Developing A/V Software for Consumer Media Products

Workshop Outline

The consumer media device
  • The big picture
Developing A/V software
  • Software subsystems
  • What’s special about codec software?
  • Numeric considerations
  • Optimization techniques
Testing
Trends and conclusions
Big Picture
Consumer media device is complex system; several software and hardware subsystems to integrate, e.g.,
- Software: player, codec(s), RTOS
- Hardware: GPP/DSP, DMA, I/O, memory

Key Big Picture Considerations:
Processor selection
- Numeric support: floating-point or fixed-point
- Development tools, e.g., IDE, HLL compiler, assembler
- Availability of OS, off-the-shelf SW, programmers, etc.

Reference implementations
- Hardware reference design
- Software components (RTOS, codec, player)

Design, development, and testing strategies
- Optimization
- Development board for early testing of software
- How testing will be performed, what resources will be utilized, and to what end: i.e., what’s good enough

Integration
- Hardware + software
- Real-time testing
Workshop Outline

The consumer media device
- The big picture

Developing A/V software
- Software subsystems
- What’s special about codec software?
- Numeric considerations
- Optimization techniques

Testing

Trends and conclusions

Software Subsystems

Primary software subsystems include:

- Player: GUI (play, stop, rewind), host helper func’s
- Codec(s): MPEG-2, WMA, RV9
- I/O: DAC, USB
- Real-Time Operating System: VxWorks, WinCE, Palm
Developing A/V Software for Consumer Media Products

Key Software Considerations

Player
- Port to target OS/hardware platform (Just compile? Unlikely)

Codec
- Starting point?
- Floating-point to fixed-point migration (if necessary)
  - Numeric considerations
- Optimize for speed, memory use, power, etc.

RTOS
- Add/remove features/device drivers

Software integration
- Player + codec(s) + RTOS

Testing
- Audio/video quality (test vectors)
- Real-time performance

Software Development

Common division of labor
- Separate teams for each subsystem
  - Teams work together to integrate and test

INTEGRATION: PLAYER + CODEC + RTOS
  Real-Time Performance Testing

TEAM 1: PLAYER
- Port to RTOS
- Helper func’s
- UI Testing

TEAM 2: CODEC
- Numeric Considerations
- Optimization
- A/V Quality Testing

TEAM 3: RTOS
- Port to Platform
- Device Drivers
- I/O Testing

Hot spot
- Codecs can pose significant development challenge
Codec Software Development

Not like other kinds of software development:

- Extreme computational demands
- Algorithm attributes
- Data access attributes
- Memory bandwidth requirements
- Testing and validation requirements
- Resource constraints
- Numeric fidelity requirements
- Standards
- Real-time requirements
- Reliability
- Specialized and complex processor architectures

Optimization is essential!

Numeric Considerations

Many important and interesting topics, e.g.,
- Float-to-fixed migration
- Numeric fidelity
- Data types
- Error propagation/analysis
- Precision and dynamic range
- Block floating-point
- Floating-point emulation
- Signal scaling
- Rounding modes

(Focus topics)
**Float-to-Fixed Migration**

Codec reference code available in different flavors, e.g.,
- Floating-point unoptimized
- Fixed-point unoptimized/optimized

Hardware platform usually fixed-point
- Fixed-point is cheaper, potentially faster, more energy efficient
- Few embedded processors support efficient floating-point

Floating-point reference code + fixed-point platform =
- Float-to-fixed migration

Float-to-fixed challenges
- Slows time-to-market
- Numeric fidelity tricky to maintain

---

**Numeric Fidelity**

**Definition**
- Numeric fidelity = accuracy of numbers

**Why do we care?**
- Computer arithmetic is not error-free
- Errors usually lead to:
  - Noise
  - Overflow—sudden change from max to min value
    - In video, e.g., white to black (bright to dark)
    - In audio, e.g., pop and click

**How do we maintain numeric fidelity?**
## Maintaining Numeric Fidelity

Minimize error & error propagation

- Analyze where errors (could) occur, and:
  - Determine tolerable level of error, e.g., one unit in last place
  - Choose alternative algorithm topology
    - To alter dynamic range requirements & error propagation
  - Choose appropriate data types and sizes
    - With sufficient dynamic range & precision for your signal
  - Scale signals
    - To prevent overflow or maintain precision
  - Select rounding modes
    - To minimize error propagation

