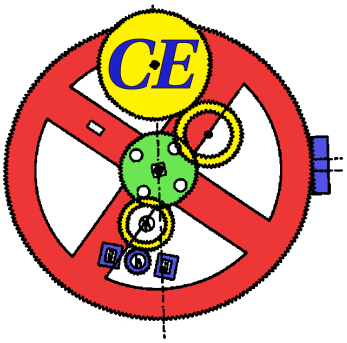


Computer Engineering
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ce.et.tudelft.nl/

2004

MSc THESIS

Accelerating the XviD IDCT on DAMP

Guido de Goede

CE-MS-2004-11

Abstract

Future increases in processor clock speed only will not contribute to performance gains as in the past. Reconfigurable hardware is believed to offer alternative ways of speeding up applications. This thesis project is part of the SMOKE project. SMOKE focusses on the acceleration of MPEG-4 operational kernels using reconfigurable hardware and is part of the MOLEN research theme. The major goal is to prove the usability of reconfigurable hardware to speed up multimedia applications. An MPEG-4 codec (XviD) is selected among four alternatives after a careful selection process. The codec is implemented on the DAMP platform. In order to create the software environment for this implementation, the Linux operating system is ported to the DAMP platform. The XviD codec is analysed and the computationally intensive parts are determined. The codec acceleration is achieved by migrating these kernels to reconfigurable hardware. Experimental results show that the IDCT kernel of XviD is accelerated by a factor of *1.86*. When the colourspace conversion is moved to reconfigurable hardware, a speedup of *3.59* is achieved. By migrating both, the IDCT and the colourspace conversion to hardware a speedup of *1.40* is accomplished for XviD. The experimental results among with the DAMP platform analysis indicate that larger performance improvements for future DAMP versions can be expected.

TU Delft

Faculty of Electrical Engineering, Mathematics and Computer Science

Accelerating the XviD IDCT on DAMP

Accelerating the IDCT in XviD by migrating computation intensive parts to hardware in the Altera Excalibur EPXA1 device on the DAMP platform.

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Guido de Goede
born in Rotterdam, The Netherlands

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Accelerating the XviD IDCT on DAMP

by Guido de Goede

Abstract

Future increases in processor clock speed only will not contribute to performance gains as in the past. Reconfigurable hardware is believed to offer alternative ways of speeding up applications. This thesis project is part of the SMOKE project. SMOKE focusses on the acceleration of MPEG-4 operational kernels using reconfigurable hardware and is part of the MOLEN research theme. The major goal is to prove the usability of reconfigurable hardware to speed up multimedia applications. An MPEG-4 codec (XviD) is selected among four alternatives after a careful selection process. The codec is implemented on the DAMP platform. In order to create the software environment for this implementation, the Linux operating system is ported to the DAMP platform. The XviD codec is analysed and the computationally intensive parts are determined. The codec acceleration is achieved by migrating these kernels to reconfigurable hardware. Experimental results show that the IDCT kernel of XviD is accelerated by a factor of *1.86*. When the colourspace conversion is moved to reconfigurable hardware, a speedup of *3.59* is achieved. By migrating both, the IDCT and the colourspace conversion to hardware a speedup of *1.40* is accomplished for XviD. The experimental results among with the DAMP platform analysis indicate that larger performance improvements for future DAMP versions can be expected.

Laboratory : Computer Engineering
Codenumber : CE-MS-2004-11

Committee Members :

Advisor:	G.N.Gaydadjiev, CE, TU Delft
Chairperson:	Stamatis Vassiliadis, CE, TU Delft
Member:	F.L. Muller, CE, TU Delft
Member:	N.V. Budko, EM, TU Delft

I dedicate this document to my family for their love and support.

"If you look for truth, you may find comfort in the end; if you look for comfort you will not get either comfort or truth only soft soap and wishful thinking to begin, and in the end, despair." – C.S. Lewis

Contents

List of Figures **vii**

List of Tables **ix**

Acknowledgements **xi**

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Project description . . . 2
1.3 Project goals and main contributions . . . 2
1.4 Thesis framework . . . 3

2 The DAMP platform and Linux environment **5**
2.1 Platform specifications and limitations . . . 5
2.2 Why Linux? . . . 7
2.3 Linux environment initialisation . . . 8
2.4 Booting the kernel . . . 9

3 History of digital video **11**
3.1 Introduction to still image compression . . . 11
3.2 Lossy compression (JPEG) . . . 12
3.3 Compressing image sequences (MPEG-1) . . . 13
3.4 Improvements to MPEG (MPEG-2) . . . 15
3.5 The ISO MPEG-4 standard . . . 15

4 Codec selection: Why XviD? **19**
4.1 Available codecs . . . 19
4.2 Codec motivation . . . 20
4.3 Installation of the XviD codec on the PC . . . 21
4.4 XviD library interface . . . 21

5 The XviD decoder evaluation **23**
5.1 Exploring the XviD source code . . . 23
5.2 Profiling the XviD codec . . . 23
5.2.1 Profiling the codec on the PC . . . 24
5.2.2 Cross-compilation for the DAMP platform . . . 24
5.3 Speeding up the codec . . . 25

6 IDCT hardware design **27**
6.1 Inverse Discrete Cosine Transform 27
6.2 Hardware software separation 28
6.3 Hardware implementation 30
6.3.1 Hardware and software communication 30
6.3.2 IDCT hardware implementation 32
6.3.3 Fitting the design into the FPGA 35
6.4 Hardware performance 36
6.5 XviD modification 37

7 Experimental results **39**
7.1 Methods of measuring 39
7.2 Acquired results 40
7.2.1 Measurements 40
7.2.2 Speedup calculation 40
7.2.3 IDCT operation time 41
7.2.4 Improving hardware IDCT speed 43

8 Conclusions and recommendations **45**
8.1 Summary 45
8.2 Main contributions 45
8.3 Recommendations for future research 46

Bibliography **48**

A IDCT source code **49**

B Source code of test program: xvid_decraw.c **55**

C Profiling results on the x86 platform **77**

D IDCT datapath and controlling FSM **89**

E Source code of XviD decoder: decoder.c **95**

F GNU General Public License **117**

List of Figures

2.1 Features designed for the DAMP platform, source page 12 of [28]. 6

3.1 Subsequent images, each containing similar parts. 14
3.2 Dependencies between I, P and B frames. 14
3.3 Example of a scene containing grouped media objects, source [18]. 16

6.1 Thinnest links in terms of communication between functions. 29
6.2 Datapath of the implemented IDCT with row/col unit. 33
6.3 States of the FSM controlling the IDCT datapath. 34
6.4 Simulation of several signals from the FSM controlling the IDCT datapath. 36

7.1 Average decoding time per frame for four reference MPEG-4 files measured for six different configurations of IDCT and colourspace conversion (CSPC). 41
7.2 Average relative speedups for six different configurations of IDCT and colourspace conversion (CSPC). 42

D.1 Quartus design of the row/col unit. 90
D.2 Quartus design of "IDCT_DEEL" block which is part of the row/col unit. 91
D.3 Quartus design of the IDCT datapath. 92
D.4 Simulation of all the signals originating from the FSM controlling the IDCT datapath. 93

List of Tables

4.1 Codec comparison 20

5.1 Average decoding time of MPEG-4 files while selectively disabling the IDCT or the colourspace conversion and maximum obtainable speedup . . 25

6.1 Cell usage of the EPXA1 device for the row-only, column-only and row/col unit 32

7.1 Average decoding time of MPEG-4 files while selectively disabling the IDCT or the colourspace conversion and maximum obtainable speedup . . 40
7.2 Average decoding time per frame 41
7.3 Average IDCT operation time per frame. 42
7.4 Total average time per IDCT for software and hardware and calculated speedup of the kernel for each file. 43
7.5 Average IDCT operation time per frame and predicted future values. . . . 44

Acknowledgements

During the different stages of my M.Sc. project, several people have helped to reflect about the direction the project was headed and to focus on the important topics. During my thesis project, which is part of the SMOKE project, I have spent a lot of time together with Jonathan Hofman, whom I was cooperating with on the SMOKE project. We helped to keep each other focussed during the months we spent together. Especially in the summer when many people went away to enjoy their vacation and the weather was nice, the fact that I had someone to share with made the project a lot more fun. I want to thank Jonathan for sticking with me when I wanted to finish in time, even when that meant giving up his vacation.

Apart from my partner in crime, I also wish to thank my advisor Georgi Gaydadjiev who spent so much time reading the documents we produced and who helped us during the project every time we had questions or needed something. He also arranged for someone to assist us on the SMOKE project when implementing the network interface and porting Linux. Thanks for helping us to have our defence in November! I also want to thank Juan Fernandez for working with us on the network interface.

I want to thank Stamatis Vassiliadis, for allowing us to start the SMOKE project. I want to thank Bert Meijs for supplying and helping us installing the computers to work on and for allowing us to print our theses. I want to thank my parents and family for supporting me in my education, both financially and emotionally. I owe a great deal to their kindness and patience! I believe in God and I want to thank Him as well for giving me the opportunity to study at this University in a free country and for the offer Jesus made to bring forgiveness for my mistakes, since I am not perfect while He is.

Guido de Goede
Delft, The Netherlands
December 7, 2004

1 Introduction

This M.Sc. project entails the speeding up of the XviD codec by implementing the IDCT in hardware. It is part of a larger project called SMOKE[1] [14] which aims to speed up XviD. For information on different implemented modifications to XviD the reader is referred to [13].

In section 1.1 the performed research is motivated. The project is described in section 1.2. Section 1.3 will elaborate on the goals of the project and the expectations for the usability of reconfigurable hardware. The thesis framework is described in section 1.4.

1.1 Motivation

The concept of using reconfigurable hardware to enhance the functionality of computer systems has gained much attention recently [20]. There are many projects on this topic, one of which is MOLEN [23]. The MOLEN project was started at the faculty of Electrical Engineering of the TU Delft, focussing on creating a paradigm to integrate the use of reconfigurable hardware into software. The SMOKE project is part of the MOLEN project. SMOKE focusses on the acceleration of operational kernels in MPEG-4 using reconfigurable hardware. There is a huge demand for computer systems capable of performing multimedia applications. For video and 3-d rendering applications faster processors are needed. Intel recently announced [19] that they will not be making processors running at 4 GHz in the near future. Instead of speeding up the processor by increasing clock speed, their approach is to utilise more than one processor core on a chip.

It evidently becomes more difficult to speed up sequential processors. Reconfigurable hardware in short range of the CPU can offer a solution. In multimedia applications, many small operations have to be performed on large quantities of data. These operations are now performed by a central processor, but they could be performed in parallel by hardware designed especially for that operation. It is envisioned that addition of reconfigurable hardware to general purpose processors can provide the means for such parallelism.

Besides the technical limitations to increase the clock speed of processors, another issue is important especially for mobile devices delivering multimedia applications. Increasing the clock speed increases the demands on power usage. Mobile devices have limited power budget, so clock speed cannot be increased to the speed needed for the application[2]. This project intends to demonstrate the usability of reconfigurable hardware for multimedia applications running on relatively slow platforms.

[1] SMOKE is an abbreviation of Speeding up MPEG-4 Operational Kernels on Excalibur [14]

[2] Contrary to personal computers containing 3 GHz processors, mobile devices generally contain processors running at speeds less than 200 MHz. For example the Sony Ericsson P800 and P900 devices contain ARM processors running at 156 MHz [4].

One type of multimedia application that is performed often in devices nowadays is video decoding. Mobile devices have limited storage space and connection speeds, so video needs to be highly compressed. The type of video encoding used in mobile devices today is MPEG-4. It will be proven that using reconfigurable hardware on such platforms can significantly improve performance for multimedia operations. To this extend an MPEG-4 decoder will be implemented and sped up using reconfigurable hardware. Section 1.2 will describe the project in more detail.

1.2 Project description

The aim of the project is to modify an existing MPEG-4 video decoder to use the reconfigurable hardware available on a specific hardware/software platform, the DAMP platform. It contains re-configurable hardware which will partly take over the decoders functionality in order to speed up the decoding process. The platform will be described in chapter 2.

The used approach is to explore which parts of the decoder will benefit the most from the migration to hardware and to design special hardware for those parts, while regarding the way the software and the hardware communicate. The project trajectory can be summarised as follows. First, the codecs will have to be evaluated in order to determine the most suitable one for this project. After that, one codec will be chosen to run on the DAMP board. The DAMP board will be running Linux as its operating system to support the codec. Finally, when the codec is working on the DAMP board it will be modified to make use of the available re-configurable hardware. The goals of the project will be described in section 1.3.

1.3 Project goals and main contributions

The research consists of determining the suitability of the reconfigurable hardware on the DAMP platform for multimedia applications. It is intended to determine this by modifying an MPEG-4 decoder in such a way that it uses the DAMP reconfigurable hardware to speed up its operation. The aim is to create a working MPEG-4 decoder in an operating environment that allows for adequate measurements and comparison. Speedup measurements will give valuable information about the possibilities for such a design in MPEG-4 products.

As seen in other projects concerning the combination of software and reconfigurable hardware [8], one needs to keep in mind the overhead produced by the necessary communication between hardware and software. The total amount of time involved in the communication and computation should be less than the original software-only approach. It is expected that reconfigurable hardware can be used in many applications, making it an advantage for processors to have reconfigurable hardware available to them [25, 6].

This project intends to prove the suitability of the use of reconfigurable hardware on the DAMP platform to speed up multimedia applications. The DAMP platform is built around an Excalibur device. It contains an ARM processor which is used in many other implementations where MPEG-4 video is used, such as in numerous mobile devices

[1]. It is conceived that the use of such reconfigurable hardware can bring the currently available devices using MPEG-4 to a higher level of performance and versatility.

The main contributions of this project are stated below.

- Addition of a network controller to the DAMP board.
- Porting Linux to the DAMP platform.
- Researching the currently most suitable codec to be used and modified for research projects.
- Determining the computation intensive parts of the XviD codec.
- Delivering a fully functional IDCT core in reconfigurable hardware.
- Speeding up the XviD codec on the DAMP platform using the available reconfigurable hardware.
- Creation of a test program to decode MPEG-4 reference files using the created IDCT and colourspace conversion cores.
- Creation of a test program to output decoded images to the VGA controller and to measure the time needed for the decoding and displaying of each frame.

1.4 Thesis framework

This document is organised as follows. Chapter 2 will describe the environment that is used to operate in. It will describe the platform, the software environment and its configuration. In chapter 3, an overview of the history of digital video is provided, introducing still image compression and various moving images compression techniques. Section 3.5 will describe the ISO MPEG-4 standard. Chapter 4 will discuss the decoder selected for this project after which it is explored further in chapter 5. To speed up the selected decoder, dedicated hardware is designed and evaluated in chapter 6. Experimental results about the obtained speedup by the modifications performed to the decoder, can be found in chapter 7. Chapter 8 concludes the document and gives recommendations for future research.

The DAMP platform and Linux environment

2

The DAMP platform will be the target platform of the MPEG-4 codec used in this project. On this platform, the speeding up of the decoding part of the codec will take place. The DAMP platform was designed as an M.Sc. project at the TU-Delft in 2003, about which additional information can be found in [28] and [9]. It contains an Altera Excalibur device which combines an ARM processor stripe with reconfigurable FPGA fabric and various peripheral devices on a single chip. Section 2.1 will describe the DAMP platform in more detail. The chosen software environment is Linux combined with an ARM boot-loader. The choice of Linux will be explained in section 2.2. Section 2.3 describes the initialisation of the software environment. Finally, section 2.4 describes the booting of the Linux kernel. More information about the software environment can be found in [13].

2.1 Platform specifications and limitations

The DAMP platform is built around the Altera Excalibur device EPXA1F672C3 as detailed in [6]. It was designed to be a platform for the creation and testing of embedded systems targeting multimedia and mobile devices. The Excalibur contains an ARM922T processor with peripherals and an Altera FPGA with 4160 logical elements (cells). Nowadays the ARM processor can be found in many portable devices. This makes DAMP a versatile development platform for such devices. Hardware and software co-designs can be created on it using the Quartus II development software [7]. The DAMP platform contains the following parts which are of importance to the targeted design.

- the Excalibur device, containing an ARM processor and a programmable logic device (PLD).
- a FLASH memory to store the bootloader software.
- a JTAG interface for configuring the hardware and programming the bootloader into FLASH.
- an SDRAM to store the multimedia data.
- a UART interface to communicate externally with the operating system.
- a VGA output which can be controlled from the PLD to output decoded images.
- a network interface for access to large files[1].

[1]The network interface was designed to be on the DAMP platform, but on the prototype it was not available yet, due to footprint errors. However for the continuation of the project, the implementation

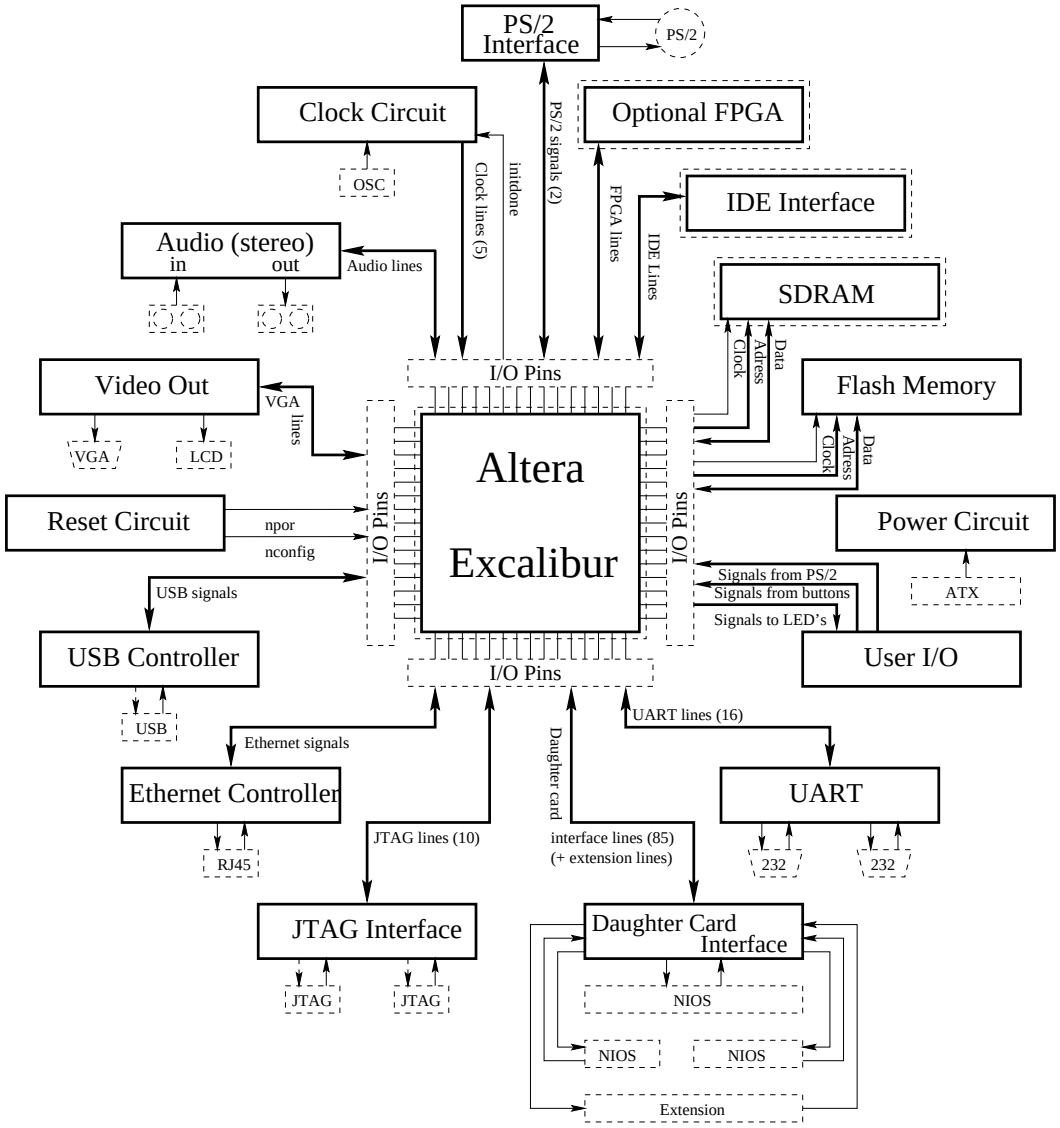


Figure 2.1: Features designed for the DAMP platform, source page 12 of [28].

