Introduction

Embedded DSP Systems Considerations with Audio Signal Processing
Introduction
DSP Architecture Background
Real-time Operating Systems (RTOS)
Memory Management
Frameworks
Control Architecture
Device Interfaces
Multi-zone Architecture
Mixed Audio/Video Architectures
Introduction

◆ Why is this topic relevant?
  ◆ More complex SOC than ever before, integration of other functions in the system
  ◆ “MIPS Hogs” (those algorithms you know and love)
  ◆ More complex peripherals and signal routing
  ◆ Time-to-market and Productivity
  ◆ Software quality

◆ What products are we talking about?

◆ What are the challenges for Embedded Audio Systems? How have those challenges evolved over the last 10 years?
SOC

- More complex SOC than ever before, integration ↑
  - Device architectures capable of not only “traditional” signal processing (MAC), but also contain complex peripheral mix (DIR, DIT, serial I/O for data/control, DMA, cache, external memory I/F, USB, SRC, etc.)
Peripherals and Signal Routing

◆ Peripherals
  ■ I/O via:
    ◆ Serial port dedicated for audio (I2S)
    ◆ Memory mapping (USB/MMC/SD/HDD/CD etc.)
    ◆ Control port (SPI/I2C)

◆ Signal Routing
  ■ Topologies: number of input/outputs simultaneously available, possibility of independent streams
  ■ Dependence on peripheral (clock zones)
  ■ Deadlines
    ◆ Real-time deadline (e.g. “pull” from D/A Converter)
    ◆ Non-real-time (e.g. “pull” from disk)
# Topologies

<table>
<thead>
<tr>
<th>Topology</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>single stream</td>
</tr>
<tr>
<td>Y</td>
<td>split stream (with outputs to two peripherals)</td>
</tr>
<tr>
<td>H</td>
<td>2 independent I processing chains (dual zone)</td>
</tr>
<tr>
<td>Z</td>
<td>primary (master) with secondary (slave) input/output</td>
</tr>
</tbody>
</table>
Different tools provided alongside embedded systems help speed prototyping and productization:

- Integrated Development Environment (IDE)
- Advanced Debugger
- Advanced C-compilation Tools
- Operating Systems
- Frameworks
- Standard APIs for algorithm, driver development
- Evaluation Kits/Reference Designs
Consumer Products

- Set-top box
- DVD player
- AV Receiver
- HDTV
- HD-DVD and Blu-ray players
- Networked AV Receiver
- PC multimedia speakers
- Digital Sound Projectors

Technology for Innovators™
More Consumer Products

HDD/Navigation systems

Home Media Servers

PVR/DVD Recorders

HDD Mini/Micro Bookshelf System

CD Ripper to MP3 player/Cell Phone/USB flash/SDMMC card

Virtual CD Changer/Automotive Jukebox

Technology for Innovators™
Embedded Solutions

- General Purpose Processors (GPP/Micro)
- Digital Signal Processors (DSP)
- Application Specific Processors (ASP)
- Application Specific Integrated Circuits (ASIC)
- Field Programmable Gate Arrays (FPGA)
Challenges for Embedded Audio Systems - Then

- Balance performance vs. cost
- Word length
- Mostly fixed-point
- Keep up with features in emerging applications (Pro Audio, A/V Receiver, DVD, HDTV)
- Digital Audio / Analog Video
- Device per function (multi-DSP products common)
- MIPS challenged
- Programming in assembly language
- “Bare-deck” DSP (no software provided)
Challenges for Embedded Audio Systems - Now

◆ Balance performance vs. cost
◆ Rapidly add features and integrate (not always signal processing tasks!)
◆ Floating point more common
◆ Programming in higher-level languages
◆ Complex peripheral mixture
◆ Power
◆ Ease-of-use, mix of tools
◆ Digital Audio / Digital Video
◆ Software and systems provided by silicon providers
◆ Figure out how to use the available MIPS!
DSP Architecture

Embedded DSP Systems Considerations with Audio Signal Processing
Topics

- Audio Serial Port
- DMA Engine
- CPU
- Memories
- Control/Status Interface
Audio Serial Port

◆ Purpose:
  ■ Gluelessly attach to audio devices such as A/D, D/A converters, S/PDIF receiver/transmitters to source and sink audio data to the DSP

◆ Standard Features
  ■ Variety of sample rates
  ■ Good number of input serializers
  ■ Different serial formats (IIS, left-justified, TDM, etc.)

