Technion – Israel Institute of Technology
Department of Electrical Engineering

The Vision and Image Science Laboratory

Project Report

Subject:

AAC Encoder on TriMedia TM-1300

Written by:
Shahar Noy **031870405**

Instructor:
Boaz Ophir

Submitting date:
June 2002

Table of Contents

Chapter 1

Abstract

The projects main target was to implement an advanced audio encoder on a DSP platform. Audio standards are widely distributed and the most popular one is MPEG-1 layer III, also known as MP3. The decision was focus on future audio format; therefore the AAC format was selected.
The DSP platform is TriMedia TM-1300, an advanced digital signal processor that was designed mainly for demanding multimedia tasks.

Project goals:

- Write an AAC encoder in ANSI C.
- Optimize the code, so it will be available for 32-Bit platform.
- Port the working code to the TM-1300.
- Achieve near “real-time” performance.

The problem:

- Optimize highly complex code to 32-Bit architecture.
- How to use properly the sophisticated compiler.

Conclusions:
Most of the project goals were achieved yet the real-time performance is still not satisfying and an assembly tuning in the complex mathematical blocks is recommended in order to achieve the desired real-time performance.

Chapter 2

Introduction

Background

Digital audio compression is a unique undertaking. If used properly, the output of the process “digitized audio” is indistinguishable from the original recording. For this reason, audio compression is sometimes referred to as a transparent technology with the benefits lying outside the realm of sound. The advantage of audio data compression is in reduced storage requirements, or in the case of transmission, reduced bandwidth requirements.

In recent years, research conducted by several different organizations has contributed to dramatic advances in audio compression. AAC, Advanced Audio Coding, is a combination of state-of-the-art technologies for high-quality multichannel audio coding from four organizations: AT&T Corp., Dolby Laboratories, Fraunhofer Institute for Integrated Circuits (Fraunhofer IIS), and Sony Corporation.

AAC has been standardized under the joint direction of the International Organization for Standardization (ISO) and the International Electro-Technical Commission (IEC), as part of the MPEG-2 specification. The MPEG-2 standard contains several audio-coding methods, including the more familiar MPEG Layer-3 or "MP3" coding scheme. When the MP3 format was defined in the early 1990s, one of its requirements was to preserve compatibility with the MPEG-1 audio coding system. After MPEG-2, Layer-3 was finalized, standardization work continued to develop a higher quality coder than could be achieved while preserving MPEG-1 backward compatibility. The result of this effort is MPEG-2 Advanced Audio Coding, or simply AAC. When AAC was first developed, it was sometimes referred to as "MPEG-2 NBC" for "not backwards compatible." It is important to know that the terms are synonymous, since several other documents use the older designation to refer to AAC.

Technical advances since the completion of the MPEG Layer-3 standard provide significantly greater levels of data reduction, as well as a more graceful handling of audio artifacts at extremely low bit rates. The MPEG audio standardization committee's goal was to develop an audio data compression scheme that achieved indistinguishable audio quality at data rates of 384 kbps for five full-bandwidth channels. Tests conducted in the fall of 1996 showed that AAC was able to meet this requirement at 320 kbps (64 kbps/channel). In contrast, MPEG, Layer 3 requires data rates of between 640 and 896 kbps to achieve an indistinguishable quality rating.

The Philips TriMedia, nicknamed “media processor” is designed as a versatile processor for video, audio and image. The processor’s main target is to deal efficiently with multiple media.

The idea behind the TriMedia is that most consumer devices are either currently, or are becoming, digital. Many audio hi-fi systems and home entertainment devices already use Digital Signal Processors (DSP) chips; telephony is now digital using ISDN and ADSL; video is digital with DVD players, personal video recorders (PVRs) and digital TVs.

This is a wide range of applications, and further complicated because the standards are not yet finally defined: for example audio compression may use the Dolby Digital AC-3 or the MPEG compression standards, and some elements of the standard for digital TV broadcast have yet to be decided. This is a classic situation where a programmable processor becomes attractive compared with a fixed hardware solution: the programmable device can be reprogrammed to match changing standards, and can be programmed differently to suit different markets or applications.

Motivation

In consumer markets, no incumbent component vendor has the advantage of defining standards. Often, standards are not set at all, and system differentiation is a key factor. Consumer electronics vendors do not want to become vendors of commodity products like the PC vendors but want to differentiate their products by adding features and functionality easily. Under these circumstances, programmability is a key factor and is preferred over hardwired ASIC-type or ASSP-type options as long as cost permits.

Programmability is one of the key differentiators between application-specific hardwired products such as ASICs or ASSPs and commodity products such as analog or memory, on the one hand; and general-purpose programmable products such as microprocessors, digital signal processors or programmable logic devices on the other hand. Theoretically, it is possible to implement any system logic on software and processors or programmable logic.

Programmability improves design flexibility as a processor can be reprogrammed as functional requirements evolve. The benefits of programmability begin in the development phase - programmable solutions have shorter development times because they can be easily modified, right up to the time the product is released. A programmable solution also allows the possibility of incorporating new functionality, even after the silicon has been included in a device. With a programmable solution, new applications can be easily downloaded and bugs fixed real-time without having to scrap the design.

The software-only implementations of primitive techniques for digital audio compression, such as μ-law and ADPCM algorithms can be easily run in real time.

The challenge lies in developing a real-time software implementation of the MPEG/Dolby audio algorithm. The MPEG/Dolby standards document does not offer many clues in this respect. There are much more efficient ways to compute the calculations required by the encoding and decoding processes than the procedures outlined by the standard.

Definitions

AAC - Advanced Audio Coding - An audio compression technology that is part of the MPEG-2 standard. It provides a greater compression and superior sound quality than MP3, which is also part of the MPEG spec (MPEG Audio Layer 3). AAC is available in three profiles: Main, Low Complexity (LC) and Scaleable Sampling Rate (SSR), with Main providing the highest quality. MPEG-4 includes a superset of MPEG-2 AAC.

ASIC – Application Specific Integrated Circuit, A chip that is custom designed for a specific application rather than a general-purpose chip such as a microprocessor.

ASSP – Application-Specific Programmable Processor/Products.

ASPP – Application Specific Standard Part, another term for an ASIC chip.

CPU - Central Processing Unit - The computing part of the computer. Also called the "processor," it is made up of the control unit and ALU. Today, the CPUs of almost all computers are contained on a single chip.
The CPU, clock and main memory make up a computer. A complete computer system requires the addition of control units, input, output and storage devices and an operating system.

Dolby - An audio research laboratory, Dolby's digital versions are advancing the art in the digital realm with movie and home theater systems that provide six and seven-channel sound.

DSP - Digital Signal Processor - A special-purpose CPU used for digital signal processing. It provides ultra-fast instruction sequences, such as “shift” and “add”, “multiply” and “add”, which are commonly used in math-intensive signal processing applications. DSP chips are widely used in a myriad of devices, such as sound cards, fax machines, modems, cellular phones, high-capacity hard disks and digital TVs.

IEC – International Electrotechnical Commission.

ISO - International Standards Organization.

