A Digital Television Receiver Constructed Using A Media Processor.

Chuck Peplinski
Philips Semiconductors
Sunnyvale, CA
Chuck.peplinski@sv.sc.philips.com

Torsten Fink
Philips Semiconductors
Sunnyvale, CA
Torsten.Fink@sv.sc.philips.com

Abstract:

This paper describes the implementation of the audio subsystem of an ATSC receiver on a Very Long Instruction Word (VLIW) processor. The audio system consists of independent, reusable, and exchangeable components which are connected to one another through a streaming software architecture. Sophisticated software development tools and a clean interface structure enabled a very short, hierarchical implementation and test period.

Section 1: Overview of an ATSC DTV Receiver

In the United States, a new standard for digital video broadcasting supporting high definition video and multi-channel audio is scheduled to replace the traditional NTSC analog television system in the coming years. Apart from the brilliant picture quality it delivers by employing MPEG-2 compression, it provides stunning Home Theater surround sound based on the Dolby Digital™ technology, also known as AC-3. The characteristics of an Advanced Television System Committee (ATSC) receiver are determined industry standards such as those given in [1].

The audio portion of this receiver must handle all of the features of AC-3, including output processing such as bass redirection, and possibly matrixed stereo decoding. The fundamental document familiar to engineers who wish to have their audio system certified as compliant by Dolby Labs is the Licensee Information Manual, or LIM, [9].

In the system constructed by the authors, all of this processing is implemented as software written in a high level language (C), and compiled to run on a VLIW machine [3], [4], [5]. This has resulted in a system that is very flexible, extensible, and very easy to maintain. The VLIW processor also handles demultiplexing of the MPEG transport stream, video decoding, and control processing for a commercially available television. The framework used to construct this system is generally applicable to any digital television broadcast receiver, and to many other types of audio processing. The software module that implements these functions is referred to as the TriMedia DTV Audio System.

While it is not unusual to implement a system like this, it is uncommon to construct a system for use in a consumer device using a high level language and a real time operating system.

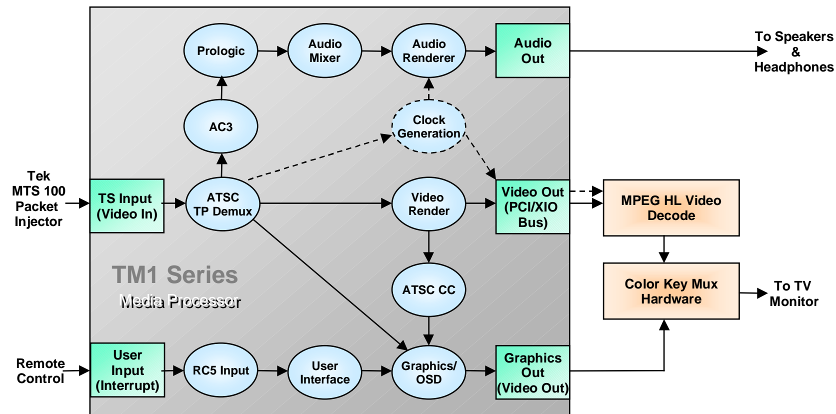


Figure 1: DTV system block diagram

Figure 1 shows a block diagram of the ATSC receiver discussed in this paper. All bubbles in the gray box represent software components running on the VLIW processor. This paper concentrates on the upper part which constitutes the DTV audio system. A detailed block diagram of the audio sub system is given at the end of section 1.

Design Choices:

While all ATSC receivers must decode AC-3 audio streams, there are still many choices to be made when designing a receiver. Will the output be stereo only, or a full six channels? Will amplifiers be built into the TV, or will the user be expected to provide his own amps? Is ATSC audio the only source, or will other sources have to be handled? Will the digital portion of the system also support the audio for analog TV? Will satellite TV be supported? Is integrated DVD support a feature?
The system under discussion is designed to handle each of these cases.

Compressed Audio Formats

The American ATSC system specifies Dolby's AC-3 system for compressed audio. Other compressed audio formats are likely to be useful in a television receiver. Some satellite broadcast systems use MPEG audio which is also used in European broadcast systems. The Japanese digital TV system has chosen MPEG AAC [8] to encode its audio.

Outputs

When it is only required to output two channels, the system can be greatly simplified. It is then expected that sophisticated users will supply an external audio decoder, and the ATSC receiver simply provides a stereo downmix of the program along with the un-decoded audio stream on an S/P DIF connection. The S/P DIF connection, described by IEC60958 [6], can carry compressed audio in accordance with IEC61937 [7]. But when the product is a complete TV set, it is desirable to include all of the nice features that audio for digital TV entails. This means the ability to decode and reproduce six channels, with amplifiers and the associated user interface.
Other common requirements are a headphone output and a VCR output. Each of these should be independently controlled.

Speaker Modes

TV's equipped to receive Dolby AC-3 signals must support a number of speaker modes. Since surround speakers are supposed to be slightly behind the listener, they cannot be built into a TV. This and other practical considerations mean that TV's must have a user interface to allow a user to describe the configuration of the speaker system. This interface controls the number of speakers, the placement of the

speakers, and the capabilities of the speakers. When the TV has built in speakers, some of these parameters may be fixed. But when a line level output is provided, full control must be offered to users.

The "speaker size" specification is used to control the bass redirection system, and this is discussed below. The "speaker position" parameter controls the delay applied to the various channels. AC-3 streams are recorded with the assumption that the speakers are placed on a circle with the user at the center. In practice, this is not the case. The rear and center speakers are moved closer to the user. Delay is used to compensate for this. Since sound travels at approximately 1000 feet per second, delay is changed by 1 millisecond per foot.

Bass Redirection

Bass redirection is discussed in section 9.8 of the Dolby LIM [9]. The concept behind bass redirection is to ensure that as much low frequency energy as possible is delivered to the user, given a particular speaker configuration. It is easy to encounter problems when implementing bass redirection. One way of looking at the problem is this. TV's are traditionally a "consumer" system, and the audio quality of an AC-3 system is that of a "high quality" or "professional" system. When a TV includes AC-3 audio, the expectation for audio quality is raised. If you build the same old television audio system, it will fail to meet the requirements that go with AC-3. A fundamental stumbling block is the wide dynamic range possible in a digital system. When bass is redirected, the low frequency signal levels become very high. If care is not taken, the reference signal level becomes very low, and subject to noise pickup.

The TriMedia DTV audio system supports a number of bass redirection options. In order to use these effectively, the TV's user interface must allow the user to specify the speaker configuration.

Volume and Tone Control

When a TV includes amplifiers and speakers, then standard features like volume and tone control are required. These features can be implemented in analog hardware, or with digital processing. The system under discussion supports these features in the digital domain. While it may seem like a simple trade off, it is easy to lose dynamic range and end up with a system that does not meet the quality specifications required for the certification of the audio sub system of the ATSC receiver by Dolby Labs. This subject is covered in more depth in a following section.

Mixing in Auxiliary Channels

Digital TV sets are likely to include multiple audio sources. A user might want to be able to mix audio from a secondary source with the primary audio channel. This secondary source might be audible feedback to the user interface (beeps), a second program channel (picture in picture, or PIP), or content from a data service (such as a web page). It is also possible to connect a microphone to support Karaoke. In any case, a sophisticated audio system should make allowances for at least one secondary channel.

Analog Television Considerations

During the transition period between analog and digital television broadcasting, the two modes will co-exist, and users will expect to be able to switch easily between the two modes. This requirement can be met, and the digital processing that is available for DTV can be applied to enhance the analog signals. In the audio system, the obvious example is matrix stereo decoding. The user has more than two speakers, so the audio system is called upon to generate the other channels.

Digital input channel

Consumer video devices like DVD's will often be used with digital TV's. When the TV includes a high quality audio system, it makes sense to use it to decode the compressed audio stored on the disk. In order to do this, an S/P DIF input is required.

The Environment

The audio system exists as a collection of software components running on a general-purpose media processor. The media processor is supported by an open software development environment. Since the media processor also handles video decoding, user interface, and control functions, it is worth a few words to explain how these elements interact.

Overview of the VLIW Processor

To better understand some of the performance considerations that will be discussed later, it is necessary to be familiar with the architecture of the TriMedia VLIW media processor family. All existing TriMedia processors consist of an internal 32-bit high-speed data bus, which is connected to external SDRAM. Attached to this highway are chip internal DMA interface blocks, the VLIW CPU, and coprocessor blocks.

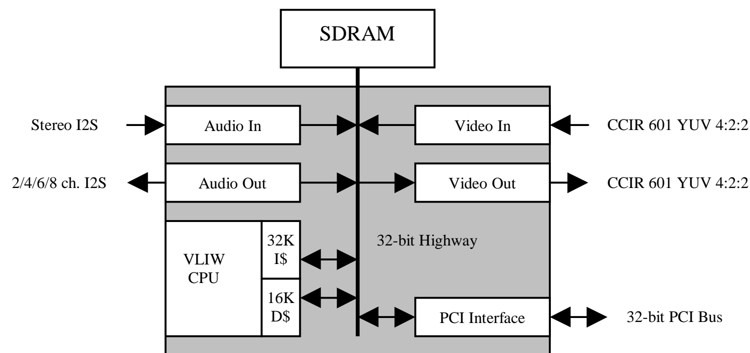


Figure 2: Block Diagram of the TriMedia Processor

Figure **2** depicts the parts of the TriMedia processor, which are of interest in this context. Other interface blocks are omitted. The entire chip is synchronized via its internal highway. All data is transferred in 64 byte blocks from or to the SDRAM. Interface blocks do not communicate directly with one another or the VLIW CPU. Between the VLIW CPU and the SDRAM, there are data and instruction caches. These caches are required because the internal highway does not provide the required memory bandwidth and latency to permit the CPU to work directly with the SDRAM. They provide the CPU with an instruction throughput of 3.7 gigabytes per second and a data throughput of over one gigabytes per second in a 133 MHz chip.