◆ Optional Features
  ■ Error recovery
  ■ Clock generation
  ■ Integrated S/PDIF receiver or transmitter
  ■ Multiple clock “zones”
  ■ Parallel input mode
DMA Engine

◆ Purpose:
  ■ Efficiently move data/code from peripheral-to-memory or memory-to-memory without interrupting the CPU
    ◆ Most applicable for block-based data movement
  ■ For external memory, latency must be considered

◆ Standard Features
  ■ A number of “channels” supporting ping/pong transfers
  ■ Highly configurable – supports chaining
  ■ DMA may be coupled with peripheral or universally available

◆ Optional Features
  ■ Circular buffer support

◆ Good overview of DMA:
  http://signal.ece.utexas.edu/seminars/dsp_seminars/02spring/DMA_TI.pdf
Model System

Digital Signal Processor

Local Mem

CoProc

Periph

Large On-Chip Memory

DMA Proc

off chip data source/sink

off chip memory

M e m o r y   L a t e n c y

a few cycles

~ 10 cycles

50 - 100 cycles

Technology for Innovators™

Texas Instruments
Peripheral signals an event to indicate it has data ready.

DMA Controller reads peripheral data

DMA Controller writes data to memory
◆ Purpose:
  ■ Signal processing functions
    ◆ Filter, scale, transform, encode/decode, correlate, etc.
    ◆ These operations are processor and data intensive

◆ How is a DSP structured?

  ■ Around MAC (core of filter/FFT)
CPU

◆ Then:
  ■ Harvard architecture
  ■ Improved variants to pipeline

◆ Now:
  ■ SIMD (Single Instruction, Multiple Data)
    ◆ Exploits data level parallelism
      ➔ Good for vector processing
      ➔ Often found as a companion to a SISD (Single instruction, single data)
  ■ VLIW (Very Long Instruction Word)
    ◆ Exploits instruction level parallelism
    ◆ Execute multiple instructions/cycle and use simple, regular instruction sets
      ➔ More parallelism, higher performance
  ■ Superscalar
    ◆ Can issue varying numbers of instructions per cycle
    ◆ Can be scheduled either statically by the compiler or dynamically by the processor
  ■ Hybrids of the above
Memories

◆ Purpose:
  ■ Design is predicated on speed
    ◆ Data/instructions must flow into numeric and sequencing sections on every instruction cycle with minimal (or no) delay
    ◆ Memory system focused on throughput

◆ Internal Memory
  ■ RAM
    ◆ Data only
    ◆ Code + data shared
    ◆ Cache (Level 1, 2)
  ■ ROM

◆ External Memory
  ■ SRAM
  ■ SDRAM
  ■ DDR

◆ Memory bus
Control/Status Interface

◆ Purpose:
  ■ Provide an interface to external host (uC) or peripheral devices (ADC, DAC, etc.)
  ■ Typically defined by a physical and logical protocol

◆ Standard Features
  ■ Support industry-standard protocols (IIC, SPI)
  ■ Flexibility to adapt to some customizations of these protocols
  ■ Translation of commands in logical protocol to application registers

◆ Optional Features
  ■ High-speed support for supporting audio data I/O in addition to control/status
  ■ Outbound signaling on status change
Real Time Operating Systems

Embedded DSP Systems Considerations with Audio Signal Processing
A simple loop can very efficiently schedule a small number of functions or applications.

- Each function is called – if it is not ready to run, it exits.

Interrupts provide the ability to handle some small asynchronous events.