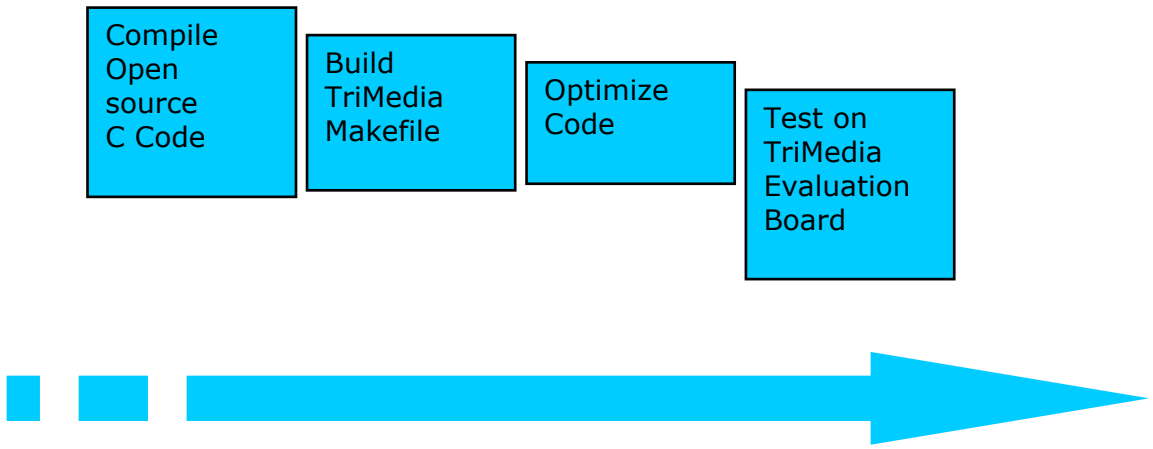
MPEG - The Moving Picture Experts Group (ISO/IEC)

Psychoacoustic - The scientific study of the perception of sound.

SIMD - Single Instruction stream Multiple Data stream - A computer architecture that performs one operation on multiple sets of data, for example, an array processor. One computer or processor is used for the control logic and the remaining processors are used as slaves, each executing the same instruction.

VLIW - Very Long Instruction Word - A CPU architecture that reads a group of instructions and executes them at the same time. For example, the group (word) might contain four instructions, and the compiler ensures that those four instructions are not dependent on each other so they can be executed simultaneously. Otherwise, it places no-ops (blank instructions) in the word where necessary.

Work Plan



Chapter 3

General Description

Introduction to the Philips TriMedia

Consumer applications are characterized by a need to be cheap. At the same time, there is new demand for high data rates applications (e.g.: video and multi channel audio), which cannot be matched by most general-purpose DSP chips.

The TriMedia tackles these needs with a well integrated set of software and hardware.

The heart of the processor is a Very Long Instruction Word (VLIW) core: this offers high speed and multiple simultaneous operations, while keeping cost down by avoiding on-chip scheduling logic. The scheduling logic is moved into an optimizing compiler which schedule the code at run time and so becomes in effect an integral part of the processor. The compiler only has to be bought once, whereas the on-chip scheduling logic would have to be paid with every chip used. In addition the TriMedia has several on-chip I/O peripherals so it will be able to cope with the multiple media. Last, but not least, is the TriMedia Co-Processor, which operate in parallel with the VLIW core and can handle demanding operations.

To maintain a high degree of re-use, TriMedia has designed a multi layered software architecture which lets application be built as a set of components that have little or no interdependency and so can be reused easily. The layered architecture also allows a separation between programmers working at low level (device drivers) and high level (operating system and application programmers).
The method of writing components and of passing data between them are very well structured and defined so that a programmer need not know the internal details of another component in order to use it; the same applies to working at different layers.

Introduction to Advanced Audio Coding (AAC)

AAC is the newest audio coding method selected by MPEG and became an international standard in April 1997. It is a fully state-of-the-art audio compression tool kit that provides performance superior to any known approach at bit rates greater than 64 kbps and excellent performance relative to the alternatives at bit rates reaching as low as 16 kbps.

The development of AAC began when researchers became convinced that significant improvements would be possible by abandoning backward compatibility to the earlier MPEG layers. The idea was to start fresh and take the best work from the world's leading audio coding laboratories. Fraunhofer Institute, Dolby, Sony and AT&T were the primary collaborators. They hoped for a result identical to *International Telecommunications Union (ITU)-R indistinguishable quality* at 64 kbps per mono channel. This was a fairly daunting requirement because it requires that no test item fall below the perceptible, but not annoying threshold in controlled listening tests.

The test items include the most difficult-to-encode audio known to researchers—isolated pitch pipe, harpsichord and glockenspiel recordings, among others. The thinking was that if a coding system passes this requirement, it would almost certainly perform well with normal program material. Pop or western classical music is tremendously easier to encode.

Compared to the previous layers, AAC takes advantage of such new tools as temporal noise shaping, backward adaptive linear prediction and enhanced joint stereo coding techniques. AAC supports a wide range of sampling rates (8–96 kHz), bit rates (16–576 kbps) and from one to 48 audio channels.

The AAC system uses a modular approach. An implementer may pick and choose among the component tools to produce a system with appropriate performance-to-complexity ratios. Three default profiles have been defined, using different combinations of the available tools:

- **Main Profile.** Uses all tools except the gain control module. Provides the highest quality for applications where the amount of random accessory memory (RAM) needed is not constrained.
- **Low-complexity Profile.** Deletes the prediction tool and reduces the temporal noise shaping tool in complexity.
- **Sample-rate Scaleable (SRS) Profile.** Adds the gain control tool to the low complexity profile. Allows the least complex decoder.

AAC is the first codec system to fulfill the ITU-R/EBU requirements for indistinguishable quality at 128 kbps/stereo. It has approximately 100% more coding power than Layer II and 30% more power than the former MPEG performance leader, Layer III.

AAC takes advantage of such tools as temporal noise shaping, backward adaptive linear prediction and enhanced joint stereo coding techniques in addition to the techniques used in ISO/MPEG Layer III.

Advanced Audio Coding Technology

Like all perceptual audio-coding schemes, AAC exploits the signal masking properties of human audio perception to achieve signal compression. AAC benefits from knowledge gained during the development of audio data compression in the 1980s and early '90s to fix weaknesses found in previous audio coders. These "fixes," referred to as either AAC modules or AAC tools, have been designed to address specific cases that occur with certain audio recordings that present encoding challenges. These modules are: filter bank, temporal noise shaping, intensity stereo, prediction, m/s stereo, quantization and coding, noiseless coding, and bitstream multiplexing.

Filter Bank

The first task of an audio coder is to break an audio sample into segments, called blocks. A time domain filter, called a window, provides smooth transitions from block to block

by modifying the data in these blocks. Choosing an optimal block size, given the wide variety of audio material, is a problem facing all audio coders. AAC handles the difficulty associated with coding audio material that vacillates between tonal (steady-state, complex spectra signals) and impulsive (transient signals) by dynamically switching between two block lengths: 2048-samples, and 256-samples, referred to as long blocks and short blocks, respectively. This, in itself, is not unique; block switching can be found in other coders, though the size of the blocks may differ. What is unique to AAC is the switching between two different types of long blocks: sine-function, and Kaiser-Bessel derived (KBD). The selection of these two long block types is based on the nature of the complex spectra of the input signal. A sine-function long block window will best represent signals of a dense spectral makeup, while signals with frequency components more widely spaced are best coded using a KBD long block window.

Temporal Noise Shaping (TNS)

The TNS technique provides enhanced control of the location, in time, of quantization noise within a filter bank window. This allows for signals that are somewhere between steady state and transient in nature. If a transient-like signal lies at an end of a long block, quantization noise will appear throughout the audio block. TNS allows for greater amounts of information to describe the non-transient locations in the block. The result is an increase in quantization noise of the transient, where masking will render the noise inaudible, and a decrease of quantization noise in the steady-state region of the audio block. Note that TNS can be applied to either the entire frequency spectrum, or to only a part of the spectrum, such that the time-domain quantization can be controlled in a frequency-dependant fashion.

Intensity Stereo

Intensity stereo coding is based on an analysis of high-frequency audio perception. Specifically, such perception is based on the energy-time envelope of this region of the audio spectrum. Intensity stereo coding allows a stereo channel pair to share a single set of spectral values for the high-frequency components with little or no loss in sound

quality. This is achieved by maintaining the unique envelope for each channel by means of a scaling operation so that each channel produces the original level after decoding.

Prediction

The prediction module is used to represent stationary, or semi-stationary, parts of an audio signal. Instead of repeating such information for sequential windows, a simple repeat instruction can be passed, resulting in a reduction of redundant information. As stated above, short blocks are utilized for non-stationary, or rapidly changing signals; it is for this reason that prediction is used only in conjunction with long blocks. The prediction process is based on a second-order backward adaptive model in which the spectral component values of the two preceding blocks are used in conjunction with each predictor. The prediction parameter is adapted on a block-by-block basis.