## Data Type Choice

Understand the attributes & implications of data types

<table>
<thead>
<tr>
<th></th>
<th>Fixed point</th>
<th>Floating point: IEEE-754 SP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>16 bit: 1 part in 64K</td>
<td>24-bit mantissa: 1 part in 16M</td>
</tr>
<tr>
<td></td>
<td>24 bit: 1 part in 16M</td>
<td></td>
</tr>
<tr>
<td></td>
<td>32 bit: 1 part in 4G</td>
<td></td>
</tr>
<tr>
<td>Dynamic range</td>
<td>16 bit: 96 dB</td>
<td>8 bit exp: 1500 dB</td>
</tr>
<tr>
<td></td>
<td>24 bit: 144 dB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>32 bit: 192 dB</td>
<td></td>
</tr>
<tr>
<td>Ease of use</td>
<td>Tricky</td>
<td>Easy</td>
</tr>
<tr>
<td>Processor cost</td>
<td>Cheap</td>
<td>Expensive</td>
</tr>
</tbody>
</table>
Developing A/V Software for Consumer Media Products

Data Types: Fixed-Point

**Signals are often represented as a fraction in a fixed-point number:** S.3 or Q.3

<table>
<thead>
<tr>
<th>Fractional Value</th>
<th>Sign</th>
<th>b₂</th>
<th>b₁</th>
<th>b₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.875</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0.75</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0.625</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0.5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.375</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0.25</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0.125</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Data Types: Fractional Add, Multiply

Fractional add ($N + N \geq N$-bit):
- Behaves the same as integer add
- Generates no error, if no overflow

Multiply ($N \times N \geq N$ bit)

```
short X, Y, Z;
Z = ((long) X * (long) Y) >> 15;
```

```
short X, Y, Z;
Z = X + Y;
```

---

Data Types: Floating-Point

```
S Exp Mantissa
```

Value = $S \times (1 + \text{Mantissa}) \times 2^{\text{Exponent}}$

<table>
<thead>
<tr>
<th>SNR (dB)</th>
<th>Precision roughly constant over dynamic range</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1500</td>
<td>Granular-Noise Region</td>
</tr>
<tr>
<td>0</td>
<td>High-Quality Region</td>
</tr>
<tr>
<td>6 dB</td>
<td>Overload Region</td>
</tr>
<tr>
<td>144</td>
<td>Signal Power (dB)</td>
</tr>
</tbody>
</table>

© 2003 Berkeley Design Technology, Inc.
Numeric Support

Fixed-point DSPs typically support fractional and integer fixed-point data types in hardware
• Fractional multiplication includes shift

Fixed-point GPPs typically do not support fractional data types in hardware
• Shift must be explicit

GPPs usually 32-bit, DSPs usually 16-bit, with some 32 bit support

Fixed-point DSPs and GPPs can emulate floating-point, but usually at a high MIPS cost

Floating-point processors are more expensive, use more energy, and have slower clock rates

No fractional support in ISO/ANSI C (coming soon?)

Optimization

Possible performance metrics:
• **Execution speed**
  • Processor-independent/different optimizations
    • High level language (HLL) optimizations
    • Hand code assembly for best performance
  • Memory access optimizations
    • Avoid cache, or L1, “thrashing”

• Memory usage (code size and data size)
  • May conflict with optimizations for speed

• Power consumption
  • Minimize off-chip memory accesses

→ Profile → Analyze → Optimize
Profiling Goal: ID S-rate Operations

80/20 Rule:
• 20% of software responsible for 80% of execution time, and vice-versa

Functions can be classified based on invocation rate:

<table>
<thead>
<tr>
<th>Class</th>
<th>Invocation rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-rate (Initialization rate)</td>
<td>≤ 1 time per second</td>
</tr>
<tr>
<td>K-rate (Control rate—parameter updates, etc.)</td>
<td>~10 to 1,000 times per second</td>
</tr>
<tr>
<td>S-rate (Sample rate)</td>
<td>~10,000 to 100,000 times per second</td>
</tr>
</tbody>
</table>

At each level:
• Rate differs by 2 to 3 orders of magnitude vs. adjacent levels
• Execution cost increases by 2 to 3 orders of magnitude
I-rate and K-rate efficiency not too important, but S-rate efficiency very important

S-rate Operation: FIR Filter Kernel

C implementation of FIR kernel

\[ y[n] = \sum_{k=0}^{T-1} x[n-k]h[k] \]

\[
\begin{align*}
N &= 40; \\
T &= 16; \\
\text{for} \ (n=0; \ n<N; \ n++) \ { } \\
\text{for} \ (k=0, \text{SUM}=0; \ k<T; \ k++) \ { } \\
&\quad \text{SUM += } x[n-k] \times h[k]; \\
&\quad y[n] = \text{SUM};
\end{align*}
\]
Analysis: Compiled ARM7 FIR Filter