A main() loop falls short because it becomes difficult to hand schedule more complex or multiple independent applications.
A simple RTOS contains two fundamental services:

- **Real-time scheduling**
  - OS can schedule an application to execute based on priority (pre-emptive multi-tasking)

- **Synchronization**
  - Enables two independent events to synchronize (semaphore)

A simple RTOS can enable one or more applications to execute independently where each application contains one or more tasks:

- Each task has an associated priority
- Each application waits for available input and output data in order to be ready to run
- The OS scheduler will execute the highest priority task that is ready to run in the system

A simple RTOS starts to fall short because it becomes difficult to handle very complex tasks that operate for unknown duration, many asynchronous tasks and difficult scheduling.
Priority Based OS Scheduling

INT 1
SWI 1
TSK 2
TSK 1
idle

Blocked!
More Advanced RTOS

◆ Includes features like memory protections and more complex scheduling
  ■ Memory protection
    ◆ Works with hardware (MMU) memory management units
      ↗ Enables each application to execute in its own virtual address space – if the application attempts to go outside its address space (bad pointer), OS will prevent and close the application without a crash
  ■ Complex scheduling
    ◆ Offers other scheduling options
      ↗ For example, time-slicing of tasks running at the same priority – required if these tasks execute for unknown amounts of time

◆ These features come at the price of performance (both MHz and memory)
Memory Management

Embedded DSP Systems Considerations with Audio Signal Processing
Many DSP applications can be thought of as requiring only a CPU and memory.

The memory configuration is likely to have a large impact on an application.

Processors will always have some amount of on-chip RAM and usually depend on external memory devices for most of the storage.
Example DSP Memories

Cache reflects contents of a larger memory further away from CPU

CPU

L1 Cache

SRAM

L2 Cache

SRAM

SDRAM, DDR, etc.

External Memory Device

CPU has the same access time to this level of memory i.e. number of wait states

SRAM is for general application use
Memory Performance/Cost Tradeoff

Memory

- Access Speed
- Size
- Cost/Byte

DSP
CPU

- L1 Cache
- SRAM

- L2 Cache
- SRAM

- SDRAM, DDR, etc.

External Memory Device

Technology for Innovators™
Accessing Memory

Cache fills load the memories closer to the CPU

CPU will directly access these memories very quickly (single cycle)

DMA engine will generally load the non-cache DSP memories
Separate Program and Data Cache

Common L1 cache configuration – separation of data and program
With this memory configuration, applications can be developed without memory placement issues – performance is subject to cache properties and external memory access.
Possible Audio DSP Configuration

- External Memory Device
- DSP (Audio Example)
- SDRAM, DDR, etc.
- CPU
- Program Cache
- SRAM
- ROM
- Direct CPU access
- Cache fills
- DMA moves

Technology for Innovators™
A DSP application will achieve peak performance when all software components (program code, constants, data, etc.) are stored in the fastest available memory.

How is this achieved?
DSP Applications

- Typical DSP applications for audio are generally algorithms that are block based
  - Examples include audio compression algorithms and PCM audio processing
- Program flow
  - Wait for input
  - Wait for output
  - Execute
  - Repeat
- Executing
  - Can be thought of as a complex function call
  - Finite amount of processing time is required (fairly predictable)
General Algorithm Memory Requirements

- Constant Data
- Program Code
- Software Stack

Input Buffer → Algorithm → Output Buffer

- Scratch Buffer
- Persistent Buffer

In some cases, the input/output buffer can be the same buffer.
DSP applications consist of the following sub-components:

- **Program code**
  - Program instruction code

- **Software stack**
  - Memory used for function calls, local variables, etc.

- **Constant data**
  - Constant tables, etc.

- **Scratch data**
  - Work buffers used during program execution

- **Persistent data**
  - State data that must be preserved between application executions

<table>
<thead>
<tr>
<th></th>
<th>Execution</th>
<th>Waiting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Code</td>
<td>Max</td>
<td>None</td>
</tr>
<tr>
<td>Software Stack</td>
<td>Max</td>
<td>Min</td>
</tr>
<tr>
<td>Constant Data</td>
<td>Max</td>
<td>None</td>
</tr>
<tr>
<td>Scratch Data</td>
<td>Max</td>
<td>None</td>
</tr>
<tr>
<td>Persistent Data</td>
<td>Max</td>
<td>Max</td>
</tr>
</tbody>
</table>
Sub-Component Relocation

- Relocate components before execution
- Use DMA moves or cache fills/flushes

Internal Memory:
- Program
- Constant
- Persistent
- Stack
- Scratch

External Memory:
Sharing Internal Memory

To enable sharing of the internal memory resources – both App ‘A’ and ‘B’ must run at the same priority and run to completion.