The VLIW CPU itself starts the execution of up to 5 RISC like operations in every clock cycle. Each of these operations is fed into one of 27 functional units. The functional units are connected to a register file with 128 general purpose 32-bit registers. Register 0 and 1 contain the hardwired values 0 and 1. Each of the functional units can be thought of as a small RISC processor that loads two operands from the register file, computes a result and writes the result back to the register file. The latency of a functional unit is the time it needs to complete an operation. This depends on the complexity of the operation and ranges from one clock cycle for simple arithmetical operations such as shifts to 17 clock cycles for floating point divisions. All functional units, except for the floating point divider, have a recovery time of 1 clock cycle. This means, that the unit can start the execution of a new operation in every clock cycle even if the latency of the operation takes more than one clock cycle. This is possible, because these units are fully pipelined.

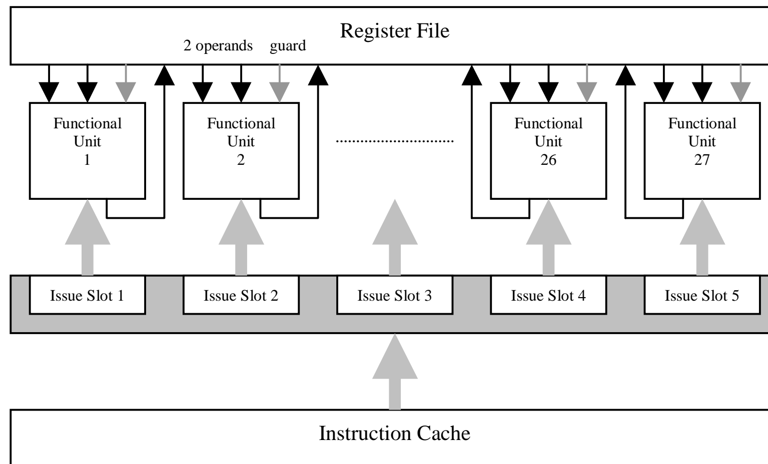


Figure 3: Simplified Internal Structure of the VLIW CPU

There are a number of restrictions on which functional unit can be utilized from each issue slot. Table 1 gives an overview on the properties of the functional units. All operations are performed on data stored in the register file except for the load/store operations. They implement the exchange of data between the data cache and the register file.
The DSP units implement special SIMD and saturation logic operations. (SIMD stands for single instruction multiple data and means that the data of the 2 32-bit operand registers is treated as 4 8-bit or 2 16-bit vectors.) A SIMD operation performs several sub-operations in parallel on the elements of these vectors. For the audio system presented, however, these instructions are not used because high quality audio requires a higher resolution than 16-bit. SIMD instructions can be used for speech processing, video and graphics applications.

Functional Unit	Quantity	Latency	Recovery Time	Issue Slots
Constant	5	1	1	1 2 3 4 5
Integer ALU	5	1	1	1 2 3 4 5
Load/Store	2	3	1	4 5
DSP ALU	2	2	1	1 3
DSP MUL	2	3	1	2 3
Shifter	2	1	1	1 2
Branch	3	3	1	2 3 4
Integer/Float Multiplier	2	3	1	2 3
Float ALU	2	3	1	1 4
Float Compare	1	1	1	3
Float SQRT/DIV	1	17	16	2

Table 1: Functional Units of TM 1X00 Processors

The hardware complexity of the VLIW CPU is relatively low because it does not perform run time scheduling and dependency checking as Super Scalar and Super Pipelined CPU's do. These tasks are implemented in the C/C++ compiler, which generates VLIW code by exploiting fine grain parallelism. It is also possible to program the CPU in assembly by obeying restrictions like those imposed by Table 1. This, however, was not required for the implementation of the software modules for the ATSC receiver. The code generated by the compiler can hardly be outperformed by hand written assembly code.

Some instructions, such as those of the DSP units, can not be generated from ANSI C. These instructions can be used in function call fashion at the C level using *custom ops*:

```
#include "custom_defs.h"
int A,B,C;
A = DSPOPERATION( B, C);
```

A close look into the table reveals another interesting property of the VLIW processor. It has three branch units and it is obvious that only one branch can be executed at a time. The trick is that the operations of the TriMedia instruction set are performed conditionally. This means that when an operation is issued a third register, called the guard, determines if the operation is executed or not. A compiler can exploit this feature to avoid jumps by generating code that performs the operations of two branches in parallel with inverted guard registers.

```
int abs_dif( int a, int b)
{
    int c;
    if( a < b)
    {
        c = b - a;
    }
    else
    {
        c = a - b;
    }
    return c;
}
```

```
(* cycle 0 *)
IF r1   isub r5 r6 -> r8      (* alu/Op6 *),
IF r1   ijmpt r1 r2           (* branch *),
IF r1   igtr r6 r5 -> r7      (* alu/Op4 *),
IF r1   nop,
IF r1   nop;

(* cycle 1 *)
IF r7   isub r6 r5 -> r8      (* alu/Op5 *),
IF r1   nop,
IF r1   nop,
IF r1   nop,
IF r1   nop;
```

The example above shows how the compiler produces code for the if-else construct without any jumps. In clock cycle 0, a – b is calculated and the result is stored in register 8. This operation is always executed because its guard is register 1, which is always 1. In the same clock cycle, it is evaluated if b is greater than a and the result of this comparison is stored in register 7. In clock cycle one, a is subtracted from b if register 7 contains the logical value true. This is only the case when b is greater than a, see clock cycle 0. The jump issued in clock cycle 0 is taken after clock cycle 3. It is the return from the function and it has nothing to do with the if-else construct.

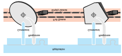
Tools used for the Development of the ATSC Receiver Software

The presented software can be written in a high level language and deployed as a commercial product only because of the quality of the development tools that are available. The code can also be re-targeted to run on any member of the TriMedia processor family because there is a strong interaction between the actual processor architecture and the development tools. All hardware dependencies are resolved at compilation time by the help of a so-called machine description file. The TriMedia processors themselves provide enough horsepower to perform audio, video, graphics, control, and communications tasks simultaneously in their DSP cores.

The audio system and the other modules used in the DTV receiver are written in the C language. Several (or many) programmers work on the system. It is imperative that the various portions of these programs co-exist peacefully. A real time operating system (RTOS) is a critical tool used to reach this end. Like any multi-tasking operating system, the RTOS allows programmers to “imagine” that they have the whole machine to themselves. The OS takes care of task switching and scheduling. The RTOS used in our ATSC receiver reference design uses a priority based scheduling algorithm.

The processor includes an on-chip PCI interface, and software is provided to operate in a mode hosted by a PC. This high speed interface can be used to speed debugging in several ways. Libraries implementing POSIX standard text I/O are available, so that functions like printf() and fread() are always available in hosted mode. A low overhead form of the printf() function (“DP()”) is provided to write to a local memory buffer avoiding the communication overhead that comes with printf(). This memory buffer can be quite large, and can be quickly dumped while the program is running. In this way, relatively long traces can be constructed to monitor the program’s behavior.

When it is necessary to trace program execution, examine variables, or monitor the state of the RTOS, a sophisticated source level debugger is available. The debugger can work in hosted mode over the PCI interface, or it can be operated over a JTAG connection for “stand alone” development.



A number of profiling tools are available. It is very easy to monitor the usage of CPU time at the granularity of the RTOS task. When more detailed profile data are required, the tool chain includes a profiling tool that can be used to identify "hot spots" for further optimization.

Finally, the fact that the source code is written in C makes it reasonable to compile the code for another target processor. This trick can be used to make use of the most sophisticated debugging tools available on PCs and UNIX workstations.

Software Architecture

The audio system for this television receiver is constructed within the framework of a software architecture optimized for streaming multimedia data. This framework allows software modules to be developed independently because it clearly defines the interface between these components. A programmer can easily integrate diverse modules as they connect in a common way. This software architecture is known as the TriMedia Streaming Software Architecture (TSSA) [4]. Several dozen TSSA components are now available, and they are used extensively in the design of the complete ATSC receiver. TSSA uses a data driven design. The RTOS provides a foundation that allows the system to be factored into independent tasks that communicate using queues and semaphores. A given task will sleep until data is available, process the data, send it along, and sleep again.

Figure 4: TSSA data flow

Scheduling is handled by the priority based scheduler of the RTOS kernel. Priorities are generally set using a rate-monotonic rule, as described by Layland and Liu [10]. A priority based scheduler is chosen over a deadline scheduler because of its predictable behavior in overloaded conditions. High priority tasks continue to meet their deadlines, while low priority tasks are deferred.

The connection points on a component are referred to as pins. The connection between tasks is implemented using the operating system's queue construct. Each connection is made up of a pair of queues, with packets full of data carried in one queue, and empty packets in the other. The empty packets signal to the sender that the associated data memory can be recycled. The number of packets that circulate in this

pair of queues determines the amount of buffering between the two tasks. With a lot of buffering, a process can handle long latencies and transient processor over-allocation in other tasks. But large amounts of buffering can also lead to longer delays as the buffers of data flow through the system. This is a tradeoff that is controlled by the system designer.

Data passes between tasks in packets, and packets use a standard structure defined by the architecture. Packets are passed by reference, so that data is not copied unnecessarily. Packets may also be time stamped to facilitate synchronization. These data packets may pass audio, video, or other forms of data. A standard means is provided to identify the format of the data. TSSA components are capable of adjusting their processing to the format of the data packets they receive. TSSA is currently supporting 13 different audio data types ranging from MPEG audio to PCM and 37 subtypes.

