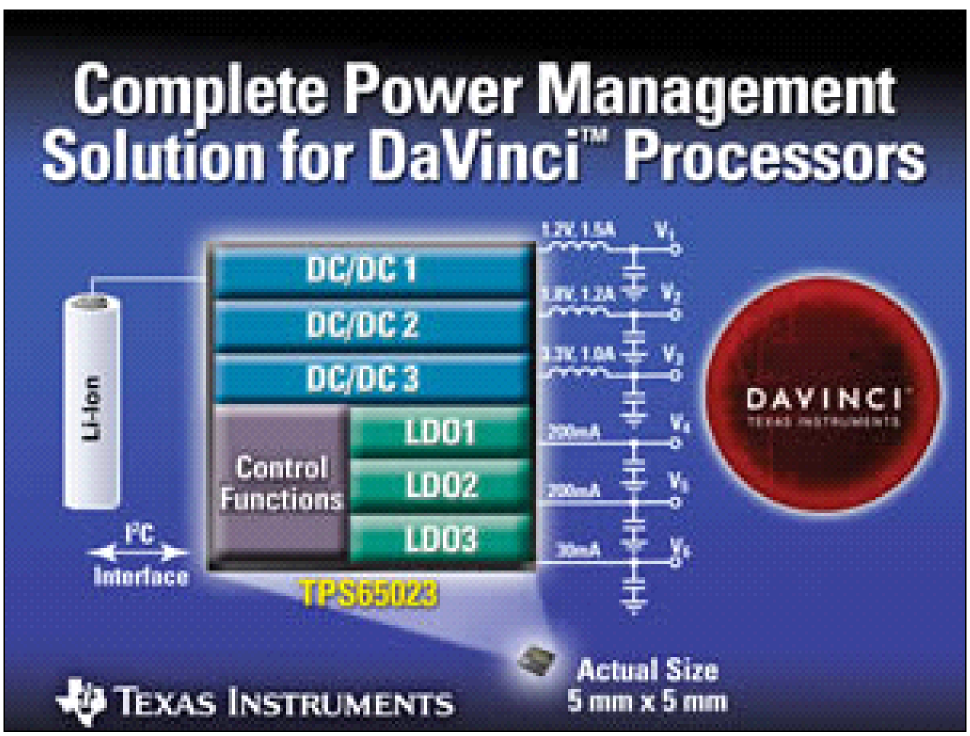
- ASPs: Companies selected to resell and fully support xDAIS algos on DaVinci technology-based and other TI DSPs
- Selected based on support and service capability, customer track record
- Additional integration and customization services available
- ASPs can act as single point of support for multiple algorithms
- Experienced in video applications

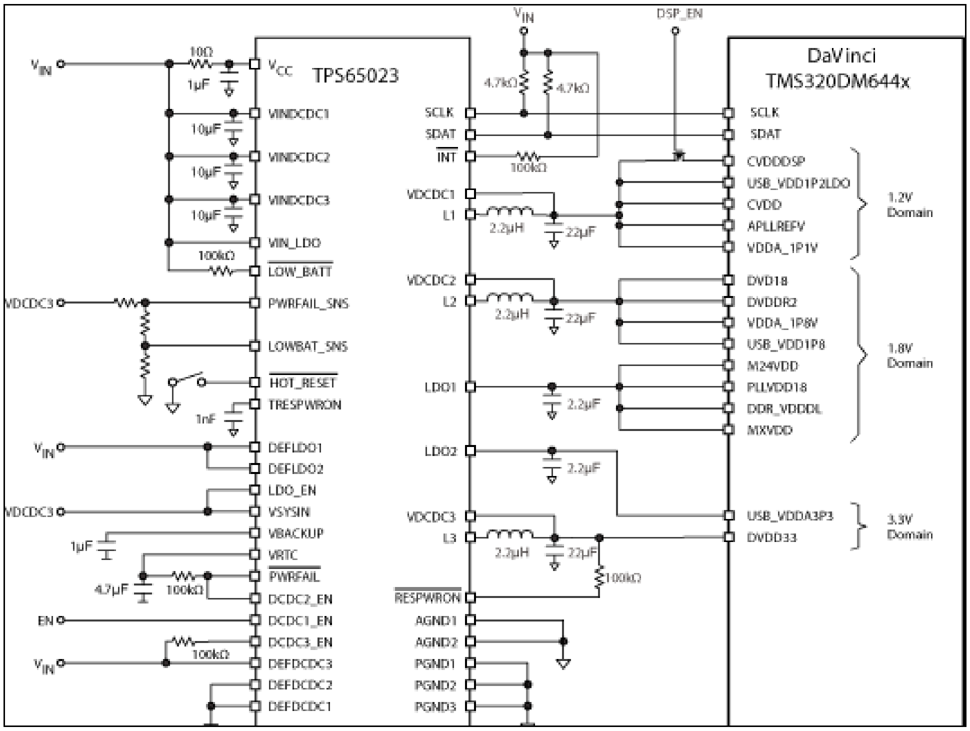
TI Digital Media Software Roadmap



Analog and Logic Companion Chips






TPS65023
DaVinci
TMS320DM644x
VCC
VINDCDC1
VINDCDC2
VINDCDC3
VIN_LDO
LOW_BATT
PWRFAIL_SNS
LOWBAT_SNS
HOT_RESET
TRESPWRON
DEFLDO1
DEFLDO2
LDO_EN
VSYSIN
VBACKUP
VRTC
PWRFAIL
DCDC2_EN
DCDC1_EN
DCDC3_EN
DEFDCDC3
DEFDCDC2
DEFDCDC1
SCLK
SDAT
INT
VDCDC1
L1
VDCDC2
L2
LDO1
LDO2
VDCDC3
L3
RESPWRON
AGND1
AGND2
PGND1
PGND2
PGND3
CVDDDSP
USB_VDD1P2LDO
CVDD
APLLREFV
VDDA_1P1V
1.2V Domain
DVD18
DVDDR2
VDDA_1P8V
USB_VDD1P8
M24VDD
PLLVDD18
DDR_VDDDL
MXVDD
1.8V Domain
USB_VDDA3P3
DVDD33
3.3V Domain
VIN
DSP_EN
VDCDC3
EN
10Ω
1µF
10µF
100kΩ
4.7kΩ
2.2µH
22µF
2.2µF
1nF
4.7µF

For More Information

For More on the C6000 Architecture

DaVinci Webpage

http://www.ti.com/davinci

Technical Documents (User's Guides, etc.)

http://focus.ti.com/dsp/docs/dspsupporttechdocs.tsp?sectionId=3&tabId=409&familyId=44

DaVinci Software – benchmarks, pricing, availability

http://www.ti.com/digitalmediasoftware

C6000 Optimization Workshop

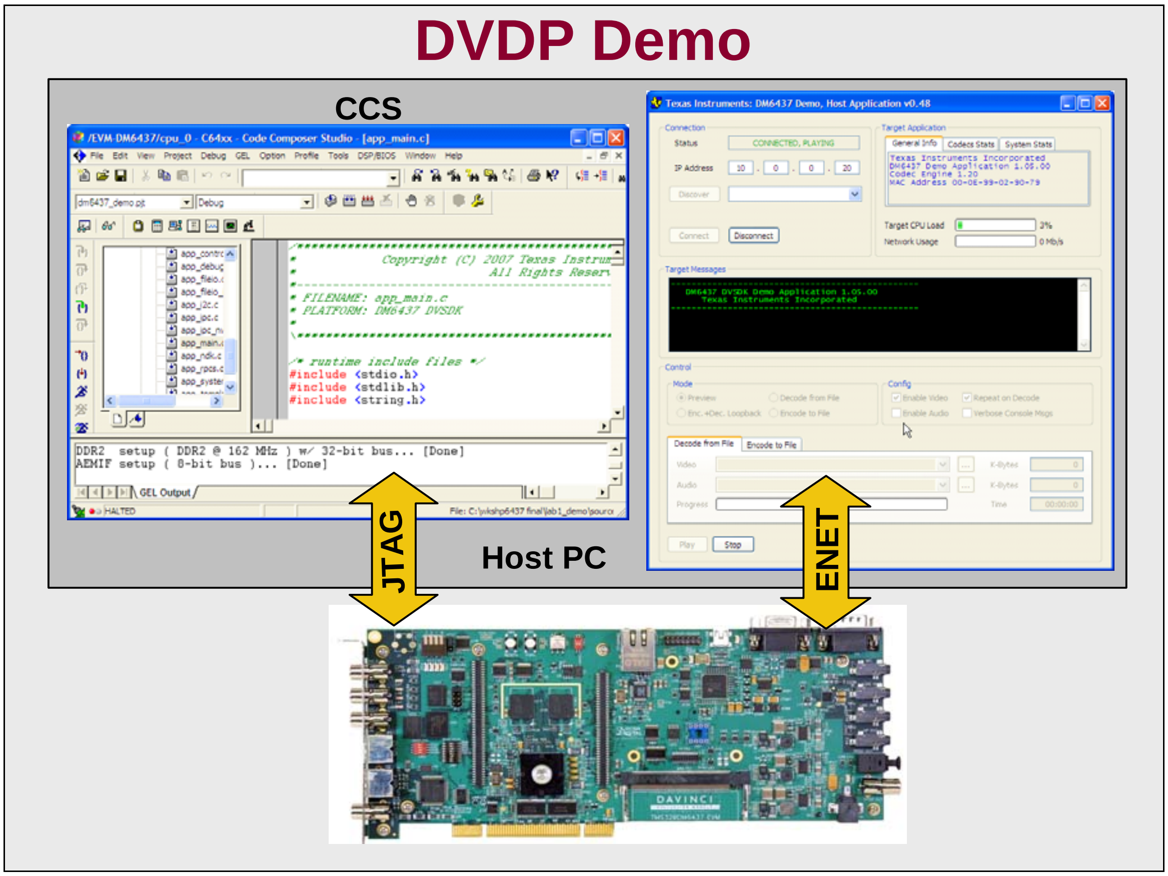
http://focus.ti.com/docs/training/catalog/events/event.jhtml?sku=4DW102260

DSP/BIOS OS Design Workshop

http://focus.ti.com/docs/training/catalog/events/event.jhtml?sku=4DW102090

Lab

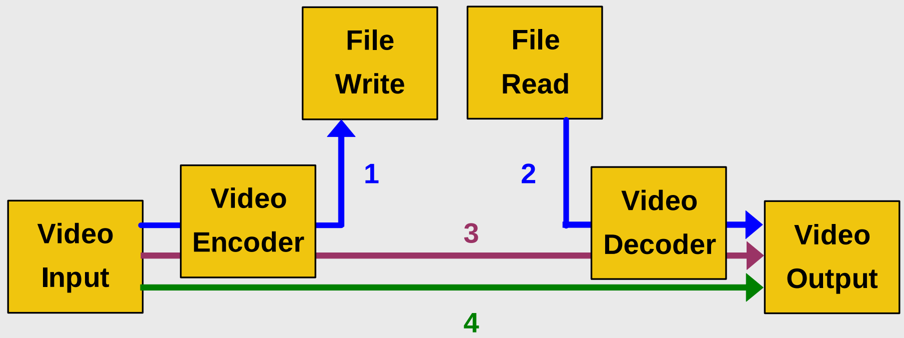
In the lab exercise for the first module, we will compile, load and execute the primary example application that is packaged with the DM6437 Digital Video Development Platform (DVDP). This example demonstrates many features available with this system, including networking, audio and video processing, and capture and display of audio and video data. The example is built upon an extensible framework that may be adapted to user applications, and is provided fully in source code with the exception of the evaluation codec libraries, the Networking Development Kit (NDK) libraries and the DSP/BIOS operating system libraries utilized.



The example demo consists of two applications – a server application which executes on the DVDP and a client application which executes on the host PC. The server application is built (compiled and linked) inside of the Code Composer Studio development system. Though Code Composer Studio itself executes on the host PC, this tool builds code which executes on the DM6437 DVDP and can be used to load this code onto the board and execute it.

The client application is a host-side application that runs on the PC host computer. This application is written in Javascript, which is in interpreted language, meaning that it does not have to be compiled before it is executed. This has the advantage that development and debugging are simpler and faster. The host-side client is used to control the operation of the demo running on the DM6437 DVDP.

Block-based Video Channels



1. **Video Recorder**
2. **Video Playback**
3. **Video loopthru with encode/decode**
4. **Video loopthru**

The client-side (i.e. executing on the DM6437 DVEVM) application is written within a modular framework that provides a simple mechanism for re-routing data between processing blocks. Four pre-defined routing structures allow for the four modes of record, playback, loopthru and loopthru with encode/decode. Selections on the host-side application are communicated via the host-to-board ethernet connection and implemented within this framework as shown on the above diagram.

Build/Load/Run the DVDP Application

1. **Start Code Composer Studio (If not already open)**

   You should have a shortcut icon on the Windows desktop

2. **Connect the DVDP board emulation**

   Debug→Connect

   or Alt-C

3. **For Convenience, Configure Code Composer Studio to always Connect at Startup**

   Option→Customize…

   In the "Debug Properties" tab, under the "Target Connection Actions", check the checkbox for "Connect to the target at startup"

   Now you won't need to do Debug→Connect every time you start CCS.

4. **Load the dm6437_demo.pjt project into CCS**

   Project→Open…

   Navigate to C:\dm6437_1day\lab1_demo_1_30

   select dm6437_demo.pjt

5. **Open the app_main.c file**

   You can expand the "Source" folder within the project view window and double click on the file name to open it.

6. **Search within app_main.c for the string "APP_GLOBAL_data.ndk"**

   Edit→Find… (or Ctrl-F)

7. **Confirm the following networking properties**

```
status = APP_SYSTEM_configSet(
   intcpy(APP_GLOBAL_data.ndk.priority      , 13 );
   strcpy(APP_GLOBAL_data.ndk.ipAddrMethod  , "static" );
   strcpy(APP_GLOBAL_data.ndk.staticIpAddr  , "192.168.1.41" );
   strcpy(APP_GLOBAL_data.ndk.subnetMask    , "255.255.255.0" );
   strcpy(APP_GLOBAL_data.ndk.gateway       , "0.0.0.0" );
   strcpy(APP_GLOBAL_data.ndk.domainName    , "tidsp.demo.net" );
   strcpy(APP_GLOBAL_data.ndk.dnsServer     , "192.168.1.39" );
   strcpy(APP_GLOBAL_data.ndk.dnsName       , "tidsp");
   intcpy(APP_GLOBAL_data.ndk.ipDiscoveryPort, 44000 );
```

   Note: This settings are correct for the workshop installation. For home use, this is where you can modify the application to match your network configuration. If connecting the board into a router, change the ipAddrMethod property to "dynamic" (IP address, gateway, domain, DNS values set in this configuration will be ignored and the system will use the values provided by the router via the DHCP protocol.)

8. **Reset the dm6437 CPU on the DVDP board**

   Debug→Reset CPU (or Ctrl-R)

   Note: It is a good idea to always reset the CPU before loading any new program

9. **Build the project**

   Project→Rebuild All

10. **Load the binary executable onto the DVDP board**

    File→Load Program… (Or Ctrl-L)

    Navigate to C:\dm6437_1day\lab1_demo_1_30\Debug

    select dm6437_demo.out

11. **For convenience, set CCS to automatically load .out files after build**

    Option→Customize…

    In the "Program/Project/CIO" tab, under the "Program Load", check the checkbox for "Load Program After Build"

    Now you won't need to do File→Load Program… every time you rebuild your project.

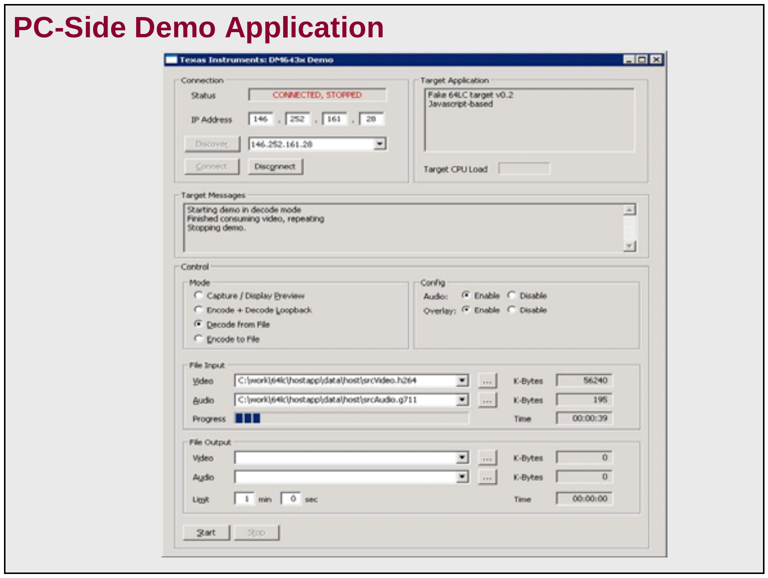
12. **Execute application on the DVDP**

    Debug→Run (or F5 key)

13. **Start DVD Player or other Video Input source**

    You should see a Loopthru of the video into your display. You will not hear audio at this time.

Run the Host Application



14. Using Windows Explorer, navigate to the hostapp folder of Lab 1

C:\dm6437_1day\lab1_demo\hostapp

15. Execute the "run.bat" batch file

Note: This script contains a full-path reference to C:\dvsdk_1_01_00_15\xdc_2_95_02\xs.exe This is correct for the in-class environment, but may need to be modified to match the installation path on other systems.

You may simply click on the file, or right-click and select Open. It should not be necessary to debug this script, but for future reference, scripts may be run from the command line by opening a command window:

start→Run... and type in "cmd" or "command"

Within a command window, you can use standard DOS commands, i.e.

dir = list contents of current directory

cd <directory> = change to the specified directory

<filename> = execute the given file, such as run.bat

16. Connect to DVDP via the Host Application

The run.bat script should bring up the Host Application window. In the top left corner of this window is the connection section. Enter the IP address of the DVDP into the IP Address window. The static IP address of the board will be whatever you specified for the ndk.staticIpAddr field in step 7 of the "Build/Load/Run the DVDP Application" section. (The instructions recommend 192.168.1.41) If this was configured for dynamic operation, then you may either consult the standard output window in CCS, where the application will print the board's IP address, or you may use the discover feature of the host application.

After you have entered the IP Address, press the Connect button. If you are properly connected, the Status field should diplay "CONNECTED, PLAYING" and you should see a welcome message in the Target Messages window.

17. **Press the Stop button to halt the current demo**

    The stop button is located at the bottom of the Host Application window.

18. **Enable audio in the demo**

    Within the config section of the window is a checkbox for “Enable Audio”

19. **Select one of the four modes of the demo**

    Within the Mode section of the window are four modes. If you have selected either the “Decode from File” or “Encode to File” modes of operation, specify the appropriate Audio and Video files. These files are stored on the harddrive of the Host PC and are transmitted from the DVDP board to the Host PC via Ethernet.

20. **Press the Play button**

21. **Observe Statistics for Operation**

    In the top left corner of the Host Application window is the Target Application section. Select various tabs for given information as well as the real-time Target CPU load and Network Usage.

22. **When you finish, press stop and repeat steps 19-21 for other modes of operation.**

I/O and Drivers

Introduction

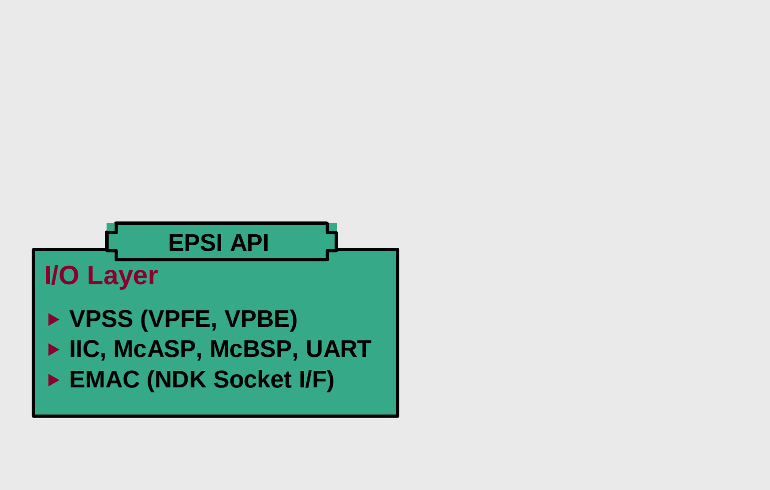
DILBERT: © Scott Adams/Dist. By United Feature Syndicate, Inc.

Data input and output is an integral aspect of any embedded system. Drivers provide a simple, modular interface for application developers to access the various peripherals of a given processor, and by compartmentalizing the work of data I/O into drivers, system developers can increase the portability and reliability of their embedded products.

And fortunately, both the efficient Linux and DSP/BIOS based drivers available for the DM6437 will never be scapegoats for system performance, as in our cartoon above!

Module Topics

TMS320DM643x Software Overview

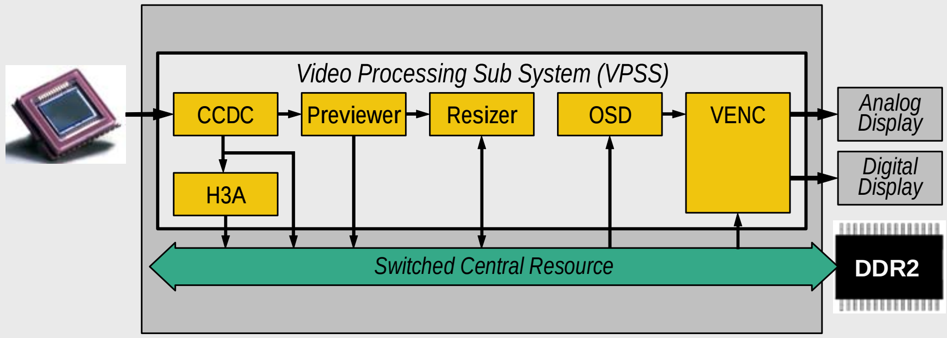


Available Peripheral Drivers

	DSP/BIOS PSP	Virtuallogix Linux
AIC33 Audio	✓	✓
TVP5146 Video Capture	✓	✓
VPBE Video Display	✓	✓
VPSS Resizer	✓	✗
VPSS H3A	✓	✗
IIC	✓	✓
UART	✓	✓
McASP	✓	(In AIC33 Audio)
McBSP	✓	✓
EDMA	(Low Level)	✓
EMAC	(NDK)	(Network Stack)
PCI	(Low Level)	✗
SPI	✗	✗
CAN Bus	✗	✗

The Video Processing Sub System

Video Processing Sub System (VPSS)



Video Processing Sub System (VPSS) is comprised of two blocks:

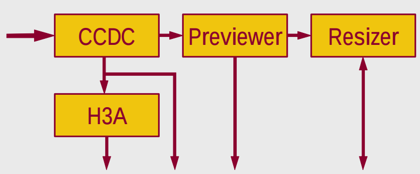
Front End (VPFE)

- **CCD Controller (CCDC)**
- **Statistics Engine (H3A)**
- **Previewer**
- **Resizer**

Back End (VPBE)

- **On-screen Display (OSD)**
- **Video Encoding (VENC)**

VPFE : 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

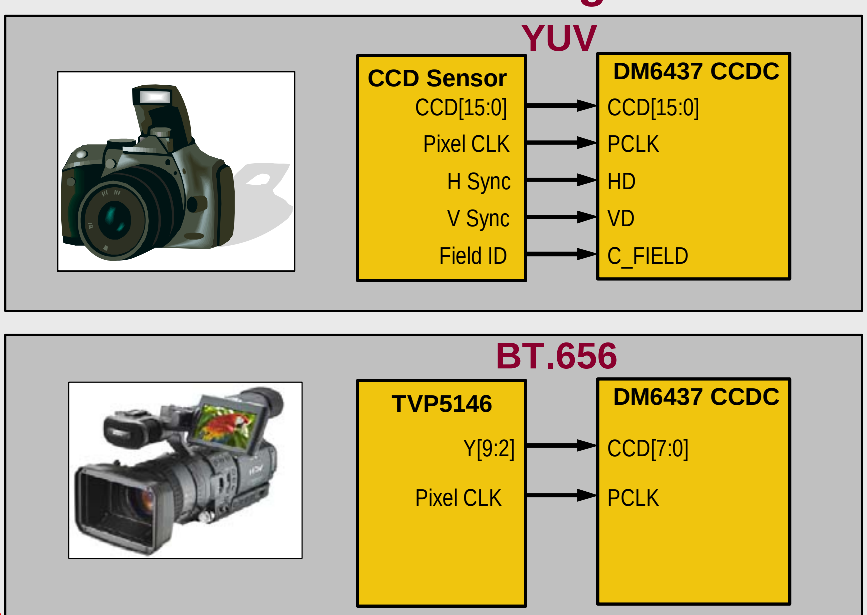
Resizer

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

Video Processing Sub-System (VPSS)