More details on the above and other DAMP features can be found in [9] and [28].

Performing the project on the DAMP platform showed that this version of the design, which is actually the first prototype, has certain limitations. The speed of the SDRAM cannot be set to the desired maximum of 83 MHz[2] supported by this Excalibur device. Instead, it has to be set to the lowest possible speed of 12.5 MHz to maintain

of a network device was necessary. An external network device has been literally soldered to the DAMP board.

[2]The current EPXA1 speedgrade 3 Excalibur device supports SDRAM at 83 MHz, but the speedgrade 1 device would support SDRAM up to 133 MHz.

system integrity. Higher speeds cause the system to become unstable and to crash within a short amount of time. It is expected that mentioned crashes are due to misalignment between lengths of data, address and control lines of the SDRAM connections on the DAMP PCB. However, transient simulation of the PCB would be required to verify this hypothesis. Future versions of the DAMP platform are expected to have overcome this problem, but for this project only the limited prototype is available.

The used prototype has footprint errors, giving rise to the need of connecting several chips off-board via wires, which would otherwise have been mounted on the PCB. For instance the FLASH memory was connected to the PCB using this method. The wires make the device more sensitive to environmental noise and mechanical movements. Because of this, only one 4 MB FLASH device was connected to the PCB. This small FLASH memory is inadequate for providing a filesystem to the operating system. Use of the network interface became evident. The network option that was designed to be on the board had not been implemented due to the incorrect footprint of the network controller. Implementing the network interface had to be done by connecting wires from the PCB to a separate network controller. This way the operating system was enabled to access large non-volatile storage media, instead of the limited FLASH memory. Details on the connection of the network device and modification of the ethernet driver in Linux can be found in [13].

The VGA interface has a simple DA-converter, consisting of a resistor network for each colour channel, to enable the use of 256 colours on an external display. In the future higher resolution DACs could provide the board with more colours. However for the demonstration of the decoding process 256 colours will suffice. The hardware to supply values to the DA converters of the VGA interface has to be built inside the PLD. This will leave only part of the PLD for the design of auxiliary hardware to speed up software on the platform. The used Excalibur EPXA1 device is rather small since it has only 4,160 logic elements (cells). From the same device family the EPXA4 exists. It has the same footprint as the EPXA1. The EPXA4 has 16,640 logic elements. Another device of the same family, the EPXA10, has more logic elements, but it does not have the same footprint as the EPXA1 and EPXA4. In the future the larger EPXA4 device could be mounted on the PCB, giving additional design freedom.

Figure 2.1 shows all the features for which the DAMP platform was designed. Not all of these features are available on the prototype. For details on which features are implemented by the designers of the prototype, the reader is referred to page 12 of [28], where figure 2.1 is explained in detail.

2.2 Why Linux?

When implementing an MPEG-4 codec for the DAMP platform, one will need to be able to validate the correct functionality of the codec. Therefore, the ability to open files and to write back the results (or possibly display the results) is required. Regardless of the way in which an MPEG-4 codec will be implemented, to properly test and compare the codecs functionality, the use of an operating system can hardly be avoided.

Linux is an operating system that can be used freely and has already been ported to comparable Excalibur boards. One could argue that implementing the codec without

the use of an operating system would also be possible. However, in the early stages of this project it would save time to port Linux to the DAMP platform for several reasons. One being that the porting of Linux is better documented but would require more or less the same effort as would porting the codec without an operating system. Secondly, since Linux has already been ported to comparable ARM boards, most of the effort of porting Linux has already been made. Using Linux allows postponing the final decision about the codec to a later stage of the project. In addition, porting Linux would give more insight in the DAMP board. This would allow for better motivating the choice of the codec. Considering the above, for this project Linux was preferred to run the codec on, instead of creating a standalone application.

2.3 Linux environment initialisation

When Linux is compiled from source code, an image file is created containing the necessary instructions for the processor to boot Linux. This image however can only be used when different parts of the platform have been properly initialised. To obtain proper initialisation, a bootloader has to be used, which is the software that initialises the hardware before the operating-system starts. The bootloader used for the DAMP board is the open source project ARM-boot.

ARM-boot initialises the memory (SDRAM) and starts a terminal which is accessible via the serial port. It can be set-up in such a way to automatically load a kernel image from the FLASH memory to the SDRAM when booting, but it also supports loading blocks of data into memory from the serial port. However the serial port is slow, so it is not suitable for uploading large amounts of data. To this extend the much faster network interface should be used. The TFTP boot protocol can be used to load an image into memory and boot from it. These images will then be used to boot the kernel as will be explained in section 2.4.

When an image containing the kernel of Linux is loaded into the FLASH memory, ARM-boot can be instructed to boot that kernel from FLASH. The FLASH memory is 4 MB (megabytes) large, of which less than 1 MB is used for ARM-boot. The Linux-kernel also takes less than 1 MB. This leaves over 2 MB of the FLASH memory for the files required after booting. These files are put together into a compressed image file, called the filesystem image. In the filesystem the cross-compiled codec and video-files to work on, also have to be present. Having the filesystem in the FLASH memory however is not practical, since the FLASH size is small and FLASH devices have a limited number of writing cycles. To provide support for a larger filesystem the network interface is needed.

For testing the correct operation of the network interface the built-in functionality in ARM-boot to use TFTP was used. Using the TFTP boot protocol ARM-boot can load a configuration file from a server, containing the image of the kernel and the file system. This removes the limitation of the size of the FLASH interface and the speed of the serial interface for uploading files to the system.

The FLASH memory is large enough to contain the kernel image without the filesystem. The kernel can then use the network interface to connect to an external filesystem. Having the kernel boot from FLASH gives two advantages. The FLASH is still used to boot, so no TFTP server is needed. Secondly, data does not have to be written to it,

because the filesystem resides elsewhere. The addition of the network interface and the modification of the bootloader are described in [13].

2.4 Booting the kernel

Normally when ARM-boot starts, the kernel-image from the FLASH is decompressed into SDRAM and executed. However, care must be taken when installing the kernel. The used Linux kernel contains settings that depend on the configured processor speed. It has been used on boards similar to DAMP, such as the EPXA1 development board. Since the processor speed on the DAMP board differs from the speed the Linux kernel was configured for, these settings are incorrect and have to be changed.

When Linux was ported to the DAMP platform a hardware configuration file has been used for configuring the hardware on the DAMP platform. It had originally been generated by the Quartus design tools from Altera. Because the file has been automatically generated it contains directives to the reader not to change this file, since all changes would be lost the next time the file is generated. However, the configuration file is distributed with the used Linux kernel (version 2.4.19-rmk7) separately from Quartus, therefore it would not be updated. In fact it has to be changed manually when one is porting Linux to the DAMP board, since the settings of this board are slightly different from the one the kernel was created for. The latter was the main reason for the problems we encountered in the early stages of the project. The settings of the clock frequency of the DAMP board in the mentioned file were wrong. The incorrect setting resulted into garbage characters on the screen coming from the DAMP board serial port. The speed at which the serial port communicates with the computer is directly dependent on the clock speed set in the file. A wrong clock speed results in a non-standard serial port baudrate, causing these garbage characters. To obtain the correct settings for the DAMP board, without again using Quartus to generate such a file, a similar file from ARM-boot has been used, which had also originally been obtained from a project compiled with Quartus. It obviously contained the correct settings, since ARM-boot already worked fine using the serial port. This made it possible to verify that Linux was indeed booting up until a certain point.

At first when trying to use the filesystem with an image in FLASH memory, Linux was unable to use the filesystem contained in the image file. It contained improper device nodes, which are files giving access to hardware devices using system calls. After creating the proper device nodes in the filesystem image and changing the clock settings Linux did boot up completely. Compiled programs could now be run on the DAMP board from within Linux, but there still was too little space for data files. Also large programs with libraries like the XviD codec did not fit. To solve this, the network connection proved the most beneficial addition. It speeds up the communication and makes the use of larger files feasible. With the working network interface, the system can function properly. It enables the use of large files to test the functionality of the codec as will be done in chapters 4 and 5. The modifications made to the device nodes and to the clock speed settings of the kernel, are explained in [13].

3 History of digital video

In the past decades different standards for recording analogue video have been introduced [21]. These video standards offer colour image information at roughly 13,5 million image pixels per second. Digitising such a video stream would require one to process 216 Mbits per second, according to the CCIR 601 standard [24]. Nowadays computers are fast enough to process data at rates of this magnitude, however storing the video on digital media would require prohibitively large storage space.

Throughout history efforts have been made to reduce the size of video streams, without losing visual quality [2]. These efforts began with the need to compress digital still images for storage as introduced in section 3.1. Section 3.2 will explain the basics of lossy still image compression. After this, section 3.3 expands the focus from still images to sequences of images and their compression using MPEG. Section 3.4 discusses improvements made to the initial lossy video encoding techniques. Section 3.5 elaborates on the ISO standard for MPEG-4 [18] as created by the International Organisation for Standardisation[1].

3.1 Introduction to still image compression

An image basically consists of a 2-dimensional plane of dots called pixels, each pixel containing colour information. A standard computer screen is at least 640 pixels wide and 480 pixels in height. This equals 307200 pixels. Each pixel consists of 3 values, one for red, one for blue and one for green colours. Mixing these three basic colours creates the impression to the human eye of a certain colour being present [3]. One can see that, when each pixel would have 24 bits for colour information (one byte for each colour), the number of bytes needed for one full screen image is over $9 \cdot 10^5$ bytes. In the early days of computer imaging this was an enormous amount of data. Even for images with only 256 colours (one byte per pixel), still $3 \cdot 10^5$ bytes were needed. Evidently, images needed to be compressed.

The first type of compression that naturally comes in mind, is using readily available data compression algorithms to compress these images. Many images contain large planes of the same colour. For images with only a small amount of colours compared to the number of pixels in the image, reasonably good compression can be obtained. This compression is called lossless compression, since it is done without any data losses. One such lossless compression technique is the Graphics Interchange Format [15]. The lossless compression of image sequences also became possible in this way, concatenating a sequence of compressed images, for instance in a format called FLIC [17].

[1]The abbreviation ISO comes from the Greek word "ισος" meaning equal. The use of this word ISO is chosen because the International Organisation for Standardisation would be abbreviated differently in different languages, for example IOS in English and OIN in French.

3.2 Lossy compression (JPEG)

When a person looks at a picture, not all parts of the picture have the same relevance. People remember pictures subjectively and are generally not capable of regenerating every detail of an image from memory. This gave rise to the idea to save only those parts of an image that people are most sensitive to, instead of keeping every pixel in an image and trying to compress the whole without loss of information.

When trying to store only relevant parts, one needs to determine when data is relevant. The first thing to look at to determine what image data is relevant for humans, is the human eye. The retina contains parts called cones sensitive to colour and parts called rods sensitive to light intensity [12]. The centre of the eye contains more cones and the larger surrounding part of the eye contains more rods. This renders the eye in general more sensitive to light intensity. There are three different types of colour sensitive parts, one type sensitive for red, one for green and the third for the colour blue. Digital pictures generally consist of a number of pixels, for which each pixel contains three values, one for the intensity of each of the three colours present.

As explained in the previous paragraph, the human eye is most sensitive to changes in light intensity and less to changes in colour. Therefore it makes sense to store images in a format which exploits this property. A pixel then contains one value, called luma[2] , for the light intensity and two values called chroma, representing the chrominance, determining the colour of the pixel. Since the changes in colour are less noticeable, the chroma values of 4 pixels together are averaged, leaving only two chroma values for a group of 4 pixels, so instead of 2 values per pixel one now has 1/2 value per pixel. This reduces the amount of information per pixel by a factor of 2, since at first 3 values for the red, green and blue components were needed for each pixel, but now 6 values per 4 pixels are needed.

When applying this process to images, the subjective quality of the image does not change much, however the process does leave out information and is irreversible. Such an irreversible process is called lossy, since information is lost. Apart from colour information, properties of human perception leave room for more optimisations by leaving out unimportant information. Another example of what can be optimised, is the human perception of changes in a picture. High frequency spatial differences between pixels are less noticeable than gradual changes, since the eye tends to blur the image. Therefore when looking at an image in the frequency domain, only the low frequencies are perceived and matter the most.

By performing a discrete cosine transform (DCT) on the image it is transformed into the frequency domain and the values that matter the most can be prioritised. Tests have shown for human perception, that after removing (or severely rounding) the higher frequency components from the image and then transforming back, the subjective quality only marginally changes. This opens opportunities for the idea of quantisation. When limiting (quantising) the values after the DCT to a certain set of values, less information is needed to describe the remaining values after quantising. This process is lossy,

[2]The use of the word luma is preferred over the word luminance to note the difference between the non-linearly transformed value used in digital images and the real luminance value from colour theory. This is explained in [22]

however the subjective quality loss can be adjusted to be acceptable for the amount of storage space that is saved in the process.

One standard using these techniques is the JPEG standard (Joint Photographic Experts Group standard) incorporated into the JPEG File Interchange Format (JFIF) [11]. Since its introduction, it has become the most used format for photographic pictures. Due to its small file sizes it is very suitable for transferring images via the internet. It is also possible to create more versions of the same picture, in different qualities, to suite different needs.

3.3 Compressing image sequences (MPEG-1)

The technique for compressing images by only keeping relevant data, can also be applied to moving pictures. These are essentially subsequent snapshots, creating a sequence of images. The images can be compressed using for instance JPEG. Compressing each frame individually is what a Motion-JPEG codec does.

Advantages of such a method are that it is just as operation intensive as JPEG compression and it can be done out of order or in parallel on different frames of a video sequence. However the compression ratio reached leaves much to be desired, since video files still are very large using such a codec, because they consist of many images. One can distinguish a fundamental difference between video compression and single image compression. The subsequent images of which a video consists are generally very similar to each other. Somehow one should take advantage of this property, but the Motion-JPEG codec certainly does not, because it only emphasises on separate images.

The Moving Pictures Expert Group (MPEG) developed a standard [5] that does use properties of the subsequent frames for achieving compression. Since many frames are alike, only the difference between one frame and the next is saved, reducing the amount of information needing to be stored. Every now and then a scene changes in such a way that a frame is completely different from the previous one. In that case the frame is saved entirely, compressed in a way similar to JPEG. Those individually compressed frames in MPEG are called I-frames. The frames that only store the difference compared to previous frames are called P-frames.

Suppose a picture contains a tree and a building in the background and a locomotive driving by. At first one sees one side of the building, but later that part of the building is covered by the moving locomotive and the part of the tree that was first covered by the locomotive is visible again. After the locomotive is gone, the image is very similar to the image that described the background before the arrival of the locomotive, so one is likely to use a P-frame and store only the difference compared to the previous frame.

There is a third way of storing compressed frames. During the presence of the locomotive, the scene changes in such a way that a frame could be stored with less information if one would refer to a future frame instead of a past frame. This is where B-frames are designed for. They can build up a current frame using information from a previous P (or I) frame and a following P-frame. The described scene with the locomotive is depicted in figure 3.1. In this figure, frame 1 could be an I frame or a P frame. Frame 3 can be stored as a P frame containing information from the previous I frame. Frame 2 contains information obtained from frame 1 and frame 3 so it can be stored as a B

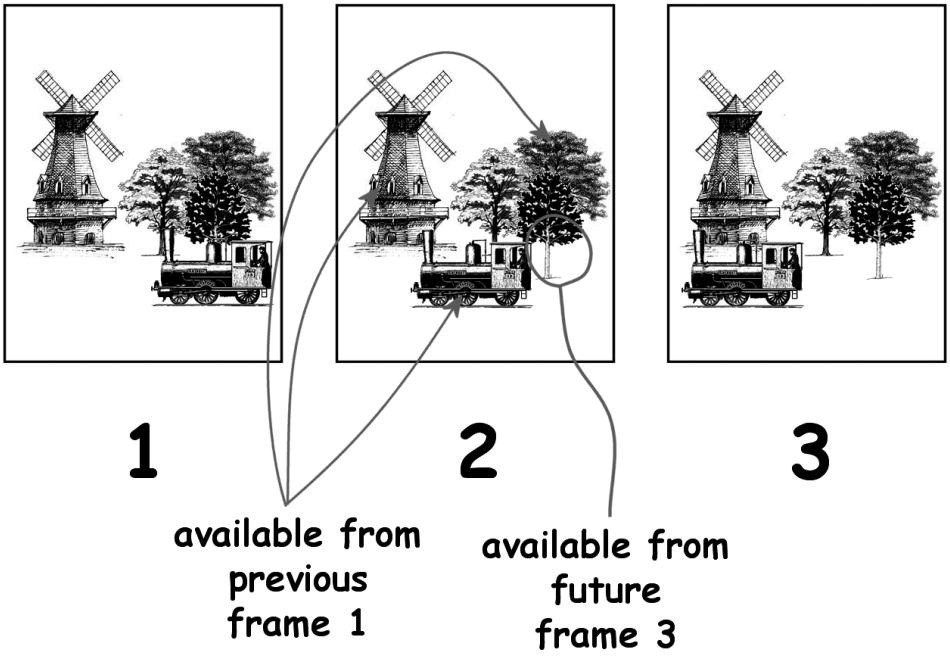


Figure 3.1: Subsequent images, each containing similar parts.

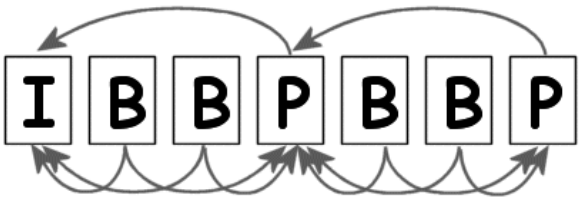


Figure 3.2: Dependencies between I, P and B frames.

frame. B frames cannot refer to other B frames, because this would make the decoders too complex. Now only an I frame and one or two P frames are needed to reconstruct a frame from a B frame. If B frames would refer to other B frames as well, all these B frames would have to be stored while decoding. The dependencies between I, P and B frames are depicted in 3.2. Groups of P, I and B frames are placed together in a predefined order into a GOP (group of pictures). The MPEG-1 standard has certain limitations, such as the maximum bitrate, as well as limitations to the size of a GOP and the number of each type of frames in it, providing an upper constraint on quality. The MPEG-1 format has been extended in various ways since its first definition, as will be explained in section 3.4.

3.4 Improvements to MPEG (MPEG-2)

A newer standard, called MPEG-2 has been developed to be more of a generic coding standard, but still to be adaptable depending on the medium that will be carrying the MPEG-2 stream. The idea is to have enough standardisation to provide compatibility between different applications, but yet to be flexible enough to permit a degree of freedom when implementing the applications.

Apart from the features designed into MPEG-1, other options have been added to MPEG-2. It contains the possibility to use other sampling standards for storing the luma and chroma information of the pictures. It also has the possibility to handle interlaced video, in which at doubled frame rate one frame contains only the even lines and the next frame contains the odd lines of a picture. Depending on whether there is much or little motion, different ways of compressing this interlaced video are supported [2].