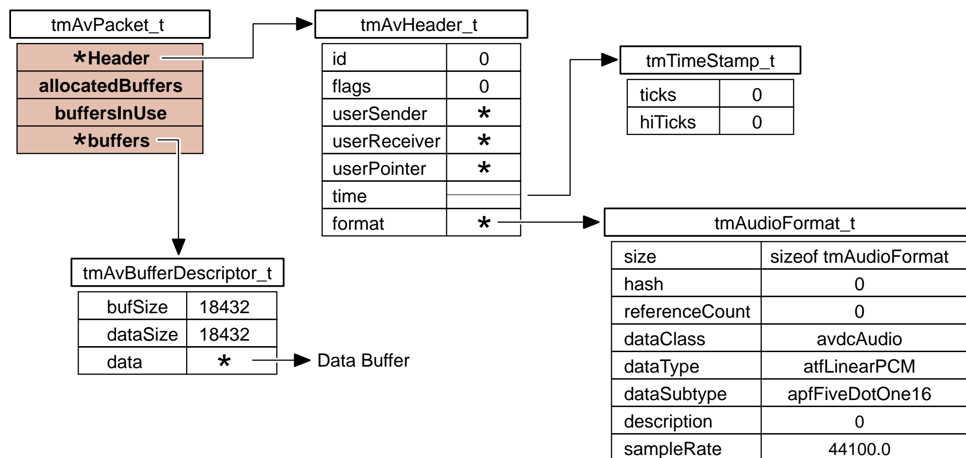


Figure 5: Composition of an Audio Data Packet

Figure 5 depicts the hierarchical composition of a TSSA data packet. This example shows an audio packet containing 6 channel 16-bit PCM samples. It carries an audio frame of 1536 samples.

The software architecture distinguishes between three different classes of components: digitizers, renderers, and processors. Digitizers are producers of data packets. They have one or multiple output pins. In most cases they represent the connection of a series of streaming components to a physical input hardware. Renderers on the other hand provide only input pins. They are the last elements in processing chains and establish the connection to the output hardware. Processors are hardware independent. They typically perform filter-like processing on input data in order to produce output data.

All TSSA components provide a set of common functions regardless of their nature. These functions are:

- GetCapabilities: returns a structure describing the capabilities of the component. Among other things, the structure contains the number of inputs and outputs and the supported data types at the respective pins.
- Open: reserves an instance of the component for the user. The function returns an instance handle and allocates memory for the new instance of the component. If no instance is available, an error is returned.
- Close: frees the resources used by the instance to be closed.
- GetInstanceSetup: returns a so-called instance setup structure which can be used to configure the component. On initialization, this structure contains default settings.
- InstanceSetup: takes an instance setup structure as parameter and configures the component according to the user’s wishes. The instance setup structure also describes the connections between this and other components.

- Start: this function initiates the data streaming of the component. A new thread of execution is launched, and the component begins to receive data packets at its active inputs. It processes this data and sends the packets to its active outputs.
- Stop: terminates the data streaming of the component.
- InstanceConfig: while the first seven functions are mandatory this one is optional. If supported it can be used to send commands to the component while the component is running (after start).
- Additional interfaces might optionally be provided, as appropriate.

The Start function of a processor component is designed in a hardware and operating system independent way. This is achieved through the use of a set of callback functions (function pointers). Every interaction of the component is implemented via these callback functions. The following callbacks are used by TSSA components:

- Datain: is called by the component when it either wants to receive a full or return an empty input packet.
- Dataout: is called by the component when it either wants to receive an empty or send a full output packet.
- Error: is called by the component when an internal error occurred during operation.
- Memalloc: is called when the component wants to allocate dynamic memory.
- Memfree: is called when dynamically allocated memory is to be freed.
- Progress: is called when the component has made a certain amount of progress. The usage of this callback function is highly dependent on the properties of the component. In most cases it is used to implement some sort of synchronization between the component and a supervising module. Examples are given in later sections.
- Completion: can be called by the component when it has stopped its processing and leaves the Start function.

TSSA provides a default implementation for each of these callback functions. An application programmer can install proprietary callback functions during instance setup. While it is not recommended to override the datain and dataout function, an application specific error and progress function should be installed. They allow the application to execute application specific code in the context of the component.

Within the development of the ATSC receiver a number of existing TSSA compliant audio components were integrated into a new ‘super’-component known as the Audio System. Like the building blocks that are used internally, the audio system also provides the set of the eight TSSA functions as its programming interface. In the following chapters the audio system and its elements are introduced.

Performance Considerations

Many factors influence the performance of a software system like the ATSC audio system, and the question is often asked whether the software architecture imposes a performance penalty. It is relatively easy to compute the penalty.

Task switch overhead = task switch time * switch rate + cache penalty

Switching tasks on the TriMedia pSOS system takes on the order of 700 processor cycles. If this happens 1000 times per second (typical in a complete DTV application), then task switching requires 0.7 million instructions per second (MIPS). This is a very reasonable price to pay for the amount of structure and flexibility provided by the RTOS. Nevertheless, it is in our interest to keep this number of task switches low because more task switches can contribute to a higher cache penalty. TSSA provides a number of tools to do this. The additional cache penalty depends on the nature of the software modules used in the system and on the granularity of the data packets travelling through the system. In the case of a task switch between complex modules like an AC-3 decoder and an MPEG-2 video decoder the entire instruction and data cache must be re-filled, which could lead to a significant performance hit. The impact of this “cache-thrashing” can be minimized by ensuring that task switches only happen at natural breaks in processing. This is discussed in a following paragraph about latency.

Performance measurement:
The CPU includes a cycle counter that can be read with a single instruction. This makes it very easy to measure how much time a basic task has taken. The operating system provides a mechanism to call a function every time that it switches tasks. Together, it is easy to measure the amount of time taken on a per-task basis. A typical trace, available at any point during development, looks like this:

```
schedules = 665 time: 3.0
idle:         48.31
demux task:    6.15
AC-3  task:   28.16
mixer task:    5.92
video task:   10.73
```

This report is inserted into the DP log once per second, and it measures the relative CPU usage of the various tasks running on the system. The idle task runs when no other task is ready to run. This example reports, that 665 task switches occurred during the third second of program execution. In this second, 48.31% of the time was spent in the idle task.

Measuring performance on a per task basis is a high level of granularity, but the fact that it comes almost for free increases its utility. When a finer granularity of performance measurement is required, the profiler is the correct tool. To use the profiler on a program that is running on the hardware, you have to accept some impact on performance. Because the profiler effectively traces all execution, it takes some CPU power. But in most cases, the extra CPU time is available, or can be made available for the purposes of profiling.
The various reports produced by the profiler can identify how many cycles are spent on a per-function, or on a "per-tree" basis. A tree is the smallest unit of scheduling, so this report tells you exactly where your time is being spent. The following example shows the processor load caused by the 9 most expensive decision trees of the AC-3 decoder. All cycle numbers must be multiplied by 1000.

```
Treename                              Executions   Total Cycles    %      I$ Cycles  D$ Cycles
--------------------------------------             -------------------   ---------  ---------
____ac3Window_d_DT_2                      393984        8594     10.77        118       2560
____ac3Unpmants_DT_67                     494532        6810      8.53        132        250
____ac3Unpmants_DT_10                     211356        4969      6.22          1        107
____ac3Unpmants_DT_36                     494532        4738      5.93         24          0
____ac3Clr_downmix_DT_4                   525312        3697      4.63         51        492
____ac3Window_d_DT_6                      393984        3277      4.11          5        908
____ac3Imdct_long_DT_10                   202496        2974      3.73         41        206
____ac3Imdct_long_DT_29                   202496        1866      2.34         11        230
__memmove_DT_2                            201781        1788      2.24         68        509
----------------------------------------------------------------------------------------------
total/average                            6428027       79829    100.00      11474      10090

NOTE: execution times are in x1000 cycles.
NOTE: threshold is set to 0.0220000

exact total machine cycles is 79828992 cycles.
```

This profile report enables programmers to easily identify bottlenecks in their software. It displays the number of times a decision tree was executed, how many cycles were spent in this decision tree, the relative percentage, and how many cycles the CPU was stalled due to instruction and data cache misses.

The only thing that the profiler cannot reliably report on when running on the hardware is cache behavior. To deeply understand the cache performance of your algorithm, the correct tool is the simulator. Since the simulator is significantly slower than the real hardware, it is best utilized to examine particular sections of code that are known to be processing-intensive. For this reason, it is often advisable to develop a test bench designed for optimization. The test bench will provide a representative set of data to the core of the algorithm so that unnecessary control and I/O code do not clutter the optimization process. Section 2 of this paper gives a more detailed insight into the optimization of a particular algorithm. Other examples are found in the TriMedia "cookbook." [11]

Latency Considerations

It is a well known fact that low latency can be purchased at the cost of extra processing time. This is because there is an overhead associated with the setup and tear down of any given task. This applies to the work that is done inside of a single loop, and it also applies to the work done by a TSSA task. Tasks must be constructed to do enough work so that the setup and tear down time are amortized over the work of the task. A longer latency on the other hand means that more buffering is required, and therefore more memory. Therefore, latency of a component must be chosen with care because it determines the processor load as well as the memory consumption. Aside from the block processing that is universally employed, a TSSA system brings up some other issues.

Are Tasks Preemptible?