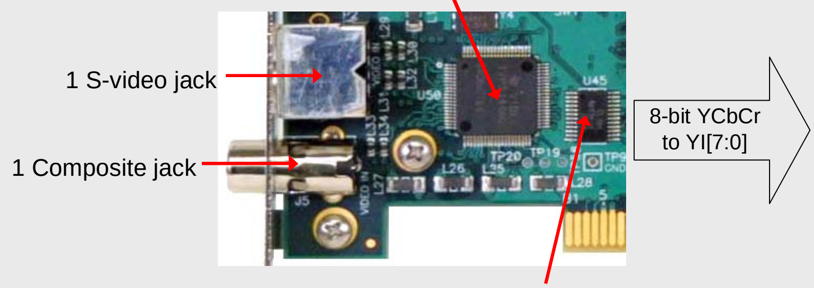
- Provides Interface to Input/Output Video
- Video Processing Front End (VPFE)
  - CCD Controller (CCDC)
  - Previewer
  - Resizer
  - Histogram/H3A
- Video Processing Back End (VPBE)
  - On Screen Display (OSD)
  - Video Encoder (VENC)
  - DAC
- VPSS Improvements Over DM644x
  - Some OSD register bits have changed location
  - Included some bug fixes from DM644x
  - New features (TBD). Stay tuned
- VPSS Reference Guides
  - VPSS FE (SPRU977)
  - VPSS BE (SPRU952)

CCDC Interface Diagrams



Video In

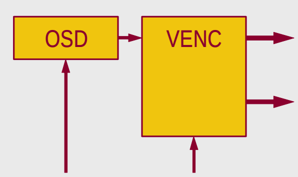
TI TVP5146M2 video decoder sources the 27MHz video clock



CI[7:0]available on header DC_P1 when CI_EMA_ENABLEn is pulled high on the daughtercard

Switch disconnects decoder from VPFE interface to allows use of VPFE on header DC_P1

VPFE : Resizer, Previewer, H3A



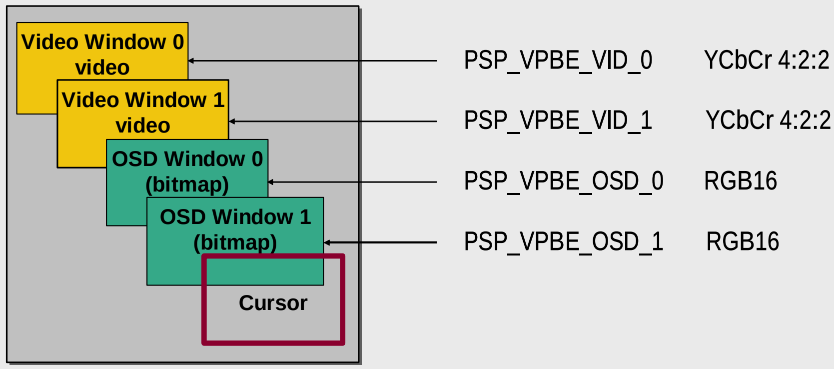
On Screen Display

- Four video/graphics planes
- Alpha Blending
- Automatic Focus Control
- Automatic White Balance Correction
- Automatic Exposure Compensation

Video Encoder

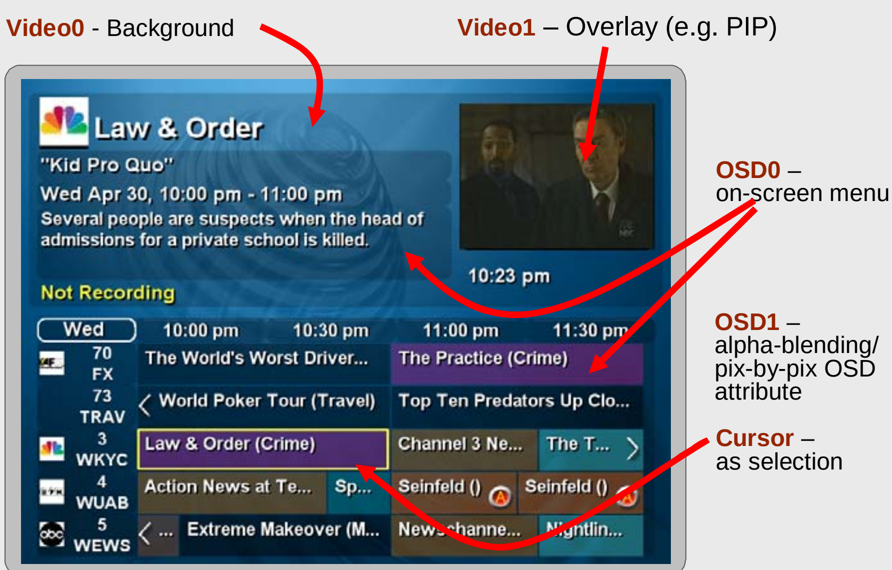
- Analog output:
  - Four 54Mhz 10-bit DACS
  - Composite NTSC/PAL video
  - S-Video
  - Component (YPbPr) or RGB
- Digital Ouputs:
  - Max Pixel clock rate: 75 Mhz (720p and 1080i use 74.25 Mhz)
  - 8/16 bit YUV
  - up to 24-bit RGB
  - BT.656 for NTSC/PAL

VPBE On-Screen Display (OSD)

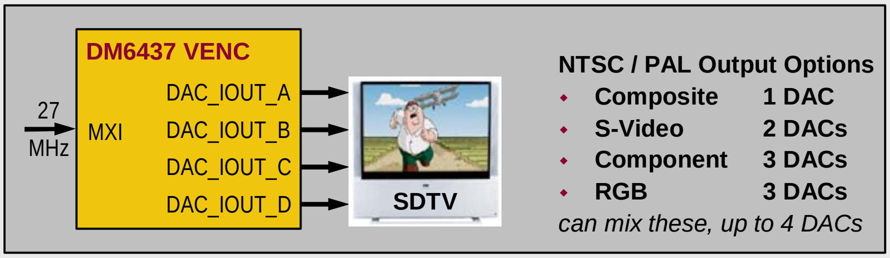


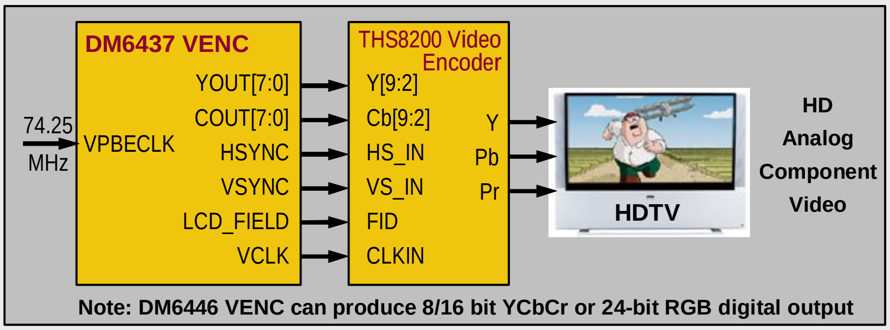
- **Two video windows for picture-in-picture**
- **Two OSD windows or one OSD window + attribute window**
- **OSD attribute mode provides pixel-level alpha blending**

Set-Top Box OSD Usage

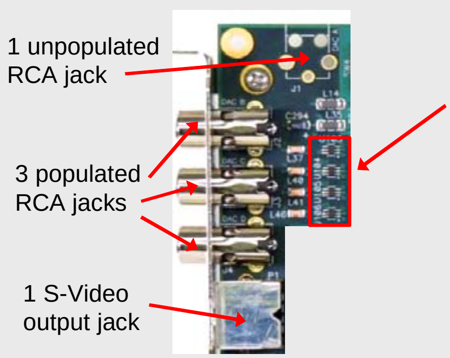


VENC : Display Options

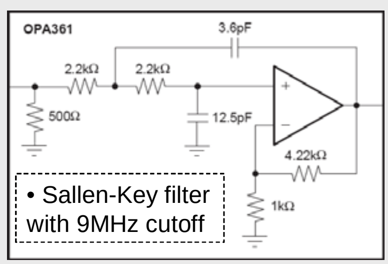




Video Output



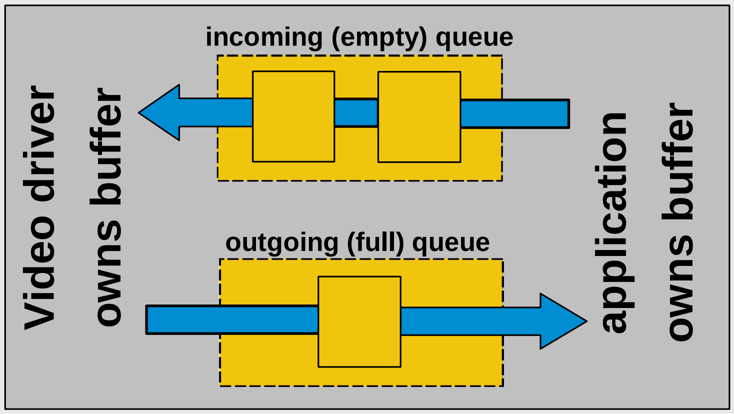
4 OPA361s with Integrated Op-amps and Reconstruction Filters



- Supports Composite, S-Video, and Component SDTV Output
- S-Video muxed with RCA jacks on DACs B and C
- Digital VPBE provided on header DC_P1

The FVID Video Driver (DSP/BIOS)

IOM Queue Structure



- Application takes ownership of a full video buffer from the outgoing driver queue using FVID_dequeue
- After using the buffer, application returns ownership of the buffer to the driver using FVID_queue
- FVID_exchange is simultaneous queue and dequeue

FVID Buffer Passing

```
status = FVID_queue(hStream, bufp);

status = FVID_dequeue(hStream, &bufp);
```

- *FVID_queue()* returns an empty buffer to the driver to be filled (input driver) or passes a full buffer to the driver to be displayed (output driver)
- *FVID_dequeue()* aquires a full buffer from the driver (input driver) or acquires an empty buffer from the driver for app to fill (output driver).
  - if no buffers are already available in the stream, this call can *block* until a buffer is made available from the driver
  - pBuf is passed by reference for FVID_dequeue to modify with the address of the return buffer

```
status = FVID_exchange(hStream, &bufp);
// is equivalent to:
// status = FVID_queue(hStream, bufp);
// status |= FVID_dequeue(hStream, &bufp);
```

FVID APIs

C

`gioChan = FVID_create(name, mode, status, optArgs, attrs)`
opens a new video channel and returns its handle through gioChan

`status = FVID_alloc(gioChan, &bufp)`
driver allocates a video buffer and returns a pointer through bufp

`status = FVID_control(gioChan, cmd, args)`
passes various control comands to driver through cmd, with argument list args

P

`status = FVID_queue(gioChan, bufp)`
grants driver access to bufp buffer by placing on incoming queue

`status = FVID_dequeue(gioChan, &bufp)`
grants application access to bufp-returned buffer by removing from outgoing queue

`status = FVID_exchange(gioChan, &bufp)`
performs simultaneous queue (bufp initial value) and dequeue (bufp return value)

`status = FVID_free(gioChan, bufp)`
frees the driver-allocated buffer pointed to by bufp (see FVID_alloc)

D

`status = FVID_delete(giochan)`
frees the video chanel specified by the giochan handle (see FVID_create)

FBVID Driver Example for VPFE

```
Void taskFunction(...)
{
    viHandle = FVID_create(...);            // Create and Initialize chan
    FVID_control(viHandle, ...);

    FVID_alloc(viHandle, &bufPtr);
    FVID_queue(viHandle, bufPtr);           // Double-buffered system
    FVID_alloc(viHandle, &bufPtr);

    while ('condition'){
        FVID_exchange(viHandle, &bufPtr);   // Exchange old buf for new

        /* Process */                       // Actually do something w/ input video buf
    }

    FVID_free(viHandle, bufPtr);
    FVID_dequeue(viHandle, bufPtr);         // Free both buffers
    FVID_free(viHandle, bufPtr);

    FVID_delete(viHandle);                  // Delete chan
}
```

Create and Initialize chan

Double-buffered system

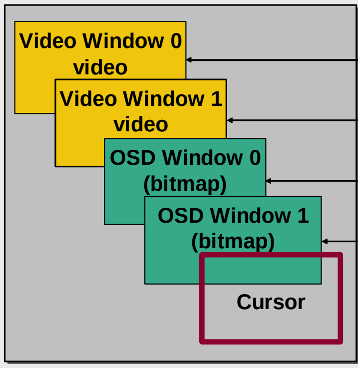
Exchange old buf for new

Actually do something w/ input video buf

Free both buffers

Delete chan

Video Port Back End Features

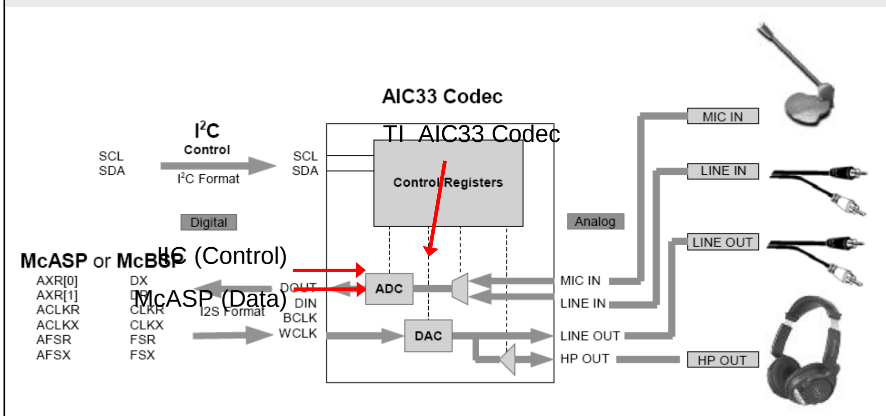


PSP_VPBE_VID_0	YCbCr 4:2:2
PSP_VPBE_VID_1	YCbCr 4:2:2
PSP_VPBE_OSD_0	RGB16
PSP_VPBE_OSD_1	RGB16

Each video/osd plane in the VPBE is opened spearately and then referenced by its own unique handle:

```
vpbeParams.id = PSP_VPBE_VID_0;
...
vo0Handle = FVID_create("/VPBE0", IOM_INOUT, NULL,
   &vpbeParams, NULL);
```

The Audio Serial Port



Optical and Coax S/PDIF output available when AIC33 not in use

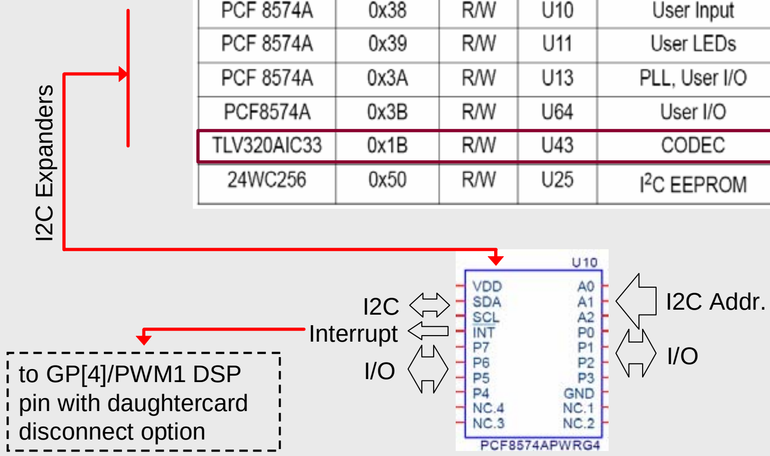
Can be disconnected from DSP via FET switch for daughtercard use of McASP and McBSP1

Multichannel Audio Serial Port (McASP)

- Serial Audio Interface
  - Transmit and receive audio to/from a DAC/ADC
    - Time-Division Multiplexed (TDM) stream
    - Inter-Integrated Sound (I^2S) stream
    - Transmit in Sony/Philips Digital Interface (S/PDIF) format
  - Up to 4 data pins to allow parallel audio streams
- Multiplexed with McBSP
- McASP Reference Guide (SPRU980)

I2C

Device	Address	R/W	Device	Function
TVP5146M2	0x5D	R/W	U50	Video Decoder
PCF 8574A	0x38	R/W	U10	User Input
PCF 8574A	0x39	R/W	U11	User LEDs
PCF 8574A	0x3A	R/W	U13	PLL, User I/O
PCF8574A	0x3B	R/W	U64	User I/O
TLV320AIC33	0x1B	R/W	U43	CODEC
24WC256	0x50	R/W	U25	I^2C EEPROM



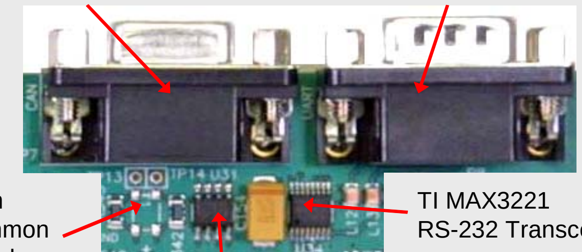
Inter-Integrated Circuit (I^2C)

- **Serial Control Interface**
  - **Used to configure Audio/Video Codecs**
  - **Communicate with other I^2C devices**
- **Features**
  - **Frequency up to 400kHz (I^2C Fast Mode)**
  - **Master/Slave**
- **Improvements over DM644x I^2C**
  - **Slew-Rate Limited Open-Drain Output Buffer meet Philips I2C Specification Revision 2.1 rise/fall time requirements**
    - **Note: Like DM644x, no fail-safe I/O buffer**
- **I^2C Reference Guide (SPRU991)**

CAN & UART

CAN Female DB-9

UART Male DB-9



CAN Population options for Common Mode Choke and ESD Suppressor

TI MAX3221 RS-232 Transceiver

TI SN65HVD235 CAN Transceiver

CAN and UART Transceivers can be disconnected from the DSP via a FET switch to allow TIMER1 and UART1 use, respectively on a daughtercard

Universal Asynchronous Receiver/Transmiter (UART)

- **Miscellaneous Serial Interface**
  - **Microcontroller Interface**
    - **Can be configured to interface through MSP430 (low cost 16-bit microprocessor):**
      - **Keypad Interface**
      - **IR Control Interface**
      - **Switches/LEDs**
  - **Smart Card Interface**
  - **Debug terminal**
  - **Miscellaneous Control Interfaces**
- **Features (same as DM644x)**
  - **Follows the TL 16C550 Industry standard**
- **Pins: Hardware Flow Control on UART0 only**
- **UART User's Guide (SPRU997)**

High-End Controller Area Network Controller (HECC)

- Serial Interface for Automotive
  - Interface through CAN Transceiver to CAN Bus
- Features
  - Compliant with the Controller Area Network (CAN) Protocol, Version 2.0B
  - Bus speed up to 1 Mbps
  - Receive/Transmit
- HECC User's Guide (SPRU981)

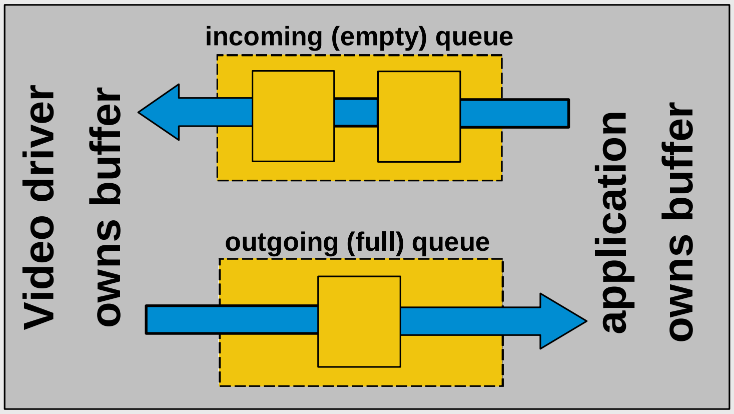
Serial Interfaces

	Audio Serial Port	I2C	SPI	UART
Features	- Full duplex - Double buffered - Standard IF for data converters, etc - 25 Mbps TX/RX - Supports IIS, SPDIF	- Compatible with Philips Spec Rev 1.1 - Fast Mode up to 400kHz - 7-bit and 10-bit Addressing - Master & Slave Functionality	- Master Mode operation - 2 chip-selects for interfacing to multiple slave SPI devices - 3 or 4 wire interface - 33 Mbps TX/RX	- Maximum Baud Rate of 128kHz - 16 Byte FIFO on Transmit and Receive - DMA FIFO Synchronization - Configurable Byte Size / Parity / Stop Bits
Driver Supports	AIC33 Audio Codec 8 – 48 kHz Sample Rates	I2C Master Mode	TBD	115,200bps / 8-bit char / no parity / 1 stop bit

DSP
TEXAS INSTRUMENTS TECHNOLOGY

The SIO Audio Driver (DSP/BIOS)

IOM Queue Structure



- Identical queueing mechanism to FVID driver
- Application takes ownership of a full video buffer from the outgoing driver queue using SIO_reclaim
- After using the buffer, application returns ownership of the buffer to the driver using SIO_issue

SIO Buffer Passing

```
status = SIO_issue(stream, bufp, nmadus, arg);

nmadus = SIO_reclaim(stream, &bufp, &arg);
```

- *SIO_issue* returns an empty buffer to the driver to be filled (input driver) or passes a full buffer to the driver to be displayed (output driver)
- *SIO_reclaim* aquires a full buffer from the driver (input driver) or acquires an empty buffer from the driver for app to fill (output driver).
  - if no buffers are already available in the stream, this call can *block* until a buffer is made available from the driver
  - pBuf is passed by reference for SIO_reclaim to modify with the address of the return buffer

```
nmadusOut = SIO_put(stream, &bufp, nmadusIn);
// is equivalent to:
// SIO_issue(stream, bufp, nmadusIn);
// nmadusOut = SIO_reclaim(stream, &bufp);
```

FVID APIs

C

```
stream = SIO_create(name, mode, bufsize, attrs)
```
opens a new audio channel and returns its handle through stream

```
nmadus = SIO_staticbuf(stream, &bufp)
```
driver allocates an audio buffer and returns a pointer through bufp, size is nmadus

```
status = SIO_ctrl(stream, cmd, arg)
```
passes various control comands to driver through cmd, with argument arg

P

```
status = SIO_issue(stream, bufp, nmadus, arg)
```
grants driver access to bufp buffer by placing on incoming queue

```
nmadus = SIO_reclaim(stream, &bufp, &arg)
```
grants application access to bufp-returned buffer by removing from outgoing queue

```
nmadus = SIO_put(stream, &bufp, nmadus)
```
performs simultaneous issue (bufp initial value) and reclaim (bufp return value)

```
status = SIO_idle(stream)
```
idle a stream – should be called before SIO_delete

D

```
status = SIO_delete(stream)
```
frees the audio chanel specified by the stream handle (see SIO_create)

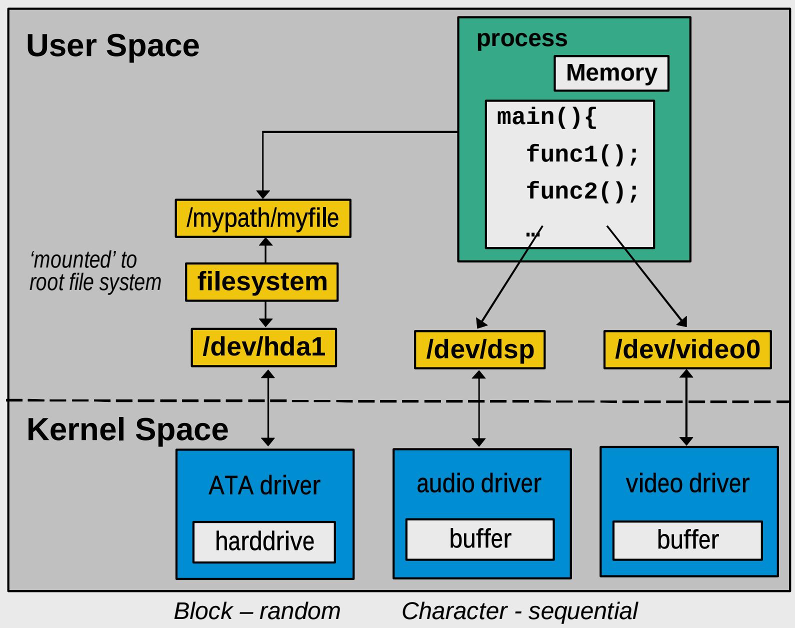
SIO Audio Driver Example

```
Void taskFunction(...)
{
    aiHandle = SIO_create(...);
    SIO_ctrl(aiHandle, ...);

    size = SIO_staticbuf(aiHandle, &bufPtr);
    SIO_issue(aiHandle, bufPtr, size, NULL);
    SIO_staticbuf(aiHandle, &bufPtr);

    while ('condition'){
        SIO_put(aiHandle, &bufPtr);

        /* Process */
    }

    SIO_reclaim(aiHandle, &bufPtr, NULL);
    SIO_idle(aiHandle);

    SIO_delete(aiHandle);
}
```

- Create and Initialize chan
- Double-buffered system
- Exchange old buf for new
- Actually do something w/ input video buf
- Reclaim and idle
- Delete chan

VirtualLogix Linux Drivers

User Access to Kernel Space



Using Character Device Drivers

Simple drivers use the same format as files:

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

len always in # of bytes

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

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

Notes:

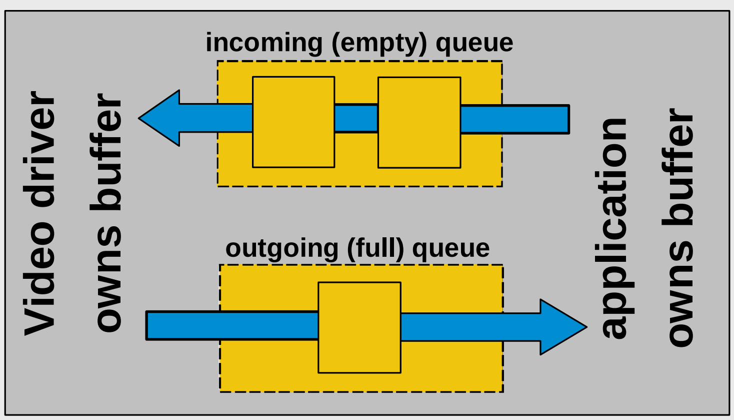
- *More complex drivers, such as V4L2 video driver, have special requirements and typically use ioctl commands to perfrom reads and writes*
- */dev/dsp refers to the "digital sound processing" driver, not the C64x+ DSP*

