Design of an MPEG audio layer 2 CODEC on a DSP

Baijens, M.C.H.

Award date:
1999

Technische Universiteit Eindhoven

Faculty of Electrical Engineering
Section of Information and Communication Systems

Master's Thesis

Design of an MPEG audio layer 2 CODEC on a DSP

M.C.H. Baijens

Coaches: ir. H. Kester, ir. J. Smeets
Supervisor: Prof. ir. M.P.J. Stevens
Date: September 1999

Summary

To reduce the amount of personnel needed for the surveillance of a building with many different rooms, Ellips B.V. is developing a Multimedia Surveillance System. The system sends audio and video information to a central control room. The audio from the different rooms has to be transmitted over a network with limited bandwidth, therefor audio compression is needed.

For the audio compression algorithm, MPEG audio layer 2 compression has been chosen. The audio compression and decompression will be done by a DSP56002 processor. This report will deal with the design of a real-time MPEG audio layer 2 CODEC for the DSP56002. This involves the software design of the CODEC as well as the determination what hardware is needed. The starting point will be the ISO/IEC example source that is distributed over the internet, called "dist10", and the Motorola DSP56002 evaluation module, called "DSP56002EVM".

From there the design will involve the following steps:

- Get the source ready and functional in a PC environment. Also remove the unnecessary code (other layers, other psycho-acoustical model, etc.).
- Convert this source to a fixed-point implementation on the PC.
- Port the fixed-point PC implementation to DSP56002.
- Make DSP source real-time by implementing time critical parts in assembly.
- Adjust hardware where necessary during these steps.

The MPEG audio layer 2 CODEC has been designed for the DSP56002. The MPEG streams generated by the encoder have been verified and are valid MPEG audio layer 2 streams. The decoder has been tested with reference streams and it decoded those streams correctly. Listening to the CODEC verified the audio quality to be quite good. To ensure high quality audio a bitrate of 128 kbit/s per channel is preferred, but operation on a lower bitrate of 64 kbit/s and thus a higher compression ratio still produces acceptable audio quality.

For single channel operation with sampling frequencies of 22 kHz and below, the CODEC achieves real-time operation. For higher sampling frequencies and stereo operation only the decoder performs real-time. Because one channel mode and a sampling frequency of 22 kHz are sufficient for the speech that will be transferred over the surveillance system, the goal of the project has been fulfilled.

The required hardware for the CODEC is a DSP56002 at least running at 66 MHz and 128 k-words (24-bit) of 15 ns or faster SRAM.

Table of Contents

1. Introduction

Surveillance of a building with many different rooms can be a very intensive task, asking for the presence of a lot of personnel. The needed manpower and thus the cost of the surveillance can be reduced by means of an electronic system. Ellips B.V. is developing such a system, more precisely a Multimedia Surveillance System. Figure 1 gives an overview of the surveillance system.

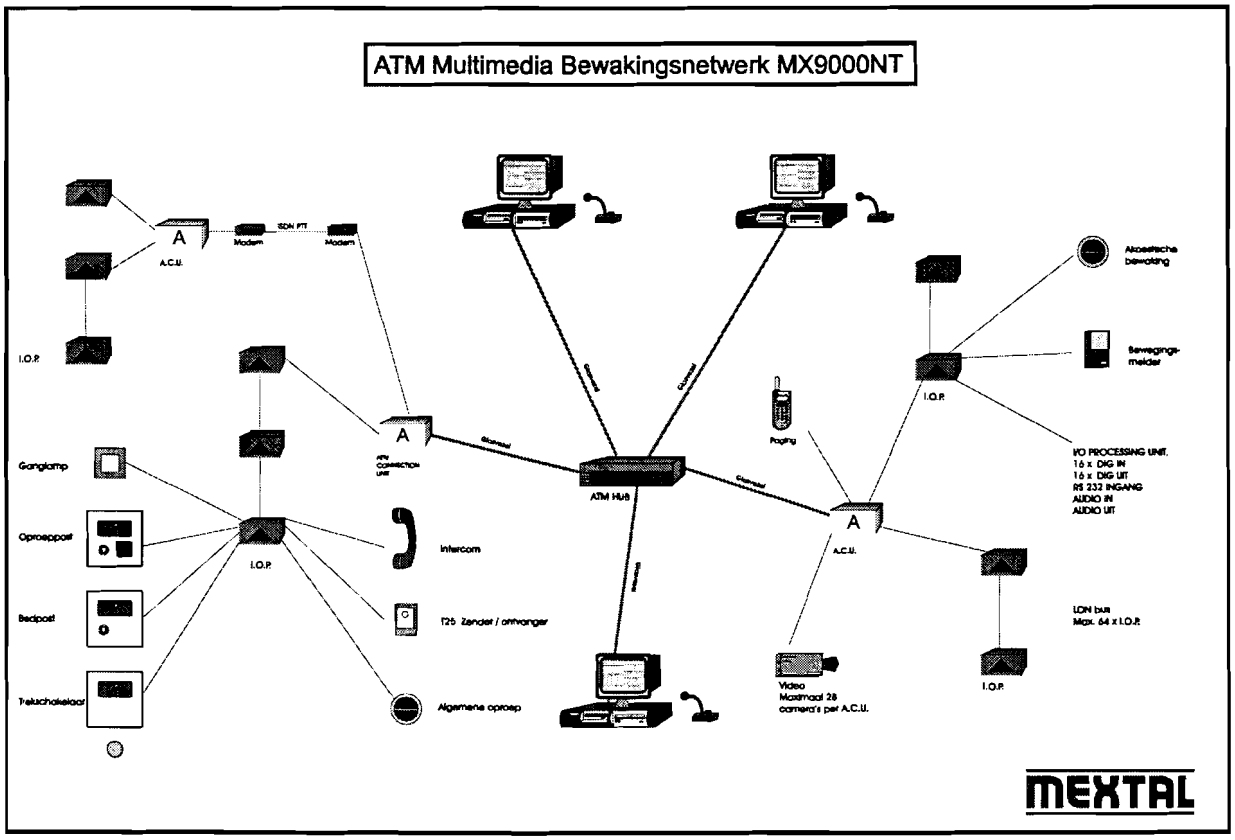

Figure 1 Overview of the surveillance system

The core of the system is an ATM HUB. The ATM HUB takes care of the data communication between the monitoring stations and the ATM Connection Units (ACU). Because the ATM network has to be able to handle video streams, it is a fiber network. The monitoring stations can communicate with all ACUs. An ACU communicates with the actual surveillance devices. These consist of cameras providing video streams of the rooms and I/O processors (IOPs). The IOPs take care of audio communication with a room and can handle several controls, like lights and camera control.

The IOPs are connected to the ACU by means of a LON network. The LON-bus allows a maximum of 64 devices to be connected to it. All data communication on the LON-bus is done via a LON-controller. So the ACU is equipped with the LON-controller and 63 IOPs can be connected to the bus. The LON-bus has a limited bandwidth of 1.25 Mbit/s. The limited bandwidth leads us to the problem.

Over the LON-bus high quality audio has to be transferred from and to the IOPs. Most of the audio stream will be speech or other vocal sounds. So a sampling frequency of 22 kHz with 16 bit precision will be sufficient. The required bitrate for transmitting such an audio stream is 352 kbit/s. So only three such audio streams can be transmitted over the LON network at the same time, which of course strongly limits the power of the total system. To increase the number of audio streams that can be transmitted compression will have to be applied to the audio streams.

Nowadays MPEG audio compression is very popular in distributing CD quality audio. Because high quality audio is wanted, MPEG audio compression is the choice for the surveillance system. The MPEG audio standard basically consists of three levels of compression, called layer 1, layer 2 and layer 3. The higher the layer, the higher the compression ratio that is achieved. However the complexity of the algorithm and thus the required processing power also increases for higher layers. Layer 3 is very popular for distributing music, however the extra compression compared to layer 2 is rather modest, while the required extra processing power is rather large. The difference in needed processing power between layer 1 and layer 2 is rather small. Considering this, layer 2 seems the best tradeoff between processing power and compression ratio and is therefor chosen for the surveillance system.

Once the layer has been chosen, the quality and compression ratio can still be adjusted. This is done by selecting a bitrate at which the CODEC will operate. Because the surveillance system requires good quality audio, the bitrate shouldn't be below 64 kbit/s per channel. On the other hand, the difference between CD audio and a MPEG layer 2 CODEC using a bitrate of 128 kbit/s per channel is hard to hear. For a surveillance system, only mono operation is interesting, so the total bitrate should be chosen somewhere between 64 kbit/s and 128 kbit/s. Compared to the original audio stream of 352 kbit/s, this will deliver a compression factor between 2.75 (128 kbit/s) and 5.5 (64 kbit/s). At a bitrate of 64 kbit/s approximately 16 audio streams can be transferred over the LON-bus at the same time and at a bitrate of 128 kbit/s approximately 8 audio streams can be transferred at the same time.

Widespread use of the MPEG compression technique has become possible by the development of fast Digital Signal Processors (DSPs). The MPEG audio layer 2 CODECs on both the ACU and the IOPs will be implemented on a Digital Signal Processor (DSP). The CODEC on the ACU will have to be able to handle multiple audio streams at the same time, while the CODEC on an IOP only needs to handle one audio-stream at the same time. Considering this, the DSP processor on the ACU needs to be a lot more powerful than the DSP processor on an IOP. Now a DSP has to be chosen that is most suitable for the application. First one has to decide between a fixed-point or a floating-point DSP. The cost-aspect favors a fixed-point DSP. To simplify the implementation and to maintain precision, a word-width of at least 20 bit is desired. Another important aspect is code compatibility for the ACU and the IOPs. The Motorola DSP56xxx series fulfills these constraints the best. The IOPs will be equipped with the DSP56002 and the ACU with the more powerful DSP56303. The DSP56303 has a code compatibility mode with the DSP56002, so the code designed for the DSP56002 can be run on the faster DSP56303. Both DSPs have a 24 bit word width.

The goal of this project is to design a real-time MPEG audio layer 2 CODEC for the DSP56002. This involves the software design of the CODEC as well as the determination what hardware is needed. The starting point will be the ISO/IEC example source that is distributed over the internet, called "dist10", and the Motorola DSP56002 evaluation module, called "DSP56002EVM".
From there the design will involve the following steps:

- Get the source ready and functional in a PC environment. Also remove the unnecessary code (other layers, other psycho-acoustical model, etc.).
- Convert this source to a fixed-point implementation on the PC.
- Port the fixed-point PC implementation to DSP56002.
- Make DSP source real-time by implementing time critical parts in assembly.
- Adjust hardware where necessary during these steps.

In the following chapters, first the available hardware will be discussed, then the MPEG audio format and the encoder and decoder algorithms will be discussed. Thereafter the implementation issues are subject of discussion, followed by some experiments.