Multi-tasking operating systems like the one used in our system support task preemption. Preemption is appropriate when a task is not designed for use in a real time system, or when the task can run for long periods without requiring new data. But the data driven tasks that make up a TSSA system do not meet these criteria. TSSA tasks block to request data, and then pass the data on. The amount of processing that can be done on a given frame is bounded, and it is known to take less time than it will take to display the data. In cases like this, preemption actually decreases performance. The optimal number of task switches has already been coded into the TSSA module by its programmer and is an inherent feature.

It is possible to declare whether any task is preemptible when the task is created. But a collection of non-preemptible tasks are likely to result in less time spent task switching.

Packets with multiple buffers

TSSA includes a feature that is designed to reduce the number of preemptions required. As an example, consider the transport stream demultiplexer. The demux constantly works with a stream of transport packets containing 188 bytes. If each of these transport packets were sent as a single TSSA data packet into the queues that connect TSSA modules, then each packet would present a scheduling opportunity to the operating system. Instead, the TSSA packet structure supports "packets" carrying multiple "buffers." This means, that multiple transport packets are combined to one TSSA data packet. Features like this allow us to reduce scheduling activity.

A function based interface, or a task based interface?

TSSA allows a user to rely on a task based interface for many operations. But there are many times, when a function based interface is also useful. TSSA supports this, as well. A typical TSSA object has a structure like this:

```
While (1) {
        Get_empty_output_packet(&outPacket);
        Get_full_input_packet(&inPacket);
        ProcessData(&inPacket, &outPacket);
        Send_full_output_packet(&outPacket);
        Send_back_empty_input_packet(&inPacket);
}
```

This buffer handling code is part of the TSSA start function. But the ProcessData() function that is used in the core can also be used to access the capabilities of the module without starting a task. This capability can be useful when a user wants to avoid using the operating system. Or more likely, the ProcessData() function can be called from another task, and this will reduce the latency of the system, as well as reducing the number of memory buffers that are required. An example of this is the Dolby ProLogic® decoder that is available on TriMedia. When a task based interface is used in the context of the audio system, buffers totaling to 30ms of delay can be required. This can cause an A/V sync problem when the input is audio from an analog TV. Using the function based interface instead saves memory, lowers latency, and lowers the amount of scheduling overhead. The cost is in the greater complexity of the mixer component that must call ProLogic as a function.

Audio System

The audio system is defined to contain all of the audio processing that is required in an ATSC receiver. The audio system is made up of a number of components, and a user could have access to these components directly. However, the audio system packages all of the setup and reconfiguration of the various components into one, simple interface. The application programmer using the audio system does not have to be concerned with the details of the audio system's internal operation. A high level interface is provided. The application only needs to connect its stream as a source, and set the audio system into the correct mode.

Inside the audio system, each independent component is represented by a thread of execution. Output is handled by an "audio renderer." Input can originate from the MPEG-2 transport stream multiplexer in compressed form, or it can come from an analog or digital (S/P DIF) input by way of an "audio digitizer". AC-3 data is decoded to PCM by one task. A "mixer" handles output processing. A "noise sequencer" task can be made active during system setup and alignment. Matrixed stereo decoding is handled by the "ProLogic" task. Finally, a task waits for events that would cause the system to be reconfigured. This might occur with a change in the input stream's format. Other factorings of this system are possible. This factoring is chosen to simplify the construction and maintenance of the independent tasks. In addition to testing the system as a whole, each module can easily be tested in isolation.

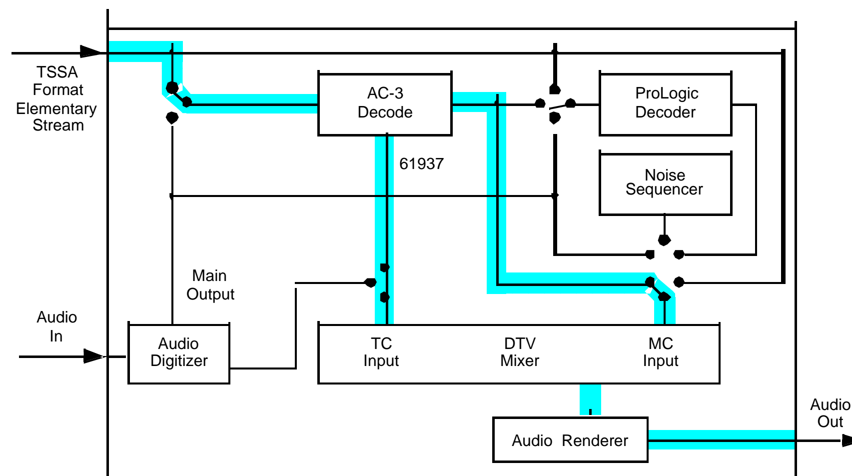


Figure 6: Basic DTV audio system block diagram. Decoding AC-3 from demultiplexer

Audio Renderer

The audio renderer is the last element in the data flow of the audio system. It represents the TSSA interface to the on-chip audio output hardware. The renderer is implemented as an interrupt service routine. The basic work of the ISR is to update the registers controlling the audio DMA hardware. A new packet of data is retrieved from the full data queue and installed into the hardware. The address of the data buffer just played is sent back in the empty queue. Packets generally contain 256 samples, which equals 5.33ms if the sample rate is 48 kHz. The consumption of data (based on the audio clock) is one of the fundamental drivers of the system's timing. Because of this, the audio renderer is a key player in the process of AV synchronization.

The renderer uses the standard TSSA "progress" callback function to allow an application to install code that is run in the interrupt service routine.

Mixer

The audio renderer is fed by a mixer. In the case of this TV, the mixer functions as a post processor and a multiplexer. But any processing that is not part of the basic decoder occurs in the mixer. The mixer

includes bass redirection and volume control. It can also include tone control. The mixer runs as an independent task. It is a typical TSSA object, in that its buffer control loop looks much like the one described above. The use of the multi-tasking operating system allows the programmer of the mixer to effectively ignore the behavior of the rest of the system. The mixer implements a number of signal processing functions, in each case maintaining more than 20 bits of precision. This code was originally written using floating point data, but recent optimizations have changed this all to 32-bit fixed point arithmetic.

The functions implemented in the mixer are:

- A pre-amp volume stage allows the main input to be controlled separately from an auxiliary input.
- A parallel headphone channel is included to format the data for the audio renderer, where eight channels are interleaved.
- Treble, bass and loudness controls are modeled on the traditional analog prototypes. Biquad sections are optimized to the TriMedia architecture using the IMULM "custom operation." A more detailed discussion of the optimization of IIR audio filters on TriMedia is included in the cookbook [11].
- The bass redirection filters are enabled based upon the speaker configuration specified by the user. If full range speakers are available, then no bass redirection is required. But the speakers built into a TV do not generally have the power to deliver full bass response, so provision is made to redirect the low frequency energy from the full range channels into a subwoofer. The crossover filters are designed to the Linkwitz-Riley [12] criteria.
- Finally, the individual channels are delayed to compensate for differences in speaker placement.

Section 4 of this paper discusses the design process that guarantees sufficient dynamic range throughout these calculations.

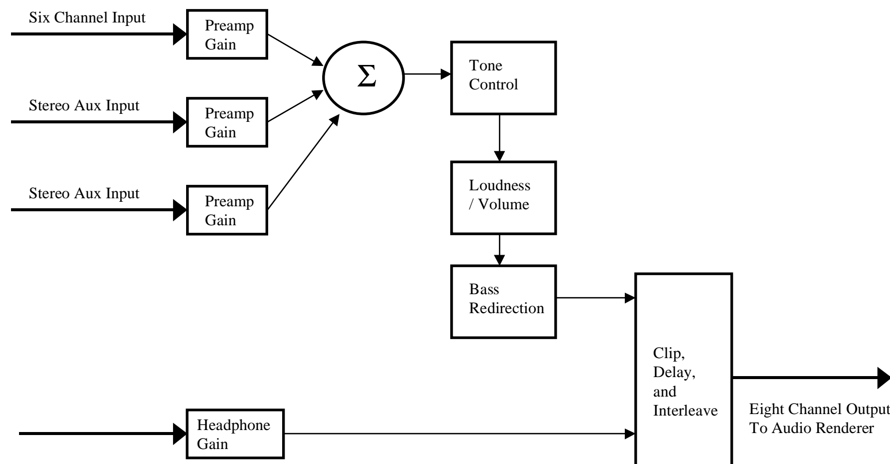


Figure 7: Mixer block diagram

Compressed Audio Decoders

In this system, the decoder is another TSSA module, and this allows it to be changed easily. The first release of the audio system supports the AC-3 decoder, including ProLogic®. Subsequent releases

incorporate MPEG audio decoders. Section 2 of this paper describes the process of porting a decoder to TriMedia.

Audio Digitizer

The audio digitizer provides a TSSA interface to A/D converters and to S/P DIF receivers. Section 3 of this paper includes some comments on the mechanism used to recover the clock from an S/P DIF data stream.

Application Programmers Interface:

The audio system presents a simple programming interface, allowing designers of digital TV's to treat the audio system as a black box, if desired. At the same time, the individual components of the audio system present published interfaces, so that customizations can be made. As is typical for a TSSA component, the standard set of entry points are provided (see page 8). The instance setup structure and the list of configuration options describe the interface to this TSSA component. Reference [5] describes this in detail. For the purposes of this discussion, suffice to say that the interface to the audio system collects the interfaces to the various underlying components.