Another improvement in MPEG-2 is that it now allows the bitrate to be varied depending on the quality required. In MPEG-1 the bitrate was kept constant by performing adaptive quantisation and thus losing more information. In MPEG-2 this adaptive quantisation can now be used to assure quality.

Finally, scalability is added by allowing different resolutions to be encoded incrementally. First the low resolution image is encoded and the higher resolution image is encoded secondly, based on the differences with the lower resolution image.

Not every MPEG-2 encoder will support each option. However they will all produce an MPEG-2 stream adhering to the standard. So all produced streams can be decoded. Thus one might choose to build a simple encoder, implementing only the possibility to create a stream of I-frames (with a complexity similar to Motion-JPEG) for example. Still this stream conforms to the standard and can be decoded. This is how specific video editing applications create MPEG-2 video, when the type of GOPs to use has not yet been specified. In that case all frames are stored as I-frames, to be converted to I, B and P frames later at a relatively small quality trade-off. MPEG-2 is the standard used in DVDs, possibly containing up to almost 9 GB of information for a few hours of video. For use on the internet video files should be smaller. Currently the most used standard for video distribution on the internet is MPEG-4. Section 3.5 will go into detail about the MPEG-4 video standard which is used in this project.

3.5 The ISO MPEG-4 standard

The MPEG-4 standard is developed by the International Organisation for Standardisation. It is the successor of MPEG-1 and MPEG-2 and has been designed to be a standard with the advantages of its predecessors, but with enhanced functionality. Researchers and engineers from different countries have cooperated in designing the MPEG-4 standard. Pieces of the standard were developed by separate teams.

Several meetings were held lasting for up to one week. In the time before these meetings, members have electronically sent in their contributions, allowing other members to review them to speed up the process. During the meetings, plenary discussions were held, as well as subgroup discussions. Occasionally a group was formed to continue until the next meeting. Their deliverables were then presented and used in the next meeting.

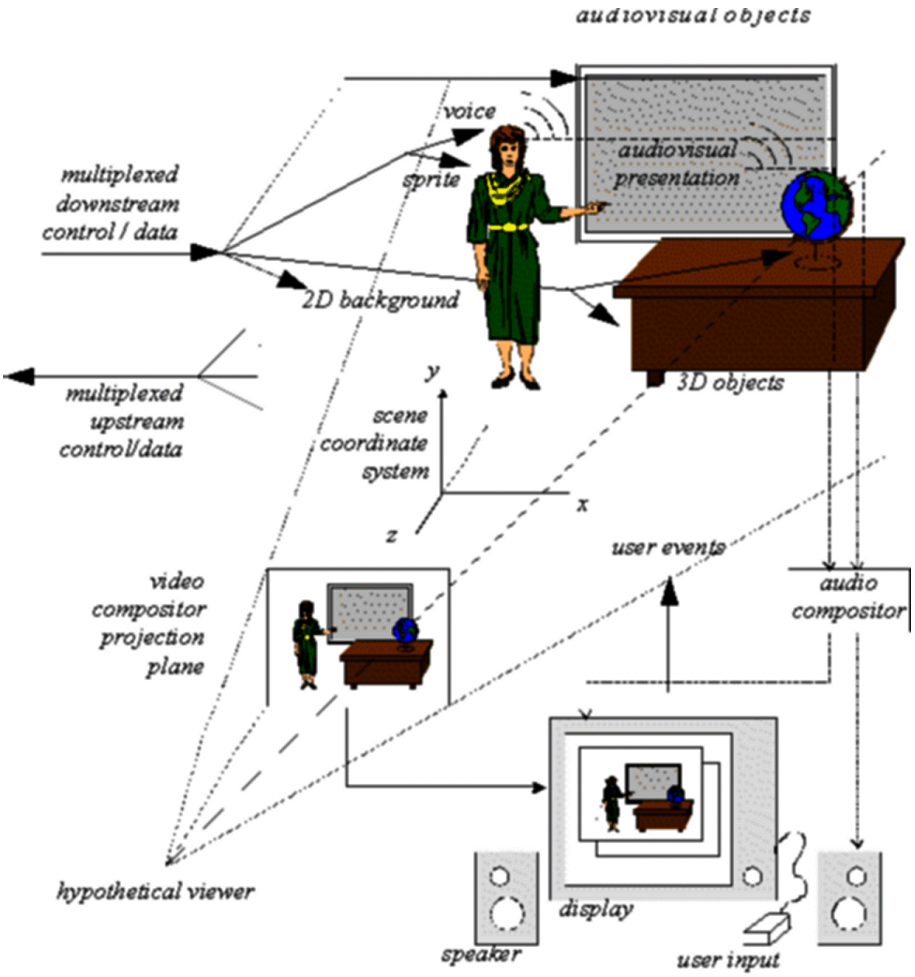


Figure 3.3: Example of a scene containing grouped media objects, source [18].

At each meeting around a hundred documents are created, of which the ones describing the standards under development are of particular importance. This section describes the major parts of the MPEG-4 standard as detailed in [18].

The idea behind MPEG-4 is to offer a framework to hierarchically combine different media objects, such as images, video, audio and text. These primitive media objects can be grouped together forming a tree structure. In this tree a subtree, which represents the grouped media objects, can again be identified as a compound media object. For instance, a video of a person talking, combined with the audio of this persons speech can be grouped together into a compound media object. Groups of compound media objects can form a 2-dimensional or 3-dimensional space. The media objects can be created by recording of images, video or sound, but they could also be generated by a computer. So either natural or synthetic media objects can be used. Different media objects such as audio, 3-d objects or sprites (still images) are multiplexed with control data into one stream. In figure 3.3, an example is given of how media objects can be grouped together to form compound media objects. In this figure the woman talking, the whiteboard

behind her, the desk and her voice as well, are separately created objects. The woman can be natural video and her voice could be recorded audio. The desk could be a 3-d rendered synthetic object and the whiteboard could be a computer generated 2-d image or an audiovisual presentation containing both images and audio. For the hypothetical viewer these objects together would form a 2-d view in the video compositor projection plane as indicated in the figure. Changing the viewpoint could be allowed by the author to change the way the grouped objects would appear to the viewer.

The MPEG-4 standard provides the author of the content the possibility to produce complex multimedia content, while making it easier to reuse content. For example the desk in figure 3.3 could be used in other content as well. Also the author is given the possibility to restrict the user of the content and to protect the owners rights. The user can be given more freedom to experience the content that is created. For example in a 3-d environment, the user can view the content from different angles only if allowed by the author. Also MPEG-4 contains standards for many applications such as standards for low bitrates. This yields the possibility to provide MPEG-4 content via low-bandwidth channels like the ones used in mobile communications. Besides the larger freedom and the greater amount of applications, the user is offered more interaction with the content. Depending on the actions of the user, different content could be provided to the user. One could imagine the user participating in the provided content, making it specifically tailored to the end user.

Apart from the author and the user, MPEG-4 also offers information to the network service providers. Because of the standardised form in which the information is broadcast, native signalling messages for the network can be relatively easily generated. MPEG-4 contains a general quality of service (QoS) descriptor for different MPEG-4 media, but the exact details of the network QoS are not a part of MPEG-4. MPEG-4 embodies several major functionalities in different profiles. XviD encompasses only natural video into the codec. It supports the encoding of rectangular video objects. The implementation of the MPEG-4 decoder in XviD is described in section 5.1.

4 Codec selection: Why XviD?

Many codecs for MPEG-4 exist, all having slightly different features. To make a reasonable decision one needs to dive into the details of the codecs. In section 4.1, the available codecs will be discussed. Section 4.2 will evaluate the suitability of the codecs and determine the one to be used. Section 4.3 will describe the installation of the XviD codec. The XviD library interface is described in section 4.4.

4.1 Available codecs

Some variants of MPEG-4 have become popular because they were used to make video available on the internet at acceptable subjective quality. These MPEG-4 videos had file-sizes several times smaller than their MPEG-2 alternatives. Microsoft developed an implementation of an MPEG-4 codec, for which only a decoder was made freely available. The Microsoft codec was modified by some people to be used also for encoding. These people made it available on the internet, which stimulated the creation of MPEG-4 content. Other codecs have been created around the same time, some based on the Microsoft codec, some built from scratch. One such MPEG-4 codec is XviD [27].

Apart from XviD several MPEG-4 codecs exist today. Another popular codec is DivX. This is a proprietary closed source codec targeting Windows and Mac OS users. One notices that the word XviD resembles the name DivX. Instead of closed source, XviD is an open source project to build an ISO MPEG-4 compliant video codec. The first key property of the codec suitable for the SMOKE project, is that it should be open source. This implies the source code of the codec should be publicly available and modifiable without infringing copyrights. Also the source code should be portable to the DAMP board (ARM-platform) as will be explained in the next chapter. A list of publicly available codec families together with their properties is shown in table 4.1[1], arranged in order of importance with the most important property on top.

The software should be open source and freely modifiable. It should be ready to be used for the ARM processor and therefore it should be easily cross-compiled. The code should be compact, so it can be understood without needing to spend weeks on traversing the source code. Finally, the codec should be mature and have the ability to decode MPEG-4 files. So all parts of the MPEG-4 decoding process should be implemented. As can be seen the MoMuSys codec and the XviD codec come out best based on these properties. The MoMuSys codec is compact and therefore the source code should be more comprehensible than other codecs. When comparing XviD and MoMuSys, it can be seen

[1] By *GPL* it is meant that the source code is available under GNU General Public License as stated in appendix F. *Compact* refers to the source code size or in the case of closed source projects to the binary size of the distribution

Codec:	DivX	ffmpeg/libavcodec	MoMuSys	sklmp4	XviD
Open Source	NO	YES	YES	YES	YES
Free to use (GPL)	NO	YES	YES	NO	YES
Ready for ARM	NO	YES	YES	Unclear	YES
Cross-compiles easily	NO	NO	YES	NO	YES
Mature	YES	YES	NO	Little	YES
Compact	Little	NO	YES	Little	Little

Table 4.1: Codec comparison

that the compactness of the MoMuSys codec is due to its immatureness. MoMuSys was not developed as far as other codecs. Therefore the compactness is a disadvantage rather than an advantage. MoMuSys has not been updated in the last years and many codecs are built from where MoMuSys has left. The newer codecs use code from MoMuSys as a base, but contain additions and more features, making those codecs more mature.

As one notices, XviD is preferred over MoMuSys. One could argue that DivX is widely known, even among people not working with MPEG-4 video and that the codec seems very mature, but it has been designed especially for the x86 platform, and it is closed source, so porting it to ARM is not possible. Also the licensing of the codec is very important and eventually only codecs that can be freely modified, will qualify for use in the project. In this document the emphasis will be on the XviD codec. XviD can be freely modified because it is available under the GNU General Public License as described in appendix F.

4.2 Codec motivation

In order to determine a suitable version of an MPEG-4 decoder to be modified and expanded for application on the DAMP platform, several codecs have been evaluated. This is described in section 4.1. A preferred property of a codec for this project is that the source code will be easily modifiable. This enables the use of certain platform specific hardware. It should be possible to create parts of the software in reconfigurable hardware, to support and speed up the software.

Easily modifiable code implies easy portability. The source of the codec needs to be compiled not just for a normal PC (the x86 platform), but for this project it will have to be compiled for the DAMP board (an ARM platform). The process of compiling source code on one machine for usage on another machine is called cross-compiling. XviD was cross-compiled with relative ease to run on the ARM platform. To test the cross-compiled codec on the ARM-platform, an operating system has to be installed. The codecs are Linux-based and Linux is a mature operating system that can relatively easily be ported to the DAMP platform. It has therefore been decided that Linux should be used on the DAMP board to create the modified codec in.

Due to limitations of this version of the DAMP board, the installed Linux operating system lacks large local non-volatile memory for installing programs. Before installing the network interface, the system initially had no fast interface to another PC for uploading files. For that reason, extensive testing of the cross-compiled codec on the

platform had to wait until installation of the network interface. At this stage it was of importance that the codec could be cross-compiled and that it would operate correctly for the stated purpose.

Testing the decoding of large files on DAMP was performed at a later stage. The codec operation has first been tested on the x86 PC. It is the build-platform, on which the cross-compiling of the codec was performed. Testing on the PC was done to assure that the codec contained the functions necessary for the project. The number of times a function is called is independent of the platform the codec is running on. Functions called often on the PC will also be called as many times on the DAMP platform, because the functional boundaries remain the same. The profiling on the x86 platform would thus give information as to which parts of the codec will benefit the most from being sped up, before having to run XviD on the DAMP board. Installation of the XviD codec on the PC is described in section 4.3.

4.3 Installation of the XviD codec on the PC

The last stable release[2] of the XviD codec version 1.0.1 was downloaded and compiled to run on a personal computer, an x86 platform running RedHat 8.0 Linux. This version of Linux was chosen to be compatible with the Quartus tools. As it turned out it was not possible to run Quartus in Linux using the available student license, so Quartus had to be run under Windows.

To compile XviD in Linux, first a bootstrap script has to be run to create the necessary configuration files. After running the bootstrap script, the created configure script will have to be run to correctly initialise the settings necessary for compilation on the current platform. Having configured all settings, the source code can be compiled using the makefile delivered with the XviD release. The compilation of the source creates a library package containing the XviD routines that can be linked with other software. The makefile also contains settings for installing the compiled XviD libraries in the correct directories on a standard Linux machine. To create a binary executable using these library functions, the library has to be linked with the source code, when compiling the source for that executable. It can be run on another x86 PC without having to install XviD on that computer as well. It is possible this way to create a program that uses the XviD library functions. Section 4.4 will go into detail on the usage of the available XviD library functions from within such a program.

4.4 XviD library interface

This section describes the way programs using the XviD library are to communicate with it. XviD contains three functions for the decoding sequence that have to be called in order. The example program `xvid_decraw` calls these functions. The first function

[2]Initially the latest release of the XviD decoder was used (the repository snapshot of June 1st 2004), but it contained bugs that wrongly changed the colours of the resulting video sequence. Therefore the stable release version 1.0.1 is used.

that is to be called is `dec_init()` to initialise the decoder environment. This function configures basic settings like the dimensions of the image, the version of XviD used and the colourspace in which to output the decoded images. This is the function that will be modified to indicate to the decoder whether or not to use the hardware kernels to speed up XviD. Which kernels are to be put into hardware on DAMP is determined in section5.2. The `xvid_decraw` program is also modified to optionally output images to the VGA driver. The design of the VGA driver is detailed in [13]

The second and most relevant function called from `xvid_decraw` is the function `dec_main()` This is the function that actually decodes a frame from the source bitstream and outputs the decoded images. After the decoding of the last image, `xvid_decraw` will know from `dec_main()` that all frames are decoded. The third function is the one which tells XviD to clean up the memory used for decoding when it has finished, is `dec_stop()`. The source of the example program calling these functions can be found in appendix B.

5 The XviD decoder evaluation

The motivation for choosing the XviD codec has been made clear in chapter 4. Section 5.1 will examine the source code of the XviD codec. The profiling of the XviD codec is elaborated on in section 5.2. Section 5.3 will explain how these profiling results will be used in the process of speeding up the XviD codec.

5.1 Exploring the XviD source code

As stated in section 4.4 the XviD codec can be used from within a program to decode MPEG-4 video, by calling functions from the XviD library. This XviD library then has to be compiled and linked together with the program. The three functions that are called from the example program have been explained. This section will elaborate on the functions worth noting for speeding up the decoder. The function which is important for this project is the function `decoder_mb_decode()`. It operates on the MPEG-4 bitstream from an MPEG-4 file and it uses quantisation matrices which have been generated during the initialisation of the decoder. The mentioned decoding function contains several other functions operating on the data one after another.

First a macro-block of encoded data is retrieved from the bitstream containing up to 6 blocks of data. This is done by the function `get_inter_block()` or `get_intra_block()` depending on the type of frame currently decoded (P-, B- or I frame). After retrieving the values they are de-quantised. As mentioned in chapter 3, values are quantised when encoding, so they need to be de-quantised when decoding. On the resulting values the Inverse Discrete Cosine Transform (IDCT) is performed by the function `idct_int32()`. This is done for each of the 6 blocks of 8x8 pixels in the macro-block. On the resulting image a colourspace conversion is performed after which the image is returned. The source code of the decoder functions is listed in appendix E.

In section 5.2, the decoder will be profiled to determine which functions require the most computation time. Section 5.3 will discuss the speeding up of the XviD decoder. In chapter 6 the hardware to speed up XviD is designed. This chapter will explain that the IDCT will be migrated to hardware. The parent function `decoder_mb_decode()` calling the IDCT and the functions before and after the IDCT will be too large to move to hardware entirely.

5.2 Profiling the XviD codec

To determine the computation intensive parts that would benefit the most from moving to hardware, the codec needs to be examined and profiled. In section 5.1, the XviD codec source was examined. Initial exploration of the codec was done on an x86 Linux

platform. The profiling of the codec to determine the computation intensive kernels will be described in subsection 5.2.1. Cross-compilation of the codec to the ARM platform is described in subsection 5.2.2. Finally, the maximum obtainable speedup for the chosen kernels is calculated in section 5.3.

5.2.1 Profiling the codec on the PC

As explained in section 4.3 the XviD codec was compiled for the x86 platform. To enable the profiling of the XviD codec, the example source code provided with the XviD codec was modified, as explained in section 4.4. This example code, `xvid_decraw` as listed in appendix B, was compiled with profiling turned on in the compiler switches. These switches tell the compiler to include profiling functions which enable the user to gain information about the time spent in each function during the operation of the program. When running the generated executable, a file `gmon.out` is created that contains said timing information. This `gmon.out` file can be read by a profiler program, in our case `gprof`, to generate human readable information as to which functions use the most time. A summary of the results of this profiling is shown below.

```
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 26.92      0.43     0.43       67     6.44     6.44  yv12_to_bgr_c
 19.73      0.75     0.32   167121     0.00     0.00  idct_int32
  8.40      0.88     0.13    90615     0.00     0.00  get_inter_block
  6.46      0.99     0.10   136681     0.00     0.00  transfer8x8_copy_c
 .....
```

The entire profiling result can be found in appendix C. As can be seen from these results the colourspace conversion `yv12_to_bgr_c` and the IDCT `idct_int32` take up the largest amount of execution time.

5.2.2 Cross-compilation for the DAMP platform

After compilation on the PC, the XviD codec was compiled for the DAMP platform. The first noticeable difference when running the decoder, was the time needed for initialisation. Due to the slower memory interface on the DAMP platform, initialising the codec took a considerable amount of time. Initialisation of XviD on the DAMP platform lasted about half a minute, while it lasted only about a second on the PC. When again compiling the codec with profiling turned on, the initialisation function of the XviD decoder indeed took most of the time. The initialisation function will not take much time for larger files, because it is only performed once when starting the decoding process. Therefore, its influence can be disregarded for determining which kernel will be sped up. The reason that initialising takes so much time, is because the DAMP platform has certain limitations in communication speed to memory. It is expected that on future versions of the DAMP platform the time needed for the initialisation is less noticeable. After the initialisation function, the colourspace conversion and the IDCT directly followed with respect to time usage. When disregarding the initialisation function, the

colourspace conversion and the IDCT are considered the most time consuming, like on the PC. Section 5.3 will explain how the decoder will be sped up.

5.3 Speeding up the codec

Tests on the ARM platform when skipping the IDCT function show considerable speedup. So the IDCT is responsible for a large part of the time on ARM as well. For the colourspace conversion, the same argument holds valid. When colourspace conversion is disabled in XviD, the decoding speeds up considerably. These speedups indicate the maximum obtainable speedup. The frame decoding times have been acquired by averaging the decoding times of four reference video files and outputting them using the VGA controller. This has been done several times for different settings. The results are summarised in table 5.1.