2. The Hardware

2.1 The IOP

IOP stands for I/O processor, meaning that the IOP will have to handle most I/O actions. Figure 2 gives an overview of the IOP. The main processor of the IOP is the

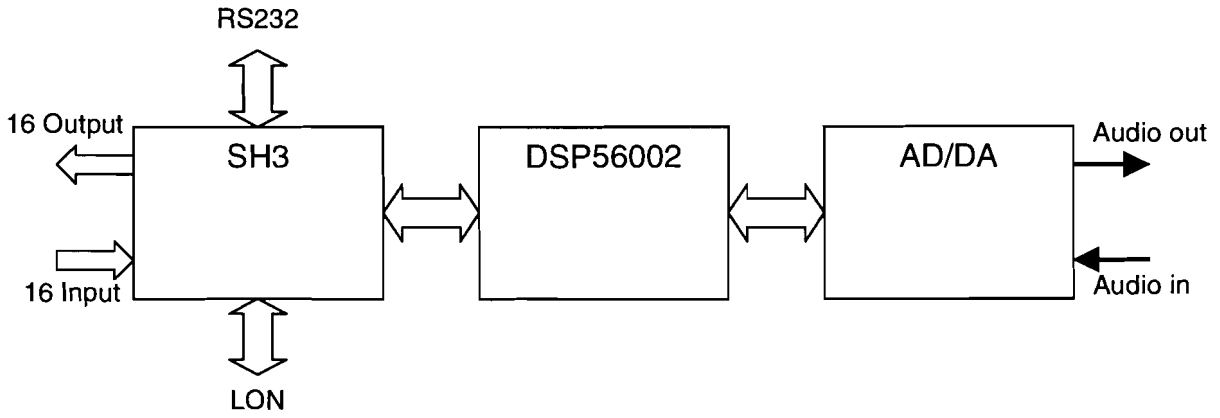

Figure 2 An overview of the IOP

Hitachi SH3. The SH3 is responsible for controlling the 16 inputs and outputs, the RS232 port and the communication with the LON-bus and the DSP. The audio path leads through an AD/DA converter, the DSP56002 running the MPEG audio CODEC and the SH3, which forms a buffer between the DSP and the LON-bus.

2.2 The DSP56002

The DSP56002 is a member of the DSP560xx series, meaning that it is based on the 24-bit DSP56000 core. The difference with other members of the family are the interfaces and the amount of on-chip memory. Figure 3 gives a picture of the DSP56002 architecture. As can be seen in the figure, the DSP56000 core has four 24 bit data buses, the Global Data Bus (GDB), the Program Data Bus (PDB), the X Data Bus (XDB) and the Y Data Bus (YDB). There are three address buses, the Program Address Bus (PAB), the X Address Bus (XAB) and the Y Address Bus (YAB), which can address three memory blocks: X memory, Y memory and Program memory. Access to Program memory occurs over the PDB, access to X memory occurs over the XDB and access to Y memory occurs over the YDB. In this way data transfers to/from X memory, Y memory and P memory can be carried out in parallel. More precisely, an instruction fetch and two data moves can be executed in the same machine cycle. The GDB is used for I/O transfers. For I/O addressing the XAB is used by offering the upper 64 bytes of X address space.

The external data bus also has a width of 24 bit. Because we are dealing with the development of a high performance application, zero-wait-state external data access is wanted. Only one data bus can be connected to the external data bus at the same time. Thus parallel moves can only be performed efficiently if the external data bus

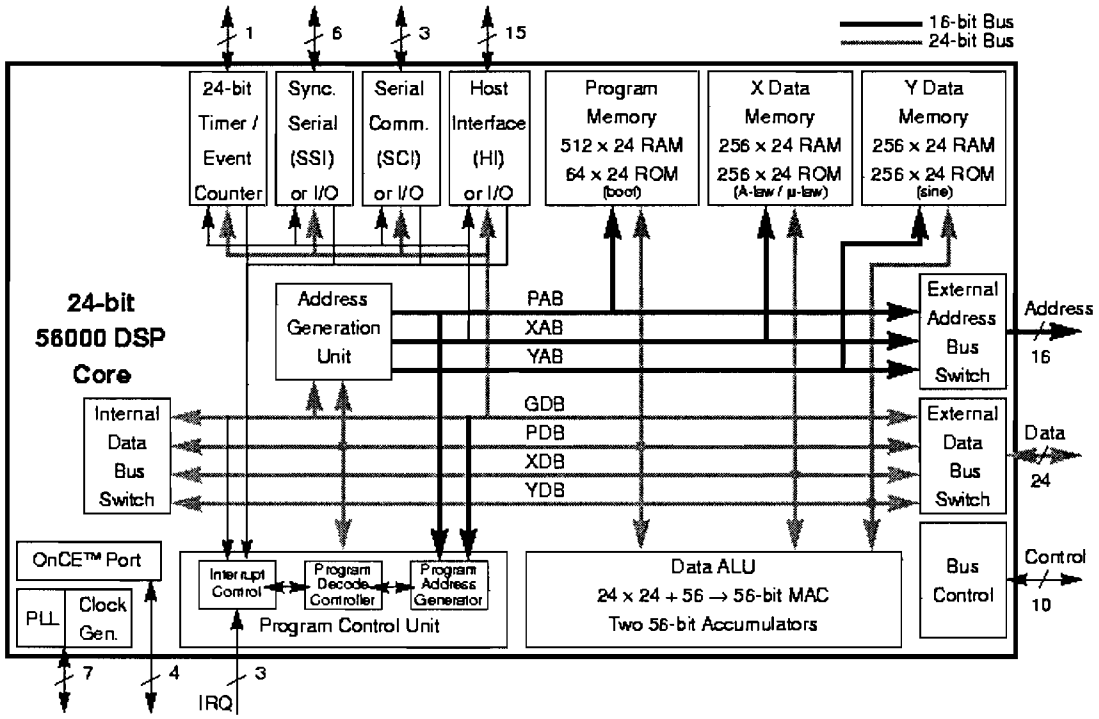

Figure 3 The DSP56002 architecture

has to be accessed at maximum one time per instruction. Extra external RAM accesses will cost one clock-cycle per access.

The DSP56002 internal memory consists of 512 words of program memory, 256 words of X memory and 256 words of Y memory. For highest performance it is important to make optimal use of the internal memory. So critical code has to be placed in the internal program memory and the variables used in this code should be placed in internal X memory or internal Y memory. The external RAM can also be divided in X memory, Y memory and Program memory. The selection is done by bus control signals. The addressing space for each memory region is 64K words. Thus the total addressing space is 192K words.

The ALU performs all arithmetic and logic operations on data operands. They are all executed in a single instruction cycle. The results can be stored in one of the two available 56-bit accumulators, allowing an 8-bit overflow protection for 48-bit operations. The Address Generation Unit (AGU) takes care of address storage and effective address calculations. This unit operates in parallel with other units.

The following interfaces to communicate with other devices are available:
- Serial Communication Interface (SCI)
- Synchronous Serial Interface (SSI)
- Host Interface (HI)

The pins used for these interfaces can also be configured as General Purpose I/O, allowing for a total of 24 GPIO pins.

To configure the onboard devices some control registers are available. Besides the control registers that are present to program and control the I/O pins, some other onboard functions can be programmed. They include a timer, memory control and the PLL. By programming the PLL the clock speed of the core can be programmed, allowing the use of low external clock speed. The memory control registers determine the availability of internal ROMs and the number of wait-states for external memory access.

The DSP56002 is equipped with an OnCE (On-chip Emulation) controller. Via the OnCE controller programs can be downloaded to the DSP56002 and the OnCE controller is also used for debugging. By sending an external command to the OnCE controller, execution of the program can be stopped at anytime and the debugging process can begin.

2.3 The DSP56002EVM

The DSP56002EVM is an evaluation board for the DSP56002 created by Motorola. The onboard DSP56002 is capable of running at a clock speed of 66 MHz. The DSP56002EVM features onboard 32Kx24 SRAM, an onboard AD/DA-converter (CS4215) and a RS-232 interface to PC debugging environment. The conversion from OnCE to RS-232 is done by a MC68HC705 microcontroller. See figure 4.

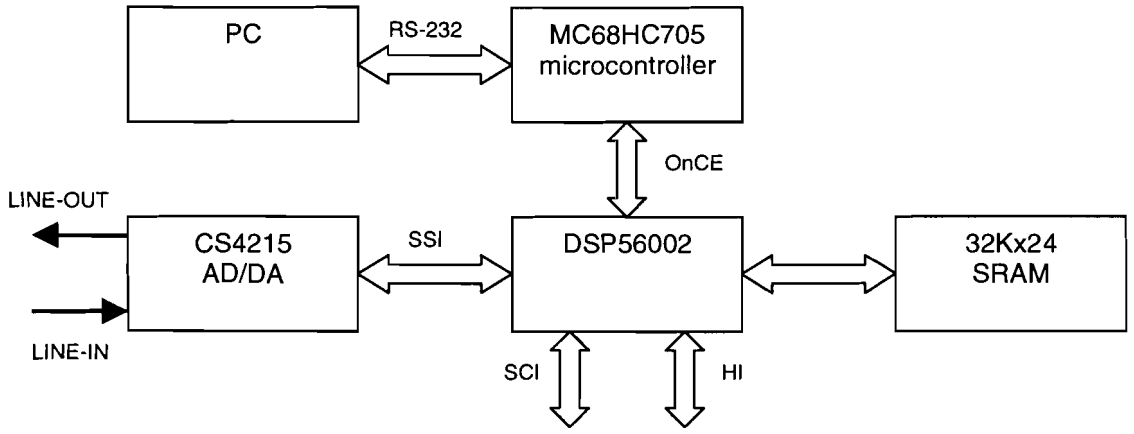

Figure 4 DSP56002EVM Block Diagram

The board as it comes from Motorola has the following problems:

(a) The download of a program to the board is very slow. A program of 64 kB will already take an hour to download.

(b) For an MPEG audio layer 2 CODEC at least 192 kB of RAM is needed and preferably more for simplifying the debugging process.

(c) The crystal-oscillator of the AD/DA-converter doesn't support 22 kHz audio and the crystal is connected to the wrong pins of the CS4215 according to the data-sheets.

Problem (a) will be solved by creating the same interface to the DSP as the DSP56002ADM. The DSP56002ADM is a more advanced debugging board also available at Motorola. The DSP56002ADM communicates through the OnCE port with a separate command converter board, which on its turn communicates with a PC interface card. The PC interface card and the command converter are ordered and a connector for the OnCE port is created on the DSP56002. The MC68HC705 has to be removed. Now the development tools for the DSP56002ADM will have to be used. These can be downloaded for free from the Motorola website. With the new software and hardware the download time for a 64 kB program has been reduced from an hour to several seconds.

The solution to problem (b) is to extend the amount of memory on the DSP56002EVM. The three 32Kx8 chips are replaced by three 128Kx8 chips. Now the total amount of memory is 128K words. The DSP56002 can only address 64K words of memory with its 16 address lines. So the 128K words of memory will have to be divided between program, X and Y memory. The C-compiler needs separate program and data memory regions. Because the C-compiler only uses Program and Y memory, two separate 64K word regions are created, one for Program memory and one for Y memory. Because the data of the program is expected to be much larger than the program code itself, the X memory is mapped over the Program memory. In this way all free program memory can be used for X data memory. The selection is done by connecting the X/Y-pin of the DSP to the highest address line (A16) of the SRAM chips. Figure 5 shows the created memory layout.

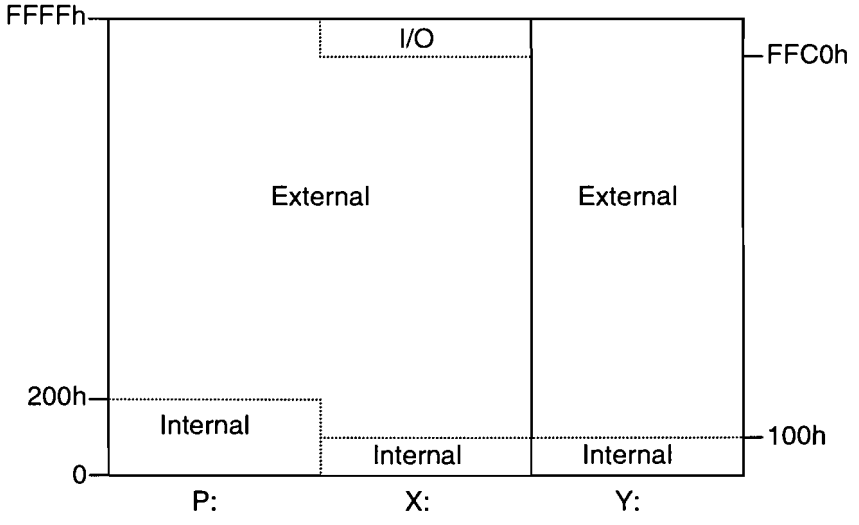

Figure 5 The DSP memory layout

Problem (c) can be solved by replacing the 24.576 MHz crystal on the XTAL2 pins of the CS4215 with a 16.9344 MHz crystal. When the removed 24.576 MHz crystal is connected to the XTAL1 pins of the CS4215 the original frequencies can still be selected, providing support for all frequencies used by the MPEG audio standard. The block diagram of the modified DSP56002EVM can be found in figure 6.