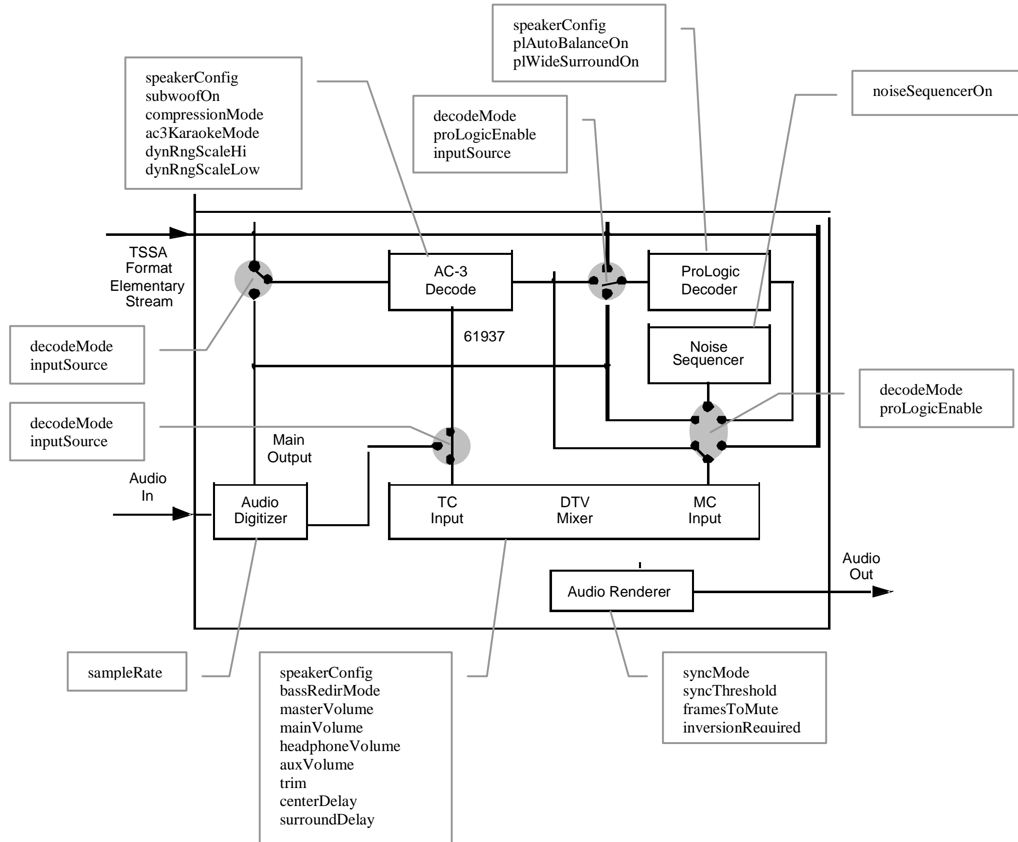


Figure 8: Configuration Parameters of the DTV Audio System

Figure 8 shows the internal components of the audio system and the set up parameters to configure them. The internal interconnection network can be configured with three basic set up parameters, all other parameters are used for the configuration of the system's sub-blocks.

- inputSource determines if the system's input comes from the audio digitizer or the elementary stream input interface. In the former case it further determines if the source is a DAC or a digital S/P DIF bit stream that may contain AC-3 data.
- decodeMode determines if the input data is fed to the AC-3 or the ProLogic decoder.
- proLogicEnable disables and enables the ProLogic decoder. If the ProLogic decoder is connected to the AC-3 decoder's output, an automatic mode can be selected. If the latter is chosen, the audio system automatically enables the ProLogic decoding if the AC-3 decoder detects that it decodes a ProLogic encoded stereo bit stream.

Section 2: Optimization of a Typical Decoder

AC-3 Decoder

Software modules are available to decode data compressed in formats such as Dolby's AC-3 and various forms of MPEG audio. The process of porting these codecs to the processor is described, with emphasis on the types of optimization that are used, and the simplicity of these optimizations. Performance measurement results are presented at several stages of the optimization process.

The AC-3 decoder was developed in two independent steps. Using the Dolby reference code, the decoder core was separated from the input/output processing. In the first development phase, the decoder core was optimized for speed and an automatic test environment was implemented. Then, a flexible TSSA application programmers interface and example applications were developed.

Testing

During the implementation process significant changes were made to the reference code. For this reason a solid quality assurance system was required to ensure that the core decoder was always fully compatible to the standard and that all interface features are functional. For this reason an automated test environment was developed for the AC-3 decoder. It performs all tests on a PC, using a hard disk for both as source of the AC-3 data and destination for the decoded data. The test application calls the reference decoder and has it decode a test vector. The PCM output is stored on hard disk. Then, the TriMedia decoder loads the same test vector and also decodes it to hard disk. After that both PCM files are compared on a sample basis. The acceptance criterion is that only the least significant bits may vary. The initial test suite consisted of 406 AC-3 streams. It was extended whenever a bug was found, which was not triggered by the existing test vectors. In addition to this core decoder regression test suite, an interface test suite was implemented. It tests all supported input/output configurations of the decoder to make sure that all internal channel pointers are assigned to the correct channels. As a third test suite, a performance test is carried out to automatically measure the load of the processor. This ensures that changes to code do not unnecessarily increase the amount of processing power that is required. A run of all tests requires about 2 hours on a standard PC.

```
File Name     : cave.ac3
Sample Rate   : 48000
Data Rate     : 448 kbps
Input Config  : 3/2 (L, C, R, l, r), Lfe
Output Config : 3/2 (L, C, R, l, r), Lfe
packet type   : apfFiveDotOne16
lowest  MIPS  : 23.593750
average MIPS  : 23.991020
highest MIPS  : 24.384370

---------------------------------------
File Name     : cave.ac3
Sample Rate   : 48000
Data Rate     : 448 kbps
Input Config  : 3/2 (L, C, R, l, r), Lfe
Output Config : 2/0 (L, R) Surround Encoded, Lfe
packet type   : apfFiveDotOne16
lowest  MIPS  : 24.615630
average MIPS  : 25.037410
highest MIPS  : 25.431250
```

Above, a part of the performance test result is shown. The first field shows the name of the test bit stream. Then, the sample rate and the data rate are reported. The next information is the channel configuration of the bit stream. In the example it is a five dot one stream. For the first test, the decoder is configured to decode all channels, which is indicated in the output configuration field. In the latter case, the decoder performs a downmixing of the five main channels to stereo including matrix encoding. In addition it decodes the subwoofer channel. The output packet type in both cases is a six channel 16-bit PCM format. This type basically describes the memory interleave of the PCM samples. Finally the decoder performance is reported on an audio frame (1536 decoded samples) basis. The decoder's performance is measured in million instructions per second (MIPS).

Decoder Core

The AC-3 core consists of two main parts, the bit stream unpacking block and the signal processing block. In the first block information on the properties of the bit stream and the applied coding strategies is retrieved from the bit stream. This information is used to control subsequent functional units of the decoder as well as to set up connected components (the sample rate information is coded in the bit stream).

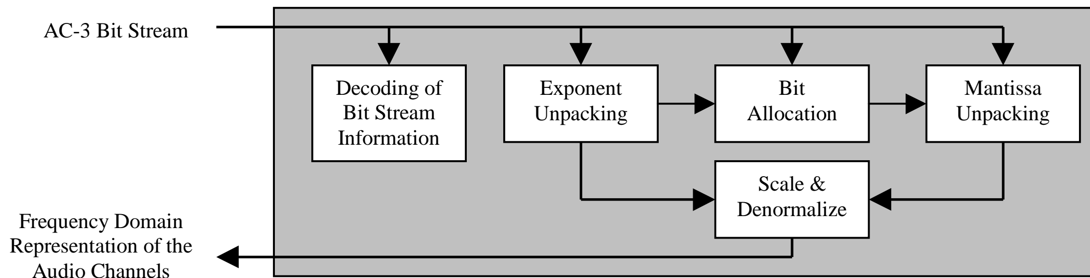


Figure 9: Bit Stream Unpacking Part of the AC-3 Decoder

The output of the first part of the decoder is a frequency domain representation of the coded audio channels. It is obtained by the decoding of its mantissas and exponents from the bit stream. While the exponents can be obtained directly from the bit stream, the coding of the mantissas is not fixed. Based on the exponents the bit allocation of the mantissas is calculated by applying the same psychoacoustic model used in the encoder.

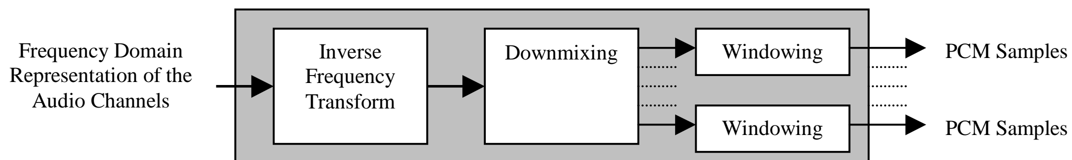


Figure 10: Digital Signal Processing Part of the AC-3 Decoder

The second part of the AC-3 decoder implements the transform from the frequency domain to the time domain, which is PCM samples. Subsequent to the inverse Time Domain Aliasing Cancellation (TDAC) filter bank, downmixing to the number of desired output channels is performed. Then, the samples of these output channels are windowed. During the windowing the scaling of the internal representation to a 16-, 18- or 20-bit interleaved PCM format is performed. The following paragraph describes the complexity of the functional blocks and how they have been optimized for speed.

Optimization for Speed

The first phase started with careful performance analysis in order to determine the algorithmic complexity of the decoder basis blocks. To do so the TriMedia simulator was used to decode a set of reference AC-3

bit streams. The simulator produces profiling information, which tells the programmer how much time is spent in which function and how well parallelism is exploited.

The initial version of the decoder consumed about 70 million instructions per second (MIPS) on a TriMedia processor. About 50% of this time was spent in the decoder's front end, where the exponents and mantissas from the AC-3 bit stream are unpacked and composed into a frequency domain representation of the audio content. The other half was spent in the signal processing part of the decoder which implements a frequency to time transform, downmixing, and windowing of the time domain audio signal. The following chart gives an overview on how the MIPS consumption of certain functions of the decoder changed during the optimization process.