The speedup is defined as the factor between the old computation time and the new

SW IDCT enabled	SW colourspace conversion enabled	Time per frame	maximum speedup
Yes	Yes	445.9 ms	1.0
No	Yes	341.1 ms	1.31
Yes	No	362.6 ms	1.23
No	No	261.5 ms	1.70

Table 5.1: Average decoding time of MPEG-4 files while selectively disabling the IDCT or the colourspace conversion and maximum obtainable speedup

computation time. It can be calculated by dividing the old computation time by the new computation time. If the new computation time is smaller, the speedup is larger than one. It can be formulated as

$$S = \frac{T_{original}}{T_{accelerated}} \tag{5.1}$$

which is a simplified version of Amdahl's law for speedup due to parallelising. When disabling only the colourspace conversion, a speedup of 1.23 is the maximum, while disabling only the IDCT would generate a maximum speedup of 1.31. This document will describe the design of hardware to speed up the IDCT. The reader is referred to [13] for the design of the hardware to speed up the colourspace conversion.

The chosen XviD codec will be enhanced to make use of the re-configurable hardware available on the DAMP platform. A certain type of MPEG-4 decoding of low resolution video on ARM-based devices, such as several mobile phones, has already been done without the use of reconfigurable hardware [1]. One is to expect that the DAMP implementation will be able to decode low resolution video as well. It is conceived that re-configurable hardware can be used to speed up similar devices which are using MPEG-4. Preliminary tests show that this version of the DAMP platform is not able to decode MPEG-4 video in real time. The current version of DAMP accesses 16 bit words from memory at 12.5 MHz. Frame-rates of the decoded files vary between 3 to 5 fps (frames per second). On hand held devices a frame-rate of 10 to 15 fps is acceptable. Future

versions of the DAMP platform should be able to access the memory at several times higher speeds. Therefore real-time decoding of video files on DAMP is expected to be feasible for mentioned frame-rates. The design of the hardware to aid in the calculation of the IDCT is described in chapter 6.

6 IDCT hardware design

As stated in section 5.2, the Inverse Discrete Cosine Transform (IDCT) is responsible for a considerable part of the decoding time of the MPEG-4 video. Therefore implementing it in hardware (an FPGA) could support the MPEG-4 decoding by performing the IDCT faster. The IDCT kernel will be implemented in hardware as a co-processor. The hardware kernels will be designed as fixed kernels instead of using an arbiter to intercept instructions meant for the hardware, like in the MOLEN paradigm [25].

In section 6.1, the IDCT algorithm is described. The separation between hardware and software will be described in section 6.2. The considerations for the implementation of the IDCT are made in section 6.3. Section 6.4 will elaborate on the performance of the hardware. The modifications made to XviD are described in section 6.5. The design of the colourspace conversion hardware kernel and considerations on using an arbiter have been described in [13].

6.1 Inverse Discrete Cosine Transform

In this section the IDCT algorithm used in XviD is explained. The C source code implementation of this algorithm can be found in appendix A. The IDCT is the inverse operation of the Discrete Cosine Transform (DCT). The 2-dimensional DCT [26] is shown in equation 6.1.

$$S_{uv} = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \sum_{y=0}^{7} s_{xy} \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16} \tag{6.1}$$

with C_u,$C_v = \frac{1}{\sqrt{2}}$ for u, $v = 0$; C_u,$C_v = 1$ otherwise.
This equation can be divided into two parts. These parts are performed after each other, consisting of eight 1-dimensional DCT operations. Such a 1-d DCT operates on each column with y constant and on each row with x constant. The 1-d DCT can be rewritten into a matrix multiplication. The inverse operation, the IDCT, can then be obtained by inverting the matrix. This inverted matrix can be factored into multiple sparse matrices. Because the matrices are sparse[1], these subsequent matrix multiplications can be replaced by a small amount of scalar operations. This way the following sequence of operations results, as documented in [16], with F_0 through F_7 the values of each row or column before the IDCT operation and f_0 through f_7 the values resulting from the

[1]Sparse matrices contain few non-zero elements. In sparse matrix multiplications most operations consist of multiplying by zero, leaving only a few relevant operations.

operation.

$$a_0 = \frac{1}{16}F_0, \quad a_1 = \frac{1}{8}F_4, \quad a_2 = \frac{1}{8}F_2 - \frac{1}{8}F_6, \quad a_3 = \frac{1}{8}F_2 + \frac{1}{8}F_6 \tag{6.2}$$

$$a_4 = \frac{1}{8}F_5 - \frac{1}{8}F_3, \quad tmp_1 = \frac{1}{8}F_1 + \frac{1}{8}F_7, \quad tmp_2 = \frac{1}{8}F_3 + \frac{1}{8}F_5 \tag{6.3}$$

$$a_5 = tmp_1 - tmp_2, \quad a_6 = \frac{1}{8}F_1 - \frac{1}{8}F_7, \quad a_7 = tmp_1 + tmp_2 \tag{6.4}$$

followed by

$$b_0 = a_0, \quad b_1 = a_1, \quad b_2 = a_2C_4, \quad b_3 = a_3 \tag{6.5}$$

$$b_4 = -(a_4C_2 + a_6C_6), \quad b_5 = a_5C_4, \quad b_6 = -(a_4C_6 + a_6C_2), \quad b_7 = a_7 \tag{6.6}$$

with C_2,C_4 and C_6 constants, and

$$tmp_3 = b_0 - b_7, \quad n_0 = tmp_3 - b_5, \quad n_1 = b_0 - b_1, \quad n_2 = b_2 - b_3, \tag{6.7}$$

$$n_3 = b_0 + b_1, \quad n_4 = tmp_3, \quad n_5 = b_4, \quad n_6 = b_3, \quad n_7 = b_7 \tag{6.8}$$

after which

$$m_0 = n_7, \quad m_1 = n_0, \quad m_2 = n_4, \quad m_3 = n_1 + n_2, \quad m_4 = n_3 + n_6 \tag{6.9}$$

$$m_5 = n_1 - n_2, \quad m_6 = n_3 - n_6, \quad m_7 = n_5 - n_0 \tag{6.10}$$

and finally

$$f_0 = m_4 + m_0, \quad f_1 = m_3 + m_2, \quad f_2 = m_5 + m_1, \quad f_3 = m_6 + m_7 \tag{6.11}$$

$$f_4 = m_6 + m_7, \quad f_5 = m_5 - m_1, \quad f_6 = m_3 - m_2, \quad f_7 = m_4 - m_0 \tag{6.12}$$

Using this 1-d IDCT on each row and after that on each column, the 2-d IDCT can be calculated. These formulas were implemented in XviD as can be seen from the IDCT algorithm in appendix A. The software contains a shortcut in the 1-d IDCT to skip some operations when the 8 input values are zero. The described IDCT algorithm has been transferred to hardware, except for this shortcut. Implementing the shortcut would cost a lot of extra hardware, but it would not make the design faster. This will be explained in section 6.3.

6.2 Hardware software separation

The key in designing software to use reconfigurable hardware is to know where to separate hardware and software. One needs to find, in terms of communication, the connection between different parts of the software where the communication overhead is the least. This will be made clear with the use of an example. Suppose one imagines all functions in a piece of software to be connected together by the data they interchange. One could then distinguish the connections through which the functions communicate and call these connections links. Those links differ from each other in terms of the amount of data that is interchanged through the links. This is graphically depicted in figure 6.1. A link will

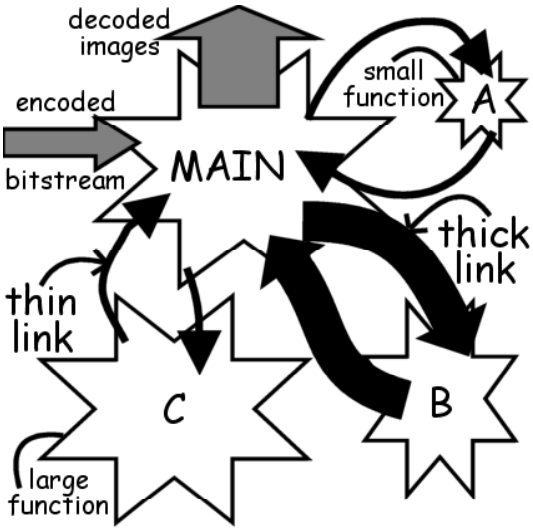


Figure 6.1: Thinnest links in terms of communication between functions.

be called *thick* when a lot of data is sent through it and *thin* when a limited amount of data is sent through it. This classification is utilised to order the different links.

The likely locations to make the hardware software separation are on those thin links.When there are several thin links, the best approach would be to choose the largest part that fits the hardware. Depending on the size of the reconfigurable hardware, larger parts of the software can be put into hardware.

When device size is limited, large parts of software cannot be put into hardware as a whole. This produces sub-optimal results compared to when the largest part with the smallest communication overhead is implemented in hardware. In our case, the part that will be put into hardware is determined by several properties. For instance its weight in execution time, where a large weight means the function is responsible for a large part of the execution time. Keeping that in mind, the part that is to be put into hardware needs to fit into the device. The part with the least communication overhead is to be found. The smallest amount of data would have to be transferred, when the encoded bitstream would be sent directly to the hardware. In figure 6.1 that would be the case if the function "main", and all the functions in it, were to be put in hardware entirely. Then if the hardware would also control the VGA connection, data would only have to be sent back to the decoder if the decoded frames are to be stored on the DAMP board. Naturally putting the entire decoding of the bitstream in hardware would require much more hardware than only computing the IDCT and it would become too large to be implemented. Therefore the function expected to have the greatest speedup, which still fits into the FPGA has to be implemented. Depending on the size of the FPGA different functions could be valid candidates.

The relative amount of data to be transferred to hardware for each frame generally becomes less when the function implemented in hardware becomes more complicated. Also there is an upper limit on the amount of data that can be transferred to hardware at

once, depending on the way the hardware and software communicate. This will become clear in section 6.3.1. As explained in that section, the size of the DPRAM will be the factor limiting the amount of data that can be transferred.

In this case, it turns out that the IDCT and the functions calling it are responsible for a large part of the execution time. The IDCT itself has such a large size in hardware that implementation of other large functions, besides the colourspace and IDCT, in hardware as well is not an option. This will become clear in section 6.3 where the IDCT datapath is designed.

6.3 Hardware implementation

When implementing the IDCT in hardware to aid the software to perform its function faster, several considerations have to be made. This section will describe the design of the IDCT hardware accelerator for the modified XviD codec.

First, the accelerator should enable the software to function as it did without hardware, showing no noticeable differences. The hardware IDCT should return the same values as would the software version. This can be tested by using a large amount of real test values and comparing the result of both implementations. Secondly, the IDCT should fit in the available hardware.

The designed IDCT hardware will be combined later with another piece of hardware to be able to control the VGA periphery and speed up the colourspace conversion [13]. This piece of hardware will be designed to take up less then 25% of the FPGA thus leaving over 75% for the IDCT hardware. Of course another constraint is that the hardware should speed up the entire calculation. The hardware should be able to finish and give back the results to the software, in a much shorter time than originally needed for the software. When designing the hardware, the amount of time needed for communication with the software should be kept as small as possible. Subsection 6.3.1 will decide on the way to communicate between hardware and software.

6.3.1 Hardware and software communication

The hardware part should remain functionally equivalent to the software, without noticeable differences (except for speed). It should fit in the available reconfigurable hardware and it has to be combined with the hardware to speed up the colourspace conversion. Apart from these constraints, the overhead in time needed for communication between the hardware and the software should be kept as small as possible.

To keep the communication overhead low, an efficient communication method is required. One IDCT requires 128 bytes to be exchanged[2] between hardware and software. In order to speed up the calculation, the transfers of the data and the IDCT operation in hardware need to be faster than the original software. In each clock cycle the maximum possible amount of data needs to be exchanged to keep the communication penalty low.

When exploring the ways for the ARM CPU to communicate with the FPGA, several

[2]Before the IDCT operation, 128 bytes need to be transferred to the hardware and when the IDCT has been calculated, 128 bytes should be transferred back.

options are to be evaluated. There is one GPIO (General Purpose Input and Output) register available for communication between the CPU and the FPGA. However, on this version of the Excalibur, only 4 output and 4 input pins are available, making this method of communication unattractive for large amounts of data, since only 4 bits can be sent each cycle. In this case transferring 256 bytes would take 512 cycles supposing each transfer uses one cycle (which actually will be more because of handshaking).

Another way of communicating with the hardware would be to use the AMBA bus and make the hardware in the FPGA a slave on that bus. The AMBA bus is a high speed 32 bit bus available in the Excalibur for connecting different devices. There is one AMBA bus, the AHB1 bus (AMBA High-speed Bus 1), operating at the same speed as the ARM CPU and another one, the AHB2, operating at half that speed. For communication, the CPU could use the AHB1 and connect to the AHB2 via a bridge and then connect to the FPGA. However, this would require the IDCT hardware to be able to communicate via the AMBA bus, causing extra overhead and increasing the size of the FPGA design. Apart from that, an option using only the high speed AHB1 bus instead of both the AHB1 and the AHB2 bus would be preferred.

A third option would be to use a DMA controller inside the FPGA to copy the data needed for the IDCT from memory accessible by the CPU, into a local memory on the FPGA. However this inhibits the same contra arguments as does the previous option. In this case as well, the AHB2 bus would be used for communication of the DMA controller in the FPGA with the memory. This would cause extra overhead on the communication and on the hardware size.

The best remaining option would be to use the available dual-port RAM (DPRAM), which is accessible by the CPU as well as by the FPGA. It does not require extra hardware inside the FPGA and it does not cause much communication overhead for the hardware as well. The CPU communicates with the DPRAM via the AHB1 bus, as it does with all periphery. In that respect, this is not a disadvantage compared to the other options. The FPGA has direct access to the DPRAM. A limitation of this approach could be the limit on the amount of data that can be transferred each time. Fortunately this is not a limitation, since for one IDCT only 128 bytes need to be exchanged.

The DPRAM is 16 Kbytes, so it is large enough for the data of 128 IDCT kernels. One IDCT will be performed each time, after which the results will be needed for the continuation of the MPEG-4 decoder. If in the future one would change the behaviour of the MPEG-4 codec to enable the pipelined computation of several IDCTs, this approach would enable one to store the results of 128 IDCTs (16 Kbytes) simultaneously in the DPRAM.

Pipelining the total IDCT operation to have the new input for the next IDCT written to DPRAM while calculating the current IDCT has two prerequisites. First, it would require the IDCT hardware to perform its operation within the time needed for the software to put the next 128 bytes in the DPRAM. In that case the time needed for communicating the data would be less than the time needed for the hardware to finish computing. As will become clear in section 6.4 this is not the case. Secondly, it would require the existence of multiple IDCT cores in the FPGA to handle the larger amount of data, which would in turn require a larger FPGA than the one currently available on

the DAMP board. The design of the IDCT hardware will be explained in subsection 6.3.2.

6.3.2 IDCT hardware implementation

In the software the IDCT is calculated by first performing an operation on each row of the 64 value matrix and then performing an operation on each column of the resulting values. The row operations can be performed independently of each other, but are performed before the column operations. The column operations can each be performed independently as well. When examining the code used by the MPEG-4 decoder to perform the IDCT, one can recognise there are only subtle differences between the operations performed on the rows and those performed on the columns. This gives rise to the possibility of designing a piece of hardware capable of performing a row operation or a column operation when requested to do so. Depending on the size of such a functional unit, control logic could use one or more of them to compute the IDCT.

A functional unit capable of computing only mentioned row operation was designed in hardware using Quartus II from Altera. It would take up about half of the available 4160 cells in the FPGA. Therefore the idea of creating more than one IDCT core to work in parallel was abandoned. When augmenting the functional unit to perform the column operation as well, about two third of the FPGA was covered. The number of cells used for the unit operating on rows or columns only and for the unit selectively operating on rows or columns is depicted in table 6.1.

Type of unit	number of cells in the FPGA	percentage of the chip
row-only unit	2030 cells	48%
column-only unit	2361 cells	56%
row/col unit	2771 cells	66%

Table 6.1: Cell usage of the EPXA1 device for the row-only, column-only and row/col unit

In figure D.1 in appendix D the design of this hardware piece is depicted. This functional unit will be called the row/col unit from now on. Using this row/col unit additional control logic is needed to load the appropriate row or column into the unit and to store the results of the row operations to be used by the column operations before writing back the results. The FPGA is capable of reading 32 bits at a time out of the DPRAM. One row contains 8 values of 16 bits, so four read operations from the DPRAM are needed before the row operation can be performed. Initial simulation tests of the row/col unit showed that results of the operation are available within 100 ns. In order to have the next 16 bytes of data ready in time, the four read operations from the DPRAM need to be performed in 100 ns to keep the row/col unit busy, since one does not want the unit to be idle.

Tests show that a 32 bit word from the DPRAM can be read out by the hardware in one clock cycle when operating at 50 Mhz. Initial designing was done for 40 Mhz which was expected to be just enough to keep the row/col unit computing all the time. Further testing shows that the row/col unit has less latency than earlier simulations indicated.

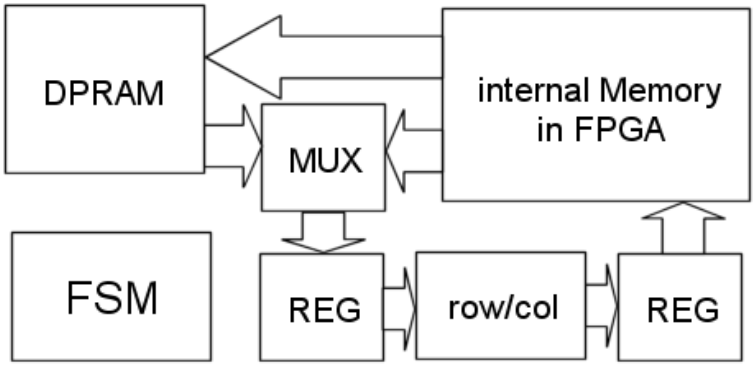


Figure 6.2: Datapath of the implemented IDCT with row/col unit.

Instead of after 100 ns the unit is ready after 75 ns. After verifying the correct operation at 40 MHz, the design has been setup to operate at 50 MHz. This frequency is the highest at which the design still fits into the device. Quartus is unable to fit the design for clock frequencies higher than 50 MHz, because higher frequencies imply stricter limitations on routing. Section 6.3.3 will elaborate on this.

The values the row/col unit is operating on are stored in registers to pipeline the design. Pipelining enables the new values to be made available to the row/col unit at the same time as the previous results are stored. The results from the row operations are stored into an internal memory on the FPGA and later fed back into the row/col unit to perform the row operations on. After each column operation, the results are stored back into the DPRAM. When the IDCT is finished, the CPU should be notified. For resetting the IDCT hardware and for notifying the CPU when the IDCT has completed, the GPIO register will be used. Using the GPIO enables the hardware to get a logic signal from the CPU directly. No extra cycle is needed to read a value from the DPRAM at the start and no extra hardware is needed to convert this value into a start or reset signal. The signal from the GPIO can be used directly to signal a reset. The data exchange will use the DPRAM, because it can be read out 32 bits at a time, but for signalling the GPIO register is used, as explained in subsection 6.3.1.