SIO Audio Driver Example

```
Void taskFunction(...)
{
    soundFd = open("/dev/dsp", O_RDWR);
    while ('condition'){
        read(soundFd, aMyBuf, length);
        /* Process */
        write(soundFd, aMyBuf, length);
    }
    close(soundFd);
}
```

Code	Description
soundFd = open("/dev/dsp", O_RDWR);	Open audio device
read(soundFd, aMyBuf, length);	Input audio buffer
/* Process */	Filter audio buffer
write(soundFd, aMyBuf, length);	Output filtered audio
close(soundFd);	Close audio device

v4l2 Driver Queue Structure



- Application takes ownership of a full video buffer from the outgoing driver queue using the VIDIOC_DQBUF ioctl
- After using the buffer, application returns ownership of the buffer to the driver by using VIDIOC_QBUF ioctl to place it on the incoming queue

FVID Buffer Passing

```
status = ioctl(v4l2Fd, VIDIOC_DQBUF, &bufp);

status = ioctl(v4l2Fd, VIDIOC_QBUF,  &bufp);
```

- *VIDIOC_QBUF* returns an empty buffer to the driver to be filled (input driver) or passes a full buffer to the driver to be displayed (output driver)
- VIDIOC_DQBUF aquires a full buffer from the driver (input driver) or acquires an empty buffer from the driver for app to fill (output driver).
  - if no buffers are already available in the stream, this call can *block* until a buffer is made available from the driver
  - pBuf is passed by reference for FVID_dequeue to modify with the address of the return buffer

```
Note: the v4l2 driver does not support an
      "exchange" function as do SIO and FVID
```

V4l2 video Driver Example

```
Void taskFunction(...)
{
    vidInFd = open("/dev/video0", O_RDWR);
    while ('condition'){
        ioctl(vidInFd, VIDIOC_DQBUF, &buf);
        /* Process */
        ioctl(vidInFd, VIDIOC_QBUF, &buf);
    }
    close(vidInFd);
}
```

- Open video device
- Acquire video buf
- Use video buffer
- Return video buffer to driver
- Free video device

For More Information

More info on DSP/BIOS Drivers

DM6437 DSP/BIOS PSP User's Manual
Included in PSP DOC folder in SDK installation

DSP/BIOS Driver Developer's Guide
Literature Number: SPRU616
http://focus.ti.com/lit/ug/spru616/spru616.pdf

User's Guides for each Peripheral:
Such as VPFE user's guide, Literature # SPRU977
http://focus.ti.com/lit/ug/spru977/spru977.pdf
See product folder at:
http://focus.ti.com/docs/prod/folders/print/tms320dm6437.html

TMS320C6000 DSP/BIOS 5.31 API Reference Guide
section 2.9 GIO module, section 2.26 SIO module
` **Literature Number: spru403n.pdf**
http://focus.ti.com/lit/ug/spru403n/spru403n.pdf

More info on VirtualLogix Linux Drivers

VirtualLogix™ VLX for Digital Multimedia v2.0 User Guide
Chapter 9: Using Linux Kernel Modules
http://www.virtuallogix.com/index.php?id=166

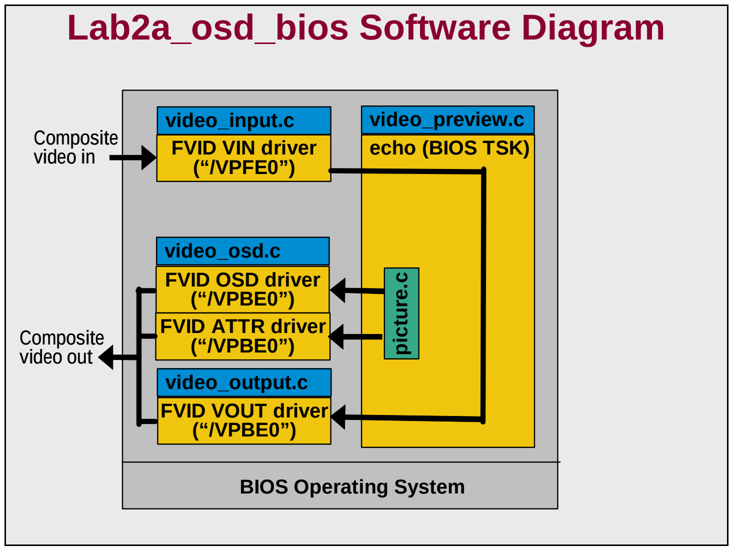
Video for linux two (v4l2) video driver online documentation
http://www.thedirks.org/v4l2/

Open sound system (OSS) audio driver online documentation
http://www.opensound.com/oss.html

Linux driver details:
Linux Device Drivers, Third Edition, **O'Reilly, Feb 2005**
ISBN: 0-596-00590-3
http://www.oreilly.com/catalog/linuxdrive3/

Lab 2a (DSP/BIOS version)

Important: Labs 2-4 of this workshop are presented in two versions, one utilizing DSP/BIOS only (Lab 2a) for the operating system, and one utilizing VirtualLogix Linux and DSP/BIOS executing concurrently (Lab 2b). You will only have time to complete one version, so please choose the lab version appropriate for your system.



In this lab, you will explore the On-Screen Display (OSD) capabilities of the Video Port Back End (VPBE) of the Video Port Sub-system (VPSS). The provided video loopback application not only captures and displays live video but overlays a custom-drawn picture (picture.c) using the DM6437 Video Port Back End On-Screen Display hardware.

Create a Custom Banner

1. **Start Code Composer Studio using the Desktop Shorrtcut**

Note: you will need to make sure that the DM6437 dsk is connected (via USB emulation cable) and powered on.

2. **Load the video_preview.pjt project from the lab2a_osd_bios directory**

Project→open…

select video_preview.pjt from the C:\dm6437_1day\lab2_osd folder.

3. **Open the picture.bmp file**

Expand the Documents folder in the project file tree and double click on the picture.bmp file. Note that the file will be opened with the Windows default editor for the file type. In your lab setup, this may be the windows picture viewer. If so, you can right click on the banner image and select “Edit” or “Open With…” from the drop down menu.

To change the editor that the file is opened with, you will need to change the default “Open with…” option for the .bmp file type in Windows. In XP this can be done by locating the file in Windows Explorer, right clicking, selecting “Open with…” and checking the “Always use this program to open this file type” option.

4. **Redraw the picture.bmp to create a custom banner**

5. **Save and close picture.bmp**

Build/Load/Run the project

6. Examine the custom build options for picture.bmp

Code Composer Studio natively knows how to build C, assembly and RTSC (more on this later) source files into a project. However, CCS does not have a native build option for bitmap source files, so we have to create a custom build option for this file.

To view the custom build option for the file, right click on picture.bmp and select “File Specific Options…”

Notice that the “Use custom build step” box has been checked, and below this checkbox, there are boxes to specify the build command, the output file and a clean command. Note that the specified output (picture.c) appears under the “Generated Files” of the project source tree and that we do not need to specify custom build options for this file because CCS natively knows how to build C source.

Examination of the Build Command
%XDC_INSTALL_DIR%\tconf.exe bmptoc.js picture.bmp picture.c gBanner
Shows that we are using a javascript script (bmptoc.js) to convert the bitmap image to a C source file. Examination of picture.c shows that this script converts the bitmapped image into a global array named gBanner and specifies the width and height of the array with global variables gBannerWidth and gBannerHeight. The command-line options specified to this script are picture.bmp (the input file), picture.c (the output file) and gBanner (the name of the global array that is generated.)

javascript was chosen to implement this utility because it is portable across most platforms and because a Javascript Virtual Machine (JVM) and various file manipulation libraries are included with the XDC tool installation (more on the XDC tool in module 3). The source code for bmptoc.js is included in this project for your reference.

7. Connect the DM6437 dsk

Debug→Connect (Alt-C)

8. Build the project

Project→Rebuild All

(You will get a number of warnings/remarks in the build output, but should get no build errors.)

9. Load the video_preview.out executable onto the DM6437 DSK

File→Load Program… (Ctrl-L)

select video_preview.out from the Debug subfolder of the C:\dm6437_1day\lab2_osd directory.

10. Run the executable

Debug→Run (F5)

CCS Debugging Tools

11. Halt execution on the dsk

Debug→Halt (Shift-F5)

12. Set breakpoint within main video loop

Open video_preview.c, locate the echo() function and scroll until you find the following code section:

```
/* loop forever performing video capture and display */
while (!done && status == 0) {

  /* grab a fresh video input frame */
  FVID_exchange(hGioVpfeCcdc, &frameBuffPtr);

  /* display the video frame */
  FVID_exchange(hGioVpbeVid0, &frameBuffPtr);

}
```

This section of code acquires a pointer to an input buffer of video data using FVID_exchange with the hGioVpfeCcdc (Video port front end) device and then immediately passes this video data to the hGioVpbeVid0 (Video port back end) display device. It is located with a while loop that will continue looping until either the “done” or “status” variables are modified.

Set a breakpoint on either of the FVID_exchange calls by selecting the appropriate line in the file, right clicking, and selecting “Toggle software breakpoint”

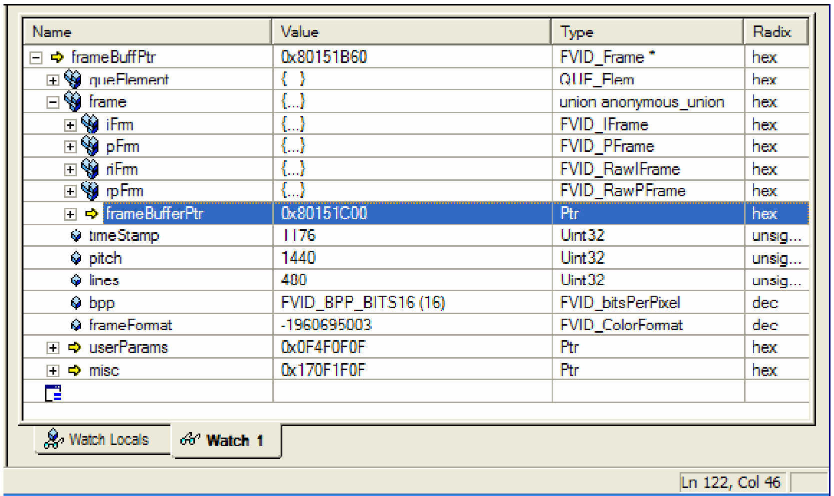
13. Run to the breakpoint that you set in step 12

The variables that we will use to specify the memory region we would like to graph are local variables. Therefore, if the program is not halted within the function that references these variables, they are out of scope and will not be recognized by the graphing utility.

Debug→Run

14. View the frame buffer structure pointed to by frameBuffPtr

Highlight the frameBuffPtr variable in the CCS editor, right click and select "Add to watch window" Expand the structure by clicking the plus sign to the left of the variable name.

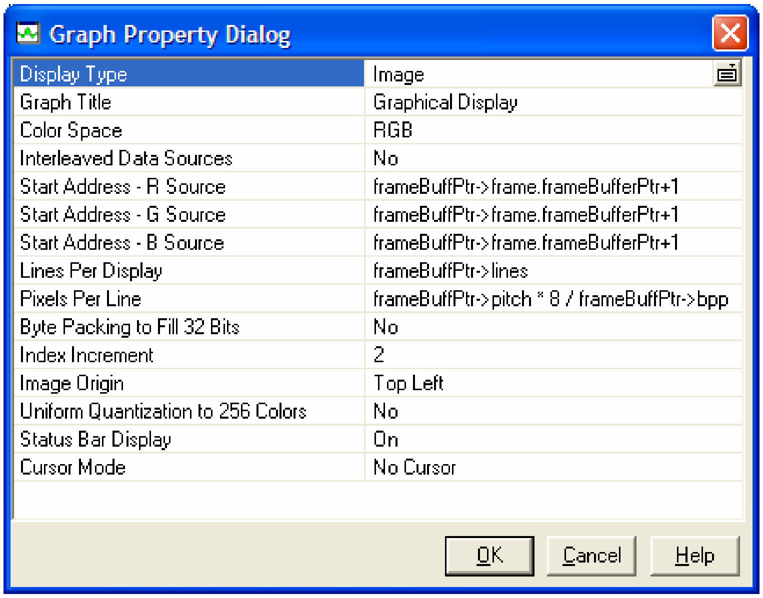


The frame buffer structure holds all of the information for a given video frame that is used by the driver, including the number of lines, the bits per pixel and the pitch (width times bits per pixel).

15. View display video buffer

View→Graph→Image…

Fill in the graph properties as follows



Display Type	Image
Graph Title	Graphical Display
Color Space	RGB
Interleaved Data Sources	No
Start Address - R Source	frameBuffPtr->frame.frameBufferPtr+1
Start Address - G Source	frameBuffPtr->frame.frameBufferPtr+1
Start Address - B Source	frameBuffPtr->frame.frameBufferPtr+1
Lines Per Display	frameBuffPtr->lines
Pixels Per Line	frameBuffPtr->pitch * 8 / frameBuffPtr->bpp
Byte Packing to Fill 32 Bits	No
Index Increment	2
Image Origin	Top Left
Uniform Quantization to 256 Colors	No
Status Bar Display	On
Cursor Mode	No Cursor

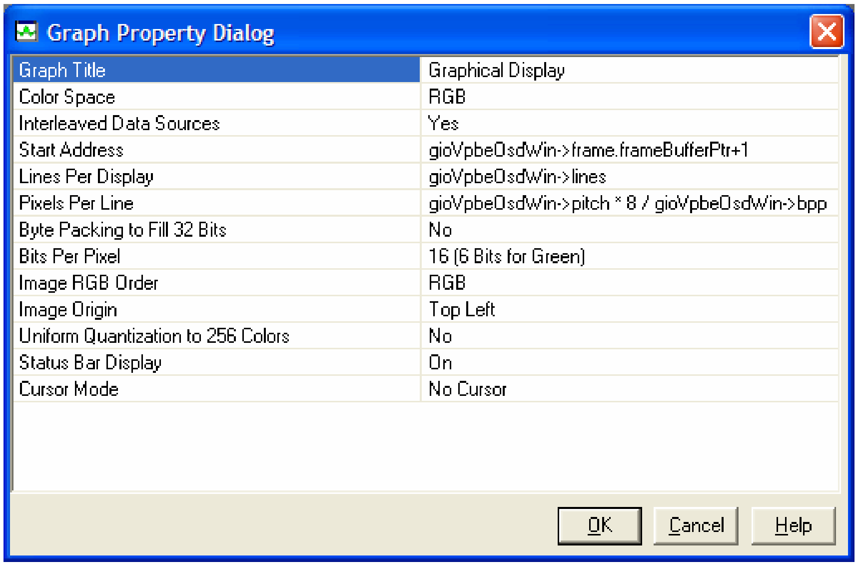
Note: We are filling in the graph to display the image in black and white format (this is why the same pointer is used for red, green and blue channels in the graph.) Although the data that we are graphing is actually a color image, it is stored in UYVY packed (interleaved) data format, where Y is luminance or intensity, which is the only data that we are using in the graph. U and V values represent red and blue chrominance (color) deviations from green.

The CCS graphing utility does have a YUV mode; however it does not currently support interleaved UYVY data (it expects the data to be planar with separate buffers for Y, U and V). This black and white graph using the RGB graphing utility is a work around. However, being able to view the image buffers directly from the device is a very useful debugging tool (even if only viewed in black and white), so we wanted to show you this technique.

16. View OSD video buffer

View→Graph→Image…

Fill in the graph properties as follows

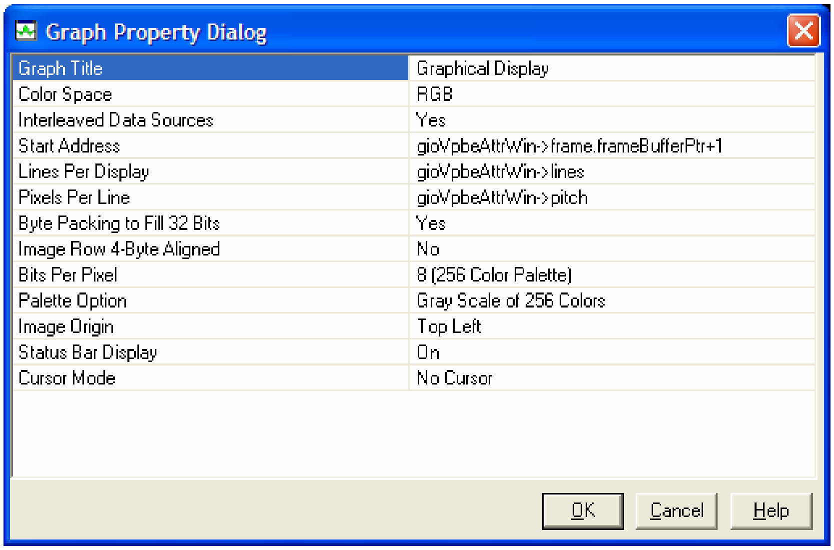


Note: the name of the frame buffer pointer, gioVpbeOsdWin, contains "Osd" standing for on-screen display. Be sure not to type in a zero instead of capitol O or you will get an "Identifier not found" error when you hit OK.

17. View alpha blending buffer

View→Graph→Image…

Fill in the graph properties as follows

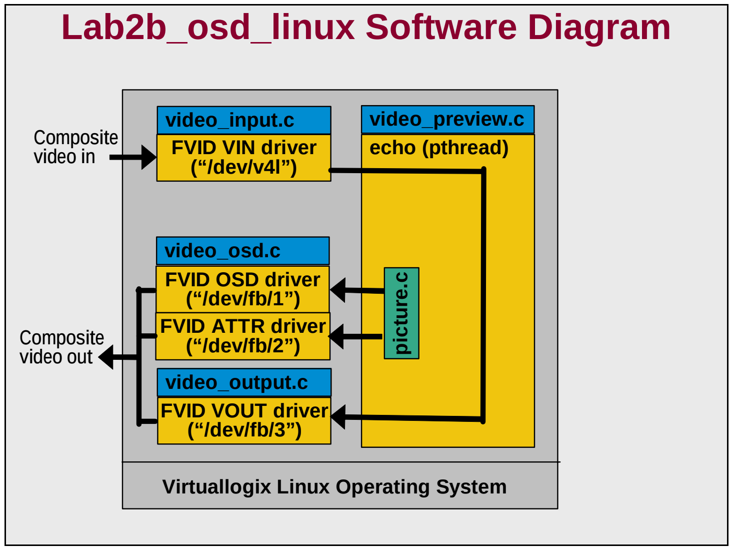


The graph will appear narrow due to the fact that we are concatenating pairs of 4-bit attribute pixels into an 8-bit value to be displayed. The shading you see is almost completely determined by the most significant pixel in the pair. For debugging purposes, this should be sufficient as the zones represented in the attribute window typically do not vary on a pixel-by-pixel basis.

You may notice a band of zero values along the right side of the graph. This is due to the fact that the attribute window must be extended beyond the 720 pixel display width in order to have a line width that is divisible by 32 bytes. This is a requirement of the driver and is purposefully done in order to make sure that each image line does not extend beyond the L1D cache line (which could cause unpredictable behaviour when caching is used.)

Lab 2b (VirtualLogix Linux version)

Important: Labs 2-4 of this workshop are presented in two versions, one utilizing DSP/BIOS only (Lab 2a) for the operating system, and one utilizing VirtualLogix Linux and DSP/BIOS executing concurrently (Lab 2b). You will only have time to complete one version, so please choose the lab version appropriate for your system.



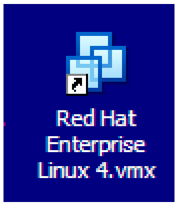
In this lab, you will explore the On-Screen Display (OSD) capabilities of the Video Port Back End (VPBE) of the Video Port Sub-system (VPSS). The provided video loopback application not only captures and displays live video but overlays a custom-drawn picture (picture.c) using the DM6437 Video Port Back End On-Screen Display hardware.

This lab is run completely within a VMware virtual machine. This tool allows us to run a Linux host operating system on our lab PC’s, even though the PC’s are actually running a Windows operating system. Currently VirtualLogix Linux build tools require a Linux host. Furthermore, debugging Linux applications is typically simpler from a Linux hosted environment.

Start the Linux Virtual Machine

1. **Start the Red Hat Enterprise Linux 4 Virtual Machine**

   There is a shortcut on your desktop that will bring up the Virtual Machine information page

   Select "Start this virtual machine" from the information page

2. **Log in with user permissions**

   There are two Linux accounts set up:

user:	user
password:	useruser
user:	root
password:	rootpw

3. **Open a terminal window**

   right-click in the desktop and select "Open Terminal"

Configure Ethernet Port in Virtual Machine

4. **Within the terminal you opened in step 3, use "su" command to switch to root permission**

   ```
   # su
   ```

   enter "rootpw" as the password when asked

5. **Use the following procedure to configure the ethernet port with static IP address 192.168.1.40**

   ```
   # /sbin/ifconfig eth0 down
   # /sbin/ifconfig eth0 192.168.1.40
   # /sbin/service nfs restart
   ```

6. **Exit out of root permission (to user permission)**

   ```
   # exit
   ```

Create a Custom Banner

We will introduce some basic linux terminal commands in this section. For those who are not familiar with linux terminal commands, you may wish to refer to the on-line manual for more information. To access the on-line manual page of a given command, use the “man” command followed by the name of the command you want more information on. For instance, to get more information on the “cd” command, type:

man cd

and to get more information on the man command itself, type

man man

(Note, here we are using the hash symbol ‘#’ to represent the command prompt. It does not need to by typed in.)

7. Change to the workshop/lab2_osd directory

Use the linux “cd” (change directory command)

cd workshop/lab2_osd

note: If the terminal gives you an error that there is no such directory, you have probably logged in as root instead of user. Log out by hitting Ctrl-Alt-Ins (not Ctrl-Alt-Del !) and selecting “Log Out” then log in with the user account as shown above

8. list the contents of the lab2_osd directory

Use the linux “ls” command

9. change to the osdfiles directory and list the contents

you will see the following files

bmptoc.js	a javascript script which converts a bitmap image to a global array in a C source file.
makefile	a gnu makefile which provides instructions to build the picture.c C source file from picture.bmp
pic.mak	referenced by makefile (This is done so that makefile can invoke pic.mak with the path variables as set in setpaths.sh)
picture.bmp	the bitmap file of a banner that will be placed in the video output of the final application using the DM6437 hardware osd capability
picture.c	picture.bmp translated into a data array for inclusion in the application

note: setpaths.sh is a shell script that is contained two directory levels up from the current directory. This shell script is used to set environment variables that define the location of various tools and libraries that are used in the lab exercises. No absolute paths are referenced in the lab exercises, but instead, these environment variables are used. When you install the lab exercises on your own host machine, you will need only to modify the paths set in setpaths.sh to specify the locations of your installed packages.

10. Use the gimp application to modify picture.bmp

gimp picture.bmp

When you are finished modifying the banner, you can save your new image with File→save or Ctrl-S

11. Change directory back to the top level of lab2_osd

cd ..

Rebuild and Install the Application

12. Change to the app directory and run gnu make specifying the “install” target

cd app

make install

Boot Linux on the DM6437 DVEVM

13. Start Tera Term Pro from the Windows Desktop

18. Start Code Composer Studio using the Windows Desktop Shortcut

Note: you will need to make sure that the DM6437 dsk is connected (via USB emulation cable) and powered on.

We will use Code Composer Studio to load the linux kernel onto the DVEVM and boot. Note, however, that the filesystem which Linux will use is a network share using the Network File Share (nfs) filesystem, and the shared path is located within the Linux environment of the virtual machine. When you ran the “make install” command in the previous section, the make utility not only rebuilt the lab2_av.out applction, but copied it to this shared directory so that it will be available to execute from the DVEVM board

19. Connect the DM6437 dsk

Debug→Connect (Alt-C)

20. Load the kernel, vlx and bootloader onto the DM6437 DSK

File→Load Program… (Ctrl-L)

Navigate to C:\dm6437_1day\lab2b_osd_linux

Load each of the three executables <u>in the following order</u>:

nkern.out	Virtuallogix vlx virtualizer
vmlinux.out	Linux kernel
bootloader.out	Bootloader program

note: the order is important because bootloader.out needs to be loaded last so that the correct entry point is set. The order of nkern and vmlinux don't actually matter.

14. Run the program

Debug→Run (F5)

You should see feedback as the Linux kernel boots, ending with a login prompt:

```
192.168.1.41 login:
```

If the kernel feedback gives an error before reaching the login prompt, ask your instructor for help.