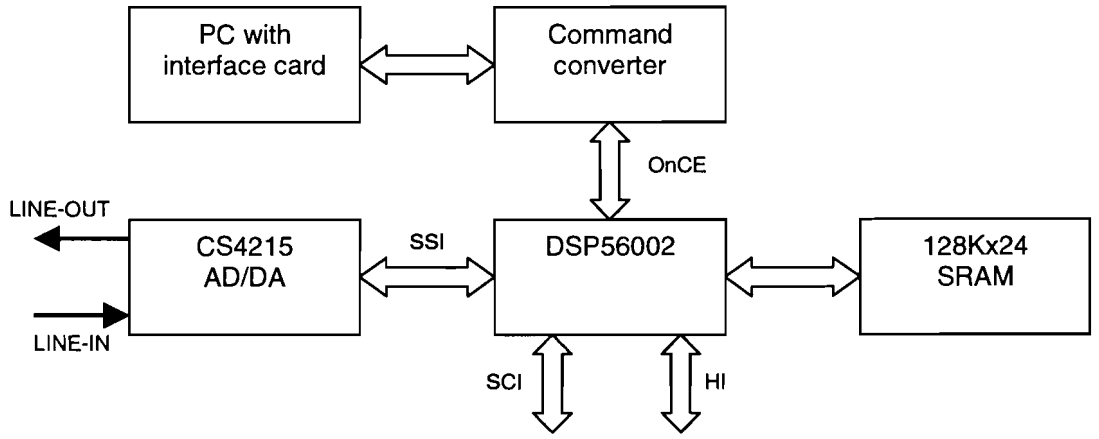

Figure 6 Modified DSP56002EVM Block Diagram

2.4 Data connection to PC

When testing the MPEG data stream needs to be transferred between DSP and PC. When the DSP is decoding the PC will have to deliver the MPEG data to the DSP and when encoding the DSP will deliver MPEG data to the PC. In the IOP the Host Interface will be used to communicate with the host processor. However the Host Interface needs a lot of lines to be connected to the PC and a rather complicated program at the PC side. Considering this, the decision has been made to create a simple interface based on handshaking. For 8 bit communication, 8 data lines are needed and two lines for handshaking. At the DSP side the Host Interface lines will be configured for GPIO of which ten are needed. The Host Interface is also referred to as DSP Port B. For fast handling of the data bytes the lower eight bits should be used as data lines. At the PC side an I/O card with an 8255, a general purpose programmable I/O device, is used. The 8255 has 24 general purpose I/O pins, mapped onto three 8 bit I/O ports (port A, port B and port C). Port A is used for the 8 data lines, while bit 0 of port B is used for the outgoing handshake signal and bit 0 of port C is used for the incoming handshake signal. Figure 7 displays the situation as it is actually used. The communication software has been written for this particular

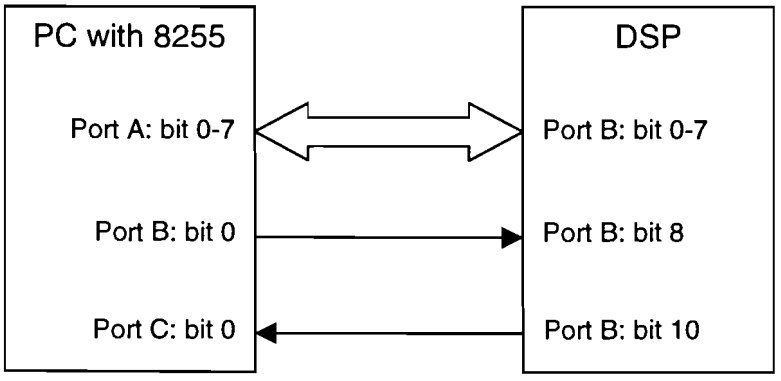

Figure 7 Communication between PC and DSP

situation. For the DSP side the transmitter software has been integrated in the encoder and the receiver software has been integrated in the decoder. For the PC side receiver and transmitter software are two separate programs.

2.5 Data-flow

Because the DSP won't have to be able to encode and decode at the same time, there are two possible operating modes: encoding and decoding.

- When encoding the CS4215 will be sampling audio using its line input. The audio samples are transferred to the DSP over the SSI bus between the CS4215 and DSP. The DSP will encode the audio samples to MPEG layer 2 audio data. Now the MPEG data is sent to the 8255 on the PC. The PC will receive the MPEG data and write it to a file. This file can be played to listen to the result.
- When decoding a valid MPEG layer 2 audio file needs to be present at the PC side. This file will be transmitted to the DSP. The DSP will decode the received MPEG data to raw audio samples. Finally these samples are sent to the CS4215 via the SSI bus. The audio can be heard at the line output or the headphone output of the CS4215.

3. The MPEG audio layer 2 format

The MPEG audio standard starts with the definition of the MPEG1 standard [1]. The MPEG1 standard defines three layers of compression (layer 1, layer 2 and layer 3) and two psycho-acoustical models (model 1 and model 2). Psycho-acoustical model 2 can be used for all layers, while psycho-acoustical model 1 should only be applied to layer 1 and layer 2 CODECs. The MPEG1 standard supports three different sampling frequencies: 32 kHz, 44.1 kHz and 48 kHz. Layer 2 combined with psycho-acoustical model 1 was chosen, because it seems to be a good tradeoff between code complexity and compression quality.

The next MPEG audio standard definition was the MPEG2 standard [2]. This is an extension to the MPEG1 standard. Actually it consists of two extensions: MPEG2-mc and MPEG2-lsf. MPEG2-mc stands for MPEG2 multi channel and is a multi channel extension to the MPEG1 standard. MPEG2-lsf stands for MPEG2 low sampling frequency and adds support for lower sampling frequencies to the MPEG1 standard. The added sampling frequencies are: 16 kHz, 22.05 kHz and 24 kHz. These frequencies were initially added to improve the quality for bitrates below 64 kbit/s. However they also seem to be very effective to reduce processor load. By using these frequencies the amount of data the processor will have to handle is halved. Considering the fact that the system is going to work with human voices, the 22.05 kHz frequency is chosen, resulting in an audio bandwidth of approximately 10 kHz.

From the above follows that the CODEC will be MPEG2-lsf layer 2 working at 22 kHz. To spare memory the code necessary to support the MPEG1 frequencies can be removed. Of course this also removes the ability to handle MPEG1 data. Now the format of an MPEG2-lsf layer 2 bitstream will be discussed.

An MPEG audio data stream consists of a series of frames. The length of these frames depends on the chosen bitrate. For layer 2 every frame corresponds to 1152 PCM samples. So when a frame is decoded, the output of the decoder will be 1152 PCM samples. For a frequency of 22.05 kHz, the frame duration can be calculated:

$$\tau = \frac{samples}{f_s} = \frac{1152}{22050} = 0.052s$$

The frame duration is the maximum time that a real-time decoder or encoder is allowed to spend for each frame. Figure 8 gives the format of a layer 2 frame.

Header (32)	CRC (0,16)	Bit Alloc (26-188)	SCFSI (0-60)	Scalefactors (0-1080)	Samples	Anc. Data

Figure 8 The format of a layer 2 frame

Now the different fields of a frame will be discussed.

- The header

The header of a frame is 32 bit wide (figure 9) and it can be detected by its first 12 bits, forming a sync word. Further the header contains all kinds of parameters about the frame. The most important ones are the ID field (distinguishes between MPEG1 and MPEG2), the layer field (tells the layer of the frame), the bitrate field (indicating the bitrate), the sampling frequency field (indicates the sampling frequency) and the mode field (stereo, joint stereo, dual channel or single channel).

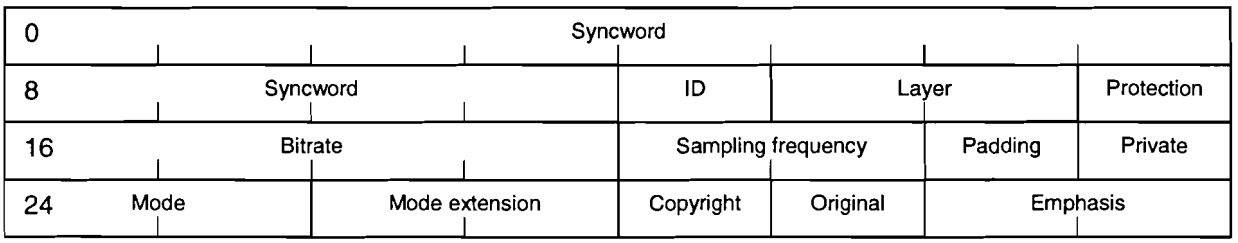

Figure 9 The header of a frame

- The CRC

If the protection bit in the header is zero, this field holds a 16 bit CRC code. Otherwise the CRC field is not present in the frame.

- Bit Allocation

Contains information about the quantizers used for the subband samples, whether the samples have been grouped and on the number of bits used to code the samples. Actually only indexes to bit allocation tables, known by the CODEC, are stored here.

- SCFSI, Scale Factor Selection Information

Gives information about the number of scalefactors transferred per subband. Also tells for which part of the frame they are valid.

- Scalefactors

Indicate the multiplication factor for dequantization of the subband samples. An index to a table known by the CODEC is stored.

- Samples

The coded subband samples. Three consecutive samples can be grouped to one code as indicated by the bit allocation.

- Ancillary data

This field is optional. The length is variable and is defined by the distance between the end of the audio data and the next frame header. This field is also used by MPEG2-mc standard to code the information for the extra channels, while staying compatible with the MPEG1 standard.

4. The Decoder

4.1 Introduction

The decoder basically exists of three parts. First the encoded bitstream is read. The decoder will extract the header information, the bit allocation, the scale factor selection information, the scale factors and the coded subband samples. In the next step the acquired information is used for the reconstruction of the subband samples. The last step takes care of the frequency to time mapping of the subband samples and thus the reconstruction of the PCM samples. Figure 10 gives an overview.

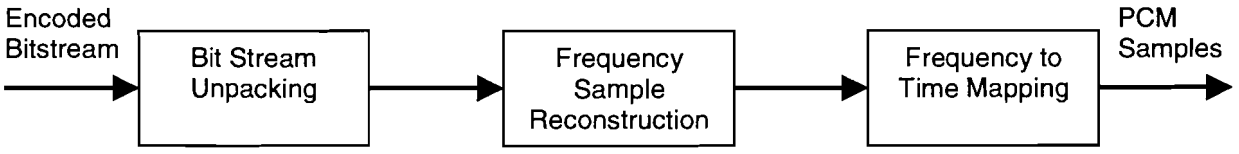

Figure 10 MPEG audio decoding

4.2 Software implementation of the decoder

To be able to implement the decoding process, a software flowchart of the decoder is created. The flowchart is based on the decoder flowchart in [1] and is shown in figure 11. Below some information about the different steps in the decoding process is given.