Mid/Side (M/S) Stereo Coding

M/S stereo coding is another data reduction module based on channel pair coding. In this case channel pair elements are analyzed as left/right and sum/difference signals on a block-by-block basis. In cases where the M/S channel pair can be represented by fewer bits, the spectral coefficients are coded, and a bit is set to note that the block has utilized m/s stereo coding. During decoding the decoded channel pair are de-matrixed back to their original left/right state.

Quantization and Coding

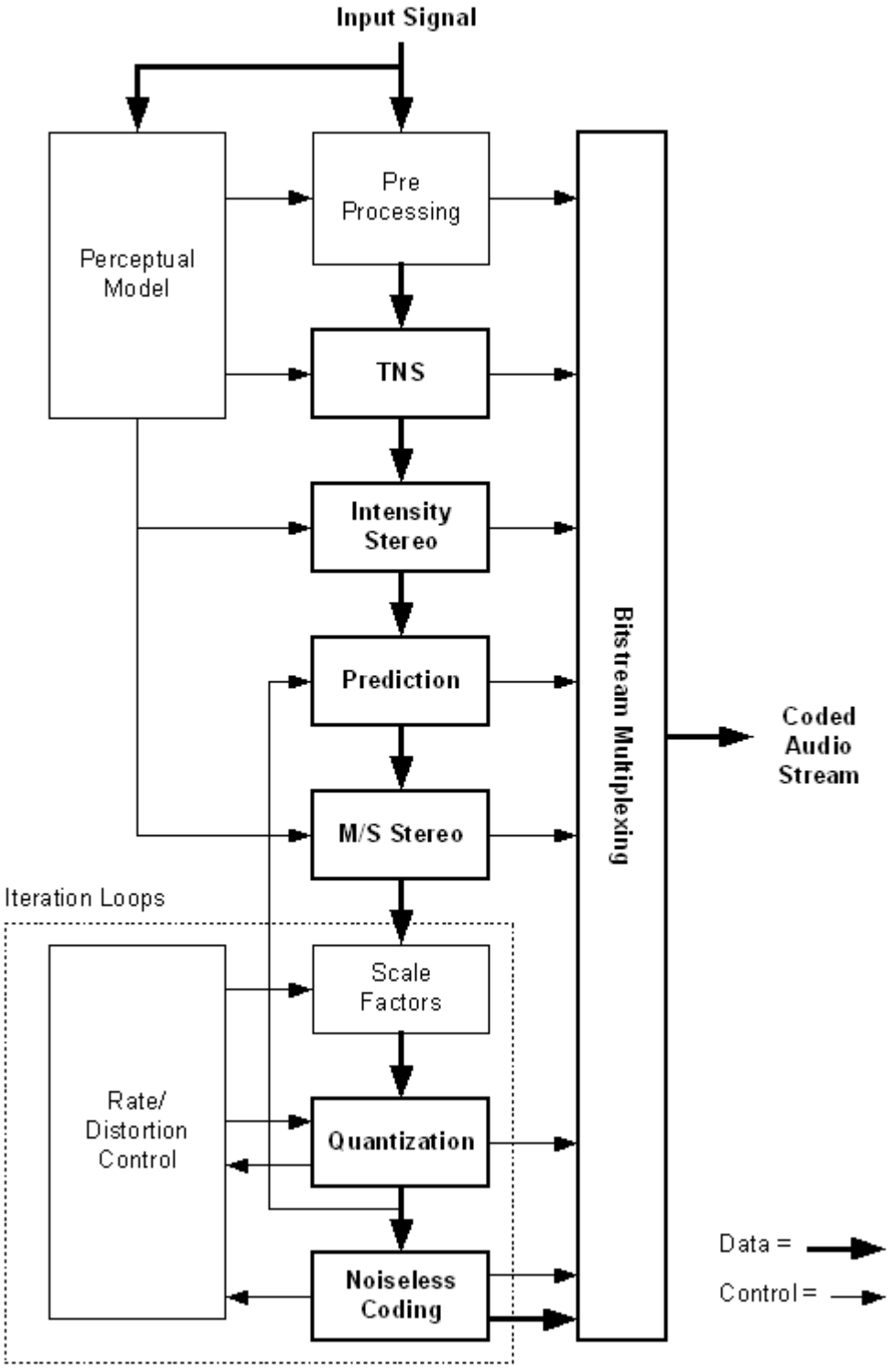
While the previously described modules attain certain levels of compression, it is in the quantization phase that the majority of data reduction occurs. This is the AAC module in which spectral data is quantized under the control of the psychoacoustic model. Its role is to determine the level and location of the resultant quantization noise. Furthermore, the number of bits used must be below a limit determined by the desired bit rate. Huffman coding is also applied in the form of twelve codebooks, allowing smaller amounts of data to represent more frequently appearing spectral coefficients. In order to increase coding gain, scale factors with spectral coefficients of value zero are not transmitted.

Noiseless Coding

This method is nested inside of the previous module, Quantization and Coding. Noiseless dynamic range compression can be applied prior to Huffman coding. A value of +/- 1 is placed in the quantized coefficient array to carry sign, while magnitude and an offset from base, to mark frequency location, are transmitted as side information. This process is only used when a net savings of bits results from its use. Up to four coefficients can be coded in this manner.

Bitstream Multiplexing

AAC has very flexible bitstream syntax. A single transport is not ideally suited to all applications, and AAC can accommodate complex audio transport logic, or can simply deliver raw data. The bitstream is broken down into two parts: transport and block. The block part consists of five sections: program configuration, audio elements, coupling elements, fill elements, and terminator. The program configuration element contains information on copyright, number of audio channels, sampling rate, etc. Audio elements are the mono, stereo, and LFE channels, which can make up anything from a mono output, to a multichannel surround output. Coupling is a form of intensity stereo coding, which allows for common information to be shared between two or more audio elements. Fill elements add bits to achieve a given bit rate, if a constant bit rate must be maintained in the decoder. The terminator element denotes the end of the block.



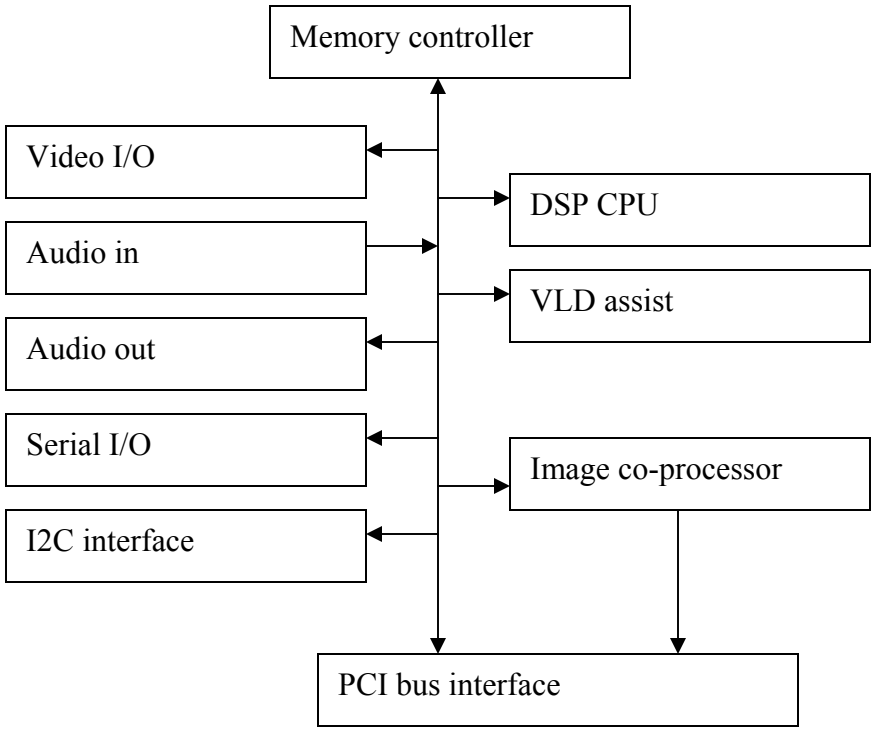
MPEG-2 AAC Encoder Block Diagram

Chapter 4

Detailed Description

The TriMedia Processor

The TriMedia processor has three independent processors, and five sets of I/O peripherals.