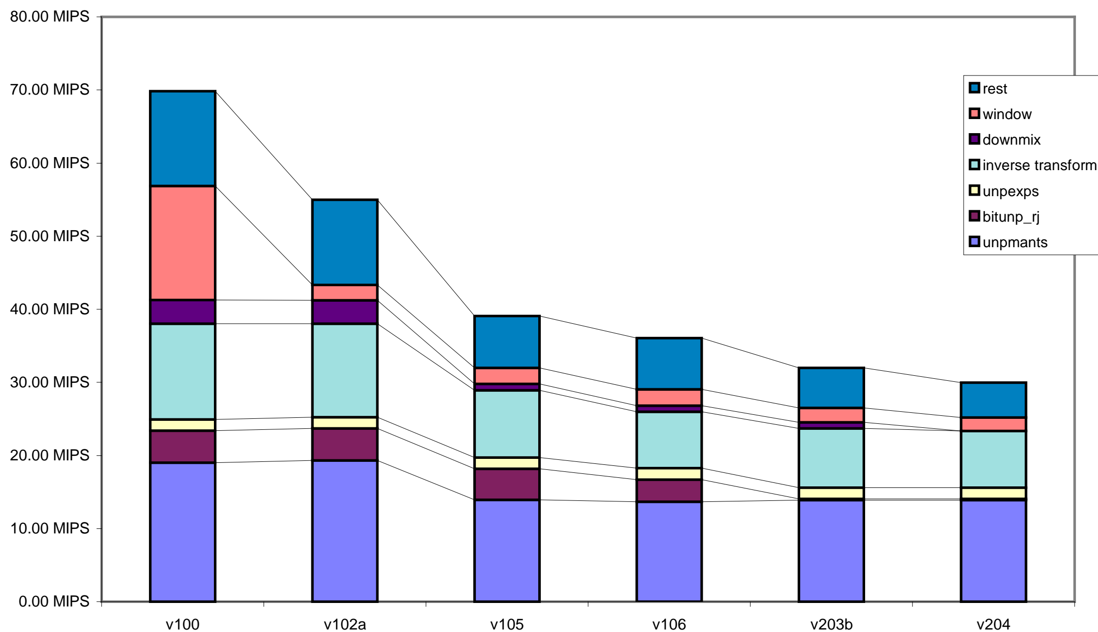


Figure 11: Improvement of the AC-3 Decoder Performance over Time

All presented performance data was measured with a test vector containing 6 audio channels with 48 kHz sample rate. This vector does not represent the worst possible input but it exercises the decoder extensively. Most real world streams seen in DVD and ATSC streams impose a lower processor load.

The chart shows the three functions that constitute the entire signal processing part of the decoder.:

- Inverse transform:
  Performs an inverse time domain aliasing cancellation filter bank implemented through an IFFT plus two complex multiply stages
- Downmixing:
  Performs the mixing of decoded audio channels to the number of desired output channels. It is performed in the time domain.
- Window:
  Implements the overlap-add windowing of the output blocks and converts the internal data representation into a PCM format.

In addition to these signal processing functions three functions belonging to the bit stream unpacking processing are depicted.

- Unpexps:
  Unpacks the exponents from the AC-3 bit stream.
- Unpmants:
  Calculates the bit allocation and unpacks the mantissas from the bit stream
- Bitunp_rj:
  Extracts the actual bits from the AC-3 bit stream. It is mainly called from unpexps and unpmants.

All other functions are summarized as in the block *rest*, see the uppermost function in Figure 11. They belong to the bit stream unpacking part of the decoder and to the I/O interface.

In the initial optimization step the arithmetic of the signal processing functions was changed from floating point to 32-bit fixed point and unnecessary limiting operations were removed. This lead to a significant performance improvement, because the TriMedia CPU provides 5 integer ALU's but only 2 floating point ALU's. Additionally, the integer ALU's require only one clock cycle to finish a calculation while the floating point ALU's require three. In general purpose CPU's the reason for a conversion from floating point to fixed point is often the difference in performance between the respective multiply units. The TriMedia CPU, however, provides two pipelined multiply units that perform either integer or floating point multiplies in three clock cycles. Therefore, the conversion was undertaken because the integer ALU's outperform the floating point ALU's in number and speed, and because the natural output format of the AC-3 decoder is fixed point integer PCM samples. Performing all processing with integer arithmetic saves a format conversion in the output stage. Another argument for working in integer arithmetic is that instructions can be scheduled more flexibly than in floating point arithmetic because of the limitations imposed by the IEEE 754 floating point arithmetic standard. The basic limitation is that the associative law is not valid in the floating point domain.

To better illustrate the porting process, it is useful to look at the system's implementation of fixed point integer arithmetic. While typical DSP's provide a MAC instruction and a high precision accumulator, the VLIW machine implements the MAC using a number of primitive operations in the same cycle. Similarly, the 32 bit register used as the accumulator limits the precision of this type of arithmetic to something over 24 bits. The processor implements 32-bit register operations without any internal states. It takes two 32-bit values as input and calculates one 32-bit result and stores it in a register. High precision fixed point arithmetic is possible on a TriMedia CPU because its integer multipliers can calculate either the high or the low half of the 64-bit result of a 32x32 multiply. A normal A*B written in the C language is translated by the compiler to the IMUL() machine operation. In high precision integer arithmetic, this operation is more or less useless because it limits the dynamic range of the data and coefficients to 16 bits. Using the IMULM() operator raises the problem that, if the arguments are signed numbers, one bit of precision is lost in the upper 32-bit result. This bit is shown in Figure 12 as SE (sign extension) bit, right next to the sign bit. When necessary, this is easily compensated for with a shift. The IMULM() operator can be used at C-code level in a function call fashion. It is not necessary to write inline assembly code to exploit these so-called "custom operators," which do not map directly to C instructions.

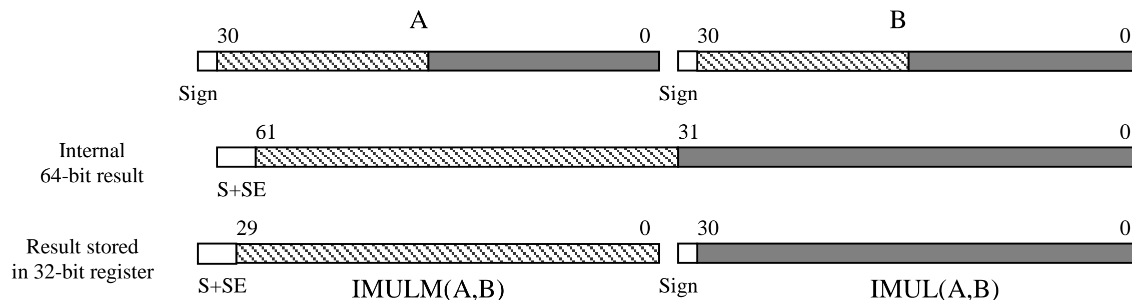


Figure 12: Multiply Instructions of the TriMedia Processor

During the conversion from floating point to fixed point code it turned out that a left shift of the multiply results is not necessary because the output of the windowing stage is still precise enough to comply with

the Dolby quality requirements. While all coefficients used for the IFFT, downmixing, and windowing use the full 32-bit word width, the intermediate processing results lose one bit per multiply stage. Additions result in an intermediate result of the same resolution as its inputs.

However, in the inverse Fourier transform a shift is necessary because in every basic butterfly an input value of a butterfly stage is added to multiply results of the stage. This means, that either the input value must be right shifted or the multiply results left shifted.

```
/* floating point implementation */
for (i = 0; i < bg; i++)
{
        ar = *bfyrptr1;
        ai = *bfyiptr1;
        br = *bfyrptr2;
        bi = *bfyiptr2;

        rtemp = br * cr - bi * ci;
        itemp = br * ci + bi * cr;

        *bfyrptr1++ = ar - rtemp;
        *bfyiptr1++ = ai - itemp;
        *bfyrptr2++ = ar + rtemp;
        *bfyiptr2++ = ai + itemp;
}
```

```
/* fixed point integer implementation */
for (i = 0; i < bg; i++)
{
        ar = *bfyrptr1;
        ai = *bfyiptr1;
        br = *bfyrptr2;
        bi = *bfyiptr2;

        rtemp = IMULM(br , cr) - IMULM(bi , ci);
        itemp = IMULM(br , ci) + IMULM(bi , cr);

        *bfyrptr1++ = (ar >> 1 ) - rtemp;
        *bfyiptr1++ = (ai >> 1 ) - itemp;
        *bfyrptr2++ = (ar >> 1 ) + rtemp;
        *bfyiptr2++ = (ai >> 1 ) + itemp;
}
```

Above, the original floating point implementation of the innermost loop of the IFFT is compared with its fixed point integer counterpart. These loops implement a complex butterfly.

The following figure shows the number of significant bits of the intermediate results throughout the signal processing blocks of the AC-3 decoder.