1. **Initialization:** The initialization is responsible for the creation and initialization of variables and memory blocks. Also the communication with the outside world, for reading the incoming bitstream and writing the outgoing PCM samples, has to be set up.
2. **Find start of frame:** The incoming bitstream is scanned for the beginning of a frame. The beginning of a frame is recognized by the presence of a sync word in the header at the beginning of each frame.
3. **Read & decode header:** The header is read from the bitstream. The parameters about the frame are decoded and stored for later use.
4. **Header ok?:** The parameters that the header gives us about the frame are verified. If the frame is not the expected type, it is declared invalid and the program returns to step 2.
5. **Read & decode bit allocation:** The bit allocation information is read, decoded and stored for later use.
6. **Read & decode scale factors:** The scale factor selection information is read and decoded. Using the scale factor selection information, the scale factors are read and stored.
7. **Read samples:** Now the subband samples are read from the bitstream. The bit allocation information indicates how many bits are allocated for each sample.
8. **Dequantize samples:** The subband samples are dequantized with the appropriate scale factors.

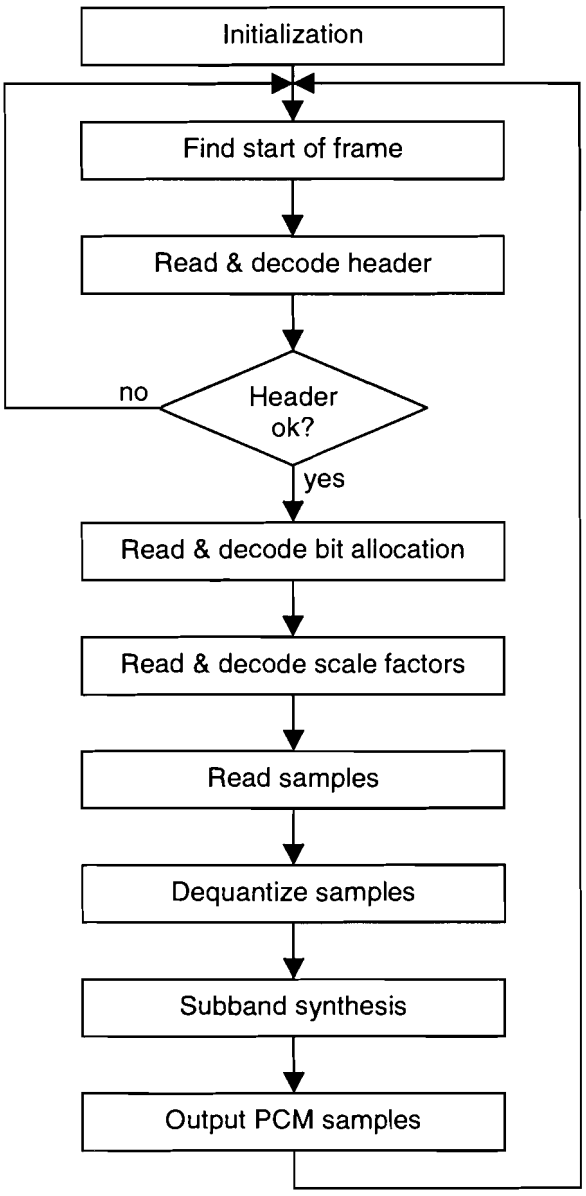

Figure 11 Flowchart of the decoder

9. **Subband synthesis:** The subband samples in the frequency domain are transformed back to the time domain, delivering 1152 PCM samples. This is done by an IDCT (Inverse Discrete Cosine Transform) algorithm.
10. **Output PCM samples:** The PCM samples are sent to an output device. Now the decoding of the frame is finished. The program flow will jump to step 2 in order to start the decoding of the next frame.

These steps are almost the same as the steps taken by the MPEG audio example source, that can be downloaded for free from the internet. The example source is called "dist10". In order to be able to follow the program flow easily in all versions of the source code, the decision has been made to hold on to the flow chart. This is achieved by creating separate functions in the program for the different steps in the

flowchart. Although the performance of the program might be affected a little, the understandability of the code is greatly improved.

5. The encoder

5.1 Introduction

Figure 12 gives an overview of the MPEG audio encoding scheme. When this figure is compared to figure 10, one can see that the main data path of the encoder is the inverse of the decoder. The input of the encoder are PCM samples. These samples are transformed to subband samples. Then the samples are quantized and coded. In the last step the header, the bit allocation, the scale factor selection information, the scale factors and the coded subband samples are written to the bitstream.

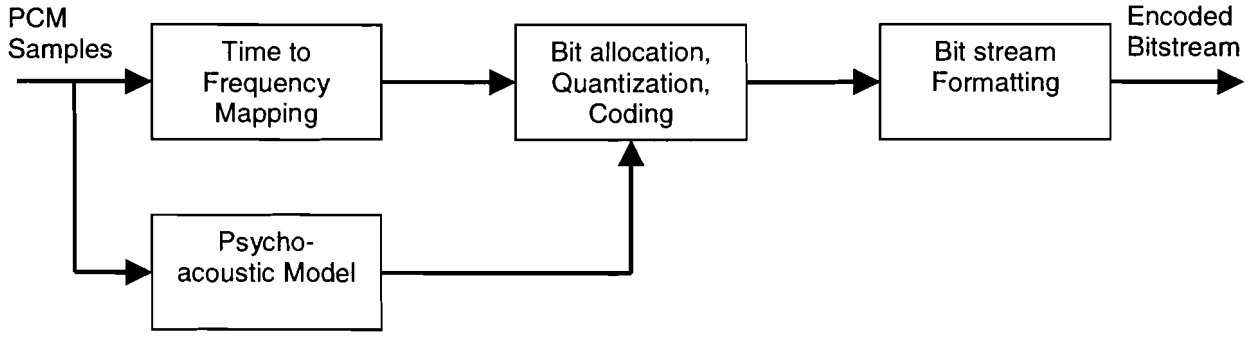

Figure 12 MPEG audio encoding

The encoder has a second path in which the psycho-acoustical analysis of the audio data is performed. Input for the psycho-acoustical model are the PCM samples, output are the masking thresholds for the different subbands. The result of the psycho-acoustical analysis is fed in the second step of the main stream and affects the number of bits that will be allocated for the different subbands.

5.2 Software implementation of the encoder

For the encoder a software flowchart is created just like for the decoder. It is based on the encoder flowchart in [1] and is shown in figure 13. Here will follow a short explanation of the different steps.

1. **Initialization:** The initialization is responsible for the creation and initialization of variables and memory blocks. Also the communication with the outside world, for reading the incoming PCM samples and writing the outgoing bitstream, has to be set up.
2. **Read samples:** From an input device 1152 PCM samples (1 frame) are read and stored in a buffer.
3. **Subband analysis:** The PCM samples in the time domain are transformed to subband samples in the frequency domain. This is done by a DCT (Discrete Cosine Transform) algorithm. The subband samples are stored for later use.
4. **Scale factor calculation:** The scale factors for the subbands are calculated and stored. Each scale factor is valid for 12 subband samples.
5. **FFT analysis:** The power density spectrum of the PCM samples is calculated, using an FFT (Fast Fourier Transform).

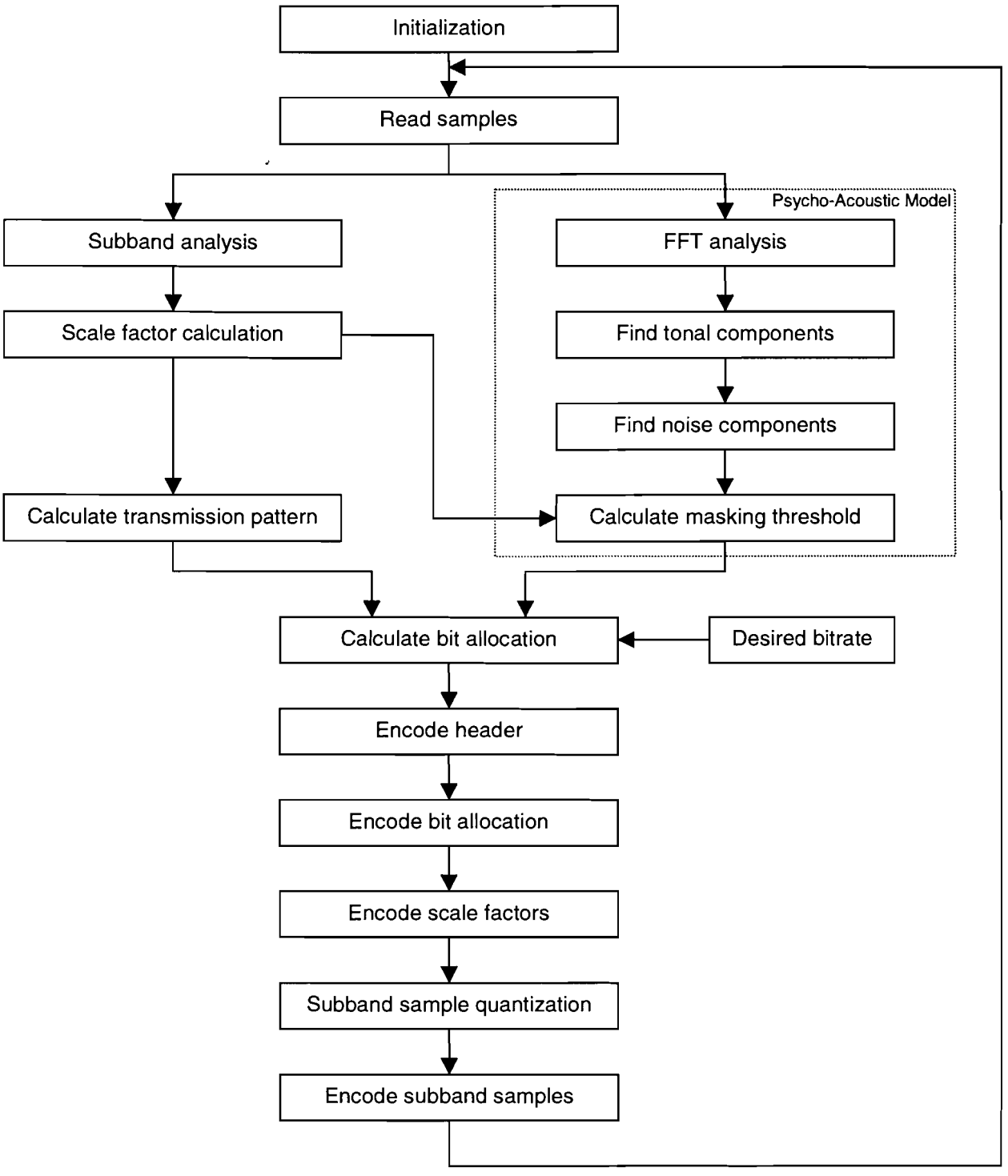

Figure 13 Flowchart of the encoder

6. **Find tonal components:** In the power spectrum the tonal components are selected and marked.
7. **Find noise components:** The noise components in the power spectrum are selected and marked.