15. Login as root user, no password

16. Change to the /opt/workshop directory

```
# cd /opt/workshop
```

17. Load the audio and video driver modules with the loadmodules.sh script

```
# ./loadmodules.sh
```

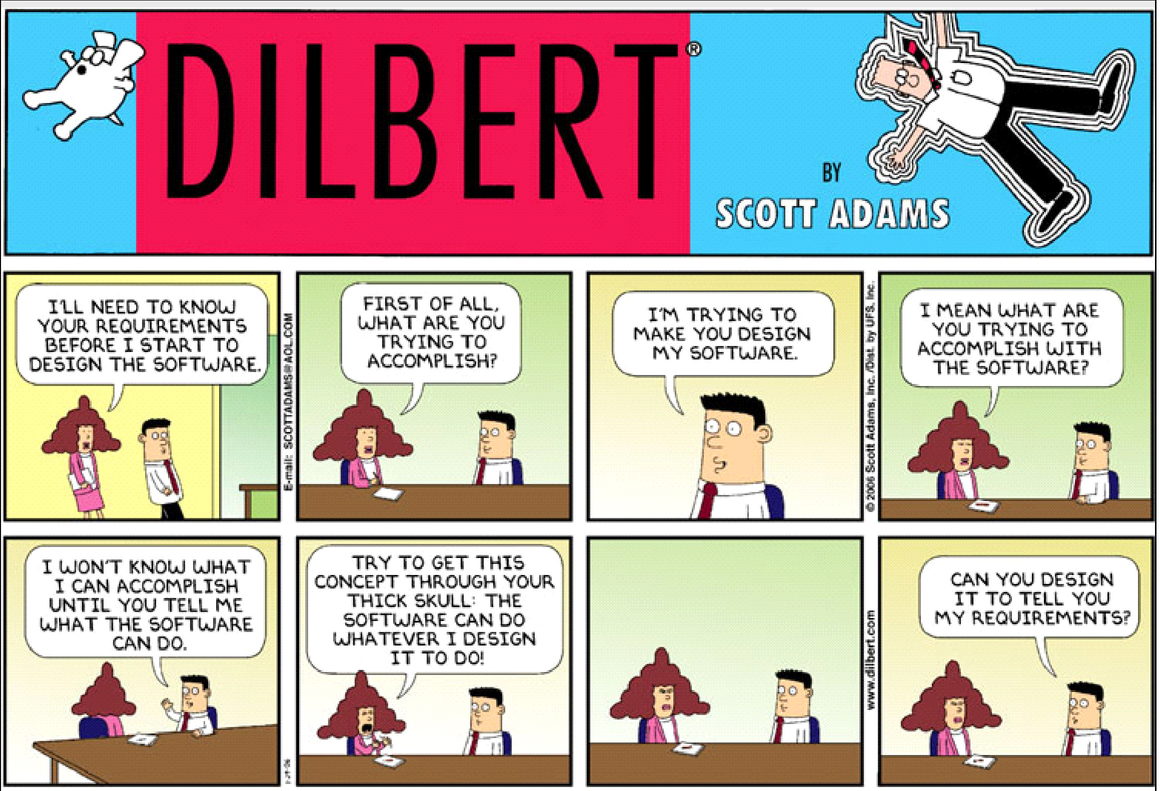
note: if you would like to see the contents of this script, type:

```
# cat loadmodules.sh
```

18. Execute the lab2_av.out application

```
# ./lab2_av.out
```

Introduction



Software design is a fluid process. A powerful and flexible framework is a great enabler in any software development, especially when requirements change suddenly or quick-turn prototyping is required. The Codec Engine is a production framework provided by Texas Instruments for use with the DaVinci series of processors. This framework provides a stable but agile base for you to build your applications upon.

All you need now is to decide what you want the Codec Engine to do for you!

Module Topics

Currently Available Codecs for DM643x:

Video
- H.263 (profile 0) D1 encode
- H.263 (profile 0) D1 decode
- H.264 BP D1 encode
- H.264 BP D1 decode
- H.264 MP@level 3 D1 Decode
- MPEG2 MP@level 3 D1 Decode
- MPEG4 SP D1 Decode
- MPEG4 SP D1 Encode
- VC1 (includes MP) Decode

Image
- JPEG Encode (baseline profile)
- JPEG Decode (baseline profile)

Audio
- AAC LC Encode
- AAC LC Decode
- AAC HE Decode
- MP3 Decode
- AC3 Decode
- WMA9 Decode
- WMA8 Encode

Voice
- G.711
- G.726
- G.729AB
- G.723.1
- G.722.2

DM6437 Video Capabilities

STANDALONE CODECS	Resolution	DSP Utilization in MHz (DM6437 @ 600MHz)
FOR CONSUMER VIDEO		
H.263 Profile 0 Decode	CIF (<2 Mbps)	60 MHz
MPEG2 MP/ML Decode	D1 (<10 Mbps)	200 MHz
MPEG4 SP Decode	D1 (<10 Mbps)	193 MHz
H.264 BP Decode	D1 (<4 Mbps)	300 MHz
H.264 MP/L3 Decode	D1 (<4 Mbps)	470 MHz
MPEG4 SP Encode, High Speed	D1 (3-5 Mbps)	312 MHz
MPEG4 SP Encode, High Quality	D1 (2.5-5 Mbps)	340 MHz
H.264 BP Encode, High Speed	D1 (2.5-4 Mbps)	530 MHz
H.264 BP Encode, High Quality	D1 (2.5-4 Mbps)	562 MHz
ADDITIONAL FEATURES		
Capture/Display	1 Video In/1 Video Out	
Network Connectivity	10/100 EMAC	
Peripheral Integration	USB 2.0, ATA, DDR2, MMC/SD, OSD, 4 DACs	

1. **All performance data is for 30fps YUV 4:2:0 unless otherwise noted. Note that the performance will vary depending on efficiency of code and data stream used.**
2. **Resolution Information: D1 (720x480) / CIF (352x288)**
3. **SP = Simple Profile / MP= Main Profile / BP=Base Profile**
4. **Image quality and bit rate were not held constant across these measurements. Current numbers are based off of independent testing for consumer types of applications. These numbers describe our current status and are undergoing further optimizations.**

DM6437 Audio Capabilities

STANDALONE CODECS	Sample Rate	DSP Utilization in Peak MHz (DM6437 @ 600MHz)
AUDIO		
MP3 L1 Decode	44.1 kHz (384 kbps)	15 MHz
MP3 L2 Decode	44.1 kHz (192 kbps)	17 MHz
MP3 L3 Decode	44.1 kHz (128 kbps)	24 MHz
AC3 Decode	48 kHz (640 kbps)	45 MHz
eAAC+ LC Decode	48 kHz (128 kbps)	27 MHz
eAAC+ LTP Decode	44.1 kHz (128 kbps)	42 MHz
eAAC+ HEHQ Decode	44.1 kHz (64 kbps)	73 MHz
eAAC+ PS Decode	44.1 kHz (320 kbps)	71 MHz
AAC LC Decode	48 kHz (128 kbps)	20 MHz
AAC LC Encode	44.1 kHz (128 kbps)	38 MHz
SPEECH		
G.711 Decode	8 kHz (64 kbps)	0.22 MHz
G.711 Encode	8 kHz (64 kbps)	0.25 MHz

DM6446 Video Decoder Performance

Video Decoders	MHz Req'd for D1 30fps	Bit Rate	Quality	30 fps max resolution w/ VICP	
				Normal mode	Turbo Mode
H.264 baseline	300-350*	< 5 Mbps		D1	D1
H.264 main	350-450	< 5 Mbps		VGA	D1
MPEG-4	80-100	< 5 Mbps		720p	SXGA
H.263	80-100	< 5 Mbps		720p	SXGA
WMV9 -MP	200-260	< 5 Mbps		D1	720p/24
WMV9 -AP	300-360	< 5 Mbps		D1	D1
MPEG2	100-150	< 15 Mbps		720p	SXGA

*All benchmarks are preliminary and subject to change.

DM6446 Video Encoder Performance

Video Encoders	MHz Req'd for D1 30fps		D1 Bit Rate	Quality	VICP	
	with VICP	w/o VICP			Normal	Turbo
H.264 baseline	410*	760*	2 - 5 Mbps	1.5 db	D1 (w/IMCOP)	D1(w/IMCOP)
MPEG-4 (high-video quality)	250	350	3 - 8 Mbps	1.0 db	D1	720p/24
MPEG-4 (high-frame rate)	180	300	4-10 Mbps	1.0 db	720p/24	SXGA/24
H.263	250	350	3 - 8 Mbps	1.0 db	D1	720p/24
WMV9	350	500	3 - 8 Mbps	1.0 db	D1	D1
MPEG2	350	500	3 - 8 Mbps	1.0 db	D1	D1

Still Capture/Playback	Mpix/sec Normal (450MHz)		Mpix/sec Turbo (600MHz)		Max video w/ VICP (size/fps)	
	with VICP		with VICP		Normal	
TI Image Pipe*	18.5 - 19.1	8.7	24.7 - 25.5	11.6	VGA/60	720p/24
JPEG enc (standalone)	45 (est.)	32.1	60 (est.)	42.9	SXGA/30	1080i/24
JPEG dec (standalone)	TBD	30-45	TBD	40-60	720p/30	SXGA/30

*All benchmarks are preliminary and subject to change.

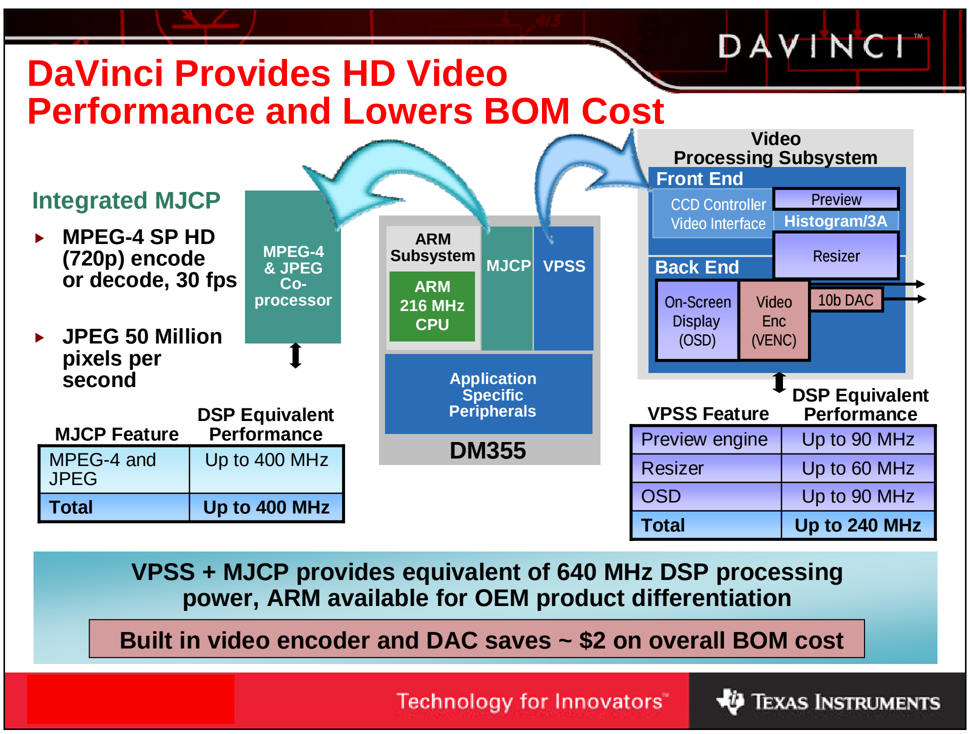
DM6446 Audio Codec Performance

Audio codecs	MHz Req'd (average)	MHz Req'd (peak)	Bit Rate	Comments
AAC-LC/Dec	9	11	128 Kbps	44.1 KHz – 128 Kbps (stereo)
AAC/+Dec (low power)	15	21	128 Kbps	44.1 KHz – 128 Kbps (stereo)
AAC/+Dec (high quality)	22	32	128 Kbps	44.1 KHz – 128 Kbps (stereo)
WMA9 dec (HBR)	7.94	15.09	320 Kbps	48 KHz – 320 Kbps (stereo)
WMA9 dec (MBR)	5.06	9.6	32 Kbps	44 KHz – 32 Kbps (stereo)
WMA9 dec (LBR)	7.45	23.5	20 Kbps	22 KHz – 20 Kbps (stereo)
WMA8 enc (32kbps)	20.08	32.47	32 Kbps	44 KHz – 32 Kbps (stereo)
WMA8 enc (48kbps)	14.78	33.95	48 Kbps	44 KHz – 48 Kbps (stereo)
WMA8 enc (80kbps)	15.36	35.82	80 Kbps	44 KHz – 80 Kbps (stereo)
MP3 dec	10	13	128 Kbps	
G.711	0.1	0.1	64 Kbps	
G.728	14.6	14.6	16 Kbps	

*All benchmarks are preliminary and subject to change.

DM647-720	Primary channels				Secondary Channels (simultaneous)			
	Density	Res.	Bitrate	fps	Density	Res.	Bitrate	fps
MPEG4 DVR High End	3–4 ch	D1	2–4 Mbps	30 fps	3–4 ch	QCIF	128kbps	30 fps
	10–12 ch	CIF	512–1024 kbps	30 fps	10–0 ch	QCIF	128kbps	30 fps
	12–14 ch	SIF/CIF	512–1024 kbps	30/25 fps	12–0 ch	QSIF	128kbps	30 fps
MPEG4 DVR Medium End	12–16 ch	QVGA	512–1024 kbps	30 fps	12–0 ch	QQVGA	128kbps	30 fps
MPEG4 DVR Low End	12 – 24 ch	CIF	512–1024 kbps	10–25 fps				
DM648-900	Primary channels				Secondary Channels (simultaneous)			
	Density	Res.	Bitrate	fps	Density	Res.	Bitrate	fps
MPEG4 DVR High End	4–5 ch	D1	2–4 Mbps	30fps	4–5 ch	QCIF	128kBps	30 fps
	12–16 ch	CIF	512–1024 kBps	30fps	12–0 ch	QCIF	128kBps	30 fps
	14–19 ch	SIF/CIF	512–1024 kBps	30/25fps	14–0 ch	QSIF	128kBps	30 fps
MPEG4 DVR Medium End	15–21 ch	QVGA	512–1024 kBps	30fps	15–0 ch	QQVGA	128kBps	30 fps
MPEG4 DVR Low End	16–32 ch	CIF	512–1024 kBps	10–25 fps				

DM647-720	Primary channels				Secondary Channels (simultaneous)			
	Density	Res.	Bitrate	fps	Density	Res.	Bitrate	fps
H.264 DVR High End	2 ch	D1	1–3 Mbps	30fps	2 ch	QCIF	128kBps	30 fps
	6–7 ch	CIF	384–784 kBps	30fps	4–0 ch	QCIF	128kBps	30 fps
	7–8 ch	SIF/CIF	384–784 kBps	30/25fps	4–0 ch	QSIF	128kBps	30 fps
H.264 DVR Medium End	7–8 ch	QVGA	384–784 kBps	30fps	7–2 ch	QQVGA	128kBps	30 fps
H.264 DVR Low End	7– 14 ch	CIF	384–784 kBps	10–25fps				
DM648-900	Primary channels				Secondary Channels (simultaneous)			
	Density	Res.	Bitrate	fps	Density	Res.	Bitrate	fps
H.264 DVR High End	2–3 ch	D1	1–3 Mbps	30 fps	8–0 ch	QCIF	128kBps	30 fps
	7–8 ch	CIF	384–784 kBps	30 fps	7–0 ch	QCIF	128kBps	30 fps
Best Regards,	8–9 ch	SIF/CIF	384–784 kBps	30/25 fps	8–0 ch	QSIF	128kBps	30 fps
H.264 DVR Medium End	9–10 ch	QVGA	384–784 kBps	30 fps	8–0 ch	QQVGA	128kBps	30 fps
H.264 DVR Low End	8–16 ch	CIF	384–784 kBps	10–25fps				

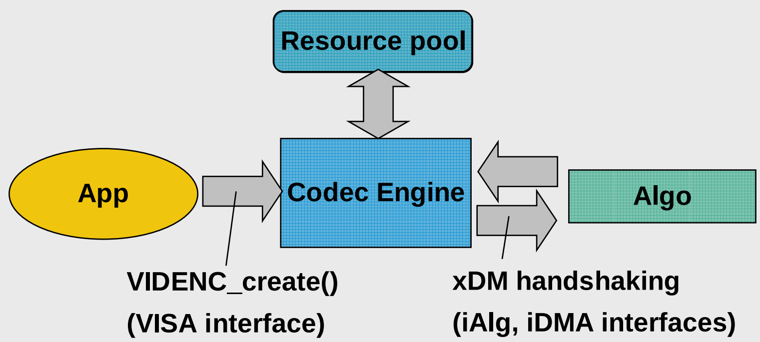


The Codec Engine Framework

The Codec Engine Framework

The codec engine provides a robust, consistent interface for:

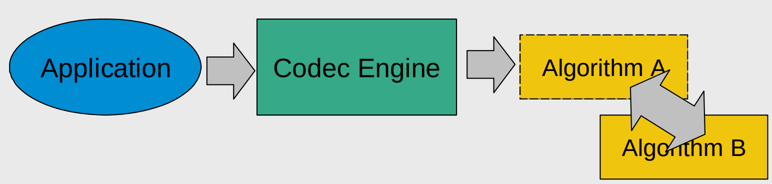
1. Dynamically creating and deleting algorithms
2. Accessing and controlling algorithm instances



Algorithm Access

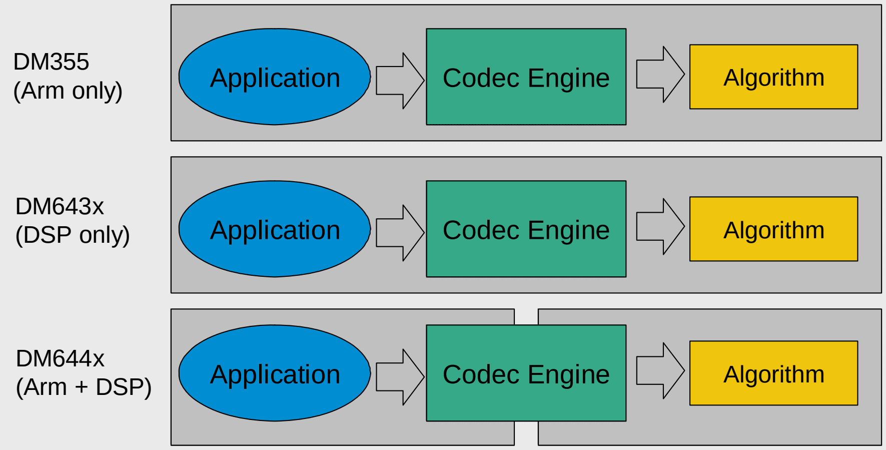
The Codec Engine provides standardized process and control calls for using algorithms it creates. This:

1. Allows algorithms of the same class to be easily exchanged without any modification to application code

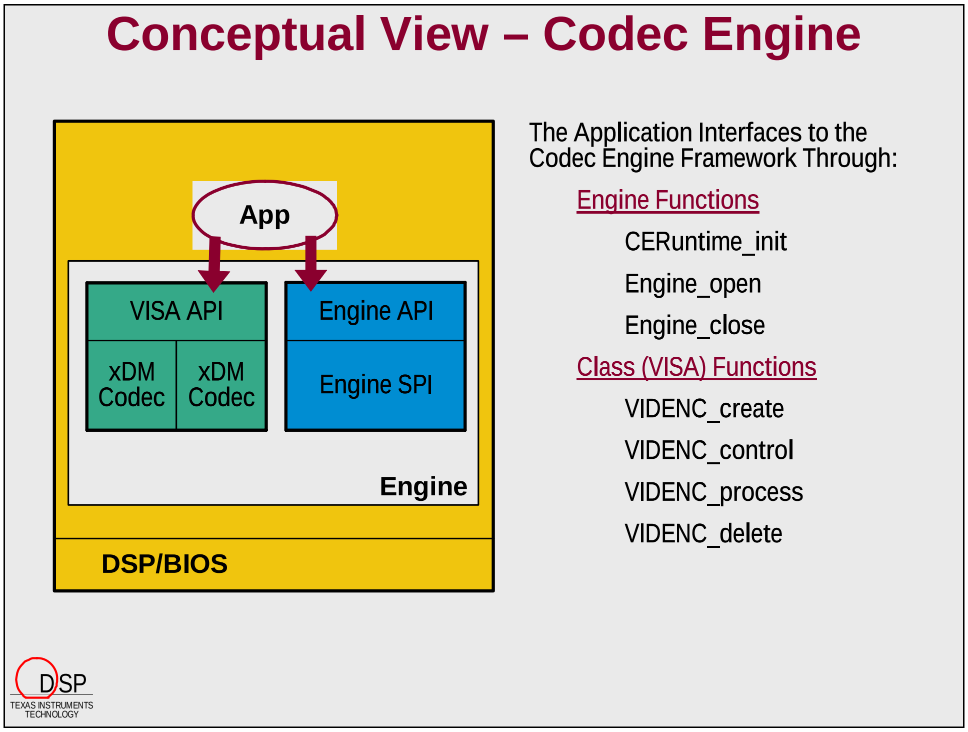


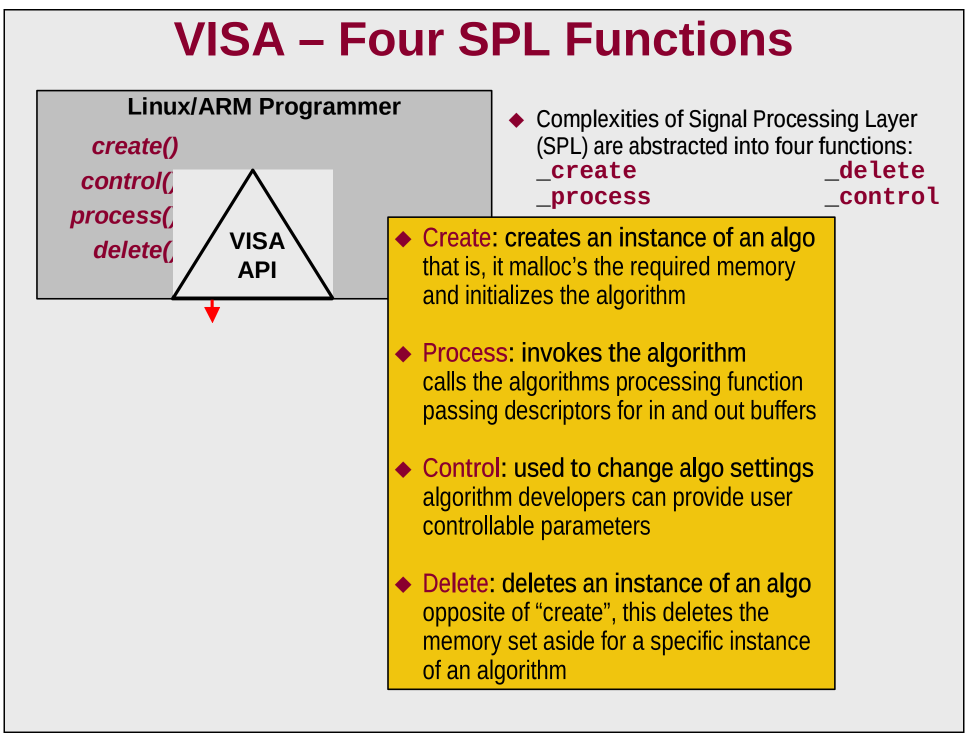
Algorithm Access

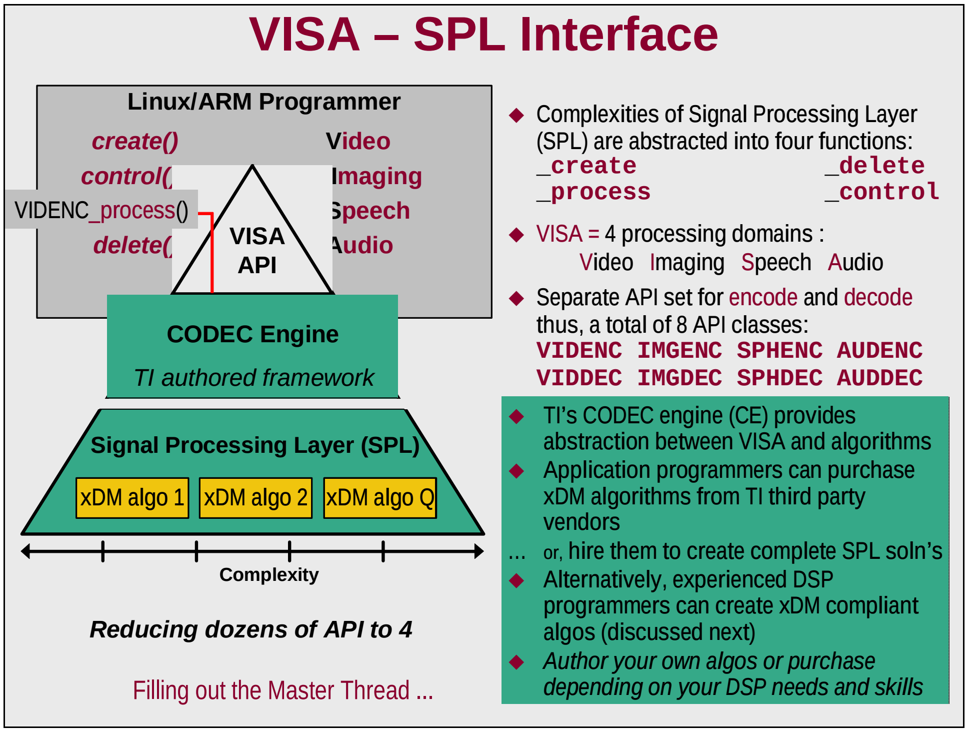
2. Allows the same application code to be used across a variety of platforms without modification