Scratch must be the max of App ‘A’ and ‘B’.
PCM Processing Example - Graphic EQ

◆ Graphic EQ
  - Parallel bank of Bandpass filters (outputs summed)
  - Fixed Q & Center frequency
  - Typically based on Direct Form I filter (biquad)
## Graphic EQ Memory Requirements

### Direct Form I

<table>
<thead>
<tr>
<th>States</th>
<th>$4 \times 4 \text{ bytes} = 16 \text{ bytes} $</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coefficients</td>
<td>$5 \times 4 \text{ bytes} = 20 \text{ bytes} $</td>
</tr>
</tbody>
</table>

### Table

<table>
<thead>
<tr>
<th>10-band EQ</th>
<th>Bytes</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Code</td>
<td>$x$</td>
<td>ISA specific</td>
</tr>
<tr>
<td>Software Stack</td>
<td>$x$</td>
<td>ISA specific (per instance)</td>
</tr>
<tr>
<td>Constant Data</td>
<td>200</td>
<td>Coefficients</td>
</tr>
<tr>
<td>Scratch Data</td>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>Persistent Data</td>
<td>200</td>
<td>States + gains (per channel)</td>
</tr>
</tbody>
</table>

*Assume all data requires 32-bits*
Two Concurrent EQs – No Resource Sharing

*Assume 128 PCM samples are processed in-place each 32-bits
Two Concurrent EQs – Shared Resources

Program x Bytes

Constant 200 Bytes

Stack y Bytes

Persistent 200 Bytes

IO Buffer 512 Bytes

EQ ‘A’

EQ ‘B’

Common

Internal Memory

External Memory

Significant reduction in more expensive internal memory usage
Maximizing Internal Memory Usage

◆ Use of internal memory can be maximized by applying the following concepts:

1. If a DSP application is run to completion, all the internal memory consumed during execution can be reused by other applications while the current application waits to execute again.

2. If a DSP application is interrupted during execution, the interrupting application cannot reuse the same internal memory.

3. All software sub-components can be stored in external memory between application executions.
Memory Allocation

◆ General thoughts on memory allocation in real-time systems
  ■ Static memory placement is fine
  ■ Malloc is even okay
  ■ Freeing is what gets you into trouble

◆ Freeing memory can cause fragmentation
  ■ Garbage collection in a real-time system is difficult to do
Typical Audio DSP Data Flow

**Input**
- DIR or A/D (I2S)
- Mc-ASP, ESAI, etc.

**Input Buffer**

**Audio Processing (CPU)**
- Input Classify/Format
- Decode
- PCM Processing
- Encode
- Output Format

**Output Buffer**

**Output**
- Mc-ASP, ESAI, etc.
- DIT or D/A (I2S)
Audio Processing “Main Loop” (CPU)

- Input Buffer
  - Input Classify/Format
  - Decode
  - PCM Processing
  - Encode
  - Output Format
  - Wait for Input Available

- Output Buffer
  - Wait for Output Free
PCM Processing

Input → Decode → Post Processing → Encode → Output

from decoder → DEM → PL → NEO → BM → DEL → ML → to encoder

- De-emphasis
- Dolby ProLogic IIx
- DTS Neo:6 2-channel Matrix
- Bass Management
- Delay
- MIPS Load
Audio Frameworks

◆ Provide Infrastructure for Audio Processing and I/O
◆ Customize for *Product Differentiation*
  ▪ Add/Modify/Delete Post-Processing Algorithms
  ▪ Change Component Parameters
◆ Standard Interfaces Allow *Component* Changes with little or no *Framework* Changes
  ▪ More Efficient Development
  ▪ Better Market Responsiveness
  ▪ Improved Software Quality
Parameter-Driven Audio Frameworks

◆ Parameters
  ■ Feature Sets
    ◆ Decoders
    ◆ Post-Processing Modules
  ■ Processing Topologies
  ■ I/O Protocols, Formats, Channels
    ◆ Audio
    ◆ Control
  ■ Memory usage
    ◆ I/O Buffers, Heap, Overlaps
◆ Run directly from ROM
  ■ Patch to add/fix
Silicon Scalable Family of Audio DSPs maximizes design reuse and reduces development costs.

AUREUS™ AUDIO DSP

Performance Audio Framework (PA/F)

CUSTOMIZATION
F/W, System S/W, and standard algorithms allow customers to focus on value added features.