8. **Calculate masking threshold:** The masking threshold for the both the tonal and noise components is calculated. From this masking threshold the signal-to-mask-ratio is computed for every subband. The signal-to-mask-ratio is stored and will be used to determine the number of bits assigned to each subband.
9. **Calculate transmission pattern:** The number of scale factors needed for the different subbands is calculated and stored. This is the scale factor selection information.
10. **Calculate bit allocation:** Based on the signal-to-mask-ratio, the transmission pattern and the desired bitrate, the number of bits allocated for each subband is computed. The bit allocation information is stored until it can be encoded.
11. **Encode header:** The header information is coded and written to the output bitstream.
12. **Encode bit allocation:** The bit allocation information is coded and written to the output bitstream.
13. **Encode scale factors:** The scale factor selection information is written to the output bitstream. Based on the scale factor selection info the appropriate scale factors are coded and written to the output bitstream.
14. **Subband sample quantization:** With the scale factor information and the bit allocation information, the subband samples are quantized and coded.
15. **Encode subband samples:** The quantized and coded subband samples are added to the output bitstream. Now the encoding of the frame is completed. The program flow will continue with step 2, where the PCM samples are read for the encoding of the next frame.

Just like the decoder, the encoder implementation will follow the steps in the flowchart and the steps can be recognized as separate procedures in all versions of the source code.

6. Implementation

6.1 Introduction

Because of the complexity of the algorithm and the limited debugging possibilities on the DSP environment, a direct implementation on the DSP is not a realistic option. First the algorithm will be implemented and tested in a PC environment. Because of the extensive debugging possibilities, the visual C++ developing environment is used for the first version. As a starting point the example source (dist10) is used. So the first step is to convert the dist10 source for operation under the visual C++ environment. The dist10 is a floating point implementation. A layer 2 only CODEC should be created, so the next step consists of the removal of all unnecessary code. The dist10 is a non-optimized implementation. Thus now the source has to be optimized to increase the performance. Because the DSP can only perform fixed-point calculations, conversion from floating-point to fixed-point is necessary. The fixed-point version is the last PC implementation and is ready for conversion to DSP. Once it is functional on the DSP, the only thing left to do, is the optimization by replacing C-code by assembly. The result will be a real-time hybrid C and assembly version. The development path is depicted in figure 14. It will have to be followed twice, once for the decoder and once for the encoder. The different steps will now be discussed separately as well as some other implementation issues to create a working MPEG audio CODEC.

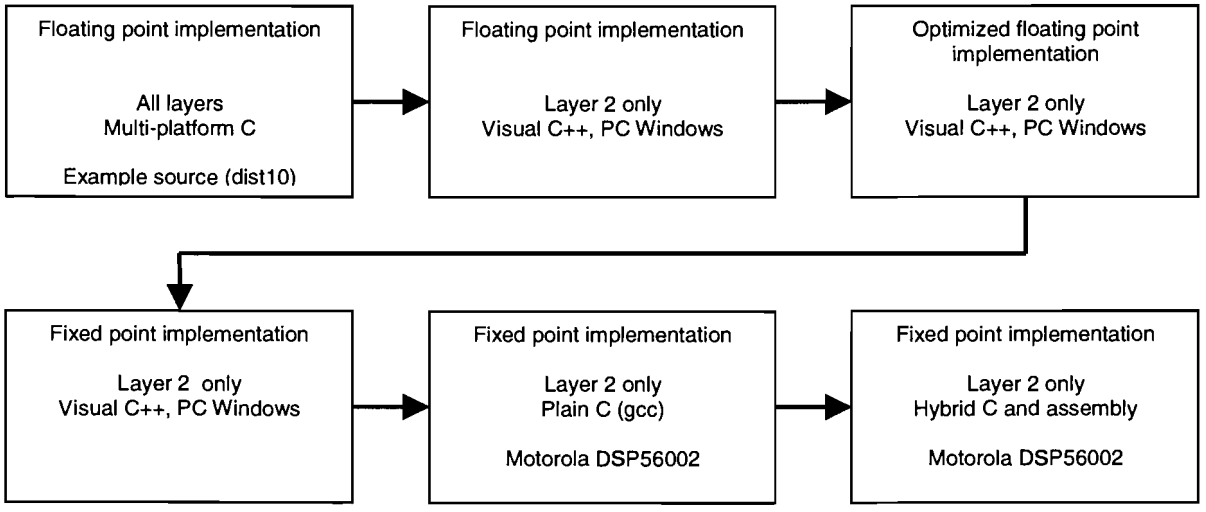

Figure 14 Development path for DSP application

6.2 Compiling the dist10 source

The dist10 source consists of a lot of C-source files and header files. Options can be defined to specify the target platform. However it has not been designed for visual C++, so all platform specific options should be disabled. The easiest way to get the source compiled, is to create an empty project, add all files to the project, set the warning level as low as possible and select the 'build project' option. The compiler will find the dependencies between the different source and header files and build a makefile based upon these dependencies. Now the warnings can be re-enabled and

after the remaining warnings that are generated by the compiler are solved, a version that can be compiled with visual C++ has been created.

6.3 First implementation

Now a first working version is to be created. In order to get this done the following steps were taken.

- Removal of layer 1 and layer 3 code. Because we are dealing with a layer 2 CODEC, the layer 1 and layer 3 code is not needed. This violates the MPEG rule that a layer 2 CODEC should be able to handle a layer 1 stream. However since layer 1 won't be used, it would be a waste of resources.
- Removal of psycho-acoustical model 2 code. The CODEC uses psycho-acoustical model 1, because model 2 is too complex for real-time implementation on the chosen hardware.
- Replacement of platform independent floating-point code by floating-point operations built in the compiler, effectively removing the whole floating-point library that is included in dist10.
- Removal of unused code from the included bitstream package. The bitstream package is used for writing and reading an MPEG stream on bit level. Also multi-platform support has been removed from the bitstream package.
- All platform specific pieces of code have been removed. So the source can only be compiled with a 32-bit compiler for the x86 series of processors.
- Replace the routines for reading and writing PCM audio files by routines supporting the ".wav"-file format.
- All remaining code is grouped into a single C-source file and all data is grouped into a single header-file.

When compiled with visual C++, the result is a 32-bit console application running under windows. The input and output is handled by means of files. So the encoder expects a "wav"-file as input and will produce a "mp2"-file as output. The decoder works the other way round, reading a "mp2"-file and producing a "wav"-file. Because no effort has been done with regard to the performance, real-time operation of the CODEC isn't possible on the test PC, a Pentium 250MMX.

6.4 Optimized implementation

The working implementation should now be optimized. The optimization is done on algorithmic level. It is only useful for the most time-consuming parts of the algorithm. In the case of the decoder most processing power is needed to perform the Discrete Cosine Transform. In the case of encoding, besides the processor power needed by the DCT, the psycho-acoustical model also consumes a lot of processing power. The majority of the processing power necessary for the psycho-acoustical is needed for the Fast Fourier Transform in this model. However the FFT is already optimized and there isn't much left to be gained.

The optimization of the DCT and IDCT is done with Lee Decomposition [3]. The following equation represents the DCT used in the MPEG audio standard.

$$x[i] = \sum_{k=0}^{31} X[k] \cdot \cos\left(\frac{i+16}{64}\pi(2k+1)\right) \qquad 0 \le i \le 63$$

It is a DCT_{32x64} . The number of calculations needed can be halved by making use of the periodicity of the DCT. It is sufficient to perform a DCT_{32x32} to get all values. The next equation gives the DCT_{32x32}.

$$x'[i] = \sum_{k=0}^{31} X[k] \cdot \cos\left(\frac{i \cdot \pi}{64}(2k+1)\right) \qquad 0 \le i \le 31$$

With the following algorithm these values can be mapped upon the original DCT_{32x64}.

$$\begin{aligned}
&for(i = 0; i \le 15; i++)\\
&\{\\
&\quad x[i] = x'[i+16]\\
&\quad x[i+17] = -x'[31-i]\\
&\quad x[i+32] = -x'[16-i]\\
&\quad x[i+48] = -x'[i]\\
&\quad x[16] = 0\\
&\}
\end{aligned}$$

The number of multiplications necessary for the DCT_{32x32} can be reduced further. In this case that is done by the Lee algorithm. The Lee decomposition provides a way to express a DCT_{nxn} in terms of two $DCT_{n/2xn/2}$ matrices. Figure 15 gives an overview of the Lee decomposition. First the input is split into an even part and an odd part. An even and an odd butterfly matrix are created, called EBF_n and OBF_n. For the odd part the calculations are somewhat more complicated than for the even part. For the odd part an extra scaling matrix $H_{n/2}$ and an output butterfly matrix $G_{n/2}$ are needed. Finally two matrices E_n and O_n are defined, that take care of the sample reordering. In formula form:

$$DCT_n = E_n \cdot DCT_{n/2} \cdot EBF_n + O_n \cdot G_{n/2} \cdot DCT_{n/2} \cdot H_{n/2} \cdot OBF_n$$

As can be seen in the formula, the Lee decomposition can be applied recursively. However with every recursion the butterfly and reordering will get more complicated. In practice the first decomposition from DCT_{32x32} to DCT_{16x16} seemed very effective, but with further decomposition little could be gained. To be able to compare the difference the decoder uses four DCT_{8x8}'s and the encoder two DCT_{16x16}'s. The difference in speed on a PC and on the DSP is minimal. However for the DSP implementation one should consider the fact that memory wait-states can greatly

influence the speed difference between the different DCT's. When slow memory is used, a DCT with more multiplications and less moves will perform better.

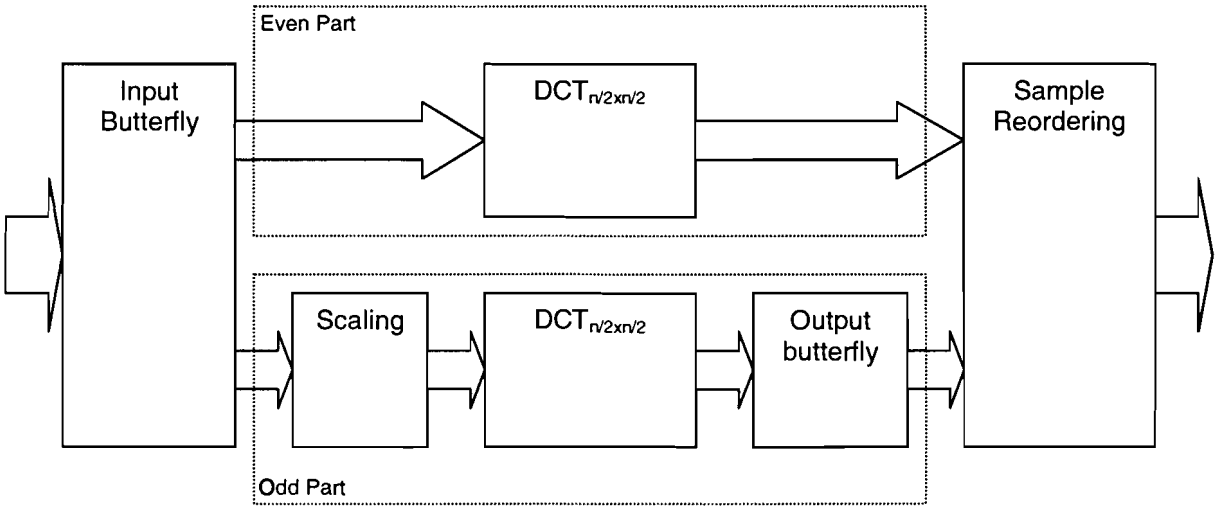

Figure 15 Lee Decomposition

Besides the implementation of a faster DCT algorithm, the following optimizations have been applied:

- Most functions from included header files have been replaced by faster ones, allowing the removal of most header files.
- The remaining functions from the bitstream package have been replaced by new and faster implementations.
- Replacement of arrays that are calculated at the initialization by pre-calculated global arrays defined in the header file. This makes the memory usage of the program more predictable.
- Replacement of arrays that are read from file by array definitions in the header file. In this way the MPEG CODEC can be provided as one executable without extra data files.
- Removal of arrays and other data structures that aren't needed for a layer 2 CODEC.