The designed datapath and the control logic (the "FSM" block) are shown in figure 6.2. In this figure the FSM is a finite state machine, written in VHDL (a Hardware Description Language used by Quartus), to control the registers and the communication with the DPRAM. In the depicted datapath data flows through the row/col unit 16 times. First 8 row operations are performed, secondly 8 column operations. In order to do this, the FSM sets the correct address for the DPRAM and enables the registers to load the data coming from the DPRAM. After 4 reads from the DPRAM a second stage of registers is enabled to load in the 4 words from the first stage at once. While the row/col unit is computing the IDCT of the read values, the next input values will simultaneously be read into the registers of the first stage. When the row/col unit is finished, the FSM enables the registers at the output of the row/col unit to read in, simultaneously with the second stage of registers at the input of the row/col unit. The row/col unit is now calculating the IDCT of the second row, while the FSM is controlling

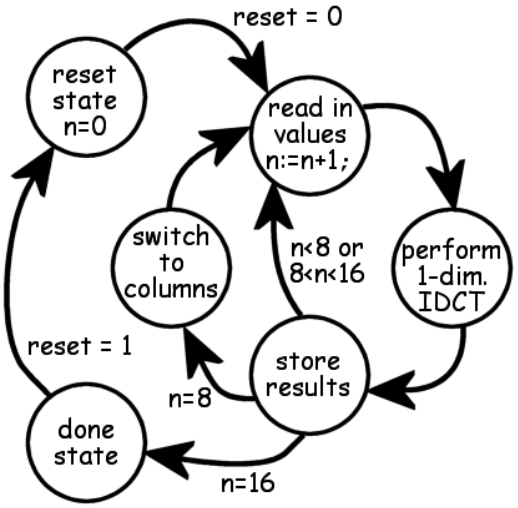


Figure 6.3: States of the FSM controlling the IDCT datapath.

the internal DPRAM to store the values from the register at the output. This process is repeated for the other rows. When the IDCT of the columns needs to be calculated, the input of the first stage of registers is switched to read from the internal DPRAM instead of the normal one.

Further on the FSM roughly repeats the steps previously mentioned, but this time the row/col unit is operating on the previously calculated values coming from the DPRAM internal to the FPGA. Finally after each row/col operation the values are now written from the internal DPRAM to the normal DPRAM. A bit in the GPIO register is set by the FSM to signal that the IDCT has been calculated. It should be noted that the described process takes up much less time than the communication time from the software to the DPRAM. The hardware IDCT will be ready before data for a next IDCT would be written to the DPRAM. As long as the communication takes more time than the calculation, only one core will be needed. Even when pipelining IDCT operations the communication would still keep the row/col unit waiting for data.This will become clear in section 7.2. The previously described operation of the FSM is depicted in figure 6.3. All the signals generated by the FSM for controlling the datapath can be found in figure D.4 in appendix D.

As mentioned in section 6.1, the software contains a shortcut to deliver the values of the 1-d IDCT on a row or column faster. This would not make sense for the hardware version, because it would require much more hardware, but the result would be the same. The hardware would not benefit from the row/col unit being ready earlier, since new input values are only available every 80 ns with a 50 MHz clock signal. Even if the IDCT hardware could have been sped up by the shortcut, it would not be implemented for two reasons. First, the hardware would become much bigger, while the IDCT and the colourspace conversion only barely fit without it. Secondly, the time needed for the hardware IDCT is very small compared to the time needed for the communication between hardware and software.

Section 6.4 will go into detail on the performance of the described datapath. The design of the hardware for speeding up the colourspace conversion and for controlling the VGA interface is described in [13]. The IDCT hardware will be combined with the hardware for the VGA driver and the colourspace conversion to fit together in the FPGA. The fitting of the design is explained in subsection 6.3.3.

6.3.3 Fitting the design into the FPGA

The designed IDCT hardware had to be combined with the hardware for the colourspace conversion and the VGA controller. Both hardware pieces have been designed separately. The design of the colourspace conversion hardware is presented in [13]. The IDCT hardware used 3218 logic cells of the available hardware and the colourspace hardware and VGA controller used 1284 logic cells. The total amount of available logic cells is 4160. It therefore seems that the two designs cannot be combined. However, the Quartus tools can try and merge logic cells from the two designs together when possible. Quartus can be configured to optimise the design for area or speed. Optimising for speed creates a design that should work, but is too large to fit into the EPXA1 device. Optimising for area creates a design that is small enough, but several signals cannot be routed to meet the speed requirements of the 50 MHz clock signals. The global settings used to configure Quartus optimisation are as stated below. Global optimisation has been set for speed. The settings which differ from the standard settings are listed.

- Optimisation technique set to speed.
- Auto global memory control signals set to on.
- Auto implement in ROM set to on.
- Auto packed registers set to minimise area.
- Remove redundant logic cells set to on.

Different parts of the design have been set to be optimised for area, overruling the global settings to optimise for speed. The global setting for speed optimisation has been turned on to maintain correct system functionality, but for each part of the design where speed was not critical, optimisation for area has been set.

Configured this way, the Quartus tools were able to compile and fit the design into the EPXA1 device. However, the routing of all the signals still was a problem for the tools, because of the requirements for clock speed. When evaluating the clock signals, one signal was used to drive several registers. These registers loaded the result of the row/col unit to pipeline the operation, as mentioned in subsection 6.3.2. The speed of the clock signal driving these registers was originally set to 80 MHz. This was because it was expected that changes at the input of the row/col unit would propagate quickly to the output. Tests have shown that the outputs of the row/col unit remain as they were for over 20 ns, so the constraint could be relaxed. Therefore the clock signal for the pipeline registers of the row/col unit could operate at 50 MHz. A 50 MHz clock had been used for the colourspace conversion as well, so the same clock signal could be used. This loosened up the constraints and enabled the Quartus tools to generate a working

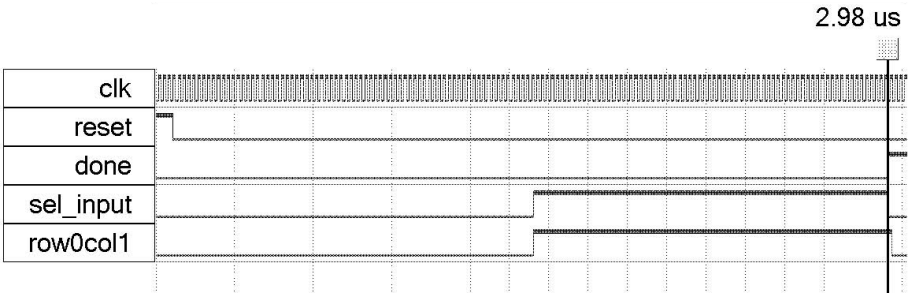


Figure 6.4: Simulation of several signals from the FSM controlling the IDCT datapath.

design containing both the IDCT hardware and the VGA controller with the colourspace conversion.

6.4 Hardware performance

The FSM controlling the IDCT hardware cycles through a similar sequence of events 16 times. First the row operations have to be performed on the input data, 8 rows after each other, then the 8 column operations have to be performed. The waveform showing all the signals originating from the FSM is shown in figure D.4 in appendix D. Figure 6.4 shows the `reset`, the `row0col1`, the `sel_input` and the `done` signal. As one can see from the signal `row0col1` the first half of the operations is on the rows and the second half is on the columns. The signal `sel_input` is a similar signal. It controls the multiplexer from figure 6.2 to select whether to read from the DPRAM or from the internal memory. The DPRAM contains the values to perform the row operations on and the internal memory contains the results of that operation, on which the column operation is to be performed.

The cycle time for one full IDCT operation can be derived from figure 6.4. The time required from the moment the input values are available until the output values are ready, is 2.98 μs. This is the time for the hardware IDCT without the time required for the software to store the values in the DPRAM and to read out the results. Therefore time for communicating with the software will have to be added. On this version of the DAMP board the maximum clock speed of the IDCT hardware is 50 Mhz, because of the speed grade 3 of the Excalibur device. In future designs a higher clock frequency could shorten the cycle time of this FSM, but even more important, on future versions of the DAMP board the communication overhead would be less. The time needed for exchanging the results on this version of DAMP is far longer than the time for the hardware to calculate the IDCT. The total cycle time for the IDCT in software compared for the one using hardware will be stated in chapter 7 which will discuss the results of the implementation in hardware.

6.5 XviD modification

The original source code for the calculation of the IDCT is shown in appendix A. The XviD code has been modified to enable the user to select whether to calculate the IDCT using hardware or to use the software only approach. The initialisation function `dec_init()` of the XviD decoder initialises the settings that are to be used for decoding. A global variable was added to tell the IDCT function whether or not to use the hardware. The piece of source code calling the IDCT function in XviD is shown below. The entire source code of the decoder is listed in appendix E.

```
start_timer();
if (USE_HARDWARE) {
      us_hardware_idct_ioctl(data,data); //the added function
       //copying the values to the DPRAM
       // and reading out the results:
       // us_hardware_idct(destination,source);
  } else {
      idct(data);
  }
stop_idct_timer();
```

The `start_timer()` and `stop_idt_timer()` functions can be used to measure the time spent in the IDCT. They are normally disabled when decoding. The function `us_hardware_idct_ioctl()` uses the Linux driver to communicate with the hardware. As can be seen the `if` statement determines whether or not to use the hardware. The global variable `USE_HARDWARE` has been set in the `dec_init()` function as explained in section 4.4. The driver for the IDCT is a modified version of the VGA driver. The VGA driver uses the SRAM to store images to be displayed, while the IDCT driver uses the DPRAM to store values on which the IDCT is to be performed and uses the GPIO registers to control the IDCT hardware. The design of Linux drivers is described in [13].

7 Experimental results

The test-results described in section 5.3 showed the theoretical maximum speedup when using infinitely fast hardware accelerators. The current chapter will describe the methods applied for measuring the actually obtained speedup. Section 7.1 will explain the methods used to measure the change in speed of the decoding process. Section 7.2 will elaborate on the results obtained by using hardware acceleration and compare those results with the theoretical maximum speedups.

7.1 Measuring methods

To measure the performance influence of the implemented hardware IDCT and colourspace conversion on XviD, the program `xvid_decraw` discussed in section 4.4 has been modified. It measures the time needed for the decoding and displaying of each video frame. The decoding time of the "software only" version of XviD has been measured, as shown in section 5.3. After implementation of the IDCT and the colourspace conversion in hardware, similar measurements have been performed with hardware acceleration turned on.

Four MPEG-4 test sequences have been generated using files available from the Technical University in Munich [10]. We will use these test sequences as benchmarks in this section. The MPEG test sequences have been used to measure the time needed for different configurations of XviD to decode the video content. Six different configurations have been used to decode each of the four files.

The first and most straightforward configuration is the software only version of XviD running on DAMP. In the second configuration, the call to the IDCT function is intentionally skipped to acquire the theoretical maximum possible speedup for the IDCT. Such a speedup would be obtained in a hypothetical case of using infinitely fast hardware with no software-hardware communication overhead. For this configuration, the colourspace conversion was performed in software. The maximum speedups shown in section 5.3 are depicted in table 7.1. The third configuration is the one using the hardware

SW IDCT enabled	SW colourspace conversion enabled	Time per frame	maximum speedup
Yes	Yes	445.9 ms	1.0
No	Yes	341.1 ms	1.31
Yes	No	362.6 ms	1.23
No	No	261.5 ms	1.70

Table 7.1: Average decoding time of MPEG-4 files while selectively disabling the IDCT or the colourspace conversion and maximum obtainable speedup

kernel to calculate the IDCT, still with the colourspace conversion in software. From the measurements with these three configurations, the time needed for the software and hardware IDCT operations will be derived by subtracting the time for XviD with "no IDCT" from the two remaining time measurements.

Apart from the mentioned three measurement configurations (software IDCT, hardware IDCT and "no IDCT") three similar measurements have been performed, with the colourspace conversion performed in hardware instead of software. The results of the above six different configurations will be compared to draw conclusions for the total speedup due to the hardware accelerations. For details about the colourspace conversion hardware implementation the reader is referred to [13]. Section 7.2 will discuss the results of the measurements and explain the derivation of the time required for a single hardware and software IDCT calculation.

7.2 Acquired results

The measurement results from section 7.1 are discussed in subsection 7.2.1. The speedup of the total application is calculated from these measurements in subsection 7.2.2. From the measurements on the total application, the operation time of the IDCT kernel implementation is derived in subsection 7.2.3. In subsection 7.2.4 the expected speedups on future DAMP versions are discussed.

7.2.1 Measurements

As described in section 7.1, six different configurations have been used to decode four MPEG-4 test sequences. In all six cases the video frames are decoded and displayed using the VGA controller. Figure 7.1 shows the decoding time per frame for the decoded reference files. The leftmost three columns indicate the times needed for XviD using software colourspace conversion while varying the IDCT configuration. For the right three columns, the hardware colourspace conversion was used while varying the IDCT configuration. Each reference file contained 20 frames of MPEG-4 video. Differences in decoding time between the reference files are due to each file requiring a different number of IDCT operations. The average decoding time per frame is stated in table 7.2.

7.2.2 Speedup calculation

The average decoding time for a frame as shown in table 7.2 can be used to calculate the average speedup compared to the software only approach. The average speedups compared to the configuration using only software, can be calculated resulting in the values which are graphically depicted in figure 7.2. For the calculation of the speedup a simplified version of Amdahl's law for speedup due to parallelising is used. The speedup is defined as the factor between the old computation time and the new computation time. It can be formulated as:

$$S = \frac{T_{original}}{T_{accelerated}} = \frac{445.86}{396.65} = 1.12 \tag{7.1}$$

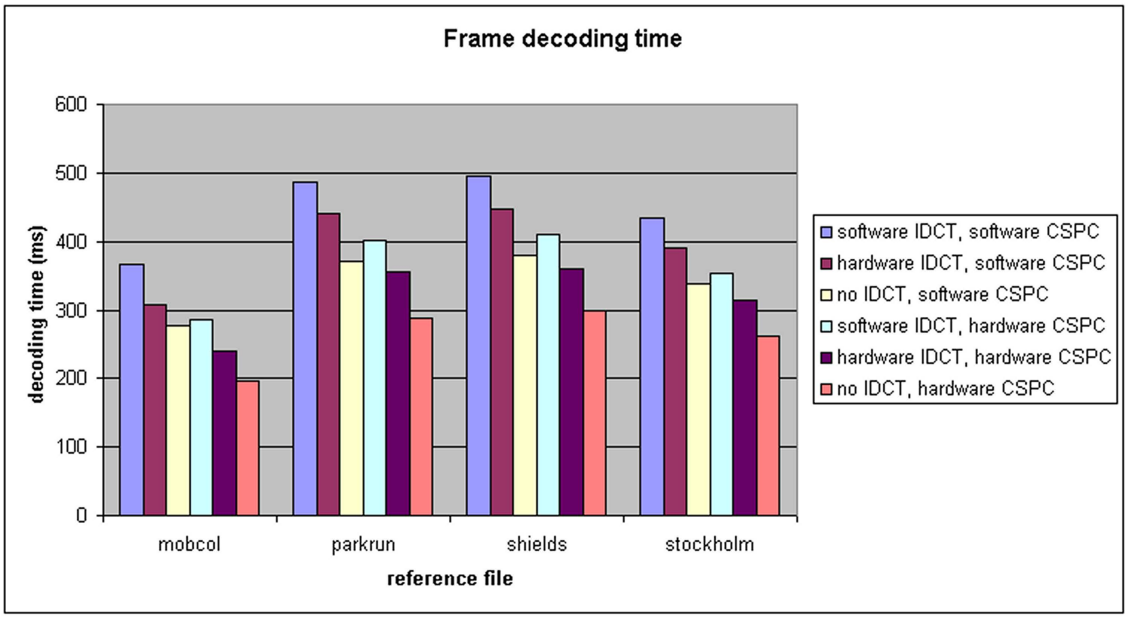


Figure 7.1: Average decoding time per frame for four reference MPEG-4 files measured for six different configurations of IDCT and colourspace conversion (CSPC).

configuration	total time per frames	
SW IDCT, SW colourspace	445.86 ms	using IDCT
SW IDCT, HW colourspace	362.64 ms	in software
no IDCT, SW colourspace	341.07 ms	the upper
no IDCT, HW colourspace	261.53 ms	limit
HW IDCT, SW colourspace	396.65 ms	using IDCT
HW IDCT, HW colourspace	317.93 ms	in hardware

Table 7.2: Average decoding time per frame

Thus a speedup of 1.12 for the application due to the hardware IDCT is obtained.

7.2.3 IDCT operation time

The speedup of 1.40 from figure 7.2 for the total decoding process is the one actually obtained when performing both IDCT and colourspace conversion in hardware. The six measured values from table 7.2 are the times needed to decode one frame. Please note that for each frame several IDCT operations are required. Comparing the frame decoding time with "no IDCT" and the time with software IDCT gives the time needed for the software IDCT operations in a frame. Similarly comparing the frame decoding time using the hardware IDCT and the one using "no IDCT" yields the time needed for the hardware IDCT operations. This results in the values of table 7.3 for the time spent in the IDCT function when decoding one frame. Please note that these values will be

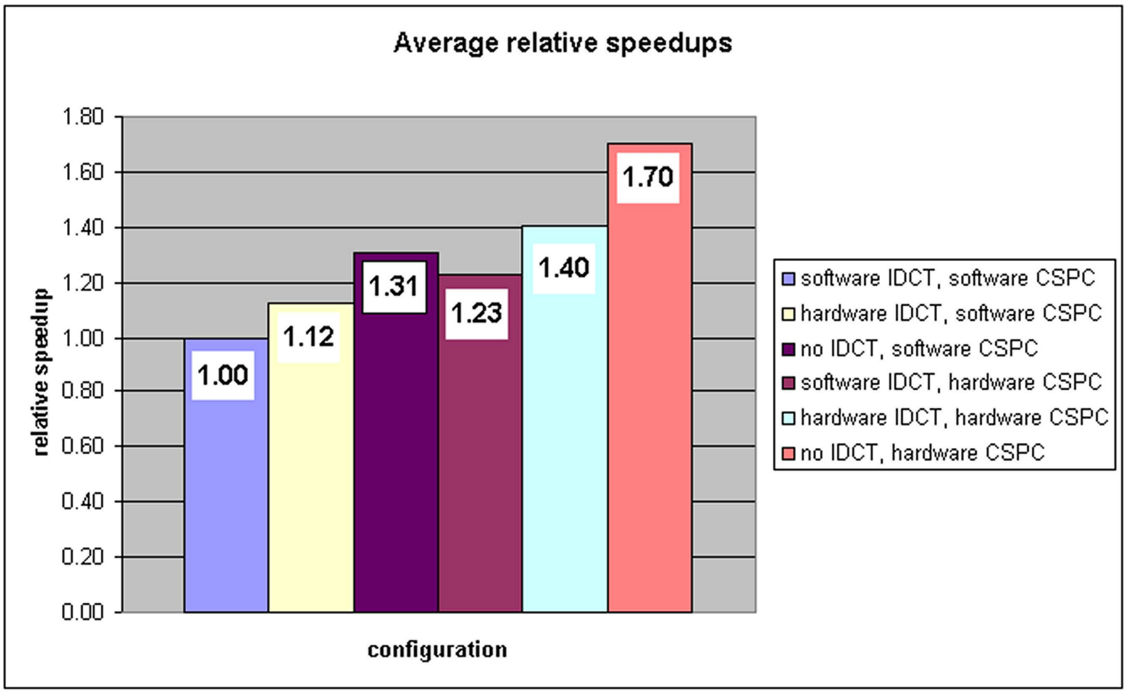


Figure 7.2: Average relative speedups for six different configurations of IDCT and colourspace conversion (CSPC).

used in subsection 7.2.4 to predict future speedups. By dividing the IDCT operation time per frame from table 7.3 by the number of IDCTs for each frame, the time of a single IDCT kernel execution can be calculated.

	IDCT operation time per frame
using software IDCT	103 ms
using hardware IDCT	56 ms

Table 7.3: Average IDCT operation time per frame.