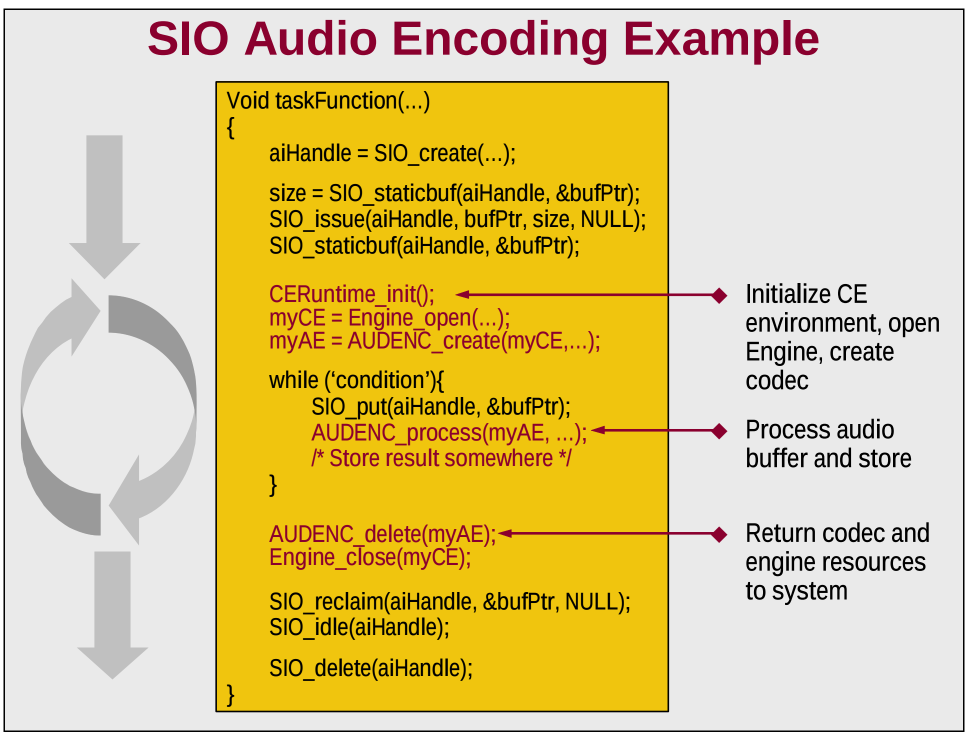


VISA API


Conceptual View – Codec Engine
App
VISA API
xDM Codec
xDM Codec
Engine API
Engine SPI
Engine
DSP/BIOS
The Application Interfaces to the Codec Engine Framework Through:
Engine Functions
CERuntime_init
Engine_open
Engine_close
Class (VISA) Functions
VIDENC_create
VIDENC_control
VIDENC_process
VIDENC_delete
DSP
TEXAS INSTRUMENTS TECHNOLOGY


VISA – Four SPL Functions
Linux/ARM Programmer
create()
control(
process(
delete(
VISA API
Complexities of Signal Processing Layer (SPL) are abstracted into four functions:
_create _delete
_process _control
Create: creates an instance of an algo that is, it malloc's the required memory and initializes the algorithm
Process: invokes the algorithm calls the algorithms processing function passing descriptors for in and out buffers
Control: used to change algo settings algorithm developers can provide user controllable parameters
Delete: deletes an instance of an algo opposite of "create", this deletes the memory set aside for a specific instance of an algorithm


VISA – SPL Interface
Linux/ARM Programmer
create()
control()
VIDENC_process()
delete()
Video
Imaging
Speech
Audio
VISA API
CODEC Engine
TI authored framework
Signal Processing Layer (SPL)
xDM algo 1
xDM algo 2
xDM algo Q
Complexity
Reducing dozens of API to 4
Filling out the Master Thread ...
Complexities of Signal Processing Layer (SPL) are abstracted into four functions:
_create _delete
_process _control
VISA = 4 processing domains :
Video Imaging Speech Audio
Separate API set for encode and decode thus, a total of 8 API classes:
VIDENC IMGENC SPHENC AUDENC
VIDDEC IMGDEC SPHDEC AUDDEC
TI's CODEC engine (CE) provides abstraction between VISA and algorithms
Application programmers can purchase xDM algorithms from TI third party vendors
... or, hire them to create complete SPL soln's
Alternatively, experienced DSP programmers can create xDM compliant algos (discussed next)
Author your own algos or purchase depending on your DSP needs and skills


SIO Audio Encoding Example
Void taskFunction(...)
{
aiHandle = SIO_create(...);
size = SIO_staticbuf(aiHandle, &bufPtr);
SIO_issue(aiHandle, bufPtr, size, NULL);
SIO_staticbuf(aiHandle, &bufPtr);
CERuntime_init();
myCE = Engine_open(...);
myAE = AUDENC_create(myCE,...);
while ('condition'){
SIO_put(aiHandle, &bufPtr);
AUDENC_process(myAE, ...);
/* Store result somewhere */
}
AUDENC_delete(myAE);
Engine_close(myCE);
SIO_reclaim(aiHandle, &bufPtr, NULL);
SIO_idle(aiHandle);
SIO_delete(aiHandle);
}
Initialize CE environment, open Engine, create codec
Process audio buffer and store
Return codec and engine resources to system

Codec Engine Details

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 DM644x users
- Provided by TI!

Many of these benefits are a direct result of the object-oriented structure of the codec engine

TTO
Technical Training Organization

Comparison of Functions and Algorithms

Why do I have to create an MPEG Algorithm?

A function:	An algorithm:
Is comprised of a set of instructions	Is comprised of one or more functions (methods) and a set of internally managed resources
Is used to modify output variables	Methods may modify output variables or internally managed resources
Using provided input variables.	Methods may utilize input variables or internally managed resources

In summary: an algorithm maintains internally managed resources which must be created an initialized for each algorithm instance.

xDM / C++ Comparison: Object

class (C++)	struct (C)
class algo{	typedef struct {
public:	
// methods	// methods
int method1(int param);	int (*method1) (int param);
int method2(int param);	int (*method2) (int param);
// attributes	// attributes
int attr1;	int attr1;
int attr2;	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.

xDM / C++ Comparison: Methods

Constructor

```
algo::algo(algo_params params)
VIDENC_create(VIDENC_params params)
```

Destructor

```
algo::~algo()
VIDENC_delete()
```

Generic Methods

```
algo::myMethod1(method_params params)
VIDENC_process(…)
VIDENC_control(…)
```

Note: with xDM, the CE Framework allocates resources on algorithm request, as opposed to a C++ constructor, which allocates its own resources.

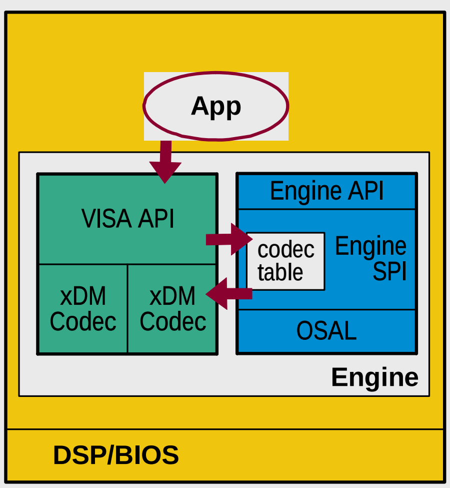
Algorithm Creation

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. **Conistent error handling when dynamic allocations have insufficient resources**

VISA Create and Delete

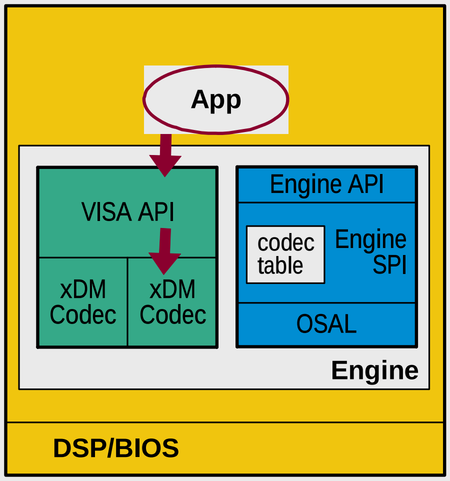


Create and Delete

- 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

DSP
TEXAS INSTRUMENTS TECHNOLOGY

VISA Control and Process



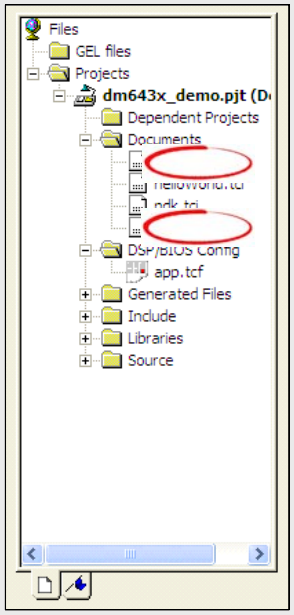
Control and Process

- 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

DSP
TEXAS INSTRUMENTS TECHNOLOGY

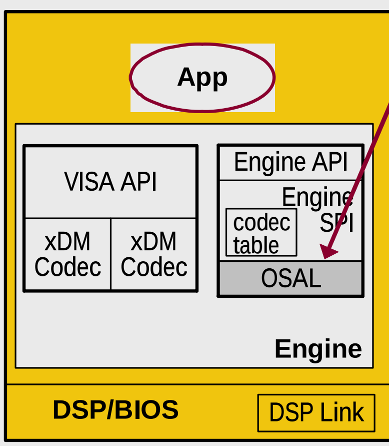
Configuring the Codec Engine

Framework Configuration



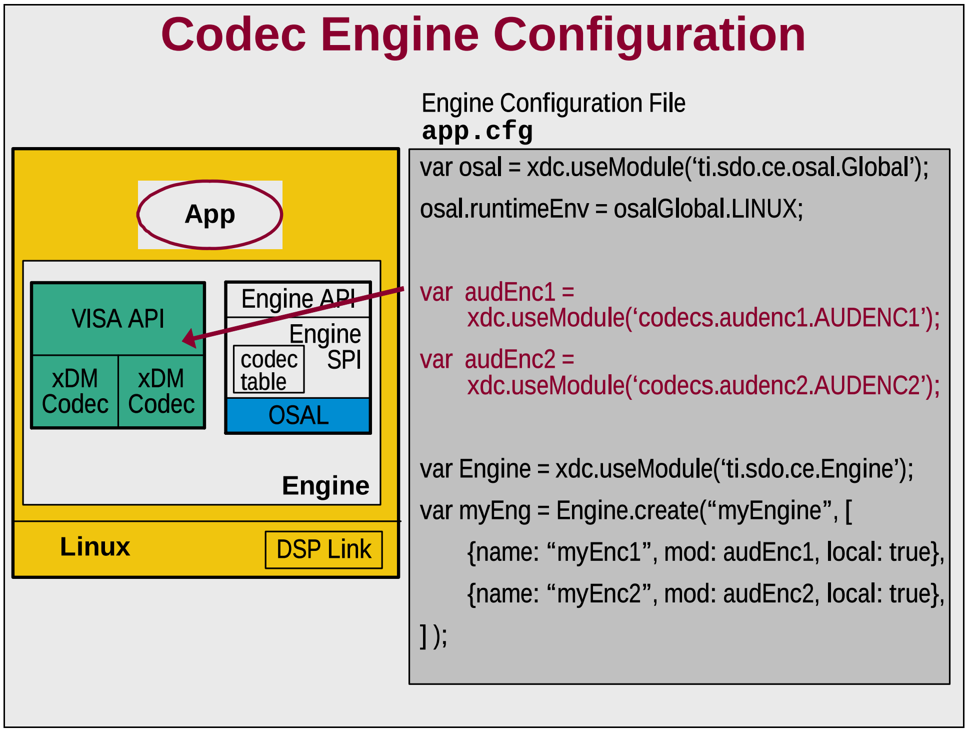
- Codec Engine, DSKT2 and DMAN3 are configured in app.cfg, similarly to DM6446 server builds.
- xdcpaths.dat provides repository search paths for all packages used in app.cfg
- app.tcf is a standard BIOS tconf file. It's name and path must match the .cfg file used in the project

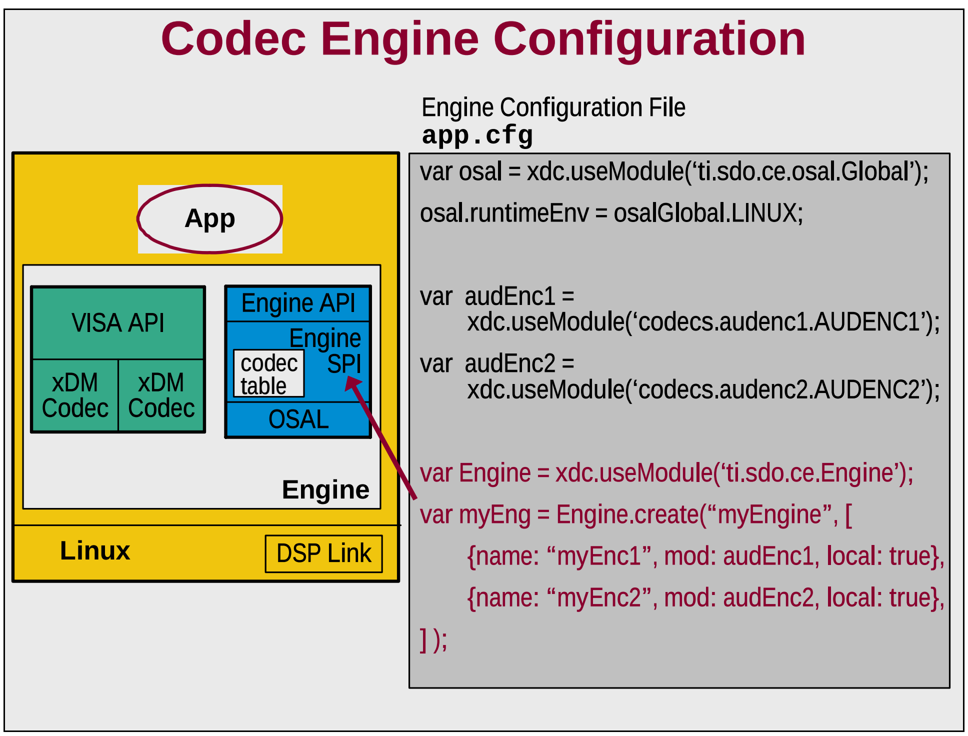
Codec Engine Configuration



Engine Configuration File
app.cfg

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

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: true},
       {name: "myEnc2", mod: audEnc2, local: true},
] );
```


Codec Engine Configuration
Engine Configuration File
app.cfg
var osal = xdc.useModule('ti.sdo.ce.osal.Global');
osal.runtimeEnv = osalGlobal.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: true},
{name: "myEnc2", mod: audEnc2, local: true},
]);
App
VISA API
xDM Codec
xDM Codec
Engine API
Engine SPI
codec table
OSAL
Engine
Linux
DSP Link


Codec Engine Configuration
Engine Configuration File
app.cfg
var osal = xdc.useModule('ti.sdo.ce.osal.Global');
osal.runtimeEnv = osalGlobal.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: true},
{name: "myEnc2", mod: audEnc2, local: true},
]);
App
VISA API
xDM Codec
xDM Codec
Engine API
Engine SPI
codec table
OSAL
Engine
Linux
DSP Link

Engine and Algorithm Names

Engine configuration file

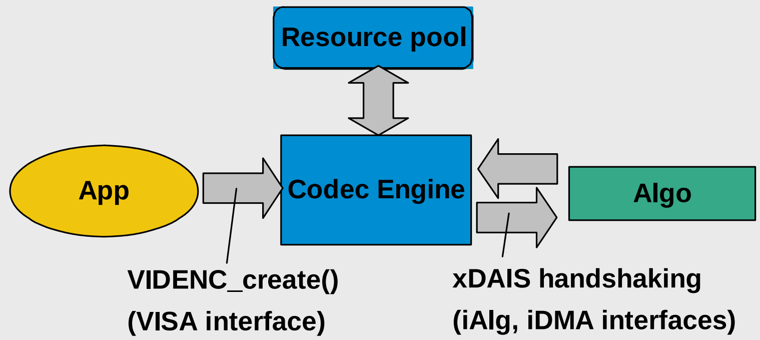
```
var myEng = Engine.create("myEngine", [
    {name: "myEnc1", mod: audEnc1, local: true},
    {name: "myEnc2", mod: audEnc2, local: true},
] );
```

Application Source File (app.c)

```
CERuntime_init();
myCE = Engine_open("myEngine", myCEAttrs);
myAE = AUDENC_create(myCE, "myEnc1", params);

AUDENC_control(myAE, ...);
AUDENC_process(myAE, ...);

VIDENC_delete(myAE);
 Engine_close(myCE);
```

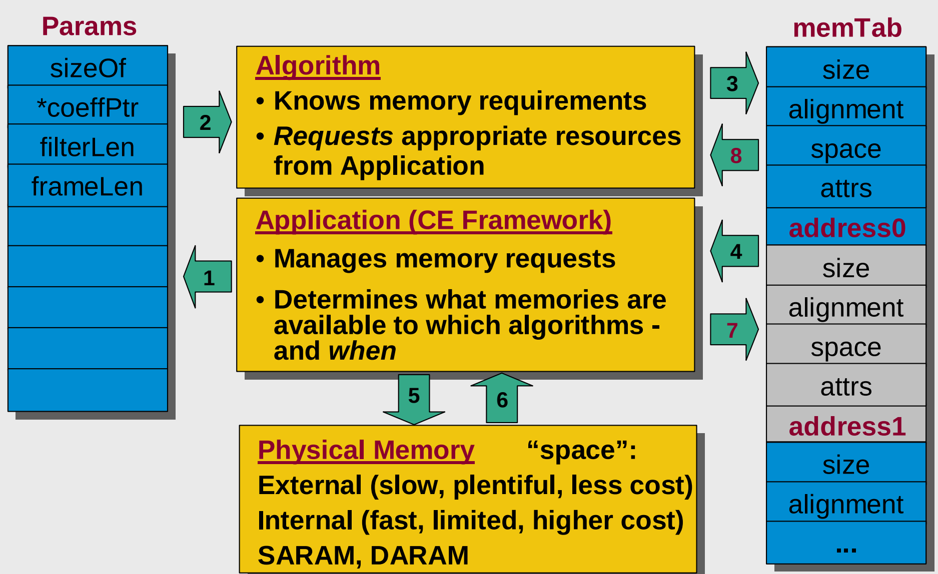
Resource Allocation

Codec Creation (Instantiation)



1. **Codec Engine gathers algorithm resource requirements from algorithm via iAlg and iDMA interfaces**
2. **Codec Engine secures resources from resource pool**
3. **Codec Engine grants resources to Algo via iAlg and iDMA interfaces**

The xDAIS Transaction



Codec Creation: xDAIS

xDM inherits xDAIS instantiation functions

Create Phase

- algNumAlloc
- algAlloc
- algInit

Create phase functions use a handshaking mechanism to request memory from the framework

Execute Phase

- algActivate
- algDeactivate
- (algMoved)

algActivate and algDeactivate are used for scratch memory sharing between algorithms.

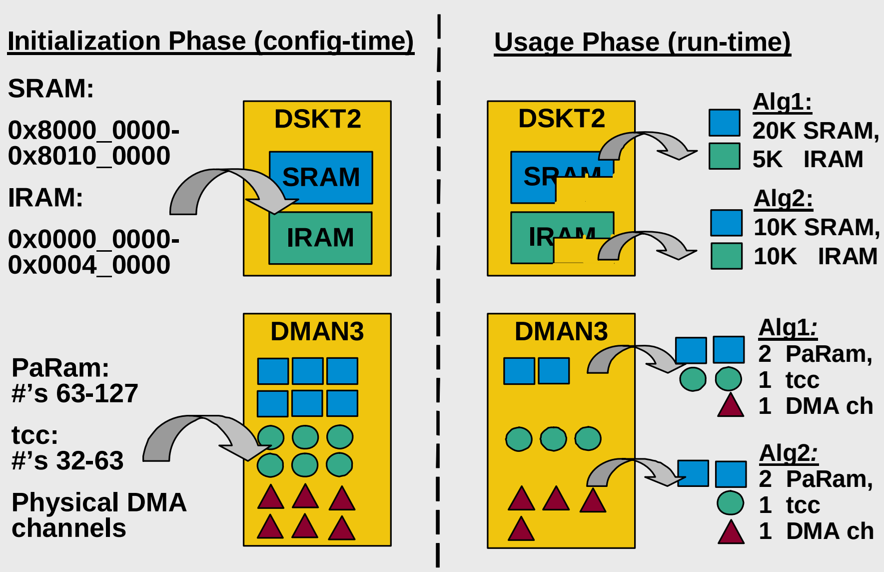
Delete Phase

- algNumAlloc
- algFree

When an algorithm instance is deleted, memory resources are returned to the framework.

DSKT and DMAN3

One last detail: The engine allocates resources to the codecs.
How does it know which resources are available?



DSKT2 Setup in app.cfg

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

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

Algorithms request memory from the framework using IALG/xDAIS identifiers (DARAM0, DARAM1, etc). Those identifiers are tied to system memory heaps here. The heap names must match heaps that are created 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.

DMAN3 Setup in app.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

For More Information

Codec Engine Information

Codec Engine Application Developer's Guide
Literature number: sprue67
http://focus.ti.com/lit/ug/sprue67c/sprue67c.pdf

Codec Engine Algorithm Creator User's Guide
Literature number: sprued6
http://focus.ti.com/lit/ug/sprued6b/sprued6b.pdf

Codec Engine Server Integrator's User's Guide
Literature number: sprued5a.pdf
http://focus.ti.com/lit/ug/sprued5a/sprued5a.pdf

xDAIS-DM (Digital Media) User Guide
Literature number: spruec8
http://focus.ti.com/lit/ug/spruec8b/spruec8b.pdf

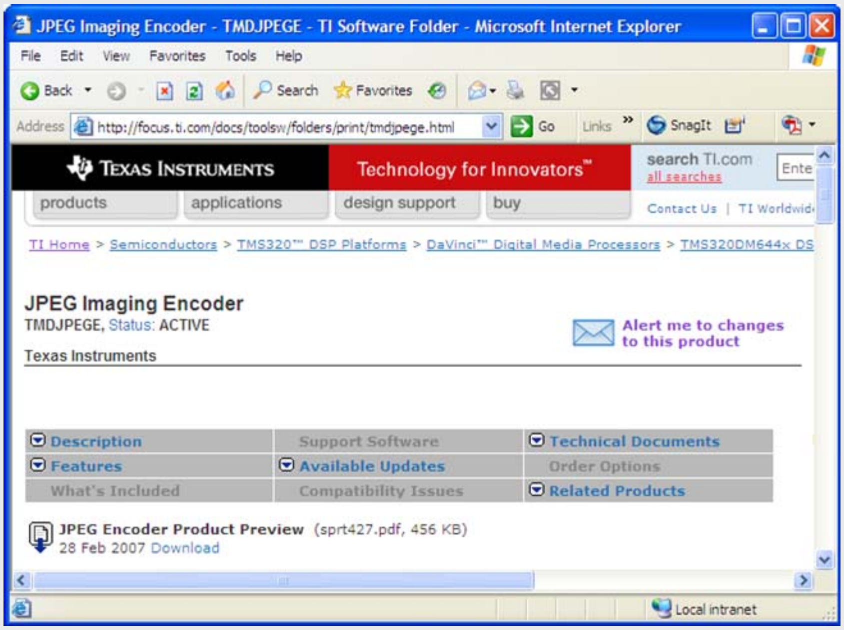
Information on Available Codecs

eXpressDSP Digital Media Software Product Bulletin
Literature Number: sprt390c
http://focus.ti.com/lit/ug/sprt390c/sprt390c.pdf

Davinci Software Book
Literature Number: sprt389
http://focus.ti.com/lit/ug/sprt389/sprt389.pdf

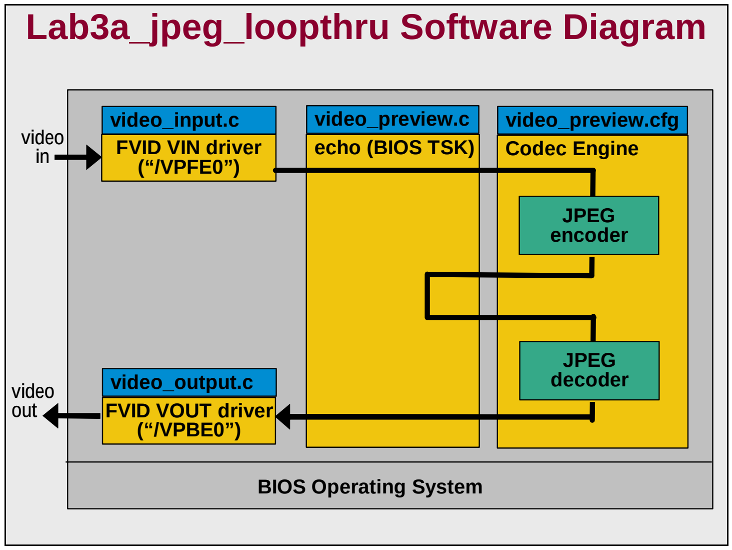
TI Digital Media Software webpage
http://focus.ti.com/dsp/docs/dspsupporto.tsp?sectionId=3&tabId=1460

Information on Available Codecs



Lab 3a (DSP/BIOS version)

Important: Labs 2-4 of this workshop are presented in two versions, one utilizing DSP/BIOS only (Lab 3a) for the operating system, and one utilizing VirtualLogix Linux and DSP/BIOS executing concurrently (Lab 3b). You will only have time to complete one version, so please choose the lab version appropriate for your system.



In this lab, you will explore the Codec Engine framework via a loopthru demo application that converts composite video input to the DM6437 DVEVM into JPEG-compressed images which are then decoded and displayed on composite video output.

Examine, Build and Test the Application

1. Start Code Composer Studio (if it is not already open) using the Desktop Shortcut

Note: you will need to make sure that the DM6437 dsk is connected (via USB emulation cable) and powered on.

2. Load the video_preview.pjt project from the lab3a_copy_loopback directory

Project→open…

select video_preview.pjt from the C:\dm6437_1day\lab3a_copy_loopback folder.

3. Open and inspect the video_preview.c file

4. Locate the create_codec( ) function within video_preview.c

edit→find…

5. Examine the create_codec( ) function

Creation and configuration of a JPEG encoder instance is as simple as:

1. declaring and configuring the appropriate IMGENC_Params structure
2. declaring and configuring the appropriate IMGENC_DynamicParams structure
3. calling IMGENC_create with the configured (static) parameters structure
4. calling IMGENC_control with the XDM_SETPARAMS instruction and the dynamic parameters structure to set the appropriate dynamic parameters

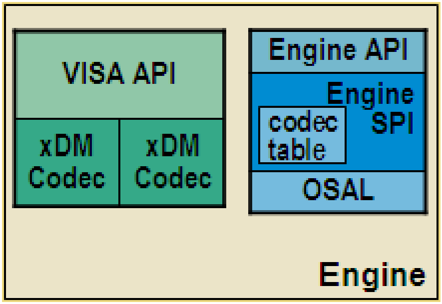
6. Locate the while loop within the video_preview function

The video_preview function is the main application task which drives the webcam application. There is some setup before entering the while loop that you may be interested in examining further at a later time, but for now, let’s focus on the main while loop.

The while loop uses the following structure:

1. FVID_exchange capture driver call to acquire an incoming video buffer
2. Configuration of the IMGENC_DynamicParams structure and setting of JPEG encoder’s dynamic params through the IMGENC_control function with the XDM_SETPARAMS command
3. Processing of the captured video frame via the IMGENC_process call
4. Cache writeback-invalidate on the output buffer (because it is stored in external memory and the codec uses the DMA)
5. Repitition of steps 2-4, but for the JPEG Decoder
6. Displaying the resultant video buffer via FVID_exchange with the display driver

7. **Examine video_preview.cfg**



The script begins by configuring the Operating System Abstraction Layer to support the DSP/BIOS operating system. Next, the JPEG encoder and decoder modules are imported via the xdc.usemodule( ) function. After that, the script imports the Engine module via the same function and configures the Engine to contain the JPEG encoder and decoder, configuring properties as appropriate.

Finally, the configuration script imports DSKT2 and DMAN3 modules as discussed during the lecture, configuring them with the available memory and DMA resources for use by the JPEG encoder and decoder when they are created.

8. **Connect the DM6437 dsk (if not already connected)**

Debug→Connect (Alt-C)

9. **Configure CCS to use the xdc toolset**

(Note, this should already be done on your lab setup, but you must be sure to do this step when running the lab exercises at home.)

Help→about…

Then select the Component Manager button

Within the Component Manager, navigate to the Target Content (XDC) branch and expand the TMS320C64XX target. Be sure that the XDC toolset is selected as shown above.

10. **Build the project**

Project→Rebuild All

(You will get a number of warnings/remarks in the build output, but should get no build errors.)

11. **Load the video_preview.out executable onto the DM6437 DSK**

File→Load Program… (Ctrl-L)

select video_preview.out from the Debug subfolder of the C:\dm6437_1day\lab3a_copy_loopback directory.

12. **Run the executable**

Debug→Run (F5)

13. Open the statistics window

DSP/BIOS → Statistics View

Note the max and average values for the stsDatarate statistics object.

STS	Count	Total	Max	Average
TSK_idle	0	0 inst	-2147483648 inst	0.00
previewTask	0	0 inst	-2147483648 inst	0.00
IDL_busyObj	1.12854e+...	-4.95506e+008	-296	-439.067
stsDatarate	5593	3.86588e+009	691200	691200

The max and average datarate (between encoder and decoder) is 691,200 Bytes per frame. This is because we are currently using a copy-based codec which does not actually encode but simply copies the input buffer to the output buffer. This codec is very useful for testing application setup before inserting a real codec as we are going to do in the next section.

The 691,200 Bytes per frame transferred corresponds to:
(720 pixels per line) * (480 lines) * (2 Bytes per pixel)
Frame size of a 4:2:2 YUV encoded standard definition frame

14. View the CPU load graph

DSP/BIOS → CPU Load Graph

Insert JPEG Encoder

Because of the xDM and VISA interface standards and the flexibility of the Codec Engine framework, changing out two xDM-compliant codecs is as easy as modifying two lines in our configuration file. This makes side-by-side comparison of codecs as well as upgrading codecs a nearly trivial exercise.

15. Halt execution

Debug → Halt (shift-F5)

16. Reset the CPU

Debug → Reset CPU (ctrl-R)

17. Open the video_preview.cfg file

18. Modify the encoder and decoder module import lines to import the JPEG codec

Locate the following lines:

```
/* get various codec modules; i.e., implementation of codecs */
var JPEGENC = xdc.useModule('codecs.imgenc_copy.IMGENC_COPY');
var JPEGDEC = xdc.useModule('codecs.imgdec_copy.IMGDEC_COPY');
```

and modify them to read:

```
/* get various codec modules; i.e., implementation of codecs */
var JPEGENC = xdc.useModule('codecs.jpeg_enc.JPEG_ENC');
var JPEGDEC = xdc.useModule('codecs.jpeg_dec.JPEG_DEC');
```

Note that the codec module names (i.e. codecs.jpeg_dec.JPEG_DEC) are part of the documentation provided with a delivered xDM encoder or decoder.

19. Rebuild, load and run the application following steps 10-12 of the previous section

20. What differences do you notice in the data rate and CPU load graphs measured as per steps 13 and 14 in the previous section

Note: You should right click in the statistics window and select “Clear” to clear the data gathered from the copy-based codec in order to gather statistics just from the JPEG encoder.

Modify JPEG Encoder Parameters

21. Halt the application if it is still running

22. Locate the dynamicParams.qValue parameter in video_preview.c <u>within the while loop of video_preview()</u>

Edit→Find… and select video_preview( )

Edit→Find… and select dynamicParams.qValue

23. Modify the qValue (quality value) from 73 to 1

24. Reset the CPU

Debug → Reset CPU (ctrl-R)

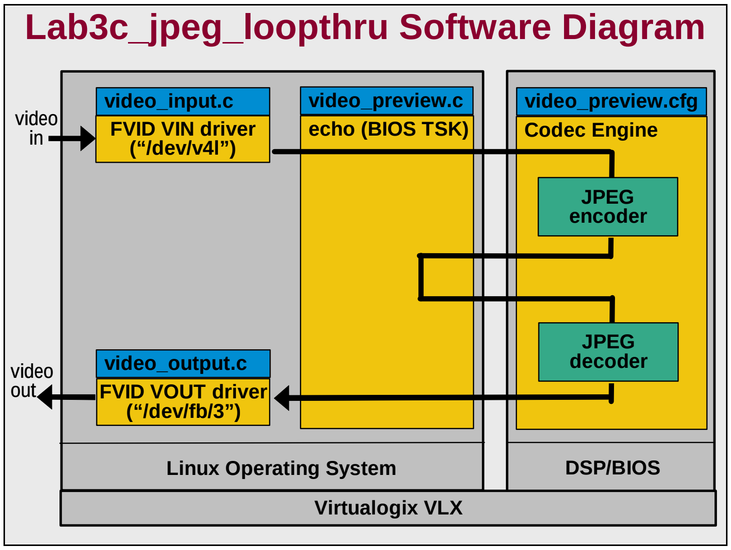
25. Rebuild, load and run the application as in step 19 of the previous section

26. Do you observe a change in the video output? How about the data rate and CPU load?

If you do not observe a noticeable difference, ask your instructor for help. Also, do not forget to clear the statistics window to ensure that you are gathering only the statistics for the current run.

Lab 3b (Linux version)

Important: Labs 2-4 of this workshop are presented in two versions, one utilizing DSP/BIOS only (Lab 3a) for the operating system, and one utilizing VirtualLogix Linux and DSP/BIOS executing concurrently (Lab 3b). You will only have time to complete one version, so please choose the lab version appropriate for your system.

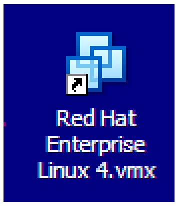


In this lab, you will explore the Codec Engine framework via a loopthru demo application that converts composite video input to the DM6437 DVEVM into JPEG-compressed images which are then decoded and displayed on composite video output.

Start the Linux Virtual Machine

1. Start the Red Hat Enterprise Linux 4 Virtual Machine (If not already started)

There is a shortcut on your desktop that will bring up the Virtual Machine information page



Select “Start this virtual machine” from the information page

2. Log in with user permissions

There are two Linux accounts set up:

user:	user
password:	useruser
user:	root
password:	rootpw

At times the instructions will ask you to switch to root permissions using the “su” (switch user) command, but generally you should be logged in to the user account.

Configure Ethernet Port in Virtual Machine

3. Open a terminal window

right-click in the Redhat Linux desktop and select “Open Terminal”

4. Within the terminal you opened in step 3, use “su” command to switch to root permission

```
# su
```

enter “rootpw” as the password when asked

5. Use the following procedure to configure the ethernet port with static IP address 192.168.1.40

```
# /sbin/ifconfig eth0 down

# /sbin/ifconfig eth0 192.168.1.40

# /sbin/service nfs restart
```

6. Exit out of root permission (to user permission)

```
# exit
```

Build the Linux-side Application

7. Change to the /home/user/workshop/lab3_jpeg_loopback directory

```
# cd /home/user/workshop/lab3_jpeg_loopback
```

8. Build and install the application via the provided script

```
# ./runmake.sh install
```

Boot Linux on the DM6437 DVEVM

9. Start Tera Term Pro from the Windows Desktop

10. Start Code Composer Studio using the Windows Desktop Shortcut

Note: you will need to make sure that the DM6437 dsk is connected (via USB emulation cable) and powered on.

We will use Code Composer Studio to load the linux kernel onto the DVEVM and boot. Note, however, that the filesystem which Linux will use is a network share using the Network File Share (nfs) filesystem, and the shared path is located within the Linux environment of the virtual machine. When you ran the “make install” command in the previous section, the make utility not only rebuilt the lab2_av.out applction, but copied it to this shared directory so that it will be available to execute from the DVEVM board

11. Connect the DM6437 dsk

Debug→Connect (Alt-C)

12. Load the server.pjt project in the lab3b_loopback_linux directory

Project→Open…

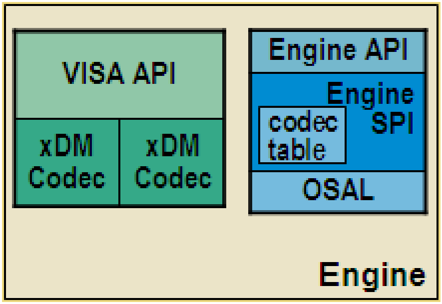
navigate to C:\dm6437_1day\labs\lab4b_webcam_linux

select server.pjt

13. Examine the mainTask function in main.c

Remember that the codec engine executes within the DSP/BIOS portion of the Virtuallogix dual-operating system environment. This task function opens the Codec Engine and creates an instance of the JPEG encoder contained in the Engine. It then loops within a while loop, pending on the SEM_ShmEvent signal to indicate that a message has been received from the Linux side via shared memory. When the message is received, the BIOS-side mainTask function then decodes the message, calls IMGENC_process to encode the incoming buffer with the JPEG encoder, and then returns the result using the nk_xirq_trigger function to send a virtual interrupt back to the linux side.

14. Examine video_preview.cfg



The script begins by configuring the Operating System Abstraction Layer to support the DSP/BIOS operating system. Next, the codec modules are imported via the xdc.usemodule( ) function. We will start with dummy copy-based codecs (provided with the Codec Engine for test purposes), but in the next section, you will add in the JPEG encoder and decoder. After that, the script imports the Engine module via the same function and configures the Engine to contain the previously imported encoder and decoder, configuring properties as appropriate.

Finally, the configuration script imports DSKT2 and DMAN3 modules as discussed during the lecture, configuring them with the available memory and DMA resources for use by the JPEG encoder when it is created.

15. Rebuild the project

Project→Rebuild All

16. Reset the CPU

Debug→Reset CPU (Ctrl-R)

17. Load the kernel, vlx and BIOS executable onto the DM6437 DSK

File→Load Program… (Ctrl-L)

Navigate to C:\dm6437_1day\labs\lab3b_webcam_linux\Debug

Load each of the three executables in the following order:

nkern.out	Virtuallogix vlx virtualizer
vmlinux.out	Linux kernel
server.out	DSP/BIOS app and Bootloader program

note: the order is important because server.out needs to be loaded last so that the correct entry point is set. The order of nkern and vmlinux don't actually matter.

18. Run the program

Debug→Run (F5)

You should see feedback on the Terra Term serial terminal emulator as the Linux kernel boots, ending with a login prompt:

```
192.168.1.40 login:
```

If the kernel feedback gives an error before reaching the login prompt, ask your instructor for help.

Run and view Application

19. Log into serial linux terminal as root user. No password is required.

20. Change to the /opt/workshop directory

```
# cd /opt/workshop
```

21. Load the audio and video driver modules with the loadmodules.sh script

```
# ./loadmodules.sh
```

note: if you would like to see the contents of this script, type:

```
# cat loadmodules.sh
```

22. Execute the lab3_loopback.out application

```
# ./lab3_loopback.out
```

Insert JPEG Encoder

Because of the xDM and VISA interface standards and the flexibility of the Codec Engine framework, changing out two xDM-compliant codecs is as easy as modifying two lines in our configuration file.

23. Halt the Linux lab3_loopback.out application

Press ctrl-C in the serial terminal

24. Halt execution of the DM6437 within Code Composer Studio

Debug → Halt (shift-F5)

25. Reset the CPU (in CCS)

Debug → Reset CPU (ctrl-R)

26. Open the video_preview.cfg file

27. Modify the encoder and decoder module import lines to import the JPEG codec

Locate the following lines:

```
/* get various codec modules; i.e., implementation of codecs */
var JPEGENC = xdc.useModule('codecs.imgenc_copy.IMGENC_COPY');
var JPEGDEC = xdc.useModule('codecs.imgdec_copy.IMGDEC_COPY');
```

and modify them to read:

```
/* get various codec modules; i.e., implementation of codecs */
var JPEGENC = xdc.useModule('codecs.jpeg_enc.JPEG_ENC');
var JPEGDEC = xdc.useModule('codecs.jpeg_dec.JPEG_DEC');
```

28. Rebuild, load and run the application following steps 15-18 and step 22 of the previous section

(Optional) Modify JPEG Encoder Parameters

29. Halt the application following steps 23 and 24 of the previous section

27. Locate the dynamicParams.qValue parameter in main.c of the server project in Code Composer Studio. <u>Make sure you find the qValue that is within the while loop of video_preview()</u>

Edit→Find… and select video_preview( )

Edit→Find… and select dynamicParams.qValue

30. Modify the qValue (quality value) from 90 to 1 and save

31. In Code Composer Studio, Reset the CPU

Debug → Reset CPU (ctrl-R)

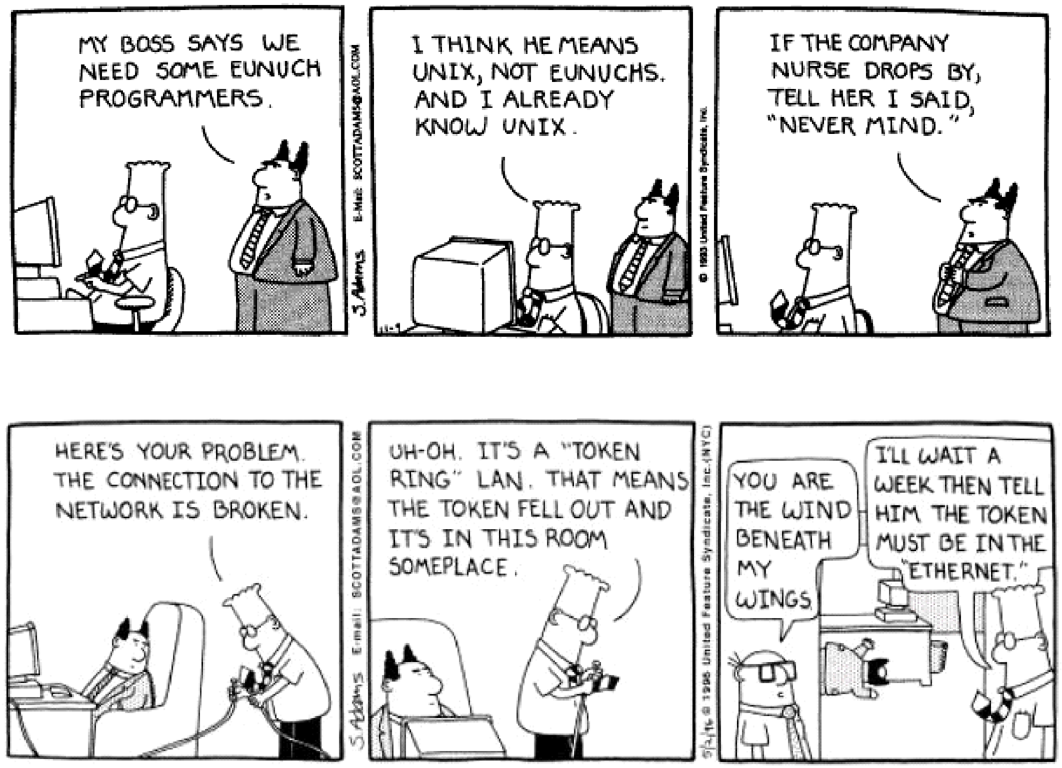
32. Rebuild, load and run the application as in step 28 of the previous section

33. Do you observe a change in the video output?

If you do not observe a noticeable difference, ask your instructor for help.

Networking and Filesystem

Introduction



DILBERT: © Scott Adams/Dist. By United Feature Syndicate, Inc.

Multi-tasking, networking, resource management and other system issues can develop into complex challenges, even for an engineer who is more in the know than Dilbert's pointy haired boss. Often these concerns are not system differentiators, simply entry-point development that is required in order to bring a production-quality product to market.

This is why many Operating Systems provide a toolset for designers to use in managing these system concerns. Effective utilization of the tools at a designer's disposal can turn design years into design months. This is why the DM6437 software framework includes two operating systems: DSP/BIOS (a real-time operating system) and Linux (a general-purpose operating system).

In this module we will compare and contrast the features of these two operating systems as well as providing a basic introduction to how their various toolsets may be used in system development.

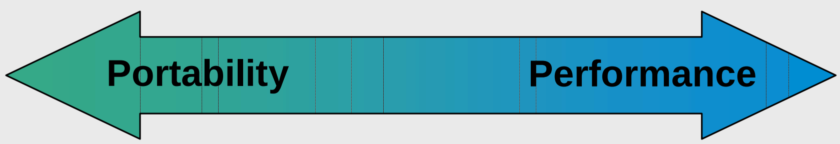
Module Topics

TMS320DM643x Software Overview

- Network Services (NDK)
- EMAC (NDK Socket I/F)

Operating System Layer (DSP BIOS and/or uC Linux)

An Operating System Tradeoff

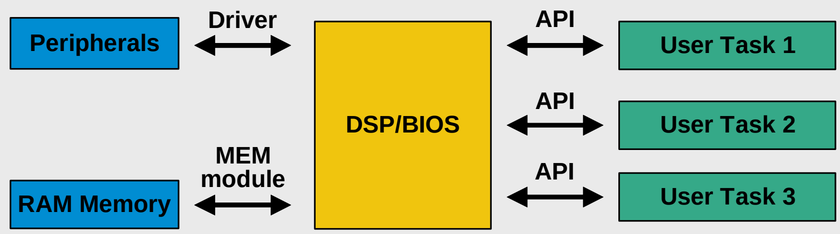


Linux | **DSP/BIOS**

Linux provides a more generic hardware interface
- Improved portability across platforms
- Many available open-source applications

DSP/BIOS provides more direct hardware control
- Improved hardware utilization / performance
- Critical for MIPS-intensive signal processing

What is DSP/BIOS?



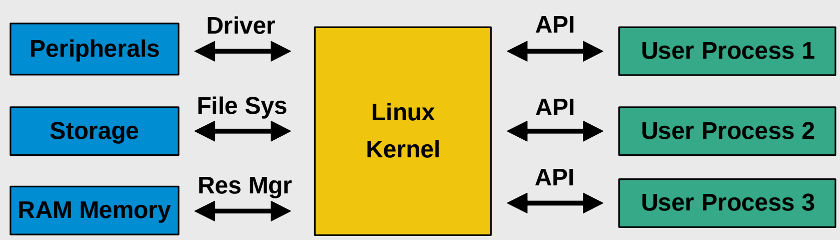
Scheduling
- Multi-tasking

Resource abstraction
- Drivers for peripherals

Resource allocation
- Arbitration between tasks

Debugging Toolset
- Real Time Data Exchange (RTDX)

What is Linux?



Scheduling
- Multi-threading

Resource abstraction
- Drivers for peripherals
- Filesystems for storage devices

Resource allocation
- Protected access to resources

Threads and Scheduling

Execution Threads

Option 1: Audio and Video in a single thread

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

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

Option 2: Audio and Video in separate threads

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

int tsk1(int argc, char *argv[]) {
      while(condition == TRUE)
            callAudioFxn();
}
```

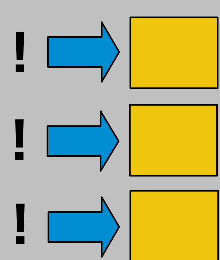
```
// video.c, handles video only

int tsk2(int argc, char *argv[]) {
      while(condition == TRUE)
            callVideoFxn();
}
```

Splitting into two threads 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)

Which Thread Runs?

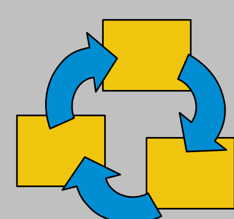
Two common schemes for sharing CPU between threads:

Real Time



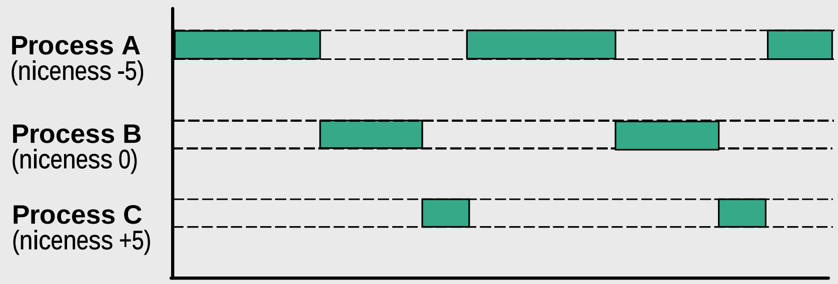
A system event drives a prioritized response

Time Sliced



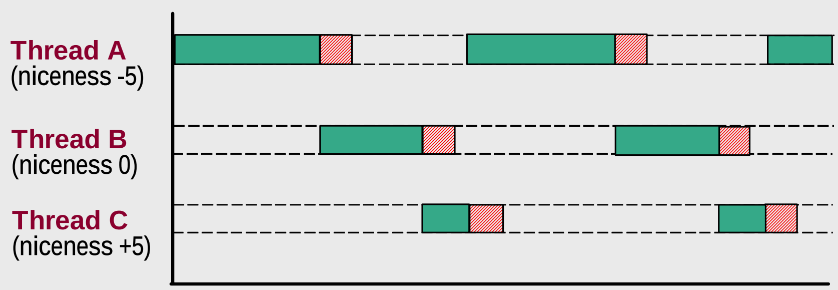
Multiple threads share CPU in round-robin circulation

Time-Slice Scheduler



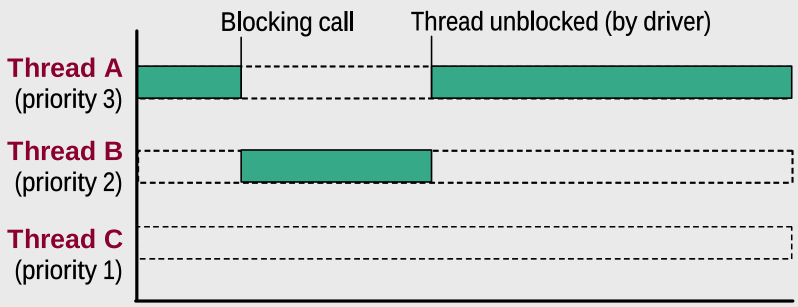
Processes are time-sliced with more time given to processes that are assigned a lower niceness value

Time-Slice Scheduler, >100% loading



- All threads share the pain of overloading, no thread has time to complete all of its processing
- Niceness values may be reconfigured, but system indeterminism may cause future problems
- A good solution for non-realtime applications such as user interface

Real-Time Scheduler

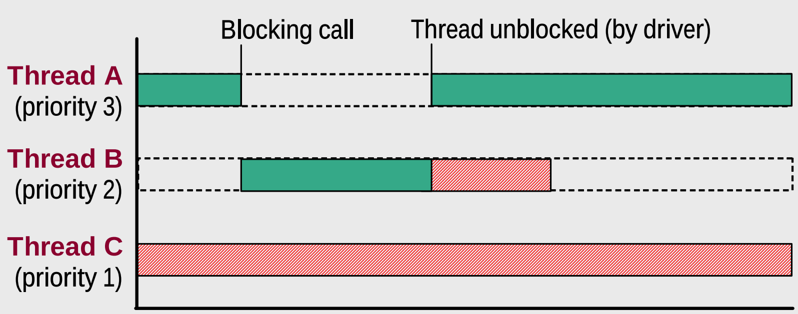


The highest priority process runs until completion or a blocking event

If external event of driver filling input buffer unblocks Thread A, it will preempt lower priority threads

Note: Time-sliced threads may also block, taking themselves out of rotation, but it is not required as with real time threads

Real-Time Scheduler, >100% loading



- Real-Time threads are scheduled according to priority
- The highest priority thread always “wins” and may cause low priority threads not to run at all
- A good solution for real-time applications such as codecs

Task Code Topology – Blocking Calls

```
Void taskFunction(...)
{
      // SIO initialization

      // CE Initialization

      while ('condition'){
            SIO_put(aiHandle, &bufPtr);

            AUDENC_process(myVE, ...);

            // Store Processed file
      }

      // SIO cleanup

      // CE cleanup
}
```

Function blocks thread execution until new data is available

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

Higher priority threads must block for lower priority threads to run

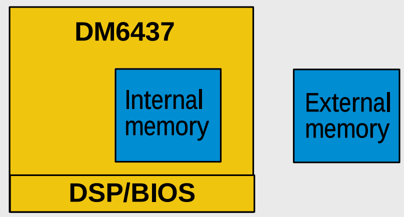
- × Requires "good citizen" threads
- × Low priority threads may starve
- ✓ Lower priority threads never break high priority threads
- ✓ Lower context-switch overhead

BIOS or Linux R/T

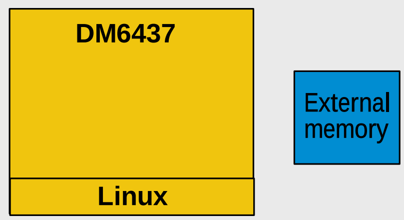
Resource Allocation

Dynamic Memory Allocation

DSP/BIOS MEM module allows memory allocations from any location in the DSP's memory map.