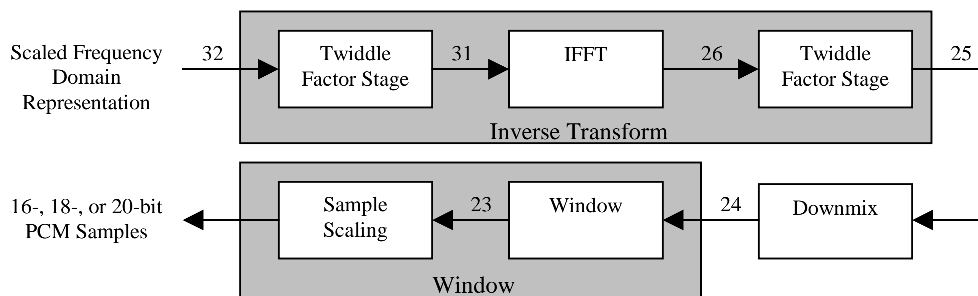


Figure 13: Precision of the intermediate results in the DSP Part of the Decoder

The straightforward fixed point conversion decreased the processor load of the three signal processing functions from 31.9 MIPS to 10.78 MIPS (see v106 in Figure 11). Further gain was achieved by integrating the downmixing with the inverse frequency transform and rearranging the code (see v204) resulting in 9.57 MIPS.

In the bit stream unpacking part of the decoder, the processor load was reduced in three basic optimization steps. In the original code all internal processing was implemented either with 16-bit integer arithmetic or single precision floating point. Also the bit stream was only accessed with 16-bit operation. The decoded mantissas were stored as floating point values and scaled by multiplies with floating point scale factors. As a first optimization step the internal 16-bit arithmetic was converted into 32-bit arithmetic. This helped to save cycles previously spent in sign extensions and it made loads from the memory more efficient. Then the mantissa denormalization and scaling was converted to 32-bit integer arithmetic. The denormalization is more efficient in integer arithmetic because it can be implemented by simple shifts (implemented in v105). Finally the function used to extract a certain number of bits from the bit stream is now used as a macro in unpmants and unpexps (see v203b in Figure 11). These optimizations reduced the processor load of the three bit stream unpacking functions from 24.9 MIPS to 15.6 MIPS.

From a precision perspective the conversion to fixed point resulted in a decreased signal to noise ration (SNR). However, the output of the decoder is still in compliance with Dolby's highest standards for accuracy. All of the above mentioned optimization has been carried out in C, and required less than 5 man months effort due to the excellent compilation tools available for the TriMedia processor.

The presented performance numbers are in the same order of magnitude as numbers measured on highly optimized DSP's, refer to [13] and [14]. This is remarkable because DSP's have specialized instructions, can execute multiple loads of coefficients concurrently and support zero overhead loops. The presented VLIW processor has to perform the DSP tasks based on its load-store architecture. Since it is capable of executing up to five RISC-like operations concurrently it can compete with DSP's performing FFT's or FIR/IIR filter. It must also be emphasized that the VLIW code is generated from C, whilst the DSP code is written in assembly language which is hardly portable or reusable.

Features of the AC-3 Decoder Library

The optimized AC-3 core decoder is encapsulated in a versatile TSSA application programmer interface. It was the goal to make the library easy to use in many different application environments. The first of two typical application scenarios is the use of the TriMedia as a pure audio decoding device. In this case an external device like a DVD or laser disk player would send AC-3 data packetized in an S/P DIF format (IEC 61937). Another scenario is to use the decoder in the context of an MPEG-2 system on the TriMedia chip as it is done in the ATSC receiver. In this case the AC-3 decoder receives a series of ES (elementary stream) packets from a system stream demultiplexer which consists of a continuous stream of encoded AC-3 frames. The library is capable of dealing with both input formats.

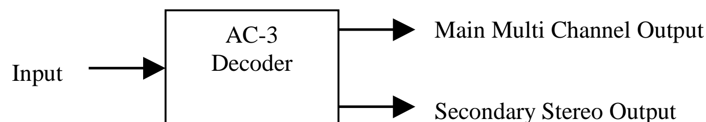


Figure 14: Input and Output Pins of the AC-3 Decoder Library Module

At its output side the AC-3 decoder library provides one main pin and one optional one. The main multi channel output is intended to carry PCM samples for the main speaker system. Aside from 5.1 channel audio it supports 7 different downmixing modes and three different PCM sample resolutions (16-, 18- and 20-bit) for all modes. The second output is foreseen to provide audio data for an external device. This could for instance be a VCR, which requires stereo data. As a second option S/P DIF formatted AC-3 data can be sent at this output. This could be used in set top box applications that provide only stereo audio jacks. Besides the analog stereo jacks, the set top box could provide a digital audio connector that can be hooked up to an A/V receiver capable of AC-3 decoding.

If the input data is time stamped the decoder attaches appropriate time stamps to the packets generated at its outputs. In the case that the presentation time stamp of an incoming packet is expired the packet is rejected and no decoding is performed on it.

Another interesting feature of the AC-3 decoder is that it is re-entrant, meaning that multiple decoder instances can run concurrently. All instances share the same code but have individual data contexts. Running more than one AC-3 decoder can be necessary due to several reasons. Firstly the ATSC standard explicitly supports multiple AC-3 streams accompanying a video stream. Secondly, if a TV set or a set top box supports the decoding of two programs at a time, one to be watched on the screen and the other to be taped by a VCR, two AC-3 decoders are required. Making the AC-3 decoder re-entrant requires the elimination of every global variable from the code, which is not read only. It is also required to allocate all the memory for all buffers dynamically when a new instance is opened. The internal structure describing the state of the instance comprises 191 variables and pointers.

The AC-3 decoder library has the following memory demands:

- size of the object code, decoder core: ca. 48 Kbytes
- size of the object code, TSSA interface: ca. 32 Kbytes
- size of static data: ca. 10 Kbytes
- size of the dynamic memory used as the instance variable: ca. 24 Kbytes
- stack size for the decoder task: ca. 10 Kbytes

This means, that the first decoder requires about 124 Kbytes of memory plus the memory required for the input and output data packets. Since the code is re-entrant, a second decoder could be run concurrently using the same object code and static data. Therefore, the second decoder requires only 34 Kbytes of additional memory.

Section 3: Audio / Video Synchronization

An important feature of a broadcast digital TV receiver is the mechanism used to synchronize audio and video. This section gives some insight into the methods used in our receiver.

In the MPEG system used by the ATSC, audio and video are synchronized using a reference clock and "presentation time stamps (PTS)." The reference clock (known as the PCR) nominally runs at 90khz. The fundamental A/V sync mechanism is to recover the PCR and then ensure that audio and video are presented when their PTS match the PCR. Assuming that the video is also locked to the PCR, then the difference between the PCR and the PTS is a measure of AV sync. This measure is referred to as "timeDiff."

Reference Clock Recovery

Samples of the PCR are periodically embedded in the broadcast stream. The demultiplexer records the exact local time when each PCR sample is received. From this data, it is possible to interpolate/extrapolate the value of the reference clock at a given local time. In this way, the exact value of the PCR is always available through a function call. Here the single chip nature of our solution reduces the complexity of communicating this time sensitive data.

In the compressed domain

Some of the AV sync process occurs in the compressed domain. AC-3 data packets (PES packets) are broken up into transport packets for transmission. Code in the receiver must reconstruct the PES packets, taking care to attach the time stamps correctly. When the AC-3 decoder receives a time stamped packet, its presentation time (PTS) is compared to the reference (PCR) before decoding. In some cases, there is not sufficient time to decode this packet, and it is rejected before decoding. This optimization speeds the A/V sync lock up considerably.

Coarse synchronization

When the audio renderer starts to play a new stream of data, it will usually find that the audio is decoded some time before it must be presented. This is done to give other parts of the system (like video and external audio decoders) time to prepare the data. The audio renderer notes this by comparing the presentation time stamp (PTS) on the packet to the local reference clock, as derived from the program clock reference (PCR).

The most obvious way to pull a signal into sync is to skip samples or play silence while waiting for the clock reference to catch up to the PTS. A mechanism like this is entirely appropriate when a new stream has just been acquired. Then the system should get into sync as quickly as possible before the user has time to notice a problem. The user will accept some short period when the audio is muted before it comes up in sync. This delay must necessarily be as long as the difference between the PCR and the PTS present in the data stream, and it is normally not more than 200ms.

Fine synchronization

Skipping samples is not appropriate during normal operation. After AV sync is achieved, it must be maintained using a less noticeable method of adjustment. This is achieved by varying the sample rate clock so as to track the broadcast clock. In the AES-3 interface, this is traditionally implemented using a hardware PLL to recover the clock encoded in the bi-phase mark signal. In this system, the sample rate clock is adjusted up or down in proportion to the timeDiff variable. The TriMedia hardware includes a convenient clock synthesizer (called a DDS) making it easy for the clock's frequency to be controlled by software. The exact algorithm can be tweaked or improved in many ways, but using this method, it is not hard to achieve AV sync reliable to within a few milliseconds.

Using TSSA Features

As a standard feature, the TSSA framework allows each component to have access to a clock to be used for synchronization purposes. In the case of an MPEG system, the components are given the handle of the recovered 90khz reference clock.
Standard TSSA components all provide a callback known as the progress function. The progress function is called in the context of the TSSA component, but the code is determined by the user. The progress function may be called at several points in the algorithm. Setup flags are used to enable or disable the various calling points. The same flag values are then used to indicate the source of the call.

In the case of the audio renderer, the progress function can be called from the interrupt service routine with a flag indicating a specific event (`AREND_PROGRESS_SyncEventCorrect`) or with a flag that tells us that the function is being called in every interrupt (`AREND_PROGRESS_ReportCount`). The former flag is used to allow a programmer to correct the clock using the mechanisms described above. The latter flag is used when a simpler type of sync is required. For instance, when a digital input is used as a source, it must be the clock master. The "report count" flag allows a user to compare the number of input and output samples and adjust the output clock to match the input clock.

Section 4: Signal Levels and Dynamic Range Considerations

The DTV audio system is designed to be used with a D/A converter attached directly to a speaker system. The system integrates loudness compensated volume control, tone control, and bass redirection in the digital domain. These choices lead to a number of interesting questions about the dynamic range of the signals. This section explores the problem and presents our solution.