The functionality and interface of the implementation are still the same. However the performance has increased dramatically resulting in the CODEC running well within real-time on a P250MMX PC.

6.5 Fixed-point implementation

The DSP56002 only supports fixed-point calculations, therefore the program must be converted to fixed-point before it can run on the DSP. Because of the better developing environment on the PC, the conversion is done on the PC. Some care has to be taken with regard to variable conversion. On the DSP an 'int' is 24-bit wide and a 'long' is 48-bit wide, while on the PC side an 'int' and a 'long' represent 32-bit integers. As a larger integer, visual C++ supports the 64-bit wide "__int64" format. So

a 24-bit integer on the DSP is mapped on a 32-bit integer on the PC and a 48-bit integer on the DSP is mapped on a 64-bit integer on the PC. For the PC implementation in the case of the 32-bit integer use of the upper 8 bits is forbidden and in the case of a 64-bit integer use of the upper 16 bits is forbidden.

Floating-point numbers consist of a mantisse (m), an exponent (e) and a base (b). For the mantisse and exponent a fixed number of bits is reserved. The base is a fixed value indicating the base of the numbers. The original number can be found with the following formula:

$$number = m \cdot b^{e}$$

A fixed-point number is represented by only one integer (x) and a fixed distance (d) between two consecutive numbers. Now a number is given by the following formula:

$$number = x \cdot d$$

As can be seen the distance between two consecutive floating-point numbers is variable while the distance between two consecutive fixed-point numbers is fixed. The result is that the floating-point numbers are able to cover a large range of numbers with an almost constant relative error, while in a fixed-point representation the precision depends on the range and the relative error certainly is not constant.

So for accurate representation of floating-point number as fixed-point numbers, the range of the floating-point representation should be limited and an appropriate distance has to be chosen. The numbers in the main part of the MPEG layer 2 CODEC are all between –4.0 and 4.0. The distance between the numbers for maximum precision can now be calculated by dividing the range of numbers by the amount of available integers (2^{24}):

$$d = \frac{8.0}{2^{24}} = \frac{8.0}{16777200} = 5 \cdot 10^{-7}$$

Because of the fixed distance, this is also the smallest number that can be represented. The DSP56002 only supports shift-operations performing one shift at a time. So conversion to another fixed-point format is a rather expensive operation and should be avoided. The easiest way to achieve this, is by only using one fixed-point format. Most of the CODEC could be implemented without changing the format, however some parts of the psycho-acoustical model require more accuracy and some parts require a larger range. So in the psycho-acoustical model other formats have been applied, amongst them 48-bit precision ones.

In the psycho-acoustical model, some very expensive operations, like power and logarithm, are present. These operations must be eliminated. To do so, there are several ways. First the applied algorithm is studied carefully, to verify whether the operations are really necessary. By altering the algorithm some expensive operations

could be avoided. For the operations that couldn't be avoided, two solutions are possible. The first one is replacing the operations by a look-up table. Unfortunately look-up tables require a lot of memory. The second possibility is to use an approximation for the operation. Generally an approximation is only accurate for a small range, but can be a very effective solution. Look-up tables and approximations have both been applied to optimize the psycho-acoustical model.

In the psycho-acoustical model concessions have been made with regard to the accuracy. Sometimes a fast but in-accurate implementation has been chosen in favor of a slower but accurate implementation. The small errors created are only present in the psycho-acoustical model. The only influence of the errors in psycho-acoustical model on the data stream is a possible change in the number of bits assigned to the different subbands.

The fixed-point version of the CODEC is running slower than the floating-point version, but it can still perform real-time on a Pentium 250MMX PC. Because it is the final PC version, the possibility to play the audio using a soundcard has been added to the decoder, effectively creating a simple "mp2-player" for the PC. The "mp2-player" can also be used for demonstration purposes or to judge the audio quality without the extra filtering that some other players apply.

6.6 DSP implementation

Now that the C-source is prepared, porting the PC implementation to the DSP platform is the next step. The C-compiler for the DSP is derived from gcc. It only accepts plain C source and it will use the p-memory region for the program code and the y-memory region for the program data.

First the source had to be compiled. Issuing the compile commando with the source file as used for visual C++ returned hundreds of errors. They were all caused by C++ conventions the gcc compiler doesn't support. Once these errors were removed the source compiled without problems.

To test the program, frame data for one frame was added to it in an array, the program was started and the output was sent to a file on the PC via the debugger. Also a PC version, that writes its output to a file, was fed with the same input frame. By comparing the two output files, differences between the CODECs can be detected. The goal is to get identical output files produced by the DSP and the PC version.

Unfortunately, the first run, both files were completely different and it was obvious that the DSP version wasn't working correctly. The problem was caused by the DSP compiler. The DSP compiler isn't treating 8-bit variables correctly. Because the DSP only supports 24-bit or 48-bit variables, 8-bit variables are stored in the lower eight bits of a 24-bit variable. Operations that cause an overflow for the 8-bit numbers, affect the upper unused bits. Most operations still work perfectly, because the upper bits will be zeroed before execution of the operations, however some operations, like shifts and typecasts, can let the upper bits appear and cause unpredictable and erroneous results. The solutions to this problem are avoiding the use of 8-bit

variables and zeroing the upper bits by an 'and'-operation before the critical operation takes place.

A second problem of the DSP compiler is its inability to index local multi-dimensional arrays correctly. Obviously the 'typedef' command only works for global variables. So the only solution is to define the arrays that malfunction globally. A third problem of the DSP compiler is, that it doesn't know the amount of memory installed on the board. This implies, that the amount of memory installed on the board has to be told to the linker explicitly.

After having dealt with the compiler problems, the DSP version produces the same output as the PC version. The version created so far can encode and decode MPEG frames on the DSP. However the I/O functions for reading and writing the data streams aren't present yet and the performance is very poor. The program clearly demonstrates how bad the code, produced by the C-compiler, performs. In order to achieve real-time operation, the speed has to be increased by a factor somewhere between 100 and 1000. Very expensive and dedicated compilers promise an increase of the performance by about 50%. Considering the fact that at least a factor 100 is needed one can conclude that such a compiler won't help much.

6.7 Hybrid implementation

Next the working DSP version has to be converted to a real-time version. The only way to do this, is by replacing time-critical parts of the C-code by assembly code. The most critical parts are the DCT for the decoder and the DCT and the psycho-acoustical model for the encoder.

After the DCT routine of the decoder had been converted to assembly, it was capable of decoding the 22kHz mono MPEG audio layer 2 stream in real-time. Because of the relative small code size of the decoder, the rest of the main-loop had also been replaced by assembly code. Now the decoder could decode a 48kHz stereo MPEG audio layer 2 stream in real-time. This is the maximum quality supported by the standard, so further optimizations are quite useless.

The situation for the encoder is much more complicated. Besides the DCT, the psycho-acoustical model consumes a considerable amount of processing power. The complete psycho-acoustical had to be converted to assembly. After the psycho-acoustical model and the DCT had been coded in assembly, 22kHz mono operation still wasn't running real-time. To achieve real-time operation in 22kHz mono mode, the functions 'subband sample quantization' and 'encode subband samples' had to be converted to assembly too.

To get an idea about how much time is needed for the different parts of the encoding process, the procedures of figure 13 are summarized in table 1 with the time needed on a DSP56002 running at 66 MHz for one frame in C-code and in assembly code (if available). As can be seen in the table, the encoding of one mono frame in C-code requires more than a second. With the most time-consuming parts replaced by assembly code, the encoding of a frame takes 37 ms. In 22 kHz mono mode, the duration of a frame is approximately 52 ms. So the encoding process is working well within real-time.

Table 1: Differences in performance between C and assembly

Function:	Time (C-code)	Time (assembly)
Read samples	1 ms	
Subband analysis	> 1000 ms	4 ms
Scale factor calculation	1 ms	
Psycho-acoustical model	> 1000 ms	13 ms
Calculate transmission pattern	1 ms	
Calculate bit allocation	2 ms	
Encode header	0 ms	
Encode bit allocation	1 ms	
Encode scale factors	2 ms	
Subband sample quantization	55 ms	1 ms
Encode subband samples	11 ms	1 ms
Rest of code	10 ms	
Total	**> 1000 ms**	**37 ms**

In order to create a functional product, some other things had to be done:

- Coding of an initialization routine. The initialization routine configures the PLL for a clock frequency of 66 MHz, sets the number of wait-states for accesses to external RAM to zero, configures port B for general purpose I/O and initializes the CS4215.
- Adding code for communication with a PC. The communication code consists of a simple handshaking protocol.
- Coding of the interrupt routines for handling data transport between the DSP and the CS4215.
- Removal of functions and operations for which the compiler includes libraries, like 48-bit multiply, divide, floating-point operations, printf-function, etc.
- Removal of the "stdio" header file. It won't be used anymore and costs a lot of memory.

6.8 Programming the CS4215

Before the CS4215 is operational, an initialization procedure is required. The initialization procedure, that can be downloaded from the Motorola website, is used. By changing some parameters it has been adjusted for the correct hardware configuration. Data communication with the DSP is done on an interrupt basis. When data transfer is completed in either direction an interrupt is generated by the CS4215. The CS4215 has two operational modes, control mode and data mode. During initialization the CS4215 is in control mode. Some configuration words are sent to the CS4215 and some control words are received from it. For the initialization two simple interrupt routines are needed. For data communication more complicated interrupt routines are required, because some buffer management of the sample buffers is necessary. Also control words and data words have to be recognized and separated.

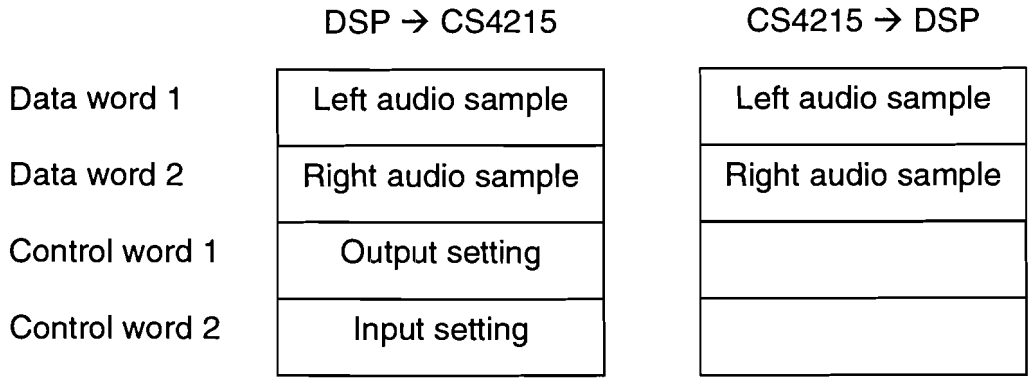

Figure 16 The input and output frames of the CS4215

Figure 16 shows the frames as they are used by the CS4215 in data mode. As can be seen, only one stereo sample can be transmitted each frame. For every word in a frame an interrupt request is generated. So for a decoder working with a sampling frequency of 48kHz stereo, 192000 interrupt requests need to be handled every second. For the encoder the scenario is even worse, fore the CS4215 requires an "output setting" and an "input setting" every frame. Thus when the CS4215 is sampling, data has to be sent at the same time in order to be able to send the control words. When encoding at 48 kHz stereo, the CS4215 will generate 384000 interrupt requests each second. This enormous amount of interrupt requests, consumes at least half of the available processing power. Because the CS4215 only supports these data frames, mono operation doesn't affect the amount of interrupt requests.