B     |L1.80|
|L1.20| MOV  a4, #0
|L1.32| MOV  lr, #0
|      | B    |L1.60|
|L1.60| SUB  v1, ip, a4
|      | MOV  v1, v1, LSL #1
|      | LDRSH v1, [a1, v1]
|      | MOV  v2, a4, LSL #1
|      | LDRSH v2, [a2, v2]
|      | ADD  a4, a4, #1
|      | MLA  lr, v2, v1, lr
|L1.60| CMP  a4, v3
|      | BLT  |L1.32|
|      | MOV  a4, ip, LSL #1
|      | STRH lr, [a3, a4]
|      | ADD  ip, ip, #1
|L1.80| CMP  ip, v4
|      | BLT  |L1.20|

ARMCC compiler known to be very good.

\[
N=40; \quad T=16; \\
\text{for } (n=0; \ n<N; \ n++) \{ \\
\quad \text{for } (k=0; \ SUM=0; \ k<T; \ k++) \{ \\
\quad \quad \text{SUM += } x[n-k] \times h[k]; \\
\quad \}
\]
\[
\quad y[n] = \text{SUM};
\]

18 instructions in kernel

4 branch instructions

2 instruction load sequence

Single instruction equivalent: LDRSH v2, [a2, a4, LSL #1]
Developing A/V Software for Consumer Media Products

Analysis:

**Algorithmic Transformation: # 1**
Reorder coefficients: single index for x & h

```
x[n-k]  x[i]  h[k]  h[i]
```

```
|L1.32| SUB      v1,ip,a4  |L1.32| SUB      v1,ip,a4
|     | MOV v1,v1,LSL #1   |     | MOV v1,v1,LSL #1
|     | LDRSH v1,[a1,v1]    |     | LDRSH v1,[a1,v1]
|     | MOV v2,a4,LSL #1    |     | LDRSH v2,[a2,v2]
|     | LDRSH v2,[a2,v2]    |     | LDRSH v2,[a2,v2]
|     | ADD a4,a4,#1        |     | ADD a4,a4,#1
|     | MLA lr,v2,v1,lr     |     | MLA lr,v2,v1,lr
|L1.60| CMP a4,v3           |L1.60| CMP a4,v3
|     | BLT |L1.32| BLT |L1.32|
```

Developing A/V Software: Optimizations

Analysis:

**Algorithmic Transformation: # 2**
Count down rather than up, branch if index \( \geq 0 \)

```
x[i]  x[i]  h[i]  h[i]
```

```
|L1.32| SUB      v1,ip,a4  |L1.32| SUB      v1,ip,a4
|     | MOV v1,v1,LSL #1   |     | MOV v1,v1,LSL #1
|     | LDRSH v1,[a1,v1]    |     | LDRSH v1,[a1,v1]
|     | MOV v2,a4,LSL #1    |     | LDRSH v2,[a2,v2]
|     | LDRSH v2,[a2,v2]    |     | LDRSH v2,[a2,v2]
|     | ADD a4,a4,#1        |     | ADD a4,a4,#1
|     | MLA lr,v2,v1,lr     |     | MLA lr,v2,v1,lr
|L1.60| CMP a4,v3           |L1.60| CMP a4,v3
|     | BLT |L1.32| BLT |L1.32|
```

© 2003 Berkeley Design Technology, Inc.
Implementing Optimizations

Combining:
- Branch reduction
- Single index offset load
- Re-order coefficients
- Count down loop

Loop:
```
add r7, r2, r4, lsl #2          ; r7 points to in(i)
mov r6, r5                     ; j = ntaps-1
mov r10, #0                    ; sum = 0
innerLoop:
    ldr r8, [r7, r6, lsl #2]    ; r8 = in(i+j)
    ldr r9, [r3, r6, lsl #2]    ; r9 = coef(j)
    mlal r10, r8, r9, r10      ; sum += in(i+j)*coef(j)
    subs r6, r6, #1            ; j--
    bpl innerLoop              ; loop until j < 0
str r10,[r1, r4, lsl #2]       ; out(i) = sum
subs r4, r4, #1               ; i--
bl loop                       ; loop until i < 0
```

11 instructions in total, compared to 18 in compiled code: ~40% less

Memory Access Optimization

Video Processing
- Video frame much bigger than typical L1 memory (cache or SRAM)

- Simplified frame buffer and L1 memory
Memory Access Optimization

Default processing sequence
- Operation 1 on entire frame
- Operation 2 on entire frame
- Etc.