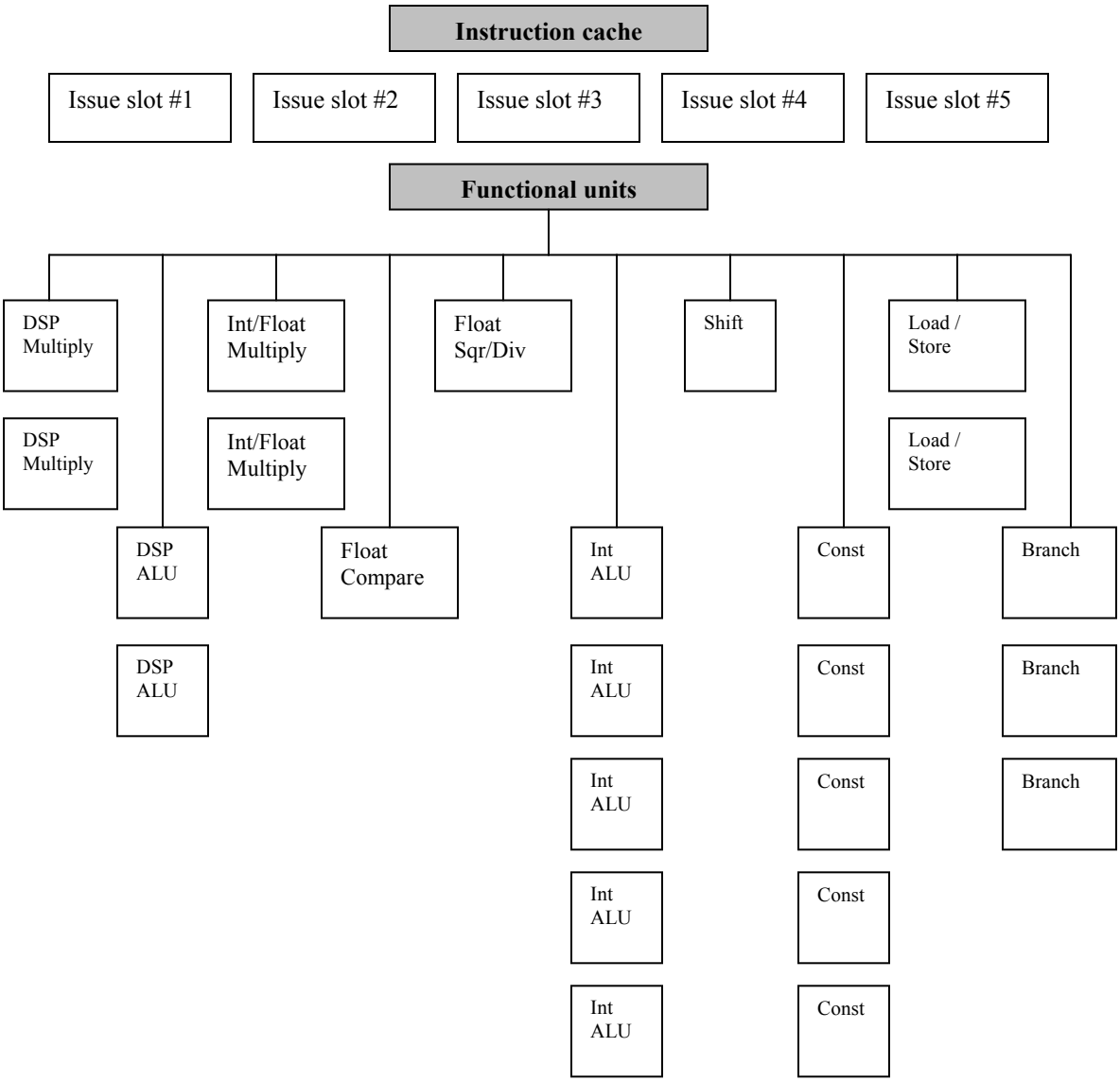
The core processor is a Very Large Instruction Word (VLIW) device with 27 functional units, 5 of which can be used in parallel at any one time. This is assisted by an image co-processor, which perform image filtering and scaling, as well handling format conversion. The image co-processor can also write direct to the PCI bus and when doing so can overlay video or graphics. The VLD assist is a special unit to help in Huffman decoding (used in MPEG AAC and video compression).

The video in and out peripherals handle CCIR 656/601 digital video data. They can also be used at up to 80 Mbyte/s for raw 8 or 10 bit data. Audio input is through a versatile interface that allows matching to any serial audio ADC and DAC – the serial format can

be programmed with control of the word length and position of data within the word. Audio output can be up to 8 channels – this is for handling multi channel surround sound (e.g.: Dolby AC-3 and MPEG AAC), which involves more than 4 speakers.

The serial I/O (synchronous interface) is designed for interaction with telecommunication standards. The I2C interface is a simple serial bus for control data

The VLIW Core



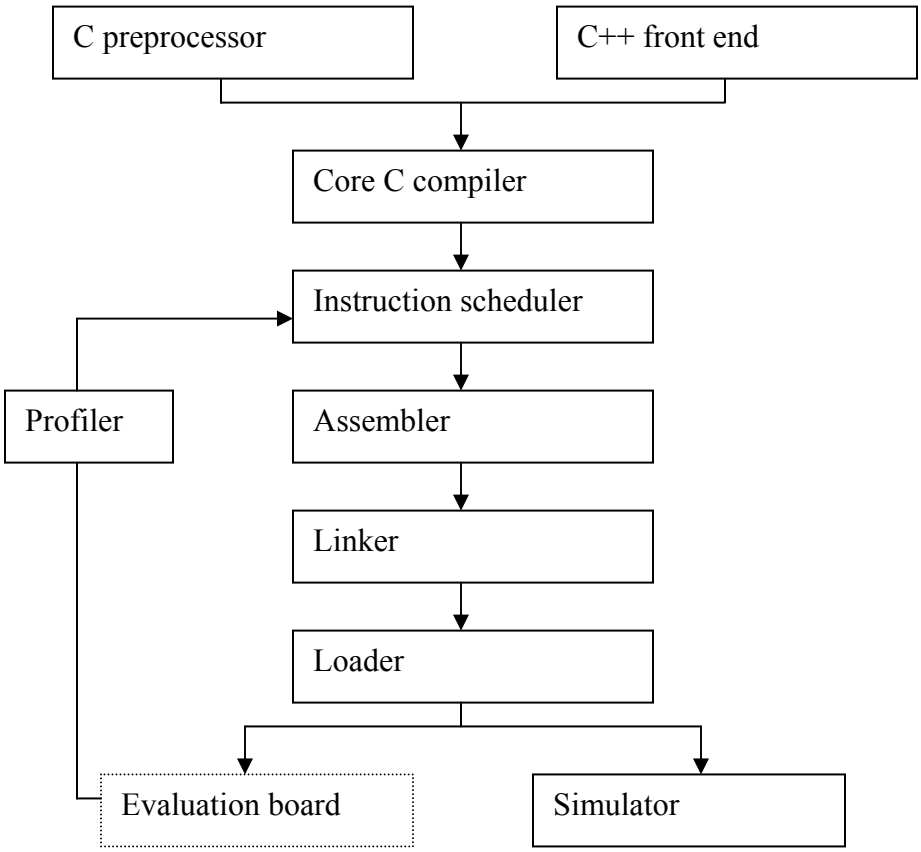
The TriMedia core uses a Very Long Instruction Word (VLIW) architecture. This has 27 functional units, of which five can be used at any time. Some operations involve a pipeline – the final result may take several cycles to emerge – but can be fed with new operands each cycle so that the throughput is one result per cycle. The processor has interlock logic so that the programmer does not have to be aware of pipeline issues. Some functional units can perform several operations in parallel when working with smaller data – for instance the data ALU can operate four 8 bit, two 16 bit or one 32 bit number at once. This means the rate processing can be higher than might at first sight appear. The very long instruction word is required simply in order to accommodate the five instructions to be issued in each cycle.