The total average software and hardware IDCT time can now be calculated as well as the time needed for the communication with the hardware. The time needed for the hardware operation is known from the simulation in section 6.4 and is 2.98μs. The measured operation time of the hardware IDCT is 180μs. Therefore the communication time from software to hardware and back is determined to be 177μs. One software IDCT operation uses 334μs. The speedup can now be calculated:

$$S = \frac{T_{original}}{T_{accelerated}} = \frac{334}{180} = 1.86 \tag{7.2}$$

These values are stated in table 7.4. The colourspace conversion has been accelerated by a factor of 3.59 as explained in [13].

	Total average time per IDCT
SW IDCT total operation time	334 μs
HW IDCT total operation time	180 μs
hardware kernel time	2.98 μs
hardware communication time	177 μs
Average speedup of the IDCT kernel	1.86

Table 7.4: Total average time per IDCT for software and hardware and calculated speedup of the kernel for each file.

7.2.4 Improving hardware IDCT speed

As calculated in subsection 7.2.3 the total application has a speedup of 1.12 when the accelerated IDCT kernel is used. The upper limit for the speedup of the total application is 1.31 when the IDCT kernel would be infinitely fast. This leaves a margin for the IDCT kernel to be improved. Currently one total hardware IDCT operation requires 180 μs as indicated in table 7.4. The communication time for this hardware IDCT to load the input values and to store the results was determined to be 177 μs. It is expected that this communication time can be decreased on future versions of the DAMP platform. The reason that the processor needs so much time to present the values to the hardware is because it needs to read these values from the SDRAM. The current communication speed to the external SDRAM is 12.5 MHz. On future versions it is expected that this speed can be increased to 133 MHz. In the most optimal case the communication time of 177 μs would therefore be decreased to less then 17 μs. Adding the 2.98 μ needed for the hardware excluding communication time, this would make the total hardware operation time 20 μs. The speedup of the IDCT kernel in this case would become:

$$S = \frac{T_{oldhardware}}{T_{fasterhardware}} = \frac{180}{20} = 9 \tag{7.3}$$

Using the values from table 7.3 we can predict what the time needed for the hardware would be with this speedup. The time the application spends performing IDCT functions for one frame, is expected to decrease 9 times from 56 ms to 6.2 ms, which is 49.8 ms less than before. Therefore it is expected that the total average decoding time per frame will decrease with 49.8 ms as well. We will use this value for predication of the future speedups. Table 7.5 shows these expectated values.

In subsection 7.2.3 the speedup of the application due to the IDCT was calculated as:

$$S = \frac{T_{original}}{T_{accelerated}} = \frac{445.86}{396.65} = 1.12 \tag{7.4}$$

The total operation time of the application using IDCT hardware was 396.65 ms. When indeed the hardware operation can save 49.8 ms due to less communication overhead, the time needed for the application would be 347.05 and the following speedup would be

	IDCT operation time per frame
currently using software IDCT	103 ms
currently using hardware IDCT	56 ms
expected faster hardware IDCT	6.2 ms
time saved per decoded frame	49.8 ms

Table 7.5: Average IDCT operation time per frame and predicted future values.

obtained:

$$S = \frac{T_{original}}{T_{fasterhardware}} = \frac{445.86}{347.05} = 1.28 \tag{7.5}$$

This speedup of 1.28 would be very close to the upper limit of 1.31 for the speedup. Of course experiments on future DAMP versions would be required to confirm this. On faster versions of the DAMP platform, the software IDCT will also become faster. The time required for the software IDCT depends on the speed of the SDRAM and on the processor speed. The processor speed can be increased by a factor of 1.5. The speed of the SDRAM communication however would increase with a factor of over 10. It is expected that a large part of the software IDCT depends on the speed of the processor, due to the large amount of computations in the IDCT. The relative speedup of the hardware version will become larger on future versions of DAMP, since it mainly depends on the communication speed to memory. Measurements on future versions will therefore also give more insight in the part of the software which depends on memory speed and the part that depends on processor speed. Chapter 8 will discuss the implications of these results and will give some recommendations for the future.

Conclusions and recommendations

8

As stated in the introduction, increasing clock speed will no longer contribute to processor performance as in the past. Reconfigurable hardware is believed to offer a better way of speeding up applications. The main goal of this project was to prove the usability of reconfigurable hardware to accelerate multimedia applications.
Section 8.1 summarises the main conclusions of this thesis. The main contributions of the project are stated in section 8.2. Section 8.3 concludes by giving several recommendations for future research.

8.1 Summary

Chapter 2 has described the DAMP platform on which the research was performed. An overview of the history of digital video was presented in chapter 3. The MPEG-4 codec used for this project was selected in chapter 4 and the basic library functions of the XviD decoder were explained. Chapter 5 continued by explaining which functions are used by XviD to decode MPEG-4 video. In this chapter the decoder was profiled and the possibilities of speeding up XviD were explored. Chapter 6 described the design of the IDCT hardware to aid in the decoding process. Finally, chapter 7 presented the results obtained when speeding up XviD. As explained in chapter 7, the IDCT kernel has been sped up by a factor of *1.86*. The kernel performing the colourspace conversion was sped up by a factor of *3.59* [13]. Combining these two kernels, for XviD a speedup of *1.40* has been accomplished.

These results show that expanding general purpose processors with reconfigurable hardware can indeed yield a large performance gain for MPEG-4 decoding. Other multimedia applications contain similar operations and are envisioned to benefit from the use of reconfigurable hardware.

8.2 Main contributions

The main contributions of this thesis are:

- Linux has been ported to the DAMP platform to create a software environment suitable for academic research.
- Currently available MPEG-4 codecs have been compared determining the most suitable codec to be used and modified in academic environments.
- The XviD codec has been analysed and profiled to determine the functions that have the largest performance impact when accelerated.

- A fully functional IDCT core in reconfigurable hardware has been designed.
- The XviD codec has been sped up by *1.40* using the available reconfigurable hardware on the DAMP platform to implement software kernels in hardware.

8.3 Recommendations for future research

Analysis of the results in chapter 7 indicates that on future versions of the DAMP board additional performance improvements are to be expected, because of the memory bandwith. Further research has to be performed to verify these expectations. Apart from verifying the performed experiments on improved DAMP versions, the following directions are recommended for future research:

- When a faster communication to memory will be available, the designed IDCT hardware can be pipelined to, reducing the idle time for the hardware.
- Besides that, for larger devices the designed hardware can be expanded with functions operating on the data before and after the IDCT (for instance the quantisation), which could increase performance gain additionally.
- The modified XviD codec can be implemented on other platforms containing reconfigurable hardware. For example the designs from the SMOKE project could be implemented on the Virtex II PRO prototype of the MOLEN project.
- Mobile devices containing ARM processors can be expanded with reconfigurable hardware to implement SMOKE on.

Bibliography

[1] Sony Ericsson Mobile Communications AB, *Sony Ericsson P800 users guide*, November 2002, publication number EN/LZT 108 6040 R1A.

[2] J. Biemond and R.L. Lagendijk, *Digitale signaal codering*, november 1996, coursebook et4089.

[3] P. Bourke, *RGB colour space*, http://astronomy.swin.edu.au/ pbourke/colour/colourspace/, May 1995.

[4] Luis González-Camino Calleja, *P900/P800 comparison table*, 2004, http://www.clubsonyericsson.com/en/products _p900_p800comparison.shtml.

[5] Leonardo Chiariglione, *Short MPEG-1 description*, June 1996.

[6] Altera Corporation, *Excalibur hardware reference manual*, November 2002.

[7] ———, *Introduction to quartus II*, June 2004.

[8] S.D. Cotofana, *Embedded systems lab. course (et3301)*, Lab course in embedded systems design taught at TU Delft.

[9] J. Eilers, *Design of the Delft Altera-based Multimedia Platform*, Master's thesis, Delft University of Technology, October 2003.

[10] Technische Universität München Lehrstuhl für Datenverarbeitung, *Mpeg test sequences*, ftp://ftp.ldv.e/technik.tu-muenchen.de/pub/test-sequences/.

[11] Eric Hamilton, *JPEG file interchange format*, september 1992.

[12] Eugene Hecht, *Optics*, 2nd ed., ch. 5.7, Addison Wesley, 1987.

[13] J. Hofman, *Speeding up MPEG-4 colorspace conversion*, Master's thesis, Delft University of Technology, November 2004.

[14] J. Hofman, G. de Goede, G. N. Gaydadjiev, and S. Vassiliadis, *SMOKE- speeding up MPEG-4 operational kernels on excalibur*, Proceedings ProRISC 2004, November 2004.

[15] CompuServe Incorporated, *Graphics interchange format (tm), a standard defining a mechanism for the storage and transmission of raster-based graphics information*, June 1987.

[16] J.Anders, *The implementation of the 2D-IDCT*, http://rnvs.informatik.tu-chemnitz.de/ jan/MPEG/HTML/IDCT.html.

[17] J. Kent, *The FLIC file format*, Dr. Dobb's Journal **18** (1992), no. 3.

[18] Rob Koenen, *MPEG-4 overview - (v.21 jeju version)*, March 2002.

[19] J. Lyman, *Intel P4 extreme boosts performance off the clock*, TechNewsWorld (2004), http://www.technewsworld.com/story/37762.html.

[20] S. Cotofana J.T.J. van Eijndhoven M. Sima, S. Vassiliadis and K. Vissers, *Field-programmable custom computing machines - a taxonomy* -, september 2002, pp. 79–88.

[21] *A brief history of digital television*, http://www.pbs.org/digitaltv/dtvtech/history.htm.

[22] Charles Poynton, *A technical introduction to digital video*, John Wiley & sons, 1996.

[23] G. Gaydadjiev K. Bertels G. Kuzmanov E.M. Panainte S. Vassiliadis, S. Wong, *The MOLEN polymorphic processor*, november 2004.

[24] ITU Radiocommunication Sector, *Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios*, October 1995, ITU-R Recommendation BT.601-5.

[25] S. Vassiliadis, G. N. Gaydadjiev, K. Bertels, and E. Moscu Panainte, *The MOLEN programming paradigm*, Proceedings of the Third International Workshop on Systems, Architectures, Modeling, and Simulation, July 2003, pp. 1–10.

[26] R. Steinmetz W. Effelsberg, *Video compression techniques*, dpunkt.verlag, 1998.

[27] XviD, *The xvid codec*, http://www.xvid.org.

[28] W. Zwart, *Testing and redesigning the Delft Altera-based Multimedia Platform*, Master's thesis, Delft University of Technology, October 2003.

IDCT source code

A

This appendix contains the C source code of the ICDT as used in XviD. It is available under the GNU General Public License, described in appendix F, as published by the Free Software Foundation. In section 6.1 the IDCT was explained and in section 6.3 the implementation of the algorithm using hardware was designed. The C source code of the IDCT is printed below. When compiled into assembly code the number of instructions varies between 400 and 800, depending on the measure of optimisation of the compiler.

```
/*****************************************************************************
 *
 *  XVID MPEG-4 VIDEO CODEC
 *  - Inverse DCT  -
 *
 *  These routines are from Independent JPEG Group's free JPEG software
 *  Copyright (C) 1991-1998, Thomas G. Lane (see the file README.IJG)
 *
 *  This program is free software ; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation ; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY ; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program ; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307 USA
 *
 * $Id: idct.c,v 1.7 2004/03/22 22:36:23 edgomez Exp $
 *
 ****************************************************************************/

/* Copyright (C) 1996, MPEG Software Simulation Group. All Rights Reserved. */

/*
 * Disclaimer of Warranty
 *
 * These software programs are available to the user without any license fee or
 * royalty on an "as is" basis.  The MPEG Software Simulation Group disclaims
 * any and all warranties, whether express, implied, or statuary, including any
```

```
 * implied warranties or merchantability or of fitness for a particular
 * purpose.  In no event shall the copyright-holder be liable for any
 * incidental, punitive, or consequential damages of any kind whatsoever
 * arising from the use of these programs.
 *
 * This disclaimer of warranty extends to the user of these programs and user's
 * customers, employees, agents, transferees, successors, and assigns.
 *
 * The MPEG Software Simulation Group does not represent or warrant that the
 * programs furnished hereunder are free of infringement of any third-party
 * patents.
 *
 * Commercial implementations of MPEG-1 and MPEG-2 video, including shareware,
 * are subject to royalty fees to patent holders.  Many of these patents are
 * general enough such that they are unavoidable regardless of implementation
 * design.
 *
 * MPEG2AVI
 * --------
 * v0.16B33 renamed the initialization function to init_idct_int32()
 * v0.16B32 removed the unused idct_row() and idct_col() functions
 * v0.16B3  changed var declarations to static, to enforce data align
 * v0.16B22  idct_FAST() renamed to idct_int32()
 *        also merged idct_FAST() into a single function, to help VC++
 *        optimize it.
 *
 * v0.14  changed int to long, to avoid confusion when compiling on x86
 *        platform ( in VC++ "int" -> 32bits )
 */

/*************************************************************/
/* inverse two dimensional DCT, Chen-Wang algorithm       */
/* (cf. IEEE ASSP-32, pp. 803-816, Aug. 1984)             */
/* 32-bit integer arithmetic (8 bit coefficients)         */
/* 11 mults, 29 adds per DCT                              */
/*                                     sE, 18.8.91       */
/*************************************************************/
/* coefficients extended to 12 bit for IEEE1180-1990      */
/* compliance                          sE,  2.1.94       */
/*************************************************************/

/* this code assumes >> to be a two's-complement arithmetic */
/* right shift: (-2)>>1 == -1 , (-3)>>1 == -2               */

#include "idct.h"

#define W1 2841 /* 2048*sqrt(2)*cos(1*pi/16) */
#define W2 2676 /* 2048*sqrt(2)*cos(2*pi/16) */
#define W3 2408 /* 2048*sqrt(2)*cos(3*pi/16) */
#define W5 1609 /* 2048*sqrt(2)*cos(5*pi/16) */
#define W6 1108 /* 2048*sqrt(2)*cos(6*pi/16) */
```

```
#define W7 565 /* 2048*sqrt(2)*cos(7*pi/16) */


/* two dimensional inverse discrete cosine transform */
void
idct_int32(short *const block)
{

/*
 * idct_int32_init() must be called before the first call to this
 * function!
 */

static short *blk;
static long i;
static long X0, X1, X2, X3, X4, X5, X6, X7, X8;


for (i = 0; i < 8; i++) /* idct rows */
{
blk = block + (i << 3);
if (!
((X1 = blk[4] << 11) | (X2 = blk[6]) | (X3 = blk[2]) | (X4 =
blk[1]) |
 (X5 = blk[7]) | (X6 = blk[5]) | (X7 = blk[3]))) {
blk[0] = blk[1] = blk[2] = blk[3] = blk[4] = blk[5] = blk[6] =
blk[7] = blk[0] << 3;
continue;
}

X0 = (blk[0] << 11) + 128; /* for proper rounding in the fourth stage  */

/* first stage  */
X8 = W7 * (X4 + X5);
X4 = X8 + (W1 - W7) * X4;
X5 = X8 - (W1 + W7) * X5;
X8 = W3 * (X6 + X7);
X6 = X8 - (W3 - W5) * X6;
X7 = X8 - (W3 + W5) * X7;

/* second stage  */
X8 = X0 + X1;
X0 -= X1;
X1 = W6 * (X3 + X2);
X2 = X1 - (W2 + W6) * X2;
X3 = X1 + (W2 - W6) * X3;
X1 = X4 + X6;
X4 -= X6;
X6 = X5 + X7;
X5 -= X7;
```

```
/* third stage  */
X7 = X8 + X3;
X8 -= X3;
X3 = X0 + X2;
X0 -= X2;
X2 = (181 * (X4 + X5) + 128) >> 8;
X4 = (181 * (X4 - X5) + 128) >> 8;

/* fourth stage  */

blk[0] = (short) ((X7 + X1) >> 8);
blk[1] = (short) ((X3 + X2) >> 8);
blk[2] = (short) ((X0 + X4) >> 8);
blk[3] = (short) ((X8 + X6) >> 8);
blk[4] = (short) ((X8 - X6) >> 8);
blk[5] = (short) ((X0 - X4) >> 8);
blk[6] = (short) ((X3 - X2) >> 8);
blk[7] = (short) ((X7 - X1) >> 8);

} /* end for ( i = 0; i < 8; ++i ) IDCT-rows */



for (i = 0; i < 8; i++) /* idct columns */
{
blk = block + i;
/* shortcut  */
if (!
((X1 = (blk[8 * 4] << 8)) | (X2 = blk[8 * 6]) | (X3 =
 blk[8 *
 2]) | (X4 =
blk[8 *
1])
 | (X5 = blk[8 * 7]) | (X6 = blk[8 * 5]) | (X7 = blk[8 * 3]))) {
blk[8 * 0] = blk[8 * 1] = blk[8 * 2] = blk[8 * 3] = blk[8 * 4] =
blk[8 * 5] = blk[8 * 6] = blk[8 * 7] =
iclp[(blk[8 * 0] + 32) >> 6];
continue;
}

X0 = (blk[8 * 0] << 8) + 8192;

/* first stage  */
X8 = W7 * (X4 + X5) + 4;
X4 = (X8 + (W1 - W7) * X4) >> 3;
X5 = (X8 - (W1 + W7) * X5) >> 3;
X8 = W3 * (X6 + X7) + 4;
X6 = (X8 - (W3 - W5) * X6) >> 3;
X7 = (X8 - (W3 + W5) * X7) >> 3;

/* second stage  */
```

```
X8 = X0 + X1;
X0 -= X1;
X1 = W6 * (X3 + X2) + 4;
X2 = (X1 - (W2 + W6) * X2) >> 3;
X3 = (X1 + (W2 - W6) * X3) >> 3;
X1 = X4 + X6;
X4 -= X6;
X6 = X5 + X7;
X5 -= X7;

/* third stage  */
X7 = X8 + X3;
X8 -= X3;
X3 = X0 + X2;
X0 -= X2;
X2 = (181 * (X4 + X5) + 128) >> 8;
X4 = (181 * (X4 - X5) + 128) >> 8;

/* fourth stage  */
blk[8 * 0] = iclp[(X7 + X1) >> 14];
blk[8 * 1] = iclp[(X3 + X2) >> 14];
blk[8 * 2] = iclp[(X0 + X4) >> 14];
blk[8 * 3] = iclp[(X8 + X6) >> 14];
blk[8 * 4] = iclp[(X8 - X6) >> 14];
blk[8 * 5] = iclp[(X0 - X4) >> 14];
blk[8 * 6] = iclp[(X3 - X2) >> 14];
blk[8 * 7] = iclp[(X7 - X1) >> 14];
}

} /* end function idct_int32(block) */
```