Load 2nd block of frame data
- Each cache line misses—very expensive—or
- DMA overhead for SRAM L1
Memory Access Optimization

Load 3rd block of frame data
- Again, cache misses or DMA overhead
- Note: still Operation 1

Memory Access Optimization

Load Nth block of frame data
- Have moved entire frame in and out of L1
  - Slow and power-hungry
Memory Access Optimization

Repeat block load and process pattern for
- Operation 2
- Ratio of memory access overhead to processing overhead very high

Optimization: process subset of frame
- Subset stays resident in L1, cutting external memory accesses
Memory Access Optimization

Load next block of frame data, etc.
• Ratio of memory access overhead to processing overhead much lower

Workshop Outline

The consumer media device
• The big picture
Developing A/V software
• Software subsystems
• What’s special about codec software?
• Numeric considerations
• Optimization techniques
Testing
Trends and conclusions
Testing

Hardware/development platform
- Vast data I/O capability
  - Capture digital output for testing

Codec software
- Audio and video quality
  - Test vectors, reference codecs
- Operating modes
  - Sample rates, frame sizes, bit rates, etc.

System level (hardware + software)
- Real-time performance under worst cases
  - Data-dependent execution time
  - Dynamic processor features
  - Interrupts enabled

Hardware/ Development Platform

A/V codecs consume and produce vast amounts of data:
- Compressed bit streams up to ~10 Mbps
- Uncompressed output streams up to 120 Mbps

Simulation model usually far too slow
- Real hardware is needed

Development board must have means to
- Supply large test vectors
- Capture potentially even larger output
Codec Software: A/V Quality

Difficult to measure quality in context of “lossy” compression algorithms
- Intentionally not bit-exact

Quality measured via reference codec + test vectors
- Supplied test vectors may not adequately stress fixed-point implementations
- May need to create tests that exercise full potential dynamic range of algorithm
  - Requires in-depth understanding of underlying algorithm

Codec Software: Operating Modes

A/V Codecs typically have several operating modes
- MPEG-1 Layer III audio has:
  - 14 bit rates
  - Three sampling rates
  - Four channel configurations

All valid combinations must be thoroughly tested
- Standard reference test vectors probably not sufficient
  - MPEG-1/2 Layer III audio has only one “compliance” test vector, so test vectors must be created
System Level: Real-Time

Real-time performance is not optional
Processor is often underpowered
  • Careful codec optimizations can pull underachievers up to real-time performance
  • But must verify worst-case performance

System Level: Worst-Case Testing

Most demanding operating mode
  • Highest bit rate, sample rate, most channels
  • Interrupts enabled and active (UI and I/O)

Most demanding data
  • Codecs have data dependent execution paths
  • Processors often have data dependent operations, e.g., multiplication

Worst case real-time stress requires:
  • Test vector which ensures worst case execution path
  • Worst case inputs to data dependent operations
**Workshop Outline**

The consumer media device
- The big picture

Developing A/V software
- Software subsystems
- What’s special about codec software?
- Numeric considerations
- Optimization techniques

Testing

Trends and conclusions

---

**Conclusions**

Successful A/V codec software development:

- Demands knowledge of the application, algorithms, and processor, and mastery of a wide range of skills and tools
- Requires careful selection of numeric data types and close attention to numeric fidelity
- Typically requires aggressive optimization in order to meet tough real-time deadlines
- Requires a well-thought-out testing strategy!
Trends & Conclusions

Trends

Processors are getting faster & compilers are getting better

- But newer A/V codecs are more demanding than previous generations

Optimized software libraries are more common

- Signal processing function level, e.g., FIR, IDCT
- Application level, e.g., A/V codec

Trends & Conclusions

Trends

Heterogeneous processors:

- Processor core + programmable logic
- Multi-processor SoCs

Heterogeneous processors may change how application workload is handled

- Proposal: off-load compute-intensive S-rate operations to custom logic or specialized DSP
- Reality: inter-processor communications and synchronization load can be deadly
Developing A/V Software for Consumer Media Products

Resources:

Processor Architecture/Software Optimization/OS:

Numerics:
Digital Signal Processing, Alan V. Oppenheim & Ronald Schafer

Audio/Video/Speech Compression:
Digital Compression for Multimedia—Principles and Standards, Jerry D. Gibson, Toby Berger, Tom Lookabaugh, Dave Lindbergh, Richard L. Baker

For More Information...
www.BDTI.com

Free Information
- White papers/presentation slides on
  - DSP software optimization
  - Streaming media implementation
  - Processor architectures and performance
  - Digital audio compression
- Article reprints on DSP-oriented processors and applications
  - EE Times
  - IEEE Spectrum
  - IEEE Computer and others
- comp.dsp FAQ