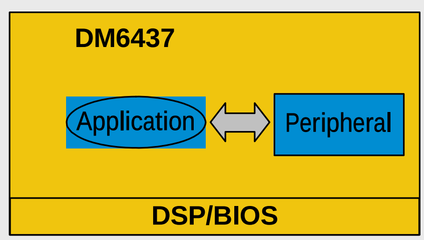
Linux malloc can only allocate memory from one system heap, usually in external memory. Internal memory often provides a data cache.



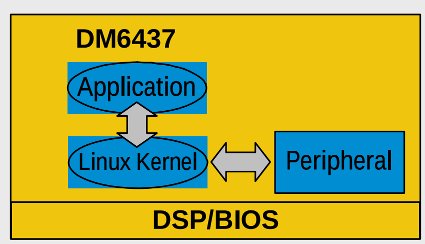
Static memory allocation has the same limitations for Linux and freedom for DSP/BIOS.

Peripherals Access

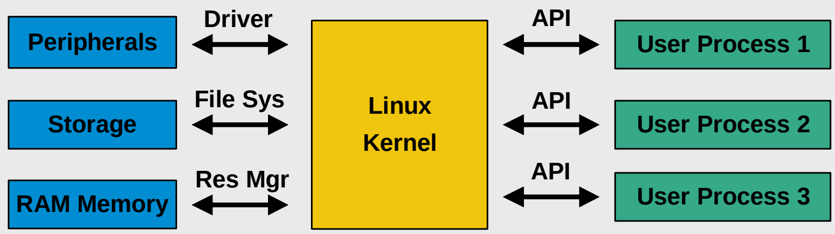
DSP/BIOS allows application code to directly modify peripheral registers.



Linux requires a context switch into kernel mode for any peripheral access.



Linux: Protected, Portable Memory/Peripheral Access



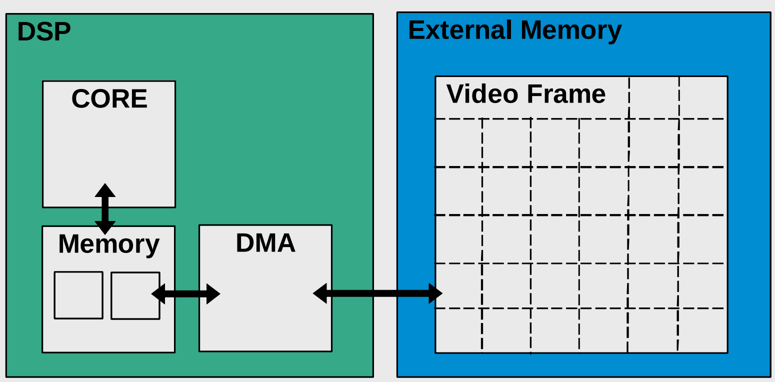
Protection

- Linux kernel acts as a proxy for process requests to peripherals

Portability

- Less specificity in memory requests allows greater portability across platforms.

DSP/BIOS: Efficient memory/peripheral access



Algorithms may utilize DMA when processing data too large to fit in internal memory. (More efficient than cache)

Linux does not allow user applications to request internal versus external memory.

Linux driver latency (kernel mode context switch) limits the efficiency of small DMA transfers.

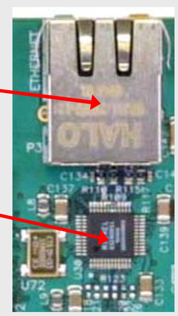
Networking Support

DVEVM Ethernet Support

- Supports 10/100 Mbit interface, sourcing the 25MHz clock
- MDIO PHY Address defaults to 0x00001

RJ-45 connector:
Green LED (link status/activity)
Yellow LED (duplex mode)

Micrel KS8001L PHY
interfaces to MII peripheral

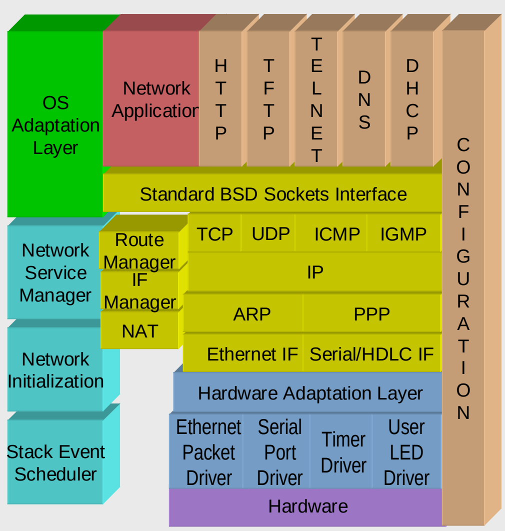


I2C EEPROM

Address	Contents
0x7F00	EMAC Address 0 (most significant)
0x7F01	EMAC Address 1
0x7F02	EMAC Address 2
0x7F03	EMAC Address 3
0x7F04	EMAC Address 4
0x7F05	EMAC Address 5
0x7F06	Reserved
0x7F07	Board Revision

Can be disconnected from the DSP via a FET switch to allow PCI use and daughtercard EMAC use.

Networking Stack



NDK (DSP/BIOS)

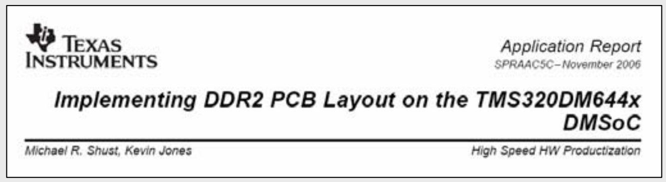
- Telnet Server
- DHCP Server and Client
- DNS Server and Client
- PPP Server and Client
- PPPoE Server and Client
- Virtual network w/ NAT
- HTTP Server
- TFTP Client
- IGMP Client

Virtuallogix Linux

- TCP/UDP
- IPv4, IPv6
- IP multicast
- IP forwarding
- DHCP/BOOTP/ RARP
- IP tunneling
- DiffServ, RSVP
- RTP/RTSP

Filesystem Support

DDR2 Interface


Texas Instruments

Application Report
SPRAAC5C–November 2006

Implementing DDR2 PCB Layout on the TMS320DM644x DMSoC

Michael R. Shust, Kevin Jones

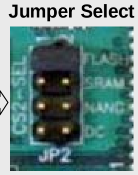
High Speed HW Productization

- 128MBytes: two 512Mbit, 16-bit wide memories
- Clock Rates up to 166 MHz
- Supports both 84-ball and 92-ball package types
- Schematics and layout dictated by DDR2 PCB Layout App. Report

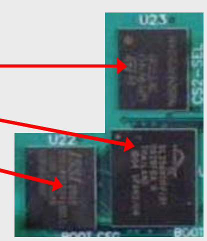
Asynchronous Memories

Jumper Select

EM_CS2

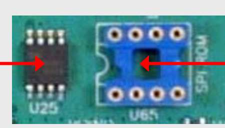


- NAND Flash (64MBytes)
- NOR Flash (16MBytes)
- SRAM (2MBytes)
- Daughtercard Header



EEPROMs

256Mbit I2C EEPROM containing DSP MAC address



SPI EEPROM socket connected to McBSP0 interface

NDK's Embedded File System

EFS **The embedded file system is the file I/O API that is used by the HTTP server and several of the example programs. Supports RAM files.**

EFS_createfile	**Create a file from a RAM array**
EFS_destroyfile	**Remove a file (if no open refs.)**
EFS_fopen	**Open a file**
EFS_fclose	**Close a file**
EFS_fread	**Read from an open file**
EFS_fwrite	**Write to an open file**
EFS_fseek	**Seek a position in open file**

VirtualLogix Linux Supported File Systems

Media File systems:

ext3	Harddrive, Robust against unexpected power-down
vfat	Harddrive, Windows FAT-32 compatible
msdos	Harddrive, Windows FAT-16 compatible
iso 9660	CD-ROM filesystem

Memory File systems:

jffs	Journaling flash filesystem
jffs2	Journaling flash filesystem (2nd generation)
cramfs	Compressed RAM filesystem

Special File systems:

nfs	Share a remote linux filesystem
devfs	Device driver filesystem
autofs	Automatic filesystem mounting with timeout

Accessing Files in Linux

Manipulating files from within user programs is as simple as…

File descriptor / handle *Directory previously mounted* *File to open…* *Permissions*

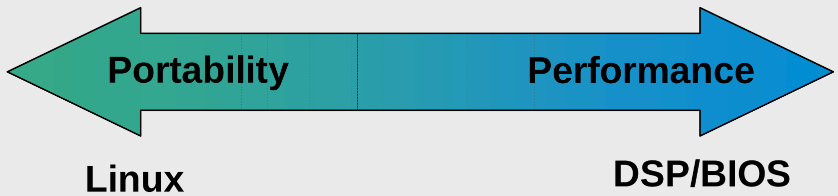
```
myFileFd = fopen("/mnt/harddrive/myfile","rw");
fread ( aMyBuf, sizeof(int), len, myFileFd );
fwrite( aMyBuf, sizeof(int), len, myFileFd );
fclose( myFileFd );
```

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

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

VirtualLogix VLX

An Operating System Tradeoff



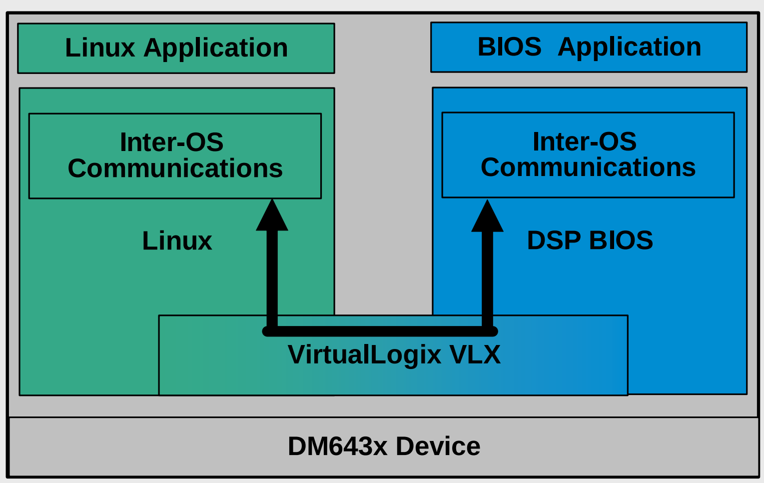
Linux:

A very good operating system fit for general pupose code, where portability and reliability requirements outweigh performance

DSP/BIOS:

A very good operating system fit for signal processing code, where performance is critical.

VirtualLogix VLX – The best of both worlds



VLX allows application programmers to run Linux and DSP BIOS simultaneously, combining the feature-rich Linux environment with the DSP BIOS hard realtime scheduler

Inter-OS Communications

Shared Memory

Large buffers of data can be passed via pointer between Linux and DSP/BIOS via a shared memory buffer

Cross Interrupts

Linux and DSP/BIOS may send virtual interrupts to each other. Useful when passing buffers via shared memory.

Virtual Ethernet

Linux and DSP/BIOS applications can communicate as if they were separate devices on an Ethernet Network.

Frame Buffer

Linux and DSP/BIOS applications may communicate as if they were separate devices connected via a video port.

Virtual Console

A Linux application can print to a standard character device, data is transmitted to DSP/BIOS side which displays as a BIOS trace log via RTDX over JTAG (can be displayed real time within Code Composer Studio).

When is VLX appropriate?

No

- **Deeply embedded devices that operate autonomously**
- **Limited or no connectivity with outside world**
- **Limited or no user interface**

Yes!

- **Real-time and general purpose computing requirements**
- **Alternative approach requires a General Purpose Processor and DSP**
- **Connected to the world**
- **Requires a complex user interface**
- **Requires fully-featured applications**
- **Have software already running on a DM644x-based solution**

For More Information

DSP/BIOS

DSP/BIOS Kernel Technical Overview

Literature Number: spra780

http://focus.ti.com/lit/an/spra780/spra780.pdf

TMS320C6000 DSP/BIOS User's Guide

Literature Number: spru303b

http://focus.ti.com/lit/ug/spru303b/spru303b.pdf

TMS320C6000 DSP/BIOS API Reference Guide

Literature Number: spru404m

http://focus.ti.com/lit/ug/spru404m/spru404m.pdf

DSP/BIOS Benchmarks, revision D

Literature Number: spraa16d

http://focus.ti.com/lit/an/spraa16d/spraa16d.pdf

TCP/IP Stack Documentation

TCP/IP Stack Getting Started Guide

The Getting Started Guide is a HTML based document providing up to date facts about the current Stack release. This includes revision history, system requirements, up to date documents, and current performance figures.

TCP/IP Stack User's Guide (SPRU523)

The User's Guide instructs the user on installing the Stack Software, going over the example applications, and starting to write their own DSP based networking application. It goes on to discuss how the Stack interacts with DSP/BIOS, and how to customize this interaction.

TCP/IP Stack Programmer's Reference Guide (SPRU524)

The Programmer's Reference Guide is an API reference for all the Stack components. In addition, there are some tutorial-like appendicles describing topics like PPP, NAT, and use of the HTTP server. It documents the Hardware Adaptation Layer, but does not discuss implementation.

TCP/IP Stack Platform Porting Guide (SPRU030)

The Porting Guide is used to move the stack from one C6000 based platform to another. It documents the mini-driver API for the Hardware Adaptation Layer, and discusses the method of porting device drivers.

VirtualLogix Information Pages

VirtualLogix Home page

http://www.virtuallogix.com/

VLX for Digital Multimedia Product Page

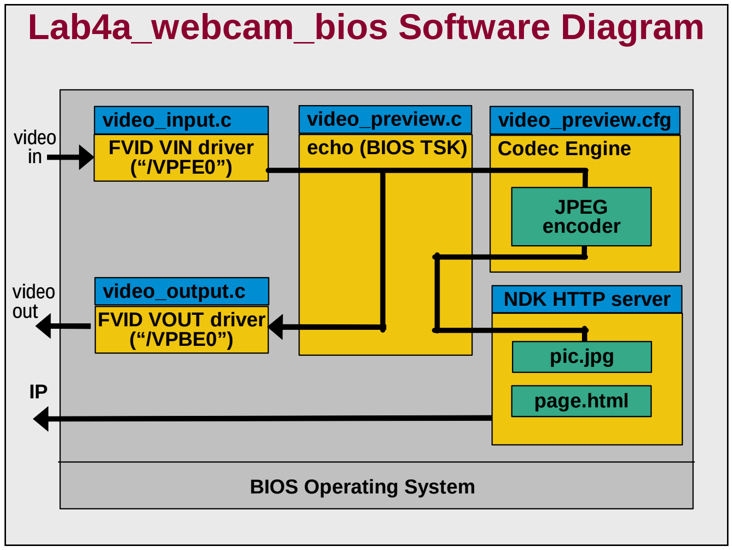
http://www.virtuallogix.com/index.php?id=21

VLXZone Support Page (patches, downloads, app notes, etc.)

http://www.virtuallogix.com/index.php?id=166

Lab 4a (DSP/BIOS version)

Important: Labs 2-4 of this workshop are presented in two versions, one utilizing DSP/BIOS only (Lab 4a) for the operating system, and one utilizing VirtualLogix Linux and DSP/BIOS executing concurrently (Lab 4b). You will only have time to complete one version, so please choose the lab version appropriate for your system.



In this lab, you will explore the networking capabilities of the DM6437 via a webcam demo application that converts composite video input to the DM6437 DVEVM into JPEG-compressed images which are displayed on the host via a webpage served from the DVEVM.

Examine the Application

1. **Start Code Composer Studio (if it is not already open) using the Desktop Shorrtcut**

Note: you will need to make sure that the DM6437 dsk is connected (via USB emulation cable) and powered on.

2. **Load the video_preview.pjt project from the lab4a_webcam_bios directory**

Project→open…

select video_preview.pjt from the C:\dm6437_1day\lab4a_webcam_bios folder.

3. **Examine the davinci.htm webpage file**

Expand the Documents folder and double click on davinci.htm

Notice that in the header of the document, the webpage defines a JavaScript function called refresh() which forces the webpage to reload the "pic.jpg" picture. Within the body, the "pic.jpg" picture is placed on the document via the <img> tag. Within this tag, the onload field is used to call a function whenever the picture is loaded into the page. We set the onload function to call setTimeout(), which instructs the page to call the refresh() function after 100mS. Thus, after every 100mS, the user-defined refresh function will reload the picture, which then triggers setTimeout() to run again. The result of this simple script is to force a reload of pic.jpg ten times per second.

Note that most browsers will cache the picture so that even when the reload is forced, the old (cached) version of the picture will be used. For this reason we will need to modify the browser settings to force a reload. (See step 15 below)

4. **Examine video_preview.c in the Source folder**

   Use the CCS search capability to search for “while”

   This is the main while loop of the application. It begins with an FVID_exchange call to load a new video buffer from the input driver. Next, IMGENC_control is used to set the desired image quality and IMGENC_process is called to JPEG encode the video buffer. Finally, the “efs_updatefilecb” function is called to update the pic.jpg file.

   Recall that efs is the embedded filesystem provided with the Networking Development Kit (NDK). This particular function updates pic.jpg with the newly encoded video frame. efs_updatefilecb is not a standard efs function call. For the purposes of this lab, the efs_createfilecb function was modified to allow a file to be updated without first being deleted. The source code for this function is provided in “os” folder of the project directory. All other efs function calls used are standard functions provided with the NDK.

5. **Examine server.c**

   In this file we configure the html server. Use the CCS search feature to find the “AddWebFiles” function. This function simply creates three new files using the efs_createfile and efs_createfilecb functions. These are standard functions provided with the networking development kit’s embedded file system (efs).

   The efs_createfile functions take as their arguments a C array and size. The binsrc.exe utility is used to convert files into C arrays. (See step 6 below)

6. **Examine davinci.htm custom build options**

   Right click on davinci.htm in the project tree and select “File Specific Options…” The window that is displayed shows a custom build command:

   ```
   “$(Proj_dir)\binsrc.exe” davinci.htm default.c DEFAULT
   ```

   This custom build command invokes the binsrc.exe utility, which is provided with the NDK at $(NDK_INSTALL_DIR)\packages\ti\ndk\example\tools\common\binsrc

   The command line options tell the utility to convert the text file davinci.htm into a standard C file named default.c. The utility will create a C array containing the text of davinci.htm converted to binary format. The name of the array is specified as the third argument, i.e. DEFAULT.

Build and Run the Application

7. Connect the DM6437 dsk (if not already connected)

Debug→Connect (Alt-C)

8. Build the project

Project→Rebuild All

9. Load the video_preview.out executable onto the DM6437 DSK

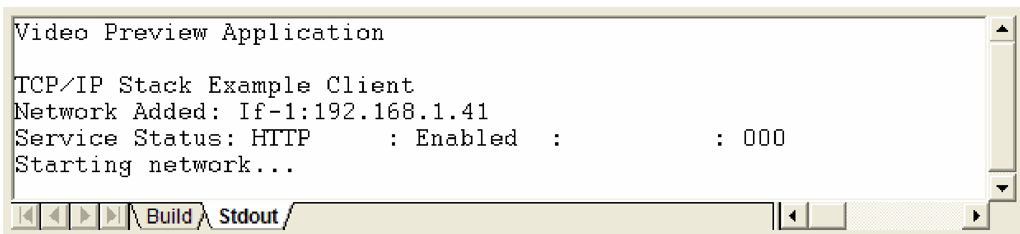
File→Load Program… (Ctrl-L)

select video_preview.out from the Debug subfolder of the C:\dm6437_1day\lab4a_webcam_bios directory.

10. Run the executable

Debug→Run (F5)

11. Note the IP address displayed in the Standard Output window of CCS



(This should always be 192.168.1.41 as it is statically assigned within the application)

View the Webcam Page in Internet Explorer

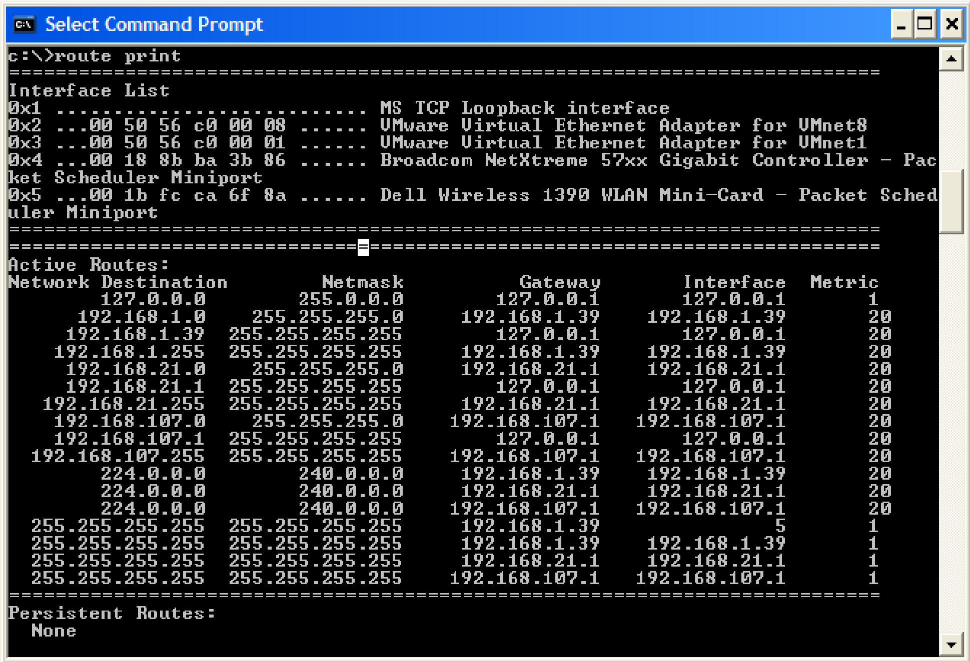
12. Check the Windows routing table

Enter windows terminal mode via start→Run… and entering “cmd”

At the windows terminal command prompt, type:

```
c:\> route print
```

You should see feedback similar to the following (will differ depending on setup):



```
Select Command Prompt
c:\>route print
===========================================================================
Interface List
0x1 ........................... MS TCP Loopback interface
0x2 ...00 50 56 c0 00 08 ...... VMware Virtual Ethernet Adapter for VMnet8
0x3 ...00 50 56 c0 00 01 ...... VMware Virtual Ethernet Adapter for VMnet1
0x4 ...00 18 8b ba 3b 86 ...... Broadcom NetXtreme 57xx Gigabit Controller - Pac
ket Scheduler Miniport
0x5 ...00 1b fc ca 6f 8a ...... Dell Wireless 1390 WLAN Mini-Card - Packet Sched
uler Miniport
===========================================================================
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
        127.0.0.0        255.0.0.0        127.0.0.1       127.0.0.1       1
      192.168.1.0    255.255.255.0     192.168.1.39    192.168.1.39      20
     192.168.1.39  255.255.255.255        127.0.0.1       127.0.0.1      20
    192.168.1.255  255.255.255.255     192.168.1.39    192.168.1.39      20
     192.168.21.0    255.255.255.0     192.168.21.1    192.168.21.1      20
     192.168.21.1  255.255.255.255        127.0.0.1       127.0.0.1      20
   192.168.21.255  255.255.255.255     192.168.21.1    192.168.21.1      20
    192.168.107.0    255.255.255.0    192.168.107.1   192.168.107.1      20
    192.168.107.1  255.255.255.255        127.0.0.1       127.0.0.1      20
  192.168.107.255  255.255.255.255    192.168.107.1   192.168.107.1      20
        224.0.0.0        240.0.0.0     192.168.1.39    192.168.1.39      20
        224.0.0.0        240.0.0.0     192.168.21.1    192.168.21.1      20
        224.0.0.0        240.0.0.0    192.168.107.1   192.168.107.1      20
  255.255.255.255  255.255.255.255     192.168.1.39               5       1
  255.255.255.255  255.255.255.255     192.168.1.39    192.168.1.39       1
  255.255.255.255  255.255.255.255     192.168.21.1    192.168.21.1       1
  255.255.255.255  255.255.255.255    192.168.107.1   192.168.107.1       1
===========================================================================
Persistent Routes:
  None
```

The important entry in this case is the one that reads:

Destination:	Netmask:	Gateway:	Interface
192.168.1.0	255.255.255.0	192.168.1.39	192.168.1.39

(For some lab setups, you may have a persistent route set specifically for destination 192.168.1.41)

This entry indicates that any IP request to the subnetwork 192.168.1.xxx (i.e. any address which, when binary &-ed with the netmask 255.255.255.0, produces 192.168.1.0) will be routed through the interface with IP address 192.168.1.39. Recall that 192.168.1.39 is the IP address that has been statically assigned to the windows network connection to the DVDP board.