Source code of test program: xvid_decraw.c

B

This appendix contains the source code of the test program used to access the XviD library functions.

```
/*****************************************************************************
 *
 *  XVID MPEG-4 VIDEO CODEC
 *  - Console based decoding test application  -
 *
 *  Copyright(C) 2002-2003 Christoph Lampert
 *               2002-2003 Edouard Gomez <ed.gomez@free.fr>
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307 USA
 *
 * $Id: xvid_decraw.c,v 1.15 2004/04/20 19:47:00 edgomez Exp $
 *
 ****************************************************************************/

/*****************************************************************************
 *
 *  Application notes :
 *
 *  An MPEG-4 bitstream is read from an input file (or stdin) and decoded,
 *  the speed for this is measured.
 *
 *  The program is plain C and needs no libraries except for libxvidcore,
 *  and maths-lib.
 *
 *  Use ./xvid_decraw -help for a list of options
 *
 ****************************************************************************/


/*****************************************************************************
 *
 *  Additional modifications by Guido de Goede and Jonathan Hofman
```

```
 *
 *  in order to enable the use of DAMP hardware to speed up XviD
 *
 ****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#ifndef WIN32
#include <sys/time.h>
#else
#include <time.h>
#endif

#include "xvid.h"
#include "VGA_user_ioctl.h"

/*****************************************************************************
 *               Global vars in module and constants
 ****************************************************************************/

/* max number of frames */
#define ABS_MAXFRAMENR 9999

#define USE_PNM 0
#define USE_TGA 1
#define USE_BMP 2
#define USE_AVI 3

//#define REF yes
//#include "stockholm20.mp4.h"
//#include "parkrun20.mp4.h"
//#include "shields20.mp4.h"
//#include "mobcol20.mp4.h"
char * mobcol_name = "mobcol";
char * parkrun_name = "parkrun";
char * shields_name = "shields";
char * stockholm_name = "stockholm";



static int USE_HARDWARE = 0;
static int USE_REFFILES = 0;
static int OUTPUT_IDCT_VALUES = 0;
static int XDIM = 0;
static int YDIM = 0;
static int ARG_SAVEDECOUTPUT = 0;
static int ARG_SAVEMPEGSTREAM = 0;
static char *ARG_INPUTFILE = NULL;
static int CSP = XVID_CSP_I420;
static int BPP = 1;
static int FORMAT = USE_PNM;
static int ARG_STARTFRAME = 0;
static int ARG_NUMBEROFFRAMES = 0;
```

```
static int CURRENTFRAME = 0;
static int BMP_VERBOSE = 0;
static int ARG_SAVEFLUSHEDFRAME= 0;
static int ARG_VGADRIVER = 0;

static char filepath[256] = "./";
static void *dec_handle = NULL;
static char * reffile_ptr = NULL;
static long int reffile_size = 0;

/* to enable writing images to AVI files */
static FILE * AVI_FILE = NULL;

#define BUFFER_SIZE (20*2*1024*1024)

/*****************************************************************************
 *               Local prototypes
 ****************************************************************************/

static double msecond();
static int dec_init(int use_assembler, int debug_level);
static int dec_main(unsigned char *istream,
unsigned char *ostream,
int istream_size,
xvid_dec_stats_t *xvid_dec_stats);
static int dec_stop();
static void usage();
static int write_image(char *prefix, unsigned char *image);
static int write_pnm(char *filename, unsigned char *image);
static int write_tga(char *filename, unsigned char *image);
static int write_bmp(char *filename, unsigned char *image);
static int write_avi(char *filename, unsigned char *image);

const char * type2str(int type)
{
    if (type==XVID_TYPE_IVOP)
        return "I";
    if (type==XVID_TYPE_PVOP)
        return "P";
    if (type==XVID_TYPE_BVOP)
        return "B";
    return "S";
}

/*****************************************************************************
 *        Main program
 ****************************************************************************/

int main(int argc, char *argv[])
{
unsigned char *mp4_buffer = NULL;
unsigned char *mp4_ptr    = NULL;
unsigned char *out_buffer = NULL;
int useful_bytes;
xvid_dec_stats_t xvid_dec_stats;
```

```
double totaldectime;

long totalsize;
int status;

int use_assembler = 0;
int debug_level = 0;

char filename[256];

FILE *in_file;
int filenr;
int i;

printf("xvid_decraw - raw mpeg4 bitstream decoder ");
printf("written by Christoph Lampert 2002-2003\n\n");

/*****************************************************************************
 * Command line parsing
 ****************************************************************************/

for (i=1; i< argc; i++) {

if (strcmp("-asm", argv[i]) == 0 ) {
use_assembler = 1;
} else if (strcmp("-debug", argv[i]) == 0 && i < argc - 1 ) {
i++;
if (sscanf(argv[i], "0x%x", &debug_level) != 1) {
debug_level = atoi(argv[i]);
}
} else if (strcmp("-d", argv[i]) == 0) {
ARG_SAVEDECOUTPUT = 1;
} else if (strcmp("-verbose", argv[i]) == 0) {
BMP_VERBOSE = 1;
} else if (strcmp("-hardware", argv[i]) == 0) {
printf("ATTENTION: Using hardware from DAMP to speed up XVID!\n");
USE_HARDWARE = 1;
} else if (strcmp("-idct_values", argv[i]) == 0) {
printf("ATTENTION: outputting IDCT values to file: array.h!\n");
printf("You must manually modify this file to be correct.\n");
OUTPUT_IDCT_VALUES = 1;
} else if (strcmp("-vga_driver", argv[i]) == 0) {
ARG_VGADRIVER = 1;
  us_hardware_vga_init_ioctl("/dev/vga_driver_yuv", "/dev/vga_driver_rgb8",
   "/dev/vga_driver_rgb24", "/dev/vga_driver_bgr24");
} else if (strcmp("-i", argv[i]) == 0 && i < argc - 1 ) {
i++;
ARG_INPUTFILE = argv[i];
} else if (strcmp("-startframe", argv[i]) == 0 && i < argc - 1 ) {
i++;
ARG_STARTFRAME = atoi(argv[i]);
} else if (strcmp("-#frames", argv[i]) == 0 && i < argc - 1 ) {
i++;
ARG_NUMBEROFFRAMES = atoi(argv[i]);
```

```
} else if (strcmp("-m", argv[i]) == 0) {
ARG_SAVEMPEGSTREAM = 1;
} else if (strcmp("-saveflush", argv[i]) == 0) {
ARG_SAVEFLUSHEDFRAME = 1;
} else if (strcmp("-c", argv[i]) == 0  && i < argc - 1 ) {
i++;
if (strcmp(argv[i], "rgb16") == 0) {
CSP = XVID_CSP_RGB555;
BPP = 2;
} else if (strcmp(argv[i], "rgb24") == 0) {
CSP = XVID_CSP_BGR;
BPP = 3;
} else if (strcmp(argv[i], "rgb32") == 0) {
CSP = XVID_CSP_BGRA;
BPP = 4;
} else if (strcmp(argv[i], "yv12") == 0) {
CSP = XVID_CSP_YV12;
BPP = 1;
} else {
CSP = XVID_CSP_I420;
BPP = 1;
}
} else if (strcmp("-ref", argv[i]) == 0  && i < argc - 1 ) {
i++;
#ifndef REF
printf("References not internally available in thsi version\n");
return(0);
#else
if (strcmp(argv[i], "mobcol") == 0) {
USE_REFFILES = 1;
reffile_ptr = mobcol;
reffile_size = mobcol_size;
ARG_INPUTFILE = mobcol_name;
} else if (strcmp(argv[i], "parkrun") == 0) {
USE_REFFILES = 1;
reffile_ptr = parkrun;
reffile_size = parkrun_size;
ARG_INPUTFILE = parkrun_name;
} else if (strcmp(argv[i], "shields") == 0) {
USE_REFFILES = 1;
reffile_ptr = shields;
reffile_size = shields_size;
ARG_INPUTFILE = shields_name;
} else if (strcmp(argv[i], "stockholm") == 0) {
USE_REFFILES = 1;
reffile_ptr = stockholm;
reffile_size = stockholm_size;
ARG_INPUTFILE = stockholm_name;
} else {
printf("Reference not internally available: %s\n", argv[i]);
usage();
return(0);
}
#endif
} else if (strcmp("-f", argv[i]) == 0 && i < argc -1) {
```

```
i++;
if (strcmp(argv[i], "tga") == 0) {
FORMAT = USE_TGA;
} else if (strcmp(argv[i], "bmp") == 0) {
FORMAT = USE_BMP;
} else if (strcmp(argv[i], "avi") == 0) {
FORMAT = USE_AVI;
} else {
FORMAT = USE_PNM;
}
} else if (strcmp("-help", argv[i]) == 0) {
usage();
return(0);
} else {
usage();
exit(-1);
}
}

#if defined(_MSC_VER)
if (ARG_INPUTFILE==NULL) {
fprintf(stderr, "Warning: MSVC build does not read EOF correctly from stdin. Use the -i switch.\n\n");
}
#endif

/*****************************************************************************
 * Values checking
 *****************************************************************************/

if (USE_REFFILES || ARG_INPUTFILE == NULL || strcmp(ARG_INPUTFILE, "stdin") == 0) {
in_file = stdin;
}
else {

in_file = fopen(ARG_INPUTFILE, "rb");
if (in_file == NULL) {
fprintf(stderr, "Error opening input file %s\n", ARG_INPUTFILE);
return(-1);
}
}

/* PNM/PGM format can't handle 16/32 bit data and our BMP and AVI format also can't */
if (BPP != 1 && BPP != 3 && FORMAT == USE_PNM) {
FORMAT = USE_TGA;
}

/*****************************************************************************
 *        Memory allocation
 *****************************************************************************/

/* Memory for encoded mp4 stream */
if (USE_REFFILES) mp4_buffer = reffile_ptr;
else mp4_buffer = (unsigned char *) malloc(BUFFER_SIZE);
mp4_ptr = mp4_buffer;
if (!mp4_buffer)
```

```
goto free_all_memory;

/*****************************************************************************
 *        XviD PART  Start
 *****************************************************************************/

printf("Settings: %s, %s, %s\n", (USE_HARDWARE)?"HW_IDCT":"SW_IDCT",
(CSP == XVID_CSP_BGR)?"SW_CSP":"HW_CSP",ARG_INPUTFILE);
printf("Initialising decoder...\n");
status = dec_init(use_assembler, debug_level);
if (status) {
fprintf(stderr,
"Decore INIT problem, return value %d\n", status);
goto release_all;
}
printf("Initialisation of decoder completed :-)\n");

/*****************************************************************************
 *                           Main loop
 *****************************************************************************/

/* Fill the buffer */
if (USE_REFFILES) useful_bytes = reffile_size;
else useful_bytes = fread(mp4_buffer, 1, BUFFER_SIZE, in_file);

totaldectime = 0;
totalsize = 0;
filenr = 0;
mp4_ptr = mp4_buffer;

do {
int used_bytes = 0;
double dectime;


/*
 * If the buffer is half empty or there are no more bytes in it
 * then fill it.
 */
if (mp4_ptr > mp4_buffer + BUFFER_SIZE/2) {
int already_in_buffer = (mp4_buffer + BUFFER_SIZE - mp4_ptr);

/* Move data if needed */
if (already_in_buffer > 0)
memcpy(mp4_buffer, mp4_ptr, already_in_buffer);

/* Update mp4_ptr */
mp4_ptr = mp4_buffer;

/* read new data */
            if(feof(in_file))
break;

useful_bytes += fread(mp4_buffer + already_in_buffer,
  1, BUFFER_SIZE - already_in_buffer,
```

```
  in_file);

}


/* This loop is needed to handle VOL/NVOP reading */
do {
/* Decode frame */
dectime = msecond();
used_bytes = dec_main(mp4_ptr, out_buffer, useful_bytes, &xvid_dec_stats);
/* write images to VGA driver and display */
if (ARG_VGADRIVER)
{
if (CSP == XVID_CSP_BGR) us_hardware_vga_bgr24_ioctl(out_buffer);
else us_hardware_vga_yuv_ioctl(out_buffer);
}
dectime = msecond() - dectime;

/* Resize image buffer if needed */
if(xvid_dec_stats.type == XVID_TYPE_VOL) {

/* Check if old buffer is smaller */
if(XDIM*YDIM < xvid_dec_stats.data.vol.width*xvid_dec_stats.data.vol.height) {

/* Copy new witdh and new height from the vol structure */
XDIM = xvid_dec_stats.data.vol.width;
YDIM = xvid_dec_stats.data.vol.height;

/* Free old output buffer*/
if(out_buffer) free(out_buffer);

/* Allocate the new buffer */
out_buffer = (unsigned char*)malloc(XDIM*YDIM*4);
if(out_buffer == NULL)
goto free_all_memory;

fprintf(stderr, "Resized frame buffer to %dx%d\n", XDIM, YDIM);
}
}

/* Update buffer pointers */
if(used_bytes > 0) {
mp4_ptr += used_bytes;
useful_bytes -= used_bytes;

/* Total size */
totalsize += used_bytes;
}

}while(xvid_dec_stats.type <= 0 && useful_bytes > 0);

/* Check if there is a negative number of useful bytes left in buffer
 * This means we went too far */
        if(useful_bytes < 0)
            break;
```

```
     /* Updated data - Count only usefull decode time */
totaldectime += dectime;


        printf("Frame %5d: type = %s, dec&disp time(ms) =%6.1f, length(bytes) =%7d\n",
   filenr, type2str(xvid_dec_stats.type), dectime, used_bytes);

/* Save individual mpeg4 stream if required */
if(ARG_SAVEMPEGSTREAM) {
FILE *filehandle = NULL;

sprintf(filename, "%sframe%05d.m4v", filepath, filenr);
filehandle = fopen(filename, "wb");
if(!filehandle) {
fprintf(stderr,
"Error writing single mpeg4 stream to file %s\n",
filename);
}
else {
fwrite(mp4_ptr-used_bytes, 1, used_bytes, filehandle);
fclose(filehandle);
}
}

/* Save output frame if required */
if (ARG_SAVEDECOUTPUT && (CURRENTFRAME>=ARG_STARTFRAME)
&& ( ARG_NUMBEROFFRAMES==0 || CURRENTFRAME<(ARG_STARTFRAME+ARG_NUMBEROFFRAMES) ))
{
sprintf(filename, "%sdec%05d", filepath, filenr);
if(write_image(filename, out_buffer))
{
fprintf(stderr,
"Error writing decoded frame %s\n",
filename);
}
}
CURRENTFRAME++;
filenr++;

} while ( (status>=0) && (filenr<ABS_MAXFRAMENR));

/*****************************************************************************
 *     Flush decoder buffers
 ****************************************************************************/

do {

/* Fake vars */
int used_bytes;
double dectime;

        do {
    dectime = msecond();
    used_bytes = dec_main(NULL, out_buffer, -1, &xvid_dec_stats);
```

```
    dectime = msecond() - dectime;
        }while(used_bytes>=0 && xvid_dec_stats.type <= 0);

        if (used_bytes < 0) {   /* XVID_ERR_END */
            break;
        }

/* Updated data - Count only usefull decode time */
totaldectime += dectime;

/* Prints some decoding stats */
        printf("Frame %5d: type = %s, dectime(ms) =%6.1f, length(bytes) =%7d\n",
   filenr, type2str(xvid_dec_stats.type), dectime, used_bytes);

/* Save output frame if required */
if (ARG_SAVEFLUSHEDFRAME && ARG_SAVEDECOUTPUT &&
CURRENTFRAME<(ARG_STARTFRAME+ARG_NUMBEROFFRAMES))
{
sprintf(filename, "%sdec%05d", filepath, filenr);
if(write_image(filename, out_buffer))
{
fprintf(stderr,
"Error writing decoded frame %s\n",
filename);
}
}

filenr++;

}while(1);

/*****************************************************************************
 *     Calculate totals and averages for output, print results
 ****************************************************************************/

if (filenr>0) {
FILE * log_file = NULL;
totalsize    /= filenr;
totaldectime /= filenr;
printf("Avg: dectime(ms) =%7.2f, fps =%7.2f, length(bytes) =%7d\n",
   totaldectime, 1000/totaldectime, (int)totalsize);
/* LOG FILE */
log_file = fopen("4tests_log.txt", "ab");
if (!log_file) printf("ERROR openeing logfile 4tests_log.txt!\n");
else
{
fprintf(log_file, "Settings: %s, %s, %s => ", (USE_HARDWARE)?"HW_IDCT":"SW_IDCT",
(CSP == XVID_CSP_BGR)?"SW_CSP":"HW_CSP",ARG_INPUTFILE);
fprintf(log_file, "Avg: dectime(ms) =%7.2f, fps =%7.2f, length(bytes) =%7d\n",
   totaldectime, 1000/totaldectime, (int)totalsize);
fclose(log_file);
}
if (ARG_VGADRIVER) us_hardware_vga_finit_ioctl();
}else{
printf("Nothing was decoded!\n");
```

```
}

/*****************************************************************************
 *      XviD PART  Stop
 *****************************************************************************/

 release_all:
   if (dec_handle) {
   status = dec_stop();
if (status)
fprintf(stderr, "decore RELEASE problem return value %d\n", status);
}

 free_all_memory:
free(out_buffer);
if (!USE_REFFILES) free(mp4_buffer);

return(0);
}

/*****************************************************************************
 *               Usage function
 *****************************************************************************/

static void usage()
{

fprintf(stderr,
"Usage : xvid_decraw [OPTIONS]\n");
fprintf(stderr,
"Options :\n");
fprintf(stderr,
" -asm           : use assembly optimizations (default=disabled)\n");
fprintf(stderr,
" -debug         : debug level (debug=0)\n");
fprintf(stderr,
" -i string      : input filename (default=stdin)\n");
fprintf(stderr,
" -d             : save decoder output (files called dec#####.PGM/TGA/BMP/AVI)\n");
fprintf(stderr,
" -c csp         : choose colorspace output (rgb16, rgb24, rgb32, yv12, i420)\n");
fprintf(stderr,
" -f format      : choose output file format (tga, pnm, pgm, bmp, avi)\n");
fprintf(stderr,
" -startframe    : the number of frames from where to start saving\n");
fprintf(stderr,
" -#frames       : the number of frames to save\n");
fprintf(stderr,
" -verbose       : extra information displayed in functions (write_bmp)\n");
fprintf(stderr,
" -saveflush     : save the frame that is in the buffer at the end when flushing\n");
fprintf(stderr,
" -m             : save mpeg4 raw stream to individual files\n");
fprintf(stderr,
" -help          : This help message\n");
```

```
fprintf(stderr,
" -hardware      : use the hardware from DAMP to speedup the IDCT conversion\n");
fprintf(stderr,
" -vga_driver    : output images to VGA driver\n");
fprintf(stderr,
" -ref      : mobcol | parkrun | shields | stockholm \n");
fprintf(stderr,
" (* means default)\n");

}

/*****************************************************************************
 *               "helper" functions
 ****************************************************************************/

/* return the current time in milli seconds */
static double
msecond()
{
#ifndef WIN32
struct timeval  tv;
gettimeofday(&tv, 0);
return((double)tv.tv_sec*1.0e3 + (double)tv.tv_usec*1.0e-3);
#else
clock_t clk;
clk = clock();
return(clk * 1000 / CLOCKS_PER_SEC);
#endif
}

/*****************************************************************************
 *              output functions
 ****************************************************************************/

static int write_image(char *prefix, unsigned char *image)
{
char filename[1024];
char *ext;
int ret;

if (FORMAT == USE_PNM && BPP == 1) {
ext = "pgm";
} else if (FORMAT == USE_PNM && BPP == 3) {
ext = "pnm";
} else if (FORMAT == USE_TGA) {
ext = "tga";
} else if (FORMAT == USE_BMP) {
ext = "bmp";
} else if (FORMAT == USE_AVI) {
ext = "avi";
} else {
fprintf(stderr, "Bug: should not reach this path code --
please report to xvid-devel@xvid.org with command line options used");
exit(-1);
}
```

```
sprintf(filename, "%s.%s", prefix, ext);

if (FORMAT == USE_PNM) {
ret = write_pnm(filename, image);
} else if (FORMAT == USE_BMP) {
ret = write_bmp(filename, image);
} else if (FORMAT == USE_AVI) {
ret = write_avi(filename, image);
} else {
ret = write_tga(filename, image);
}

return(ret);
}

static int write_tga(char *filename, unsigned char *image)
{
FILE * f;
char hdr[18];

f = fopen(filename, "wb");
if ( f == NULL) {
return -1;
}

hdr[0]  = 0; /* ID length */
hdr[1]  = 0; /* Color map type */
hdr[2]  = (BPP>1)?2:3; /* Uncompressed true color (2) or greymap (3) */
hdr[3]  = 0; /* Color map specification (not used) */
hdr[4]  = 0; /* Color map specification (not used) */
hdr[5]  = 0; /* Color map specification (not used) */
hdr[6]  = 0; /* Color map specification (not used) */
hdr[7]  = 0; /* Color map specification (not used) */
hdr[8]  = 0; /* LSB X origin */
hdr[9]  = 0; /* MSB X origin */
hdr[10] = 0; /* LSB Y origin */
hdr[11] = 0; /* MSB Y origin */
hdr[12] = (XDIM>>0)&0xff; /* LSB Width */
hdr[13] = (XDIM>>8)&0xff; /* MSB Width */
if (BPP > 1) {
hdr[14] = (YDIM>>0)&0xff; /* LSB Height */
hdr[15] = (YDIM>>8)&0xff; /* MSB Height */
} else {
hdr[14] = ((YDIM*3)>>1)&0xff; /* LSB Height */
hdr[15] = ((YDIM*3)>>9)&0xff; /* MSB Height */
}
hdr[16] = BPP*8;
hdr[17] = 0x00 | (1<<5) /* Up to down */ | (0<<4); /* Image descriptor */

/* Write header */
fwrite(hdr, 1, sizeof(hdr), f);

#ifdef ARCH_IS_LITTLE_ENDIAN
/* write first plane */
```

```
fwrite(image, 1, XDIM*YDIM*BPP, f);
#else
{
int i;
for (i=0; i<XDIM*YDIM*BPP;i+=BPP) {
if (BPP == 1) {
fputc(image[i], f);
} else if (BPP == 2) {
fputc(image[i+1], f);
fputc(image[i], f);
} else if (BPP == 3) {
fputc(image[i+2], f);
fputc(image[i+1], f);
fputc(image[i+0], f);
} else if (BPP == 4) {
fputc(image[i+3], f);
fputc(image[i+2], f);
fputc(image[i+1], f);
fputc(image[i+0], f);
}
}
}
#endif

/* Write Y and V planes for YUV formats */
if (BPP == 1) {
int i;

/* Write the two chrominance planes */
for (i=0; i<YDIM/2; i++) {
fwrite(image+XDIM*YDIM + i*XDIM/2, 1, XDIM/2, f);
fwrite(image+5*XDIM*YDIM/4 + i*XDIM/2, 1, XDIM/2, f);
}
}


/* Close the file */
fclose(f);

return(0);
}

static int write_bmp(char *filename, unsigned char *image)
{
FILE * f;
char hdr[54];
char * RGBQUAD;
unsigned long filesize, imagesize;
int i, width32, height_adapted, width, height, YUV_SCALE;

if (BPP!=1 && BPP != 3 || BPP == 1)
{
printf("For now only 24 bit supported in BMP! (normaly 8 or 24 bit)");
return -1;
}
```

```
if (BPP==1) YUV_SCALE = 1;
 else YUV_SCALE = 0;

if(YUV_SCALE) height=YDIM*1.5;
else height = YDIM;
width = XDIM;

width32 = (width * 4 + 3) /4; /* round width up to 4 byte value */

if (BPP==3) imagesize = width*height_adapted*3; /* 3 bytes for 24 bits */
else imagesize = width32*height_adapted;
/* imagesize depends on number of 32 bit words written per line */
if(BMP_VERBOSE) printf("imagesize: %i, %i x %i\n", imagesize, width32, height_adapted);
filesize = 54 + ((BPP>1)?0:1024); /* 1024 more if color table present in file */
filesize += imagesize;
if (BPP==1) RGBQUAD = (char*)malloc(sizeof(char)*1024);
/* create RGBQUAD array if color table is necessary */
f = fopen(filename, "wb");
if ( f == NULL) {
return -1;
}
if ( BPP != 1 && BPP != 3) {
printf("Only BMPs supported of 8 bits per pixel or 24 bits per pixel.");
return -1;
}
hdr[0]  = 'B'; /* Bitmap signature BM */
hdr[1]  = 'M';
hdr[2]  = (filesize>>0)&0xff; /* LSB filesize */
hdr[3]  = (filesize>>8)&0xff;
hdr[4]  = (filesize>>16)&0xff;
hdr[5]  = (filesize>>24)&0xff; /* MSB filesize */
hdr[6]  = 0; /* (not used) */
hdr[7]  = 0; /* (not used) */
hdr[8]  = 0; /* (not used) */
hdr[9]  = 0; /* (not used) */
hdr[10] = 54; /* LSB Offset */
hdr[11] = (BPP>1)?0:4;
/* Specifies the offset from the beginning of the file to the bitmap data, */
hdr[12] = 0; /*  which is 54 in case of no color table */
/* and 1078 in case of 256 colors (1 byte) * 32 bits*/
hdr[13] = 0; /* MSB Offset 1078 or 54, for both LSB=54*/

hdr[14] = 40; /* LSB Size BITMAPINFOHEADER */
hdr[15] = 0; /* pecifies the size of the BITMAPINFOHEADER structure, in bytes. */
hdr[16] = 0; /*  which is 40 normally */
hdr[17] = 0; /* MSB Size BITMAPINFOHEADER */

hdr[18] = (width>>0)&0xff; /* LSB Width */
hdr[19] = (width>>8)&0xff;
hdr[20] = (width>>16)&0xff;
hdr[21] = (width>>24)&0xff; /* MSB Width */

/* if (BPP > 1) {*/
hdr[22] = (height>>0)&0xff; /* LSB Height */
hdr[23] = (height>>8)&0xff;
```

```
hdr[24] = (height>>16)&0xff;
hdr[25] = (height>>24)&0xff; /* MSB Height */
/* } */

hdr[26] = 1; /* specifies the number of planes of the target device, */
hdr[27] = 0; /*  which is 1 normally */

hdr[28] = BPP*8; /* specifies the number of bits per pixel, */
hdr[29] = 0; /* 8 for 256 colors, 24 for 16 milion colors */

hdr[30]  = 0; /* Specifies the type of compression, */
hdr[31]  = 0; /* --- */
hdr[32]  = 0; /* --- */
hdr[33]  = 0; /*  usually set to zero (no compression). */

hdr[34]  = (imagesize>>0)&0xff; /* LSB imagesize */
hdr[35]  = (imagesize>>8)&0xff; /* Specifies the size of the image data, in bytes. */
hdr[36]  = (imagesize>>16)&0xff;
/* If there is no compression, it is valid to set this member to zero. */
hdr[37]  = (imagesize>>24)&0xff; /* MSB imagesize */

hdr[38]  = (BPP>1)?0xC4:0;
/* Specifies the the horizontal pixels per meter on the designated targer device, */
hdr[39]  = (BPP>1)?0x0E:0; /* --- */
hdr[40]  = 0; /* --- */
hdr[41]  = 0; /* usually set to zero. */

hdr[42]  = (BPP>1)?0xC4:0;
/* Specifies the the vertical pixels per meter on the designated targer device, */
hdr[43]  = (BPP>1)?0x0E:0; /* --- */
hdr[44]  = 0; /* --- */
hdr[45]  = 0; /* usually set to zero. */

hdr[46]  = 0; /* Specifies the number of colors used in the bitmap, */
hdr[47]  = (BPP>1)?0:1; /* 256 colors if BPP==1 */
hdr[48]  = 0; /* --- */
hdr[49]  = 0;
/* if set to zero the number of colors is calculated using the biBitCount member. */


hdr[50]  = 0; /* Specifies the number of color that are 'important' for the bitmap, */
hdr[51]  = (BPP>1)?0:1; /* 256 colors if BPP==1 */
hdr[52]  = 0; /* --- */
hdr[53]  = 0;  /* if set to zero, all colors are important. */

/* Write header */
fwrite(hdr, 1, sizeof(hdr), f);

if (BPP==1)  /*color table necessary if BPP == 1, make color table */
{
for(i=0;i<256;i++)
{
/* fill color table, only RGB->G values change, rest 0 */
RGBQUAD[i*4+0] = i; /* RGB->B value */
RGBQUAD[i*4+1] = i; /* RGB->G value */
```

```
RGBQUAD[i*4+2] = i; /* RGB->R value */
RGBQUAD[i*4+3] = 0; /* reserved value must be zero*/
}
/* Write color table */
fwrite(RGBQUAD, 1, 1024*sizeof(char), f);
printf("Writing color table \n");
}

/* Write Y and V planes for YUV formats */
if (YUV_SCALE) {
int i,j;
printf("Writing YUV planes\n");
/* Write the two chrominance planes */
for (i=(height/3); i>=0; i--) {
   for(j=0;j<BPP*width/2;j++) {
    fputc(*(image+i*2*width*BPP+j*2),f);
    /*fputc(((i*width*BPP+j+0)%255),f);*/
    /*fputc((unsigned char)*(image+i*width32+j*2), f);*/
/*fwrite(image+XDIM*YDIM + i*XDIM/2, 1, XDIM/2, f);
fwrite(image+5*XDIM*YDIM/4 + i*XDIM/2, 1, XDIM/2, f);*/
   }
   for(j=0;j<BPP*width/2;j++) {
    fputc(*(image+(i*2+1)*width*BPP+j*2+1),f);
   }
}
height = height/1.5;
}


/* write RGB or first yuv plane */
{
int i,j,k;
if(BMP_VERBOSE==1)printf("Writing imagedata:\n");
for(j=height-1;j>=0;j--){
   for (k=0; k<width*BPP;k+=BPP) {
i=j*width32*BPP+k;
if (BPP == 1) {
fputc(*(image+i), f);
if(BMP_VERBOSE==1)printf("%i ",(unsigned char)*(image+i) );
} else if (BPP == 2) {
printf("Shouldn't get here, 16 bit BMP not yet supported.");
fclose(f);
return -1;
/*fputc(image+i+0, f);
fputc(image+i+1, f);*/
} else if (BPP == 3) {
fputc(*(image+i+0), f);
fputc(*(image+i+1), f);
fputc(*(image+i+2), f);
} else if (BPP == 4) {
printf("Shouldn't get here, 32 bit BMP not supported.");
fclose(f);
return -1;
/*fputc(image+i+0, f);
fputc(image+i+1, f);
```