AUDIO ALGORITHMS
Comprehensive list of optimized audio decode, encode, post processing and I/O algorithm implementations.

FRAMEWORK + SYS S/W
Enables high-quality, complete audio system while delivering flexible design platform.

AUREUS™ AUDIO DSP
Scalable Family of Audio DSPs maximizes design reuse and reduces development costs.

CUSTOMIZATION

AUDIO ALGORITHMS

FRAMEWORK + SYS S/W

AUREUS™ AUDIO DSP

Technology for Innovators™
◆ Processing is organized into Tasks (TSK)
  ■ Task priorities facilitate real-time artifact-free audio
◆ I/O is performed using SIO drivers
  ■ Stacking SIO drivers used for additional “logical” processing
◆ Memory management
  ■ IRAM & SDRAM heaps (creation only — no fragmentation)
  ■ One stack per task and one system stack (quasi-dynamic memory)
  ■ Global data
    ◆ DSP/BIOS configuration
    ◆ Static function tables
**PA/F Task & Idle Function Organization**

- **Audio Stream Task**
  - Calls `audioStream1Task()`. Main audio stream task (audio I/O, decode/encode, ASP).

- **Alpha File Processing Task**
  - Calls `AlphaFileProcessingTask()`. Performs alpha-code processing of messages from external controller (e.g., SPI, I2C).

- **Alpha Interval Processing Task**
  - Calls `alphaIntervalProcessing-Task()`. Implements “at boot” and “at time” processing of alpha code.

- **Idle Task**
  - Calls `audioStream1Idle()`. Performs background management of main audio stream task.
  - Calls `DAP_watchDog()`.
    - Monitors McASP device status and initiate/performs cleanup/recovery.
  - The idle task is a standard part of, and managed by, DSP/BIOS. It provides various system services, such as MIPS measurement, as well as user-defined services.
What is the Audio Stream Task?

- Initialization of components
- Control and definition of audio streams
Multiple Tasks

Real Time Operating System

<table>
<thead>
<tr>
<th>Task 1</th>
<th>User-defined Task</th>
<th>System Stream 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio Stream Task 1</td>
<td>HDD Control</td>
<td>RTA Tools</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AFP</th>
<th>ACP</th>
<th>AIP</th>
</tr>
</thead>
</table>
TMS320 DSP Algorithm Standard (xDAIS)

- TI’s standard guideline for DSP S/W development
- Standardized way of coding algorithms
- Allows plug-and-play portability/flexibility
- Object-oriented style framework
- Re-entrant, multiple-instance
- CCS supports easy xDAIS coding
- All PA/F components are built to xDAIS standard
PA/F Algorithms — Interface

- Based on xDAIS’ ALG & (newer) ALGRF
- Application-specific public functions added
  - E.g., for ASPs: reset(), apply()
- PAF_ALG enhancements
  - Enable safe “common” memory sharing between mutually exclusive algos.
PAF_ALG Common Memory Types

From paf_ialg.h

- PAF_IALG_COMMON_STEREO_DECODE_PROCESSING
- PAF_IALG_COMMON_MULTI_DECODE_PROCESSING
- PAF_IALG_COMMON_FRONT_SURROUND_PROCESSING
- PAF_IALG_COMMON_RESERVED0
- PAF_IALG_COMMON_BACK_SURROUND_PROCESSING
- PAF_IALG_COMMON_ENCODE_PROCESSING
- PAF_IALG_COMMON_VIRTUAL_PROCESSING
- PAF_IALG_COMMON_BASS_MANAGEMENT
- PAF_IALG_COMMON_DELAY_MANAGEMENT
- PAF_IALG_COMMON_RESERVED1
- PAF_IALG_COMMON_RESERVED2
- PAF_IALG_COMMON_RESERVED3
- PAF_IALG_COMMON_CUSTOM1
- PAF_IALG_COMMON_CUSTOM2
- PAF_IALG_COMMON_CUSTOM3
- PAF_IALG_COMMON_CUSTOM4
- PAF_IALG_COMMON_MEM0 IALG_SCRATCH
- PAF_IALG_COMMON_MEM1
XDAIS compliant C code using above template
- Serial commands communicate with rest of system and algorithm
- Serial commands sent either at initialization time, regular intervals, from external host, or from user-defined task
Achieving High (MIPS) Efficiency