It is worth nothing that VLIW architecture is used here to make the chip cheap. A superscalar chip (like a Pentium) has multiple functional units but the scheduling (the decision which operation to perform in parallel) is done at run time, using scheduling silicon on the processor. The scheduling logic costs money and is needed on each chip. The VLIW architecture requires that the instruction scheduling be done at compile time, and is then fixed at run time. This means the scheduling logic can be omitted so the chip is cheaper than a similar superscalar device.

Scheduling Compiler

Because instruction scheduling must be done at compile time, the compiler is critical to the efficiency of the TriMedia. The compiler essentially becomes an integral part of the processor.

The compiler has to optimize instruction scheduling to use the multiple functional units to best advantage. This makes the compiler complex: but the compiler has to be paid for only once, and not for every chip bought. Also, the chip efficiency can be improved by improving the compiler, using the same silicon.



The TriMedia compiler can work on C or C++ code. This code is first turned into a sort of 'linear assembler' which specifies the operations to be done and their dependency – "Do X then do Y and add Z to W after doing A". The instruction scheduler takes this 'tree' code and creates parallel VLIW instructions.

In order to optimize for the conditions that obtain when the program is actually running, a program run on the simulator or the actual processor can be made to produce profiling information – this can be automatically fed back to the compiler to guide a further step of optimization. For complex code several iterations can produce progressively better tuned code.

The DSP CPU

The DSP CPU is the core of the TriMedia processor. It is a 32 bit processor with 27 'functional units' – instructions can be issued to five of these in a single instruction cycle. There are 10 types of functional unit – there is more than one of most types of unit as shown in the diagram. It Is worth noting that loads and stores, as well as branches, are counted as 'functional units' – and five units are concerned with generating constant values – these operations take up some of the five possible operations per cycle have to be explicitly coded for. The assembly language is quite straightforward, consisting of five instructions written in line, but to write efficient code that takes advantage of the possible parallel operations requires a lot of care and thinking ahead.

Some functional units have a pipeline – the result takes several cycles to emerge – but can be fed with new operands each cycle so that the throughput is one result per cycle. The processor has interlock logic so that the programmer does not have to worry about pipeline effects – but if a program instruction tries to access a result that is in a pipelined unit, the processor will cause a pipeline stall cycle to wait for the result.

The DSP functional units can perform several operations when working with smaller data – the DSP ALU and the DSP multiply can operate on one 32 bit number, or on two 16 bit or four 8 bit numbers, at once.

The TriMedia uses VLIW architecture. Each VLIW word issue five instructions – the instructions specify multiple, independent single operations each of which is RISC like. The VLIW architecture permits powerful parallel processing at relatively low cost. It is helpful to compare VLIW with the other main parallel processor – Superscalar.

	VLIW	**Superscalar**
Pros.	Cheaper! Make parallelism explicit in instructions. Do not require scheduling	Requires less compiler support. Binary compatibility is preserved.

	hardware	
Cons.	The compiler has to schedule parallelism. The compiler is complex and critical.	Exploit parallelism at run time. Requires scheduling hardware.

Both VLIW and Superscalar processors have multiple functional units that can be used in parallel. Both can fetch, issue, and complete more than one instruction per clock cycle – the difference lies in when the use of the units is scheduled. The Superscalar processor has special scheduling logic on chip – at run time, the scheduler looks ahead and decides how best to schedule the instructions so that it makes best use of the parallel architecture. This incurs a cost in silicon with every chip bought. The VLIW processor requires the programmer to schedule the parallel operations at compile time. This makes life harder for the programmer, but each chip is cheaper because it doesn't have the costly scheduling logic. To automate the scheduling, VLIW chips are supported by software schedulers. The TriMedia scheduler works with the C compiler. The compiler is complex, but has to be paid for only once and not with every chip bought. Also, the software can be much more sophisticated than on chip scheduling logic and so can better use the processor.

With VLIW, the compiler is an essential part of the chip. The compiler's efficiency determines the effective speed of the chip and so the two can't be evaluated in isolation. VLIW compilers are good but not perfect, so a raw measure of chip speed alone is not very helpful. On the other hand, the effective chip speed can be improved through improvements in the compiler, which can be very useful.

Instruction Format

Instructions are stored in a compressed format to economies on memory. Common operations are coded into very few bits – for example a NOP takes only two bits. The

instructions are loaded into the instruction cache in a compressed form, and decoded automatically when dispatched.

There are five slots, but there are some restrictions on which units may be used in which slots – and on how many of the same type of operation can be issued in one cycle. For example – only two load/store operations can be issued in one cycle – and no more than five results from previous operations can be written in one cycle.

The TriMedia has 128 registers of 32 bit each (this is one of TriMedia's key advantages). The register bank has 15 read and 5 write ports – three read ports and one write port for each issue slot. The write port is for writing the result from the functional unit to a register. Two of the read ports carry registers, which provide the operands for the functional unit. The third read port carries a register called the 'guard' register – this is executed or not depending on the content of this 'guard' register. Conditional execution is important in VLIW architecture – it allows the scheduler to execute chain of instructions in advance to make efficient use of 'spare' issue slots, but to use the results only if needed, perhaps as a result of a conditional branch.

The C Compiler

The C compiler is critical to the TriMedia architecture; it not only converts C/C++ programs to object code, but also schedules assembly instructions to make the most efficient use of the parallel processing units. The intention is that programmers should not need to write code at the assembler level – the C compiler aims to produce code that is efficient enough to be acceptable without hand tuning. Because it is quite hard to schedule VLIW instructions efficiently by hand, it is reasonable to expect that the majority of code will be written in C and so the compiler efficiency becomes an integral contribution of the speed of the processor.

The TriMedia compiler converts C/C++ code into a special sort of code ('tree' code) that specifies what must be done, and the dependency on previous operations having been

completed. For example, one might specify that A must be added to B, but only after the calculation of B has been completed by another operation.

The ‘tree’ code is passed to the scheduler, which in some ways the most important part of the compiler. The scheduler schedules assembler instructions to exploit the parallel units. It produces assembly language code.

The assembler code is assembled into object code, linked and can be loaded either into TriMedia hardware or into simulator for execution. Object modules can be dynamically loaded at run time, when needed – this reduces the overall system memory requirement.

The results of code execution or simulation can be profiled automatically and fed back to the scheduler to guide a further iteration. Successive iterations of scheduling and profiling can improve the efficiency of code by giving the scheduler information on where the program spends most of its time in practice

The TriMedia Audio I/O

Each peripheral I/O device in the TriMedia is configured through its own set of MMIO (Memory Mapped I/O) registers. The details of configuration are unique to each peripheral, but he basic method is consistent across all peripherals.

All peripherals work to and from memory – there are no direct connections between peripherals or between peripherals and the DSP CPU.

Technical specification regarding the audio I/O:

- Works in parallel with the DSP CPU
  - Data in and out via DMA
  - Standard serial data format
  - Supports multiple DAC outputs
- Separate over sampling clock

- For use with over sampling converters

Left[n]	…	…	…	…	…	…	Left[n+7]

Left[n]	Right[n]	…	…	…	…	Left[n+3]	Right[n+3]

Left[n]	…	…	Left[n+3]

Left[n]	Right[n]	Left[n+1]	Right[n+1]

- Frame modes
  - Philips I2S: 32 bit frame, arbitrary data start bit position and data length within the frame (MSB / LSB).
  - SuperFrame: 4,6 or 8 channels of audio – up to 512 bits per frame.
  - Simple 16 bit LSB mode.