If your network routing table is not properly set, you can add a routing entry via:

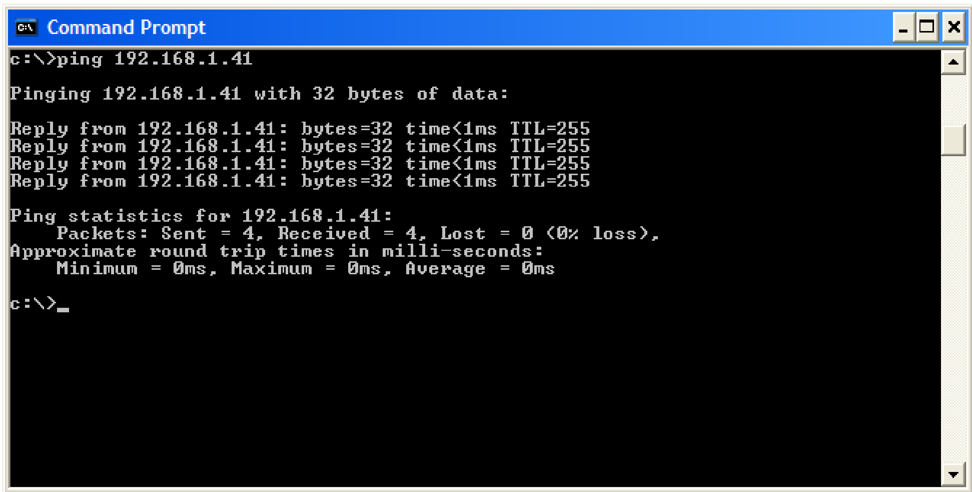
```
c:\> route add 192.168.1.41 192.168.1.39
```

13. Test your network connection to the board via the ping utility

At the Windows terminal command prompt, type:

```
c:\> ping 192.168.1.41
```

You should see a feeback similar to the following:



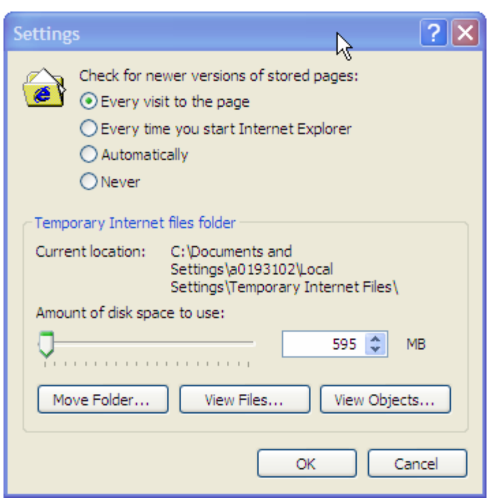
You can close the Windows terminal window when finished.

14. Open the Internet Explorer Web Browser and check the page refresh options

tools→Internet Options…

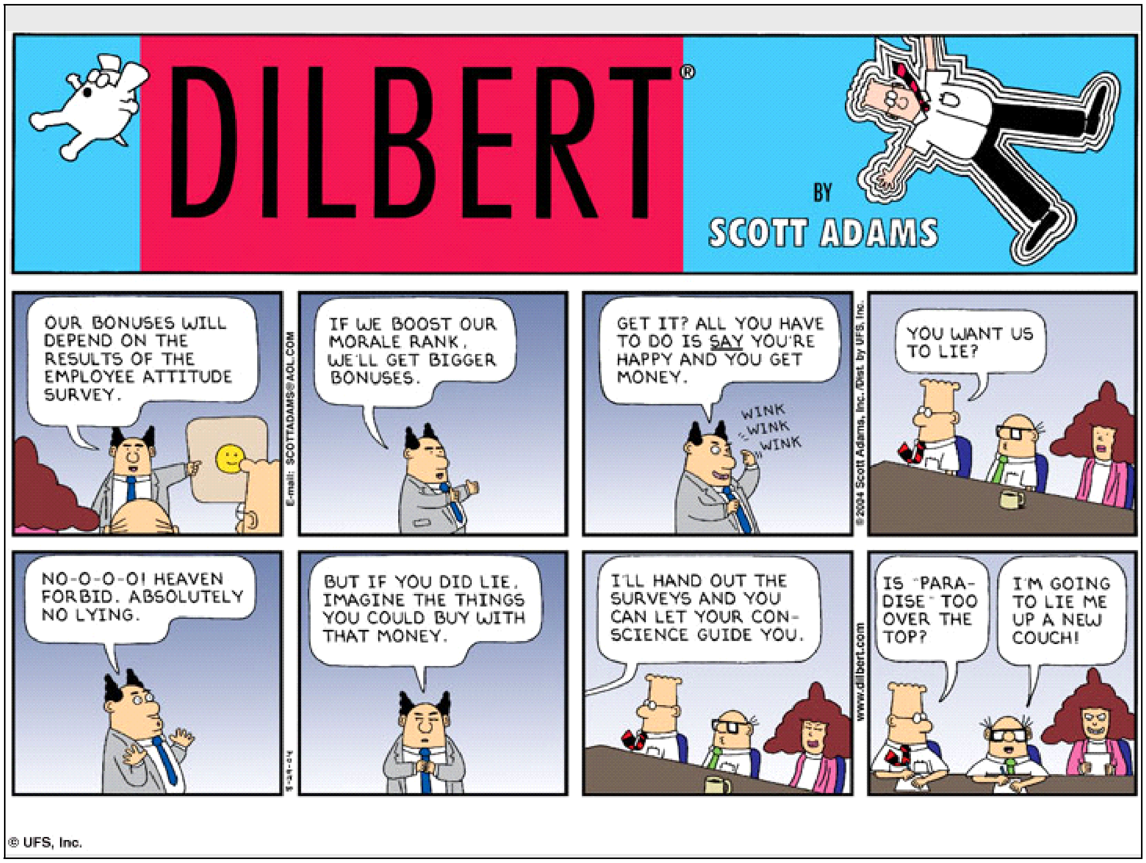
Click the "Settings" button under the Temporary Internet Files heading

15. Make sure the "Check for newer versions of stored pages": option is set to "Every visit to the page" as shown below:



16. Enter **http://192.168.1.41** into the browser's url locator

We Hope You Have Had an Informative Day

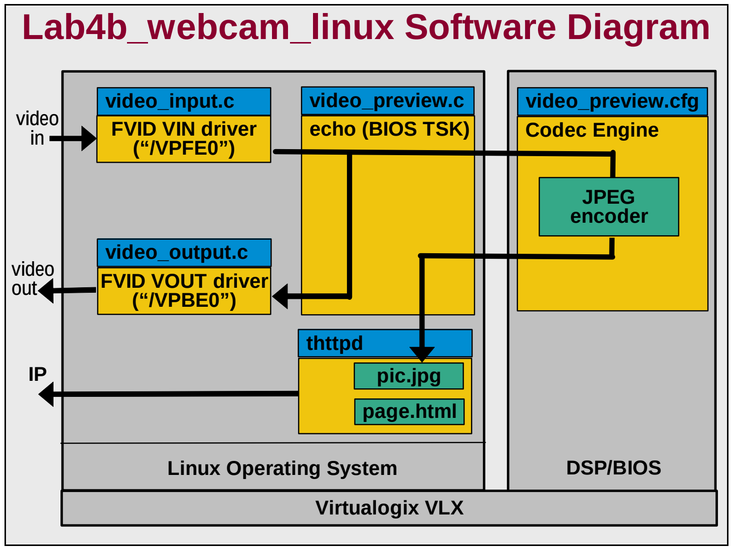


Please do not forget to fill out an evaluation of the DM6437 1-day workshop before you leave. Your feedback is crucial in helping us improve our course materials.

If you have not been provided with instructions for filling out an evaluation, please ask your instructor.

Lab 4b (Linux version)

Important: Labs 2-4 of this workshop are presented in two versions, one utilizing DSP/BIOS only (Lab 4a) for the operating system, and one utilizing VirtualLogix Linux and DSP/BIOS executing concurrently (Lab 4b). You will only have time to complete one version, so please choose the lab version appropriate for your system.

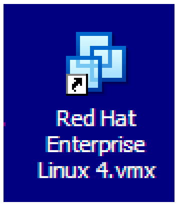


In this lab, you will explore the Codec Engine framework via a webcam demo application that converts composite video input to the DM6437 DVEVM into JPEG-compressed images which are displayed on the host via a webpage served from the DVEVM.

Start the Linux Virtual Machine

1. Start the Red Hat Enterprise Linux 4 Virtual Machine (If not already started)

There is a shortcut on your desktop that will bring up the Virtual Machine information page



Select “Start this virtual machine” from the information page

2. Log in with user permissions

There are two Linux accounts set up:

user:	user
password:	useruser
user:	root
password:	rootpw

At times the instructions will ask you to switch to root permissions using the “su” (switch user) command, but generally you should be logged in to the user account.

3. Open a terminal window

right-click in the desktop and select “Open Terminal”

4. Within the terminal window use “su” command to switch to root permission

```
# su
```

enter “rootpw” as the password when asked

5. Use the following procedure to configure the virtual machine’s network connection with static IP address 192.168.1.40

```
# /sbin/ifconfig eth0 down
# /sbin/ifconfig eth0 192.168.1.40
# /sbin/service nfs restart
```

6. Exit out of root permission (to user permission)

```
# exit
```

Examine and Build the Linux-side Application

7. Change to the /home/user/workshop/lab4_webcam directory

```
# cd /home/user/workshop/lab4_webcam
```

8. Examine the davinci.htm webpage file in the websrc folder

```
# gedit websrc/davinci.htm
```

Notice that in the header of the document, the webpage defines a JavaScript function called refresh() which forces the webpage to reload the "pic.jpg" picture. Within the body, the "pic.jpg" picture is placed on the document via the <img> tag. Within this tag, the onload field is used to call a function whenever the picture is loaded into the page. We set the onload function to call setTimeout(), which instructs the page to call the refresh() function after 100mS. Thus, after every 100mS, the user-defined refresh function will reload the picture, which then triggers setTimeout() to run again. The result of this simple script is to force a reload of pic.jpg ten times per second.

Note that most browsers will cache the picture so that even when the reload is forced, the old (cached) version of the picture will be used. For this reason we will need to modify the browser settings to force a reload.

9. Build and install the application via the provided script

```
# ./runmake.sh install
```

Boot Linux on the DM6437 DVEVM

10. Start Tera Term Pro from the Windows Desktop

11. Start Code Composer Studio using the Windows Desktop Shortcut

Note: you will need to make sure that the DM6437 dsk is connected (via USB emulation cable) and powered on.

We will use Code Composer Studio to load the linux kernel onto the DVEVM and boot. Note, however, that the filesystem which Linux will use is a network share using a Network File Share (nfs) filesystem located within the Linux environment of the virtual machine.

12. Connect the DM6437 dsk

Debug→Connect (Alt-C)

13. Load the server.pjt project in the lab4b_webcam_linux directory

Project→Open…

navigate to C:\dm6437_1day\lab4b_webcam_linux

select server.pjt

14. Rebuild the project

Project→Rebuild All

15. Reset the CPU

Debug→Reset CPU (Ctrl-R)

16. Load the kernel, vlx and BIOS executable onto the DM6437 DSK

File→Load Program… (Ctrl-L)

Navigate to C:\dm6437_1day\lab4b_webcam_linux\Debug

Load each of the three executables <u>in the following order</u>:

nkern.out	Virtuallogix vlx virtualizer
vmlinux.out	Linux kernel
server.out	DSP/BIOS app and Bootloader program

note: the order is important because server.out needs to be loaded last so that the correct entry point is set. The order of nkern and vmlinux don't actually matter.

17. Run the program

Debug→Run (F5)

You should see feedback as the Linux kernel boots, ending with a login prompt:

```
192.168.1.41 login:
```

If the kernel feedback gives an error before reaching the login prompt, ask your instructor for help.

Run the Application

18. Log into serial linux terminal as root user, no password

19. Change to the /opt/workshop directory

```
# cd /opt/workshop
```

20. Load the audio and video driver modules with the loadmodules.sh script

```
# ./loadmodules.sh
```

note: if you would like to see the contents of this script, type:

```
# cat loadmodules.sh
```

21. Execute the lab4_html.out application

```
# ./lab4_html.out
```

View Webcam Page in Internet Explorer

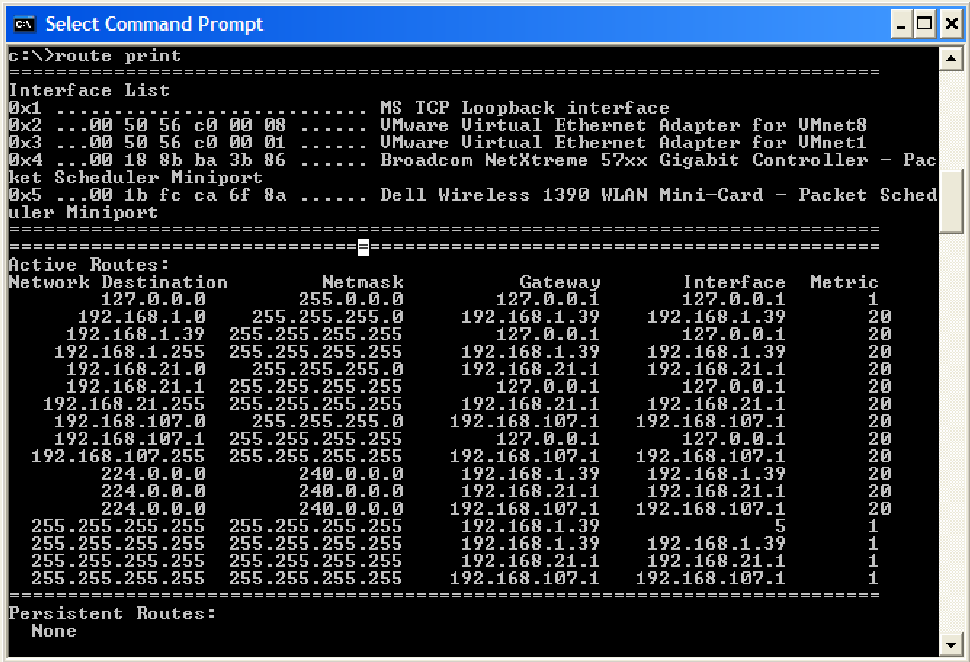
17. Check the Windows routing table

Enter windows terminal mode via start→Run… and entering “cmd”

At the windows terminal command prompt, type:

```
c:\> route print
```

You should see feedback similar to the following (will differ depending on setup):


Select Command Prompt

```
c:\>route print
===========================================================================
Interface List
0x1 ........................... MS TCP Loopback interface
0x2 ...00 50 56 c0 00 08 ...... VMware Virtual Ethernet Adapter for VMnet8
0x3 ...00 50 56 c0 00 01 ...... VMware Virtual Ethernet Adapter for VMnet1
0x4 ...00 18 8b ba 3b 86 ...... Broadcom NetXtreme 57xx Gigabit Controller - Pac
ket Scheduler Miniport
0x5 ...00 1b fc ca 6f 8a ...... Dell Wireless 1390 WLAN Mini-Card - Packet Sched
uler Miniport
===========================================================================
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
        127.0.0.0        255.0.0.0        127.0.0.1       127.0.0.1       1
      192.168.1.0    255.255.255.0     192.168.1.39    192.168.1.39      20
     192.168.1.39  255.255.255.255        127.0.0.1       127.0.0.1      20
    192.168.1.255  255.255.255.255     192.168.1.39    192.168.1.39      20
     192.168.21.0    255.255.255.0     192.168.21.1    192.168.21.1      20
     192.168.21.1  255.255.255.255        127.0.0.1       127.0.0.1      20
   192.168.21.255  255.255.255.255     192.168.21.1    192.168.21.1      20
    192.168.107.0    255.255.255.0    192.168.107.1   192.168.107.1      20
    192.168.107.1  255.255.255.255        127.0.0.1       127.0.0.1      20
  192.168.107.255  255.255.255.255    192.168.107.1   192.168.107.1      20
        224.0.0.0        240.0.0.0     192.168.1.39    192.168.1.39      20
        224.0.0.0        240.0.0.0     192.168.21.1    192.168.21.1      20
        224.0.0.0        240.0.0.0    192.168.107.1   192.168.107.1      20
  255.255.255.255  255.255.255.255     192.168.1.39               5       1
  255.255.255.255  255.255.255.255     192.168.1.39    192.168.1.39       1
  255.255.255.255  255.255.255.255     192.168.21.1    192.168.21.1       1
  255.255.255.255  255.255.255.255    192.168.107.1   192.168.107.1       1
===========================================================================
Persistent Routes:
  None
```

The important entry in this case is the one that reads:

Destination:	Netmask:	Gateway:	Interface
192.168.1.0	255.255.255.0	192.168.1.39	192.168.1.39

(For some lab setups, you may have a persistent route set specifically for destination 192.168.1.41)

This entry indicates that any IP request to the subnetwork 192.168.1.xxx (i.e. any address which, when binary &-ed with the netmask 255.255.255.0, produces 192.168.1.0) will be routed through the interface with IP address 192.168.1.39. Recall that 192.168.1.39 is the IP address that has been statically assigned to the windows network connection to the DVDP board.

If your network routing table is not properly set, you can add a routing entry via:

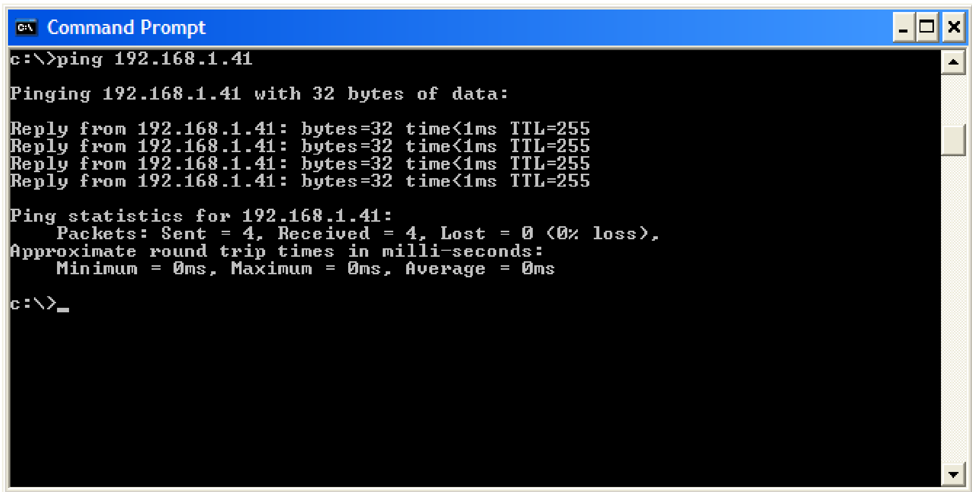
```
c:\> route add 192.168.1.41 192.168.1.39
```

18. Test your network connection to the board via the ping utility

At the Windows terminal command prompt, type:

```
c:\> ping 192.168.1.41
```

You should see a feeback similar to the following:



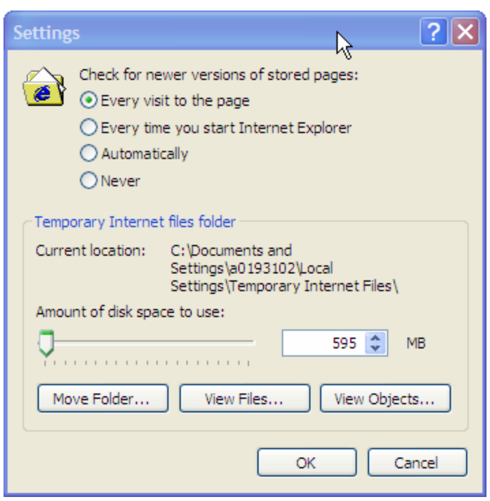
You can close the Windows terminal window when finished.

19. Open the Internet Explorer Web Browser and check the page refresh options

tools→Internet Options…

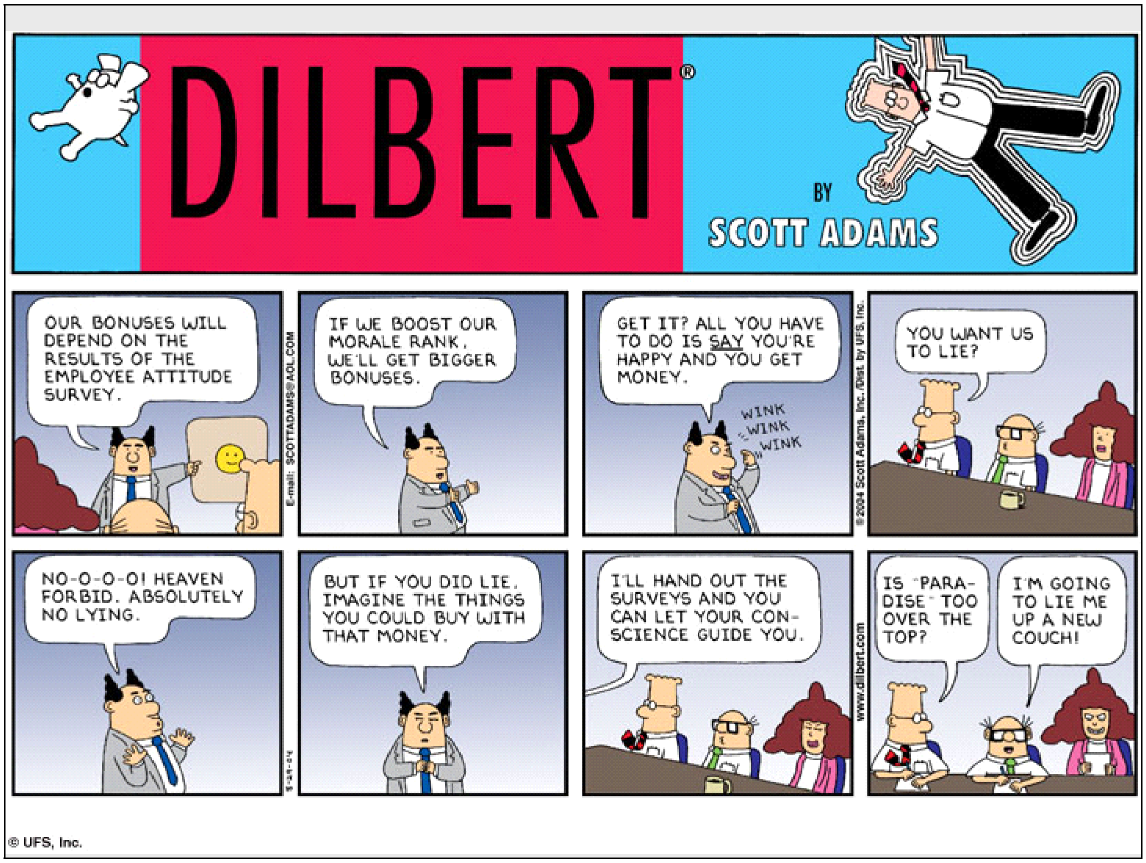
Click the "Settings" button under the Temporary Internet Files heading

20. Make sure the "Check for newer versions of stored pages": option is set to "Every visit to the page" as shown below:



22. Browse the http://192.168.1.41/davinci.htm url

We Hope You Have Had an Informative Day



Please do not forget to fill out an evaluation of the DM6437 1-day workshop before you leave. Your feedback is crucial in helping us improve our course materials.

If you have not been provided with instructions for filling out an evaluation, please ask your instructor.