Compliance Requirements

Dolby Labs sets certain quality requirements for their licensed products. These are outlined in the Dolby Labs Licensee Information Manual (LIM) [9]. Section 6.2 of the LIM outlines some signal to noise ratio (SNR) requirements. In short, televisions must demonstrate a signal to noise ratio of at least 55db.

For a device with built in amplifiers, the signal to noise ratio must be measured under a specific set of circumstances:

- 1 watt into 8 ohms, at the speaker connections.
- Input signal 20 db below full scale.
- Tone and trim controls flat.

If all of the gains in the system are not aligned properly, then the user could experience premature clipping of the signal, or if too much headroom is provided, a low level signal will result in a very low signal to noise ratio. Ideally, these rules can be applied:

1. Align the system so that the analog amplifiers clip just before the D/A converter clips.
2. In the digital domain, ensure that intermediate results do not overflow, and are never clipped. Then clip the output to the limits of the D/A just before presentation.

3. Adjust the nominal operating point of the digital processing chain to make a good trade off between dynamic range, clipping, and loudness. At high volumes, the system may clip. But the user must always be able to eliminate clipping by turning down the volume.

Required Headroom

Features of the TV can require headroom, that is dynamic range above the nominal value to protect against clipping. The contributions of these sources can be summed together to compute the required performance of the DAC used in the system.

DAC Dynamic range >= Dolby Spec + 20db + sum of headroom. (1)

The most useful measure of the dynamic range of the DAC is given with the measure of SINAD (Signal to Noise plus Distortion) for a reference signal -60db below full scale. Together, these give an honest appraisal of the capabilities of a DAC.

Summing Channels

When multiple channels are summed together, the result cannot exceed the capabilities of the accumulator.

Tone Control

If the system will allow bass and treble boost, then these functions will require headroom. Assuming that a full scale signal is given as input. Applying a bass boost using the tone control must not result in clipping. Hence headroom is required to match the boost capability.

Trim Control

AC-3 systems are required to have a range of at least +/- 6db gain that is variable on a per channel basis to trim the balance between the various channels. The Dolby tests are made with trim at its center position. If trim is to be provided in the digital domain, then the output signal will need six dB of additional headroom.

Volume Control

If the volume control is to be performed in the digital domain, then some headroom may be required to account for this. The amount of headroom depends on the desired output power. In the test, the system provides one watt when the signal is 20db below full scale. By Ohm's law, this implies that 2.8V RMS will be measured across the 8 ohm resistor. We will consider this the fundamental reference voltage.

$P = E^2/R$ (2)
2.88 = square root of 8.

When the voltage (recorded in the digital domain as PCM audio) goes up by 20db, the voltage measured on the resistor will be ten times that at reference, and the power will be 100 times the reference.

dB = 20 log (V1/V2) = 10 log (P1 / P2). (3)

Since 100 watts are more power than we expect to have in our TV, we will need to take gain on the reference signal in order to reach the measurement voltage. And if that gain is left enabled when a full scale signal is presented, the signal will be clipped, unless we can provide 100 watt amps. This gain will probably be taken in the analog domain. The amount of gain required to deliver the 1W test signal can be computed with this equation:

Gain = 10 log (Amp Power / Reference power) (4)

Since reference power is 100 watts, this table shows the gain required for some common output powers:

Maximum Watts RMS output	*DB Gain*
100	0dB
50	3dB
20	6.9dB
10	10dB

Table 2: Relationship between Gain and Output Power

When the actual required analog gain is computed, you must also factor in the difference between your reference voltage (DAC output) and the standard reference voltage used above.

As a result of this investigation, we see that with lower powered amps we will expect to run the Dolby test with the volume control set wide open. Hence the volume control does not require headroom.

Bass Redirection

As demonstrated in figure 9.9 of the LIM, [9] Dolby expects bass redirection to include some analog processing. Note in particular the 10 db of extra gain that is applied to the bass signal. This gain is part of the spec for a playback system, and it is required in order to provide as much low frequency energy as the authors of the bitstream expect. When the bass signals are redirected to the main speakers, this extra gain places a heavy burden on the channel to which the bass is being sent. If you thought that 10 watts would be enough for your main speakers, but now you want to redirect bass into those main speakers, you need to provide three times (3.16) that wattage to the speakers that will be handling redirected bass.

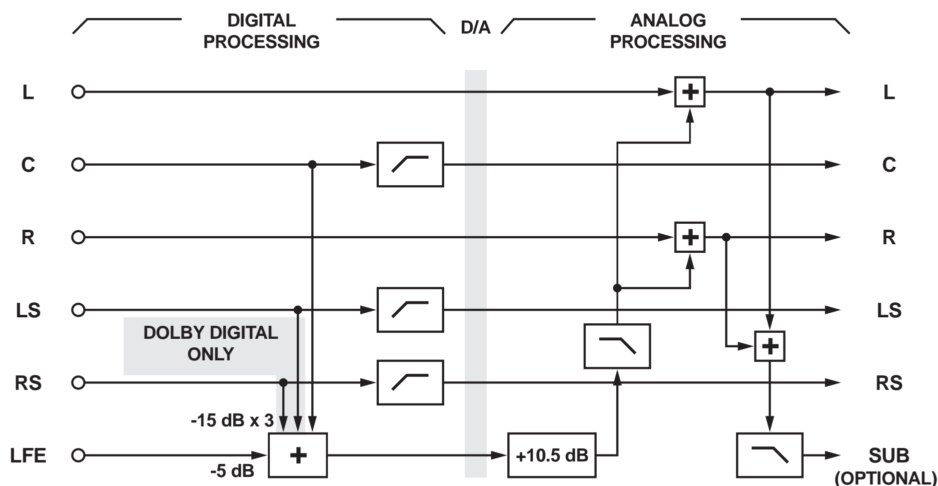


Figure 15: Bass Redirection Signal Flow Diagram

The additional output power requirement can not be circumvented. But it is possible to move the summation and gain from the analog domain to the digital domain at the cost of another 10db of headroom.

Signal Levels

We can turn equation (1) around to give the result that we can expect from Dolby's measurement:

$$\text{Measured SNR} = \text{DAC SNR} - 20 - \text{Headroom} \qquad (5)$$

As an example:

100db DAC.
6db headroom for trim control
12db headroom for tone control

12db headroom for bass redirection

Measured SNR = 50db.

Clearly, this provides too low of an operating point for a real system. Some gain must be added to move the operating point back up to the range where we can pass the dynamic range test. This gain is added in the final stage. Nominally, 15db of gain would seem appropriate.

Mixing ATSC and NTSC signals

In a digital TV that can receive both analog and digital signals, another gain correction is required. The dialog normalization feature of the AC-3 system will ensure that the dialog portion of a signal will be reproduced 31db below full scale. This stands in contrast to the NTSC world, where dialog is normally modulated 20db below clipping. The so-called "11db problem" results when you switch between these two signals. Because the ATSC broadcasters simulcast their analog signal, it is common to be able to switch between the same program material in the analog and digital domains. To keep the same loudness, the NTSC signal must be attenuated, or the ATSC signal boosted.

Section 5: Summary

The TriMedia DTV Audio System illustrates many aspects of the design of a digital television receiver. It also illustrates the capabilities of a mature software development environment when deployed in conjunction with a media processor and a sophisticated streaming software architecture. All key components are fully reusable and can be employed without any changes in different target applications. The use of an operating system greatly helps to implement load balancing and synchronization of the components in a very general way, applying well known techniques from the general purpose, multi-tasking arena.

In addition to the system level description of the DTV audio system, this paper has shown how a complex signal processing algorithm can be optimized at the C level without lagging behind highly specialized digital signal processors.

Acknowledgements

The authors wish to acknowledge the help and guidance given by Dolby Labs, whose Dolby Digital system greatly determines the outline of a system like this. The Dolby reference code is used under a license agreement.

References

[1] ATSC standards (A65, etc), refer to http://www.atsc.org/Standards/stan_rps.html
[2] AC-3 standard (A-52), refer to http://www.atsc.org/Standards/stan_rps.html
[3] TM1000 PCI Media Processor (TriMedia) Hardware data book. ©1998, Philips Semiconductors
[4] TriMedia Programmers Reference Manual (SDE 2.0) ©1999, Philips Semiconductors
[5] TriMedia DTV Programmers Reference Manual (v1.3) ©1999, Philips Semiconductors
[6] IEC60958
[7] IEC61937
[8] ISO/IEC 13818-7, MPEG-2 Advanced Audio Coding
[9] Dolby Licensee Information Manual. Version 2.0, April 1997. ©Dolby Labs
[10] C.L. Liu, James W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment", Journal of the ACM, Jan 1973
[11] TriMedia Programmers Cookbook (SDE 2.0) ©1999, Philips Semiconductors
[12] Siegfried Linkwitz, Russ Riley, AES journal Volume 33 no.3, march 1985

[13] Steve Vernon, Vlad Fruchter, Sergio Kusevitzky, “A Single-Chip DSP Implementation of a High-Quality , Low Bit-Rate, Multi-Channel Audio Coder”, 95th Convention of the Audio Engineering Society, New York, October 1993
[14] Steve Vernon, “Design and Implementation of AC-3 Coders”, IEEE Transactions on Consumer Electronics, Vol. 41, No. 3, August 1995
[15] Linkwitz, Seigfried H, "Active Crossover Networks for Noncoincident Drivers", J. Audio Eng. Soc., Vol.24, No. 1, pp. 2-8, Jan/Feb 1976
[16] Linkwitz, Seigfried H, "Passive Crossover Networks for Noncoincident Drivers", J. Audio Eng. Soc., Vol.26, No. 3, pp. 149-150, Mar 1978