The audio interface uses a versatile serial interface that permits multiple channels of audio.

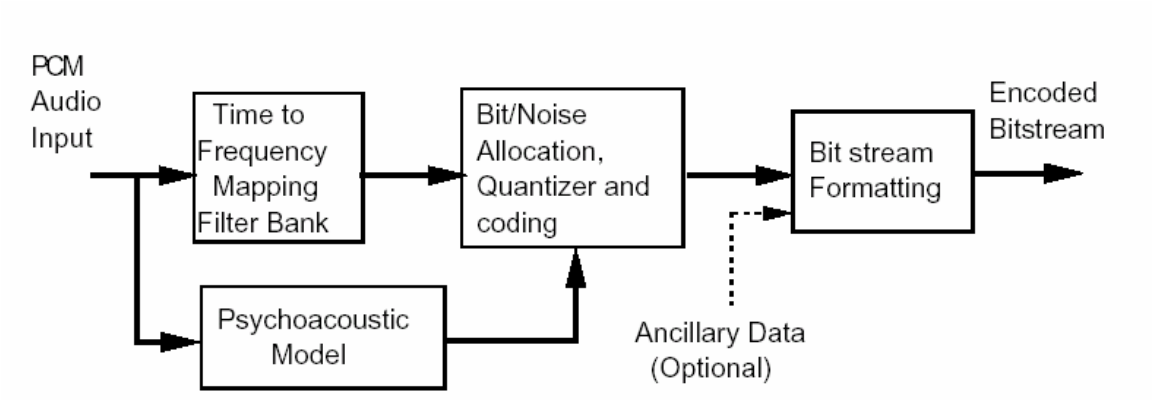
The Serial interface is synchronous – the data line is accompanied by a separate clock and word synchronization. The Philips I2S serial format supports a 32 bit frame, within which the start bit position and word length of the audio data can be specified. Stereo formats can be supported within the 32 bit frame as two 16 bit values or four 8 bit values; this can also be specified as multi channel formats using “Superframe” mode with 4, 6 or 8 channels of audio up to 512 bits per frame.

The serial interface has a separate over sampling clock – this can be extremely useful in synchronizing to over sampling converters.

MPEG Audio

The key to MPEG/audio compression is quantization. Although quantization is lossy, this algorithm can give "transparent", perceptually lossless, compression. The MPEG/audio committee conducted extensive subjective listening tests during the development of the standard. The tests showed that even with a 6-to-1 compression ratio (stereo, 16 bits/sample, audio sampled at 48 kHz compressed to 256 kbits/sec) and under optimal listening conditions, expert listeners were unable to distinguish between coded and original audio clips with statistical significance. Furthermore, these clips were specially chosen because they are difficult to compress.

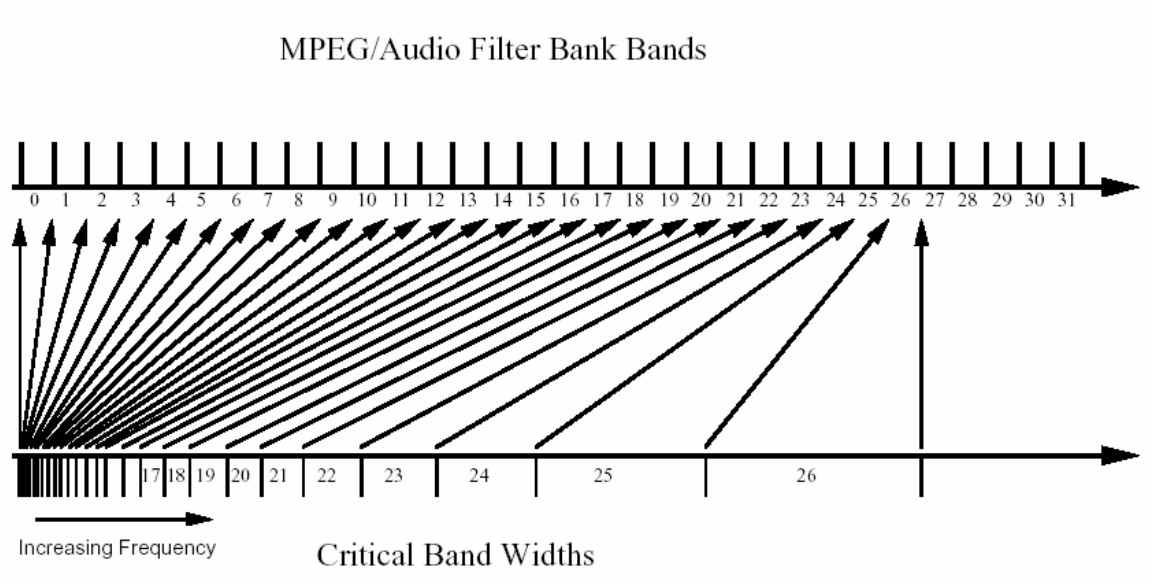
The block diagram of MPEG audio codec is as follows - the input audio stream passes through a filter bank that divides the input into multiple subbands of frequency. The input audio stream simultaneously passes through a psychoacoustic model that determines the ratio of the signal energy to the masking threshold for each subband. The bit or noise allocation block uses the signal-to-mask ratios to decide how to apportion the total number of code bits available for the quantization of the subband signals to minimize the audibility of the quantization noise. Finally, the last block takes the representation of the quantized subband samples and formats this data and side information into a coded bitstream. The decoder deciphers this bitstream, restores the quantized subband values, and reconstructs the audio signal from the subband values.



MPEG/Audio Encoder

The Polyphase Filter Bank

The polyphase filter bank is the key component common to all layers of MPEG/audio compression. This filter bank divides the audio signal into number of equal-width frequency subbands. The filters are relatively simple and provide good time resolution with reasonable frequency resolution. The design is a good compromise with three notable concessions. First, the equal widths of the subbands do not accurately reflect the human auditory system's frequency dependent behavior. The width of a "critical band" as a function of frequency is a good indicator of this behavior. Many psychoacoustic effects are consistent with a critical band frequency scaling. For example, both the perceived loudness of a signal and its audibility in the presence of a masking signal is different for signals within one critical band than for signals that extend over more than one critical band.



At lower frequencies a single subband covers several critical bands. In this circumstance the number of quantizer bits cannot be specifically tuned for the noise masking available for the individual critical bands. Instead, the critical band with the least noise masking dictates the number of quantization bits needed for the entire subband. Second, the filter bank and its inverse are not lossless transformations. Even without quantization, the inverse transformation cannot perfectly recover the original signal. However, by design