During operation another problem with the CS4215 arose, namely the fact that it requires a very prompt service of its interrupt requests. If a pending interrupt isn't serviced in time, the CS4215 will shutdown itself, generating a terrible 'plop'-sound on its output. The 'rep'-instruction, being one of the DSPs most powerful loop control instructions, because it only performs one instruction fetch for the whole iteration, isn't interruptible. So the 'rep'-instruction can only be used for small amounts of data and had to be replaced by a 'do'-loop in most cases. Of course the performance of the CODEC is affected.

In the IOP another AD/DA-converter will be applied, which generates less interrupts by removing the control words from the data frames.

6.9 Software for communication with the PC

The communication over the 10-bit cable to the PC is done with a simple handshake protocol. There are 8-bits for data transfer and two bits for hand-shaking, one is used for the request signal and the other for the acknowledge signal. The timing diagram for the handshake protocol is shown in figure 17. For the explanation of the working, assume that the DSP is decoding an MPEG stream and thus wants MPEG data from the PC. The PC is programmed for sending data to the DSP and to allow faster communication it already places the first data byte on the data-bus. Now the PC has to wait for a data-request from the DSP. When the DSP needs data from the PC, it

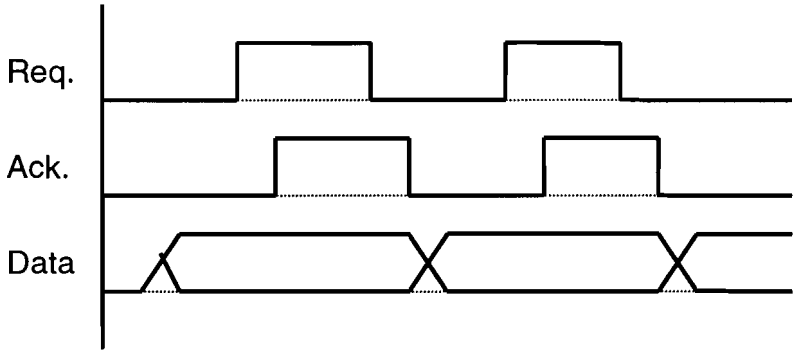

Figure 17 Timing diagram of the handshake protocol

will raise the request-line and wait for an acknowledge from the PC. The PC, continuously scanning the request-line, will notice the request and answer by raising the acknowledge-line. Once the DSP has received the acknowledge signal, it will copy the data from the data-lines and release the request-line. The PC will notice the end of the request and will respond by releasing the acknowledge-line. At the same time the PC will place the next data byte on the data-lines. Now the PC is waiting for a request from the DSP for the next transfer.

A disadvantages of handshaking in software, is the fact that both sides are continually polling and can't do anything else. Because the data requests are coming in bursts, this problem is intensified. Another problem is, that if a bit-error occurs on one of the two hand-shaking lines, the system will end in a dead-lock situation. In practice the handshaking was good enough for testing the system and in the IOP, it will be replaced by communication via the host interface.

7. Experiments

7.1 Introduction

To verify the implementation the following experiments have been performed.

- Testing the PC decoder.
- Testing the DSP decoder.
- Testing the PC encoder.
- Testing the DSP encoder.
- Duration test for the CODEC.
- Error sensitivity.
- MPEG compliance.
- Considerations about an optimal PC version.

They will be discussed in the following paragraphs.

7.2 Testing the PC decoder

The sound quality of the decoder is judged by listening to it. Some MPEG layer 2 test files are created with the floating-point encoder, of which the functionality is equal to the dist10 source. So the quality of the test files is the maximum that the CODEC can reach. The files use a bitrate of 64 kbit/s per channel or 128 kbit/s per channel. These test files are played back with the floating-point version of the decoder and with the fixed-point version of the decoder. No difference between the sound produced by the floating-point version and the fixed-point version is heard.

To be able to verify the sound quality, the windows media-player is used as a reference decoder. A difference in sound quality between the windows media player and the PC decoder couldn't be heard. As expected a difference between 64 kbit/s per channel and 128 kbit/s per channel MPEG streams can be heard.

7.3 Testing the DSP decoder

For testing the DSP decoder the test configuration as shown in figure 18 is used. The

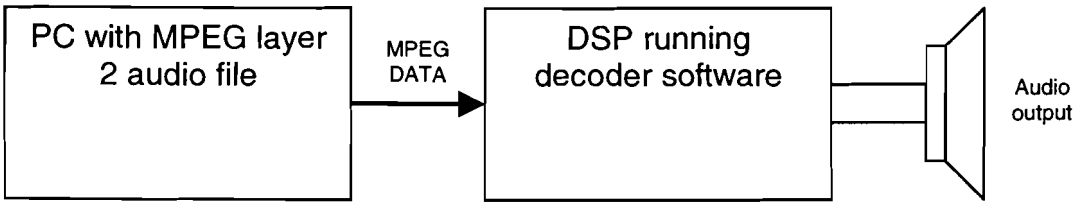

Figure 18 Test configuration for the decoder

PC transfers an MPEG layer 2 stream to the DSP and the DSP receives the MPEG data, decodes it to raw audio samples and plays these samples via the CS4215.

Two tests are performed:
- A real-time operation test to verify whether the decoder is capable of real-time decoding.
- A listening test to verify the sound quality of the decoder.

For both tests the files created for testing the PC version are used. The real-time operation test is done by playing the test files and measuring the time it takes to play them. Then the number of frames of the test files is counted and the duration of the test file is calculated. Now the calculated duration and the measured duration are compared and should be equal. With a bitrate of 64 kbit/s per channel the decoder passed the test at 48 kHz stereo, so all other sampling frequencies won't be a problem. However if a 48 kHz stereo stream with 128 kbit/s per channel (256 kbit/s total bitrate) was fed to the decoder, a slight difference between measured time and calculated time occurred. The difference can be explained be the fact that the handshaking protocol used for the connection between the PC and the DSP, is consuming a lot of processing power at a bitrate of 256 kbit/s. In the IOP the problem won't occur and the decoder will be able to handle all sampling frequencies and bitrates supported by the MPEG1 standard.

The listening test is performed by comparing the sound of the DSP version with the PC version. The DSP version sounds better than the PC version, although they produce the same output. This can be explained by the poor quality of the PC sound card, when compared to the CS4215, used on the DSP board to play the audio.

7.4 Testing the PC encoder

Besides the verification of the sound quality, the bitrate has to be determined by listening tests. The sound quality can be improved by increasing the bitrate. There is a preference for a bitrate of 64 kbit/s per channel or 128 kbit/s per channel, so a choice between these two has to be made. For comparisation with CD audio a sampling frequency of 44.1 kHz was used. For the other tests the sampling frequency of 22 kHz that will be used in the final design was chosen.

The audio quality of a 64 kbit/s per channel stream sounds acceptable, however the sound quality of the 128 kbit/s per channel stream certainly is better. So a bitrate of 128 kbit/s is preferred. The difference between an 128 kbit/s per channel MPEG stream and the original CD audio is difficult to hear.

7.5 Testing the DSP encoder

For testing the encoder a more complicated test setup had to be built. The problem is that the PC running the handshaking protocol should devote as much processing power to it as possible, otherwise the performance of the encoder will be affected. Another PC is required to provide the MPEG encoder with an audio stream. This PC can also be used to decode the MPEG stream and thus simulating real-time operation. See figure 19. Here PC2 collects the MPEG data. PC2 is connected to PC1 by means of an ethernet network. PC1 provides the MPEG encoder with audio data and can play the MPEG data that PC2 collects via the ethernet network. Now it is also possible to switch the output of PC1 between the original CD audio and the audio that has passed the CODEC. For DSP encoder testing a sampling frequency of

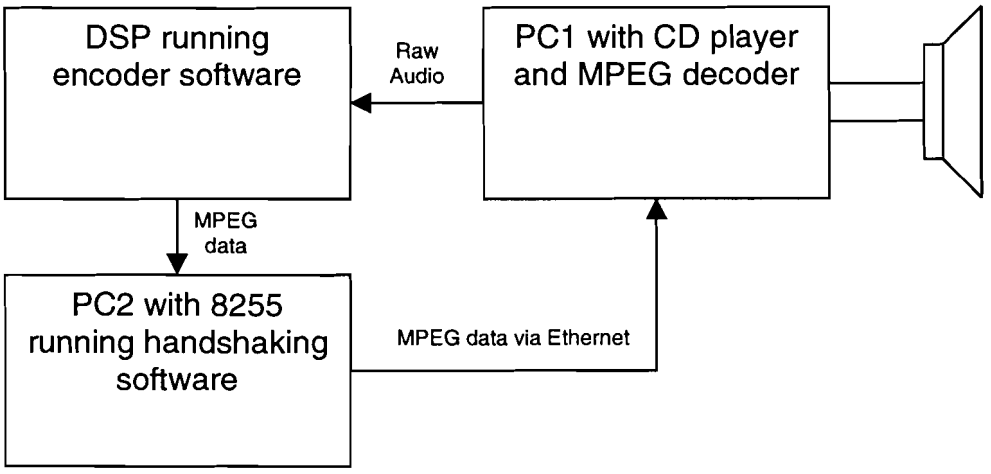

Figure 19 Test configuration for the encoder

22 kHz and only one channel was used, because the DSP encoder can't handle a higher frequency or more channels in real-time.

Two tests are now performed:

- A real-time operation test to verify whether the encoder is capable of real-time encoding.
- A listening test to judge the sound quality of the encoder.

For the real-time operation test, the encoder is started to encode with a certain bitrate and a few seconds later, the decoder on PC1 is started. The decoder will stop as soon as it can't read any data anymore (it reaches the end of the file on PC2). So as long as the encoder is running the decoder shouldn't stop. As an audio source a normal audio CD is used. At 64 kbit/s operation the test was passed without a problem, however when the bitrate was increased to 128 kbit/s the test failed. The failure is caused by the doubled bitrate doubling the data-rate over the connection between PC2 and the DSP. By this double data-rate, the processor load at the DSP side is increased by such an amount, that real-time encoding isn't possible anymore. This problem won't occur in the IOP, because then the communication is done by means of the host interface, which is a lot faster.

The listening test is performed by encoding an audio track and then playing it back and compare it with the original. Unfortunately the test can only be performed at a bitrate of 64 kbit/s, because at a bitrate of 128 kbit/s real-time encoding is not possible. A difference between the 64 kbit/s and the original audio track could easily be heard, however the quality of the 64 kbit/s stream is certainly acceptable and in accordance with the quality of the PC encoder. In order to get high quality audio in the final version a bitrate of 128 kbit/s is most likely to be chosen. Based on the results of the 64 kbit/s stream, the expected audio quality of the 128 kbit/s stream on the DSP will be comparable to the quality of the 128 kbit/s stream on the PC.

7.6 Duration test for the CODEC

The stability of the CODEC is now subject of a test. For the decoder the test is performed by first starting the decoder on the DSP. Then a few MPEG files are placed on the PC, that communicates with the DSP. Finally the PC2 is programmed to repeatedly send the MPEG streams to the DSP. Extensive tests showed that the decoder isn't totally stable. The average time before a crash for the setup of figure 18 is about a week. Inspection after a crash showed the program code to be intact and no data corruption to have occurred either, so there is a deadlock situation. Two causes of the deadlocks were found.

The first one is a bit error on the request or acknowledge line of the cable between the PC and DSP. Such an error can get the DSP and PC in a state where they are both waiting for an answer from the other. To confirm the situation a bit can be flipped manually and in some cases the program continued normal operation, proving the bit error to be the problem. A possible solution is to program a timer to check for this situation once in a while and to restart the protocol if necessary.