◆ Re-use efficient code “building blocks”
  ■ Especially Filter algo. and Filter functions
  ■ Potentially minimizes ROM, RAM, MIPS
◆ Keep the (8) ALU functional units busy
  ■ Combine operations in cascade or parallel (e.g. multi-channel)
◆ Minimize number of read/write “passes” through data to reduce read/write instructions
  ■ Don’t divide processing into too many post processing functions
◆ Locate frequently used code in IRAM
◆ Typically use in-place processing for memory saving
◆ Block Processing ➔ Natural fit with …
  ■ Compressed-bitstream audio (input frames, output blocks)
  ■ DSP Architectures
    ◆ Register setup cycles can be amortized over block of audio samples in efficient loop kernels
    ◆ Enable use of DMA for buffered I/O transfers
Control Architecture

◆ Allows run-time monitoring and control (“user interface”)
◆ Direct communication with individual system components
◆ Use DSP to organize and assimilate information to be reported
◆ Regularize multi-task, DSP/micro, DSP/DSP communications
◆ Consider synchronization requirements
Application — Conventional A/V Receiver

Front Panel

Control Microprocessor

Audio DSP

Messaging (SPI or I2C)
System Information

- Sample Rates
- Listening Modes
- Channel Configurations
- Input, Decode, Output Processing
- Volume Control
- Input/Output Switching (IOS)
Application — Cost-Reduced A/V Receiver

Front Panel

Audio DSP

Control Task

Messaging (Inter-Task)

Audio Task

PA/F
Application — High-End A/V Receiver

Front Panel

Control Microprocessor

Audio DSP

Secondary DSP

Messaging (SPI or I2C)

Messaging (SPI or I2C)
PA Messaging (PA/M)

- Multiple message types provide
  - Application flexibility
  - Communications bandwidth efficiency (8/16/32/vector data)
- Most messages perform memory Read or Write
- Remote function calls
- Presets facilitated by customer-defined “shortcuts”
- OEM Extensions implicitly supported
- Future messaging extensions possible
- Host/DSP, Intra-, Inter-DSP communications
- Symbolic control of algorithm parameters/status
### PA/M — Write Messages

<table>
<thead>
<tr>
<th>Length (16-bit wds)</th>
<th>Message Type</th>
<th>MSB</th>
<th>LSB</th>
<th>MSB</th>
<th>LSB</th>
<th>MSB</th>
<th>LSB</th>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Function Invocation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Type 0, Write</td>
<td>Function num.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Encapsulation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Type 1, Write</td>
<td>Len. (Words)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8-bit Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Type 2, Write</td>
<td>Beta ID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>16-bit Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Type 3, Write</td>
<td>Beta ID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>32-bit Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Type 4, Write</td>
<td>Beta ID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;= 1</td>
<td>Extended Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Type 5, Write</td>
<td>Ext. Sub-type</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2+Length/2</td>
<td>Var.-Length Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Type 6, Write</td>
<td>Beta ID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4+Length/2</td>
<td>Physical Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Type 7, Write</td>
<td>Access Type</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Function 0**: Calling type 8-bit data

**Function 1**: Word 1 [Word 2] [Word 3]

**Function 2**: Byte Offset 8-bit data

**Function 3**: Byte Offset 16-bit data

**Function 4**: Byte Offset 32-bit data

**Function 5**: [Word 1] [Word 2] [Word 3]

**Function 6**: Byte Offset [Len. ( Bytes)] Byte 1 [Byte 2] [Byte 3] [Byte 4]

**Function 7**: Length (Bytes) Physical Address Data …
Each addressable component has a unique “ID” (0-255)
Device Interfaces

◆ Strive for orthogonality between I/O and Processing
  ■ Allow for changes to one without affecting the other
  ■ Layered (Stacking) I/O Drivers
    ◆ Physical
    ◆ Logical
◆ Physical I/O Driver Parameters
  ■ Audio I/O
    ◆ I2S, TDM
    ◆ Clock direction
    ◆ Channel interleaving
    ◆ Map physical pins to logical channels
  ■ Control/Status I/O
    ◆ SPI, I2C
    ◆ Clock settings
◆ Logical Parameters
  ■ What content types are recognized? (Auto-detection)
  ■ IEC header analysis
  ■ Restart conditions based on I/O
Multiple Audio Zones

Embedded DSP Systems Considerations with Audio Signal Processing
Topics

◆ Why is a multi-zone architecture relevant?
◆ How to realize multi-zone?
Why is Multi-Zone Relevant?