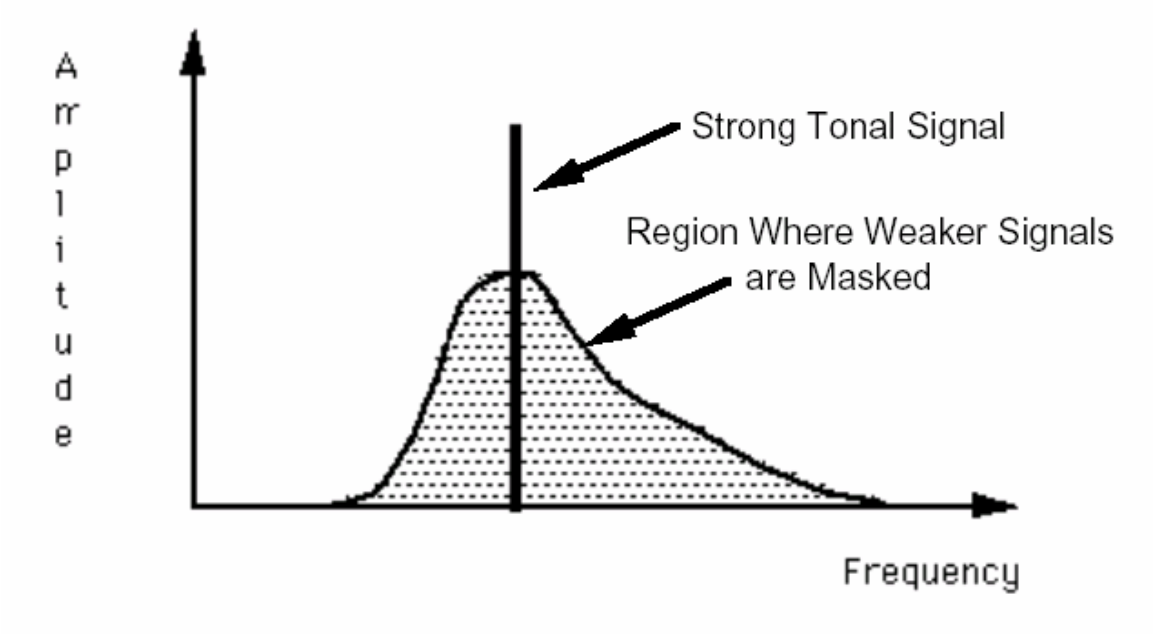
the error introduced by the filter bank is small and inaudible. Finally, adjacent filter bands have a major frequency overlap. A signal at a single frequency can affect two adjacent filter bank outputs.

Psychoacoustics

The MPEG/audio algorithm compresses the audio data in large part by removing the acoustically irrelevant parts of the audio signal. That is, it takes advantage of the human auditory system's inability to hear quantization noise under conditions of auditory masking. This masking is a perceptual property of the human auditory system that occurs whenever the presence of a strong audio signal makes a temporal or spectral neighborhood of weaker audio signals imperceptible. A variety of psychoacoustic experiments corroborate this masking phenomenon.

Empirical results also show that the human auditory system has a limited, frequency dependent, resolution. This frequency dependency can be expressed in terms of critical band widths which are less than 100 Hz for the lowest audible frequencies and more than 4 kHz at the highest. The human auditory system blurs the various signal components within a critical band although this system's frequency selectivity is much finer than a critical band.

Because of the human auditory system's frequency-dependent resolving power, the noise-masking threshold at any given frequency is solely dependent on the signal energy within a limited bandwidth neighborhood of that frequency.



MPEG/audio works by dividing the audio signal into frequency subbands that approximate critical bands, then quantizing each subband according to the audibility of quantization noise within that band. For the most efficient compression, each band should be quantized with no more levels than necessary to make the quantization noise inaudible.

Bit Allocation

The bit allocation process determines the number of code bits to be allocated to each subband based on information from the psychoacoustic model.

The MPEG/audio standard provides tables that give estimates for the signal-to-noise ratio resulting from quantizing to a given number of quantizer levels.

Once the bit allocation unit has mask-to-noise ratios for all the subbands, it searches for the subband with the lowest mask-to-noise ratio and allocates code bits to that subband. When a subband gets allocated more code bits, the bit allocation unit looks up the new estimate for the signal-to-noise ratio and recomputes that subband's mask-to-noise ratio. The process repeats until no more code bits can be allocated.

The AAC encoder uses noise allocation. The encoder iteratively varies the quantizers in an orderly way, quantizes the spectral values, counts the number of Huffman code bits required to code the audio data and actually calculates the resulting noise. If, after quantization, there are still scalefactor bands with more than the allowed distortion, the encoder amplifies the values in those scalefactor bands and effectively decreases the quantizer step size for those bands.

Stereo Redundancy Coding

The MPEG/audio compression algorithm supports two types of stereo redundancy coding: intensity stereo coding and Middle/Side (MS) stereo coding. Both forms of redundancy coding exploit another perceptual property of the human auditory system. Psychoacoustic results show that above about 2 kHz and within each critical band, the human auditory system bases its perception of stereo imaging more on the temporal envelope of the audio signal than its temporal fine structure.

In intensity stereo mode the encoder codes some upper-frequency subband outputs with a single summed signal instead of sending independent left and right channel codes for each of the 32 subband outputs. The intensity stereo decoder reconstructs the left and right channels based only on a single summed signal and independent left and right channel scale factors. With intensity stereo coding, the spectral shape of the left and right channels is the same within each intensity-coded subband but the magnitude is different.

The MS stereo mode encodes the left and right channel signals in certain frequency ranges as middle (sum of left and right) and side (difference of left and right) channels. In this mode, the encoder uses specially tuned threshold values to compress the side channel signal further.

Chapter 5

Software Structure

Directories and files

/faac

This is the project root directory; it contains the makefile and test file.

/faac/libfaac

This is the AAC directory; it contains the encoder files.

/faac/libsnd

This is the project addendum, it support several different raw audio files identification and conversion.

On this version you can find support to the following standards: WAV, AIFF, AU, RAW, PAF and SVX.

Makefile

The following make file is in UNIX type format, it must be used as argument to the attached “make.exe”.

```
#
#
#  Makefile for TriMedia pSOS components
#
#
#  Module name            : Makefile  1.14
#
#  Last update            : 12:00 - 15/06/02
#
#  Author                 : Shahar Noy
#
#
#  This is a simple makefile for a pSOS application.
#  This makefile is designed to be used in the Unix environment.
#  See its usage in $(PSOS_SYSTEM)/apps/demo1 as an example.
#  Care was taken to make it portable for use with Microsoft nmake.
```

```
#
#   Note on using the pSOS monitor in tmdbg:
#   1) add -g to CFLAGS
#   2) add $(TCS)/lib/$(ENDIAN)/psosmon.o to the link line of your application:
#      @ $(CC) \
#          $(OBJECTS) $(PSOS_SYSTEM)/sys/os/psos_tm_$(ENDIAN).o bsp.a \
#          $(TCS)/lib/$(ENDIAN)/psosmon.o $(LDFLAGS) $(CFLAGS) -o $(APPLICATION)
#   3) remove linker optimizations in LDFLAGS
#


#################### Compilation Environment Flags ####################

# Fill in these appropriately for your application and host configuration
# HOST: Win95, WinNT, MacOS, tmsim, nohost
# ENDIAN: el, eb
# add '-fp64'to CFLAGS if using 'long double'

TCS         = C:\TriMedia
#HOST        = tmsim
HOST        = WinNT
ENDIAN      = el
TARGET_TYPE = tm2
APPLICATION = a.out

# You normally should not need to change the following

CC          = $(TCS)/bin/tmcc
LD          = $(TCS)/bin/tmld
AR          = $(TCS)/bin/tmar
CINCS       = -I. -Ilibfaac/include -Ilibsnd/include
LDFLAGS     =
OFLAGS      =
CFLAGS      =  -host $(HOST) -target $(TARGET_TYPE) -$(ENDIAN)  -B -x -Xchar -Xalign
$(OFLAGS) $(DEBUG)




#################### Project Compilation Flags ####################

FAACSRC = libfaac\src
SNDSRC  = libsnd\src

LIBFFC_OBJECTS = $(FAACSRC)\util.o $(FAACSRC)\tns.o $(FAACSRC)\psych.o
$(FAACSRC)\ltp.o $(FAACSRC)\joint.o $(FAACSRC)\huffman.o $(FAACSRC)\frame.o
$(FAACSRC)\filtbank.o $(FAACSRC)\channels.o $(FAACSRC)\bitstream.o
$(FAACSRC)\backpred.o $(FAACSRC)\aacquant.o $(FAACSRC)\fft.o
LIBSND_OBJECTS = $(SNDSRC)\aiff.o $(SNDSRC)\alaw.o $(SNDSRC)\au.o
$(SNDSRC)\common.o $(SNDSRC)\paf.o $(SNDSRC)\pcm.o $(SNDSRC)\raw.o
$(SNDSRC)\sndfile.o $(SNDSRC)\svx.o $(SNDSRC)\ulaw.o $(SNDSRC)\wav.o
$(SNDSRC)\wav_float.o $(SNDSRC)\wav_ima_adpcm.o $(SNDSRC)\wav_ms_adpcm.o

MAIN_OBJECTS = test.o

OBJECTS = $(LIBFFC_OBJECTS) $(LIBSND_OBJECTS) $(MAIN_OBJECTS)
```

```
target: $(APPLICATION)


$(APPLICATION): $(OBJECTS) Makefile
        @ echo "Linking $(APPLICATION)"
        @ $(CC) $(OBJECTS) $(CFLAGS) $(LDFLAGS) -o $(APPLICATION)


%o: %c
        @ echo "Compiling $(*)c"
        @ $(CC) -tmsched -report=treestat -reportlog=$(*)inf -- $(CFLAGS) $(CINCS) -c $(*)c -o
$@


clean:
        del *.out
        del *.o
        del $(FAACSRC)\*.o
        del $(SNDSRC)\*.o
```