The second cause is a problem with the communication with the CS4215. For an unknown reason the CS4215 can stop operating so the decoder can no longer get rid of the decoded audio data. Once the buffers are full the decoder will stop operation, until a buffer is free again. By manually freeing a buffer the situation was verified. The only way to get the CS4215 operational again, is by a new initialization. However the interrupt routines needed for initializing the CS4215 are replaced after the initialization has completed successfully. So even a reset of the software won't help, but the original software will have to be downloaded.

For the encoder the testing is a little bit more complicated. The problem is that the encoded MPEG data has to be written to a file before it can be played, making tests of very long duration impossible, because the space on the harddisk of the PC is not infinite. However there is no reason why the same problems that caused the decoder to malfunction won't occur with the encoder. During the tests of a few hours that have been done with the encoder, it performed well.

7.7 Error sensitivity

The sensitivity to bit errors of the CODEC is tested by delivering a defective stream, some randomly chosen bits are flipped, to the decoder. It doesn't matter whether the error is generated by the encoder, or if it is a network error or if it happened at the decoder, the impact of the error will be the same. However the effect of the error does depend on the part of the stream where the error is situated. See figure 8. If the header is affected the result is most likely disastrous. The frame will be decoded with the wrong options, causing terrible sound and possibly making the decoding of the next frame impossible too. Decoders supporting all possible formats have even more trouble with errors in the header. Such decoders might stop operation or perform bad decoding for multiple frames.

If the bit allocation information or scalefactor selection information is affected, the result will also be disastrous and the next frame might also be invalidated. However the problems will be restricted to a few frames. When the error is situated in the

scalefactors or the subband samples, annoying 'clicks' are likely to occur. The next frame is not affected by the error.

In the test situation for encoding, the receiving side (in this case PC2), doesn't know what is received. So if a whole byte is missed or in the IOP if a whole network packet is missed, the result will also be disastrous when decoding the defective frame.

To avoid these problems error detection should be applied and defective frames should be discarded. The error detection built in the MPEG standard isn't good enough, because a bit in the header will indicate whether the error detection is active. The error can also affect that particular bit and thus disable the error protection and defect the frame at the same time, because the error detection bits will be interpreted as bit allocation bits. So there is a need for separate error detection.

7.8 MPEG compliance

In order to be able to test the MPEG compliance of the CODEC, reference files and test programs can be downloaded from the internet. The decoder does decode the MPEG layer 2 streams and the test programs do accept the MPEG layer 2 streams generated by the encoder. Because the dist10 source is used as a starting point, and no concessions have been made with regard to the code of the main data stream for the encoder and decoder, it is to be expected that the CODEC will correctly decode any MPEG audio layer 2 compliant stream and that the stream generated by the encoder will be accepted by any MPEG audio layer 2 compliant decoder.

The only concessions that have been made with regard to the standard are in the implementation of the psycho-acoustical model. To increase the performance of the model a limited error in the calculations has been introduced. This could result in a different bit allocation for the subbands.

Although it wasn't the goal of the project to create a fully MPEG compliant CODEC, the psycho-acoustical model of the CODEC can be modified to an MPEG compliant one with very little effort. Another problem is that the layer 2 decoder can't handle a layer 1 stream, so it still violates one of the rules and isn't MPEG compliant.

7.9 Consideration about an optimal PC version

The monitoring stations of the surveillance system also need to be able to decode and encode MPEG audio in real-time. These monitoring stations are PCs running a Graphical User Interface (GUI). For a good performance of the GUI, these PCs should not devote too much time to the MPEG algorithms.

To get an idea about the suitability of a PC processor for an MPEG CODEC the optimization of the algorithm for the PC has also been studied. Most of the processing power is needed for the DCT, so only the implementation of the DCT is considered. Of the PC versions, the floating-point version is the fastest one. When this version is optimized and converted to assembly, one multiply-accumulate will take approximately 4 cycles. The DSP will do the same operation in 2 cycles, showing a DSP to be the right type of processor for these operations.

However the in C-code slower fixed-point PC version is, when converted to assembly, suitable for implementation with the MMX instruction set, allowing two multiply-accumulate operations to be executed in one cycle. So a PC processor with MMX instructions is also well suited for creating a fast implementation of the algorithm. See also [3].

8. Results

8.1 The decoder

The decoder is capable of decoding a 48 kHz stereo MPEG audio layer 2 stream in real-time. For the 48 kHz stream a 66 MHz DSP56002 is required in the test setup, so the decoding only can also be done with a 40 MHz DSP56002. The 40 MHz version is the slowest and cheapest of the DSP56002 series. When decoding a 22 kHz mono stream, like the IOP should be able to handle, a DSP56002 running at 66 MHz can decode multiple streams.

Table 2 gives the memory usage of the decoder. As can be seen, the decoder fits in the 32 k-word of memory that was present on the original board. Also the choice of shared p-memory and x-memory seems a good one, because the total amount of needed p-memory and x-memory is still below the total amount of y-memory. For debugging purposes there is still some redundancy present in the decoder, allowing the amount of necessary memory for the decoder to be reduced to about 20 k-words (60 kB).

Table 2 Memory usage of the decoder

Memory region	24-bit words
p-memory	2245
x-memory	8704
y-memory	16384
Total	**27333**

The decoder is almost completely optimized. Some performance can be gained, by using different scaling for different parts of the algorithm. Because of the performance that is already reached, further optimizations aren't necessary. Perhaps it is an idea to convert the last remaining C-code of the decoder to assembly. The advantage is that the C-compiler isn't necessary anymore and it is easier to weave the decoder between the encoder code.

8.2 The encoder

The encoder is capable of encoding a 22 kHz mono audio stream in real-time, if the DSP56002 is at least running at 66 MHz. When a DSP56002 running at 80 MHz is used, 44 kHz mono encoding should be possible in real-time.

The memory constraints for the encoder are given in table 3. The encoder needs at least 64 k-word of memory. The spreading of the memory usage is again as expected, so combining the p-memory and x-memory regions is definitely the right choice.

Table 3 Memory usage of the encoder

Memory region	24-bit words
p-memory	7972
x-memory	16384
y-memory	32768
Total	**57124**

In contrast to the decoder, the encoder has not been fully optimized yet. One of the main reasons for not converting most of the encoder to assembly is the greater complexity and code size. Of the encoder only the critical parts have been converted to assembly. If faster operation is necessary one should think of one or more of the following optimizations:

- Replace more C-code by assembly code. Although all time critical code has been converted, by replacement of the remaining C-code by assembly an estimated gain in performance of about 33% is expected.
- Optimize the DCT. The DCT is not optimal yet. It should be coded like the decoder, but with adjusted scaling.
- Replace the FFT in the psycho-acoustical model with an FFT routine optimized for the DSP56002.
- Optimize the other parts of the psycho-acoustical model. The psycho-acoustical model including the FFT uses the most processing power, so here the most speed can be gained.

If real-time 48 kHz stereo encoding is desired, the previous steps might not be sufficient. In that case the psycho-acoustical model will have to be replaced by custom model.

8.3 Requirements for the CODEC

Considering the fact that a real-time CODEC with a sample frequency of 22 kHz is needed, the hardware requirements can be determined. Because a speaker is used for the audio, full duplex operation is not possible, meaning that encoding and decoding won't take place at the same time. So for the processor, the requirements of the encoder are considered. The encoder needs a 66 MHz DSP56002, so 66 MHz is the minimum clock speed for the processor. Zero wait-state access to the external RAM is also needed for optimal operation. For a clock speed of 66 MHz, 15 ns or faster SRAM will be needed.

An upper bound for the amount of RAM needed for the CODEC is calculated by adding the memory constraints of the decoder and the encoder. So a total amount of 84457 k-words of RAM is needed. The smallest power of two in which this value fits, is 128 k-words. So three RAM chips of 128 kB should be used, providing a total memory size of 384 kB, divided in two regions of 192 kB each. One is reserved for p-memory and x-memory, the other for y-memory.

The AD/DA-converter on the test board (CS4215) should be replaced by another one that produces less interrupts per sample. Replacement of the CS4215 can decrease processor load considerably.

9. Conclusions

An MPEG audio layer 2 CODEC has been designed for the DSP56002. The MPEG streams generated by the encoder have been verified and are valid MPEG audio layer 2 streams. The decoder has been tested with reference streams and it decoded those streams correctly. Listening to the CODEC verified the audio quality to be quite good. To ensure high quality audio a bitrate of 128 kbit/s per channel is preferred, but operation on a lower bitrate of 64 kbit/s and thus a higher compression ratio still produces acceptable audio quality.

For single channel operation with sampling frequencies of 22 kHz and below, the CODEC achieves real-time operation. For higher sampling frequencies and stereo operation only the decoder performs real-time. Because one channel mode and a sampling frequency of 22 kHz are sufficient for the speech that will be transferred over the surveillance system, the goal of the project has been fulfilled.

The required hardware for the CODEC is a DSP56002 at least running at 66 MHz and 128 k-words (24-bit) of 15 ns or faster SRAM.

10. Recommendations

Although the designed MPEG audio layer 2 CODEC is well suited for operation in the surveillance system, one could consider further development of the CODEC. Development in a few directions is possible.

- Further optimization of the current CODEC and thus being able to provide real-time 48 kHz stereo audio in the system. This might be used to improve the audio quality or perhaps new features can be added to the system.
- Adjust the CODEC to allow full duplex operation. At the moment an IOP can only transmit or receive audio, but not both at the same time. The fact that the IOP uses a speaker for the MPEG audio, prohibits full duplex operation, but perhaps in future applications, full duplex support might be useful.
- Further development of the CODEC to support MPEG layer 3 audio. MPEG layer 3 audio is very popular nowadays and a layer 3 CODEC can be used for many applications.

11. Literature

[1] ISO/IEC 11172-3 standard
Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s, Part 3: Audio
1993

[2] ISO/IEC 13818-3 standard
Information Technology – Generic coding of moving pictures and associated audio information, Part 3: Audio
1998

[3] Intel Application Note AP-533
Using MMX [tm] Instructions to Implement a Synthesis Sub-Band Filter for MPEG Audio Decoding
1999

[4] Hoekstra, E. and M. Shaaban, R. Czernikowski, S. Dianat
Design and Implementation of DSP based MPEG-1 Audio Encoder
IEEE Transactions on Consumer Electronics, Vol. 45 (1999), no. 1, pp. 31-35

[5] Murphy, C.D. and K. Anandakumar
Real-Time MPEG-1 Audio Coding and Decoding on a DSP Chip
IEEE Transactions on Consumer Electronics, Vol. 43 (1997), no. 1, pp. 40-47

[6] Chen, J. and HM Tai
Real-Time Implementation of the MPEG-2 Audio CODEC on a DSP
IEEE Transactions on Consumer Electronics, Vol. 44 (1998), no. 3, pp. 866-871

[7] Pan, D.
A Tutorial on MPEG/Audio Compression
IEEE Multimedia, Vol. 2 (1995), no. 2, pp. 60-74

[8] Pan, D.
Digital Audio Compression
Digital Technical Journal, Vol. 5 (1993), no. 2, pp. 28-42

[9] Shlien, S.
Guide to MPEG-1 Audio Standard
IEEE Transactions on Broadcasting, Vol. 40 (1994), no. 4, pp. 206-218

[10] Okabe, T. and T. Sakamoto, T. Hase
Full Audio Software Solution for a 16-Bit DSP Core for Digital Audio Decoder LSI
IEEE Transactions on Consumer Electronics, Vol. 44 (1998), no. 1, pp. 117-124

[11] Motorola
DSP56000/DSP56001 Digital Signal Processor User's Manual
1990

[12] Motorola
DSP56002 Digital Signal Processor User's Manual
1993