◆ Multi-Zone Home Entertainment System
  - Multiple sources, multiple sinks using a centralized processor
Why is Multi-Zone Relevant?

- Two-Way Viewing Angle (TWVA) TV
- “Directed” Audio is necessary for dual sources to arrive at distinct listening positions coordinated with the video.
Why is Multi-Zone Relevant?

- Jukebox/Network/Disk-based Media Systems
How to Realize Multi-Zone on a DSP?

◆ Hardware
  ● Requires support of peripherals
    ◆ Multiple, unrelated clock zones
    ◆ DMA events

◆ Software
  ● Requires use of operating system to manage resources
    ◆ Would you want to hand schedule this?
  ● Memory reuse
  ● Thread protection
    ◆ Drivers
    ◆ DMA resources
  ● Scheduling algorithm
Multi-Zone Scheduling

◆ System:
  ■ Two asynchronous audio threads at same priority
◆ Option 1: Run each thread until blocked (co-operative multitasking)
  ■ + Simple
  ■ – May miss real-time (unless additional latency introduced)
  ■ – Separate memory space might be required for each thread
◆ Option 2: Use time-slicing to allow equal time (pre-emptive multitasking)
  ■ + Good for threads with similar processing load
  ■ + Able to use Simple RTOS
  ■ – Separate memory space for each thread
  ■ – Overhead for context switch
  ■ – May leave MIPS on the table
◆ Option 3: More advanced options (deadline scheduling)
  ■ Use more Advanced RTOS
Multi-Zone Realization

◆ I/O concerns
  ■ Buffering required to keep up with peripherals when task is inactive
  ■ Latency
  ■ Simplify scheduling by synchronizing threads (asynchronous rate conversion)

◆ Memory
  ■ Cache and scratch (inefficient with pre-emptive scheduling)

◆ RTOS
  ■ Good method to realize pre-emptive scheduling
Multi-Zone Messaging

◆ Multiple Streams
  ➞ Multiple Components
  ➞ Unique IDs for each component

Stream #1

Input & Auto Detect → DEC-ode → PL → NEO → CUS → BM → DEL → ML → ENC-ode → 8-ch PCM Output

Stream #2

PCM Input → DEC-ode → PL → NEO → OEM → BM → DEL → ML → ENC-ode → Network Output
A/V Systems

Embedded DSP Systems Considerations with Audio Signal Processing
A/V Systems

- Architectures that can support concurrent audio/video are possible
  - Hardware differences include:
    - External memory is mandatory
      - Image buffers are large
    - Video related peripherals
      - Display controllers, Analog video DACs, Digital video outputs, OSD
  - Software differences include:
    - Video processing has higher memory and MHz requirements compared to audio processing
    - Video does not require as much precision as audio
Example Digital Video System

Demux
MPEG PS/TS
ASF

Video Decode
MPEG-1/2/4,
WMV

Audio Decode
AC-3, DTS,
MPEG, WMA

Video Post
Resize, I2P,
Deblocking

Audio Post
EQ, Matrix,
VTBF, SRC

AV Sync

Example configuration:
- DVD-V: MPEG Program Stream with MPEG-2 Video + AC-3/DTS Audio
- ATSC HDTV: MPEG Transport Stream with MPEG-2 Video + AC-3 Audio
- Internet Video: ASF with WMV Video + WMA Audio
Compressed Video Decoders

- External memory accesses are more random compared to audio so the on-chip cache performance will degrade – keep the video data from flowing through the cache
- Large amount of memory access can be required
  - MPEG-2 video decoder will peak at ~75 MB of external memory access per second for standard definition

Image buffers can be several MB at even standard definition (720x480 @ 30 fps)
AV System Considerations

- A relatively complex subsystem like this will require several tasks each with different priorities – RTOS can handle task scheduling
- Most of the components in this system have fairly linear access which will provide good cache performance. This limits the amount of memory moves required by the system