The test file

This is the project main file; any changes in the sample audio file must be updated here

```
#include <faac.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <sndfile.h>
#include <faac.h>


#ifndef min
#define min(a,b) ( (a) < (b) ? (a) : (b) )
#endif

/* globals */
char* progName;

int StringCompI(char const *str1, char const *str2, unsigned long len)
{
   signed int c1 = 0, c2 = 0;

   while (len--) {
      c1 = tolower(*str1++);
      c2 = tolower(*str2++);

      if (c1 == 0 || c1 != c2)
         break;
   }
```

```
    return c1 - c2;
}


int main(int argc, char *argv[])    /* start here !  */
{
    int frames, currentFrame;
    faacEncHandle hEncoder;
    SNDFILE *infile;
    SF_INFO sfinfo;

    unsigned int sr, chan;
    unsigned long samplesInput, maxBytesOutput;

    faacEncConfigurationPtr myFormat;
    unsigned int mpegVersion = MPEG2;
    unsigned int objectType = LOW;
    unsigned int useMidSide = 1;
    unsigned int useTns = 0;
    unsigned int useAdts = 1;
    unsigned int cutOff = 18000;
    unsigned long bitRate = 64000;

    char *audioFileName;
    char *aacFileName;
    char percent[200];

    short *pcmbuf;

    unsigned char *bitbuf;
    int bytesInput = 0;

    FILE *outfile;

    float totalSecs;
    int mins;

    /* point to the specified file names */
    audioFileName = "music.wav";
    aacFileName = "music.aac";

    /* open the audio input file ,get header info ,get starting location of data*/
    infile = sf_open_read(audioFileName, &sfinfo);
    if (infile == NULL)
    {
        fprintf(stderr, "Couldn't open input file %s\n", audioFileName);
        return 1;
    }

    /* open the aac output file */
    outfile = fopen(aacFileName, "wb");
    if (!outfile)
    {
        fprintf(stderr, "Couldn't create output file %s\n", aacFileName);
        return 1;
    }
```

```
/* determine input file parameters */
sr = sfinfo.samplerate;
chan = sfinfo.channels;

/* open the encoder library */
hEncoder = faacEncOpen(sr, chan, &samplesInput, &maxBytesOutput);

pcmbuf = (short*)malloc(samplesInput*sizeof(short));
bitbuf = (unsigned char*)malloc(maxBytesOutput*sizeof(unsigned char));

/* put the options in the configuration struct */
myFormat = faacEncGetCurrentConfiguration(hEncoder);
myFormat->aacObjectType = objectType;
myFormat->mpegVersion = mpegVersion;
myFormat->useTns = useTns;
myFormat->allowMidside = useMidSide;
myFormat->bitRate = bitRate;
myFormat->bandWidth = cutOff;
myFormat->outputFormat = useAdts;
if (!faacEncSetConfiguration(hEncoder, myFormat)) {
   fprintf(stderr, "Unsupported output format!\n");
   return 1;
}

if (outfile)
{

   frames = (int)(sfinfo.samples/1024+0.5);
   currentFrame = 0;

   /* encoding loop */
   for ( ;; )
   {
      int bytesWritten;

      currentFrame++;

      bytesInput = sf_read_short(infile, pcmbuf, samplesInput) * sizeof(short);

      /* call the actual encoding routine */
      bytesWritten = faacEncEncode(hEncoder,
         pcmbuf,
         bytesInput/2,
         bitbuf,
         maxBytesOutput);

      sprintf(percent, "%.2f encoding %s.",
         min((double)(currentFrame*100)/frames,100), audioFileName);
      fprintf(stderr, "%s\r", percent);

      /* all done, bail out */
      if (!bytesInput && !bytesWritten)
         break ;

      if (bytesWritten < 0)
```

```
            {
                fprintf(stderr, "faacEncEncode() failed\n");
                break ;
            }

            /* write bitstream to aac file */
            fwrite(bitbuf, 1, bytesWritten, outfile);
        }

        /* clean up */
        fclose(outfile);
    }

    faacEncClose(hEncoder);

    sf_close(infile);

    if (pcmbuf) free(pcmbuf);
    if (bitbuf) free(bitbuf);

    return 0;
}
```

Chapter 6

Examples

The following explain how to compile and run the project:

1. Put the “make.exe” file in the *TriMedia\bin* directory.
2. Update the “TCS” variable in the makefile to the TriMedia’s software library.
3. Run “make makefile” where the ‘makefile’ is located, the output should be the object code ‘a.out’
4. Type ‘tmrun a.out’ to run the object code on the evaluation board.

Chapter 7

Summary & Conclusions

Here are my conclusions regarding the AAC encoder on TriMedia –

1. The current version works as fast as a Pentium. The tested streams produced the same results as the original code generated on the PC version.
2. According to my calculation, a fully optimized AAC encoder on the TriMedia that samples a 48KHz raw music file, should be decoded at 128Kbps in less than 10MIPS requiring ~40KB ROM (program memory) and 30KB ~RAM (data memory).
3. Regarding TriMedia's announcement on their sophisticated compiler, I believe that there is some place for improvement. There is no ultimate compiler; it can't be efficient enough in terms of optimizing code, not more than a human hand can produce. Writing in machine language (assembler) level is a must in order to achieve real time performance.
4. The VLD feature in the TriMedia is not efficient enough. The stalls caused due to frequent memory access are costly; therefore it's highly recommended to build a clever software algorithm than to use the co-processor.
5. According to the profiler the most demanding function is the "Filter Bank" process. In order to achieve real-time encoding it's highly recommended to write this function in assembler. The "Filter Bank" is a fundamental component in the audio coding process – it represents the conversion of the time domain signals into time-frequency representation. This conversion is done by a forward modified discrete cosine transform (MDCT).

6. There are well known techniques to accelerate DCT based algorithm using multiply and accumulate instructions. In addition, the TriMedia VLIW architecture and its 128 registers can easily take the advantage of fast MDCT implementation.
7. Further improvement can be reached while reading the whole raw file to a buffer and producing the output into another buffer. The read/write operations to the hard disk are expensive in terms of cycles. In order to prepare the project for real-time voice/audio coding (e.g. from microphone input) it will be recommended to define buffer that is large enough to hold the current processed frame (bitstream) in any give moment.

Chapter 8

Appendixes

Chapter 9

Bibliography

1. ISO/IEC International Standard IS 11172-3 "Information Technology - Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5 Mbits/s - Part 3: Audio"
2. ISO/IEC International Standard IS 13818-7 "Information Technology - Coding of Moving Pictures and Associated Audio Information – Part7: Advanced Audio Coding (AAC) ".
3. Techniques & Standards For Image, Video & Audio Coding – K.R. Rao, J.J. Hwang.
4. TriMedia's Compiler User Manual
5. C code based on open source from SourceForge - http://www.audiocoding.com/index.php
6. AAC official website - http://www.aac-audio.com/