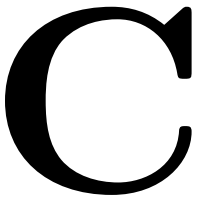
```
fputc(image+i+2, f);
fputc(image+i+3, f);*/
}
    }
}
}

/* Close the file */
fclose(f);

return(0);
}

static int write_avi(char *filename, unsigned char *image)
{
/*
if (CURRENTFRAME==ARG_STARTFRAME)
open file and create AVI header;
else if (CURRENTFRAME == (ARG_STARTFRAME + ARG_NUMBEROFFRAMES-1)
write last frame and close;
else write current frame;
*/
}



static int write_pnm(char *filename, unsigned char *image)
{
FILE * f;

f = fopen(filename, "wb");
if ( f == NULL) {
return -1;
}

if (BPP == 1) {
int i;
fprintf(f, "P5\n#xvid\n%i %i\n255\n", XDIM, YDIM*3/2);

fwrite(image, 1, XDIM*YDIM, f);

for (i=0; i<YDIM/2;i++) {
fwrite(image+XDIM*YDIM + i*XDIM/2, 1, XDIM/2, f);
fwrite(image+5*XDIM*YDIM/4 + i*XDIM/2, 1, XDIM/2, f);
}
} else if (BPP == 3) {
int i;
fprintf(f, "P6\n#xvid\n%i %i\n255\n", XDIM, YDIM);
for (i=0; i<XDIM*YDIM*3; i+=3) {
#ifdef ARCH_IS_LITTLE_ENDIAN
fputc(image[i+2], f);
fputc(image[i+1], f);
fputc(image[i+0], f);
#else
fputc(image[i+0], f);
```

```
fputc(image[i+1], f);
fputc(image[i+2], f);
#endif
}
}

fclose(f);

return 0;
}

/*****************************************************************************
 * Routines for decoding: init decoder, use, and stop decoder
 ****************************************************************************/

/* init decoder before first run */
static int
dec_init(int use_assembler, int debug_level)
{
int ret;

xvid_gbl_init_t   xvid_gbl_init;
xvid_dec_create_t xvid_dec_create;

/*------------------------------------------------------------------------
 * XviD core initialization
 *----------------------------------------------------------------------*/

/* Version */
xvid_gbl_init.version = XVID_VERSION;

/* Assembly setting */
if(use_assembler)
#ifdef ARCH_IS_IA64
xvid_gbl_init.cpu_flags = XVID_CPU_FORCE | XVID_CPU_IA64;
#else
xvid_gbl_init.cpu_flags = 0;
#endif
else
xvid_gbl_init.cpu_flags = XVID_CPU_FORCE;

xvid_gbl_init.debug = debug_level;

if (OUTPUT_IDCT_VALUES) xvid_global(NULL, -2, &xvid_gbl_init, NULL);
else if (USE_HARDWARE) xvid_global(NULL, -1, &xvid_gbl_init, NULL);
else xvid_global(NULL, 0, &xvid_gbl_init, NULL);

/*------------------------------------------------------------------------
 * XviD encoder initialization
 *----------------------------------------------------------------------*/

/* Version */
xvid_dec_create.version = XVID_VERSION;

/*
```

```
 * Image dimensions -- set to 0, xvidcore will resize when ever it is
 * needed
 */
xvid_dec_create.width = 0;
xvid_dec_create.height = 0;

ret = xvid_decore(NULL, XVID_DEC_CREATE, &xvid_dec_create, NULL);

dec_handle = xvid_dec_create.handle;

return(ret);
}

/* decode one frame  */
static int
dec_main(unsigned char *istream,
 unsigned char *ostream,
 int istream_size,
 xvid_dec_stats_t *xvid_dec_stats)
{

int ret;

xvid_dec_frame_t xvid_dec_frame;

/* Set version */
xvid_dec_frame.version = XVID_VERSION;
xvid_dec_stats->version = XVID_VERSION;

/* No general flags to set */
xvid_dec_frame.general          = 0;

/* Input stream */
xvid_dec_frame.bitstream        = istream;
xvid_dec_frame.length           = istream_size;

/* Output frame structure */
xvid_dec_frame.output.plane[0]  = ostream;
xvid_dec_frame.output.stride[0] = XDIM*BPP;
xvid_dec_frame.output.csp = CSP;

ret = xvid_decore(dec_handle, XVID_DEC_DECODE, &xvid_dec_frame, xvid_dec_stats);

return(ret);
}

/* close decoder to release resources */
static int
dec_stop()
{
int ret;

ret = xvid_decore(dec_handle, XVID_DEC_DESTROY, NULL, NULL);

return(ret);
```

```
}
```

Profiling results on the x86 platform



This appendix contains the results of the initial profiling performed on the x86 platform before porting the XviD code to the DAMP platform. The first column shows the relative time in percentages spent in each function. Functions can be called many times by their parent functions, so even though the time spent in a child is not much each call, the total time spent executing this child function can be large. As for instance can be seen with `idct_int32` where the relative time is almost 20% because it is called 167121 times by its parent. The profiling results are automatically generated by the profiler `gprof`. The terms used by the profiler are explained further on in the file.

```
Flat profile:

Each sample counts as 0.00195312 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 26.92      0.43     0.43       67     6.44     6.44  yv12_to_bgr_c
 19.73      0.75     0.32   167121     0.00     0.00  idct_int32
  8.40      0.88     0.13    90615     0.00     0.00  get_inter_block
  6.46      0.99     0.10   136681     0.00     0.00  transfer8x8_copy_c
  3.78      1.05     0.06    55797     0.00     0.00  interpolate8x8_halfpel_hv_c
  3.53      1.10     0.06    90615     0.00     0.00  dequant_h263_inter_c
  3.53      1.16     0.06    90615     0.00     0.00  transfer_16to8add_c
  3.05      1.21     0.05    76506     0.00     0.00  transfer_16to8copy_c
  2.80      1.25     0.04    76506     0.00     0.00  dequant_h263_intra_c
  2.56      1.29     0.04    32110     0.00     0.00  get_intra_block
  2.07      1.33     0.03    62490     0.00     0.00  interpolate8x8_avg2_c
  1.95      1.36     0.03       80     0.39     0.39  image_setetinydges
  1.95      1.39     0.03       25     1.25    17.31  decoder_bframe
  1.71      1.42     0.03    76506     0.00     0.00  predict_acdc
  1.34      1.44     0.02    32096     0.00     0.00  interpolate8x8_halfpel_h_c
  1.22      1.46     0.02    76506     0.00     0.00  add_acdc
  1.22      1.48     0.02    36378     0.00     0.00  interpolate8x8_halfpel_v_c
  0.97      1.49     0.02  1603803     0.00     0.00  DPRINTF
  0.97      1.51     0.02    21721     0.00     0.02  decoder_mb_decode
  0.85      1.54     0.01    10415     0.00     0.02  decoder_bf_interpolate_mbinter
  0.73      1.55     0.01    20021     0.00     0.00  get_pmv2
  0.73      1.56     0.01       30     0.39    16.49  decoder_pframe
  0.61      1.57     0.01       82     0.12    19.41  decoder_decode
  0.37      1.58     0.01    12751     0.00     0.03  decoder_mbintra
  0.24      1.58     0.00    70532     0.00     0.00  get_mv
  0.24      1.58     0.00    51004     0.00     0.00  get_dc_size_lum
  0.12      1.59     0.00   136608     0.00     0.00  DPRINTF
  0.12      1.59     0.00    45828     0.00     0.00  check_resync_marker
  0.12      1.59     0.00    19572     0.00     0.00  get_mcbpc_inter
  0.12      1.59     0.00       80     0.02     0.02  BitstreamReadHeaders
  0.12      1.59     0.00       67     0.03     0.03  type2str
  0.12      1.60     0.00        1     1.95     3.91  decoder_create
```

```
  0.12      1.60     0.00        1     1.95     1.95  init_noise
  0.12      1.60     0.00                             get_qpmv2
  0.12      1.60     0.00                             output_slice
  0.12      1.60     0.00                             transfer_8to16copy_c
  0.00      1.60     0.00   184051     0.00     0.00  DPRINTF
  0.00      1.60     0.00    33610     0.00     0.00  get_dc_dif
  0.00      1.60     0.00    27780     0.00     0.00  get_cbpy
  0.00      1.60     0.00    25502     0.00     0.00  get_dc_size_chrom
  0.00      1.60     0.00     8208     0.00     0.00  get_mcbpc_intra
  0.00      1.60     0.00      862     0.00     0.00  DPRINTF
  0.00      1.60     0.00      164     0.00     0.00  msecond
  0.00      1.60     0.00       84     0.00     0.00  image_swap
  0.00      1.60     0.00       84     0.00    19.00  xvid_decore
  0.00      1.60     0.00       83     0.00     0.00  emms_c
  0.00      1.60     0.00       82     0.00    19.00  dec_main
  0.00      1.60     0.00       67     0.00     6.44  image_output
  0.00      1.60     0.00       66     0.00     0.00  write_image
  0.00      1.60     0.00       66     0.00     0.00  write_tga
  0.00      1.60     0.00       23     0.00     0.00  xvid_free
  0.00      1.60     0.00       23     0.00     0.00  xvid_malloc
  0.00      1.60     0.00       12     0.00    18.41  decoder_iframe
  0.00      1.60     0.00       12     0.00     0.00  image_destroy
  0.00      1.60     0.00        6     0.00     0.00  image_create
  0.00      1.60     0.00        1     0.00     0.00  colorspace_init
  0.00      1.60     0.00        1     0.00    19.00  dec_init
  0.00      1.60     0.00        1     0.00    19.00  dec_stop
  0.00      1.60     0.00        1     0.00     0.00  decoder_destroy
  0.00      1.60     0.00        1     0.00     0.00  decoder_resize
  0.00      1.60     0.00        1     0.00     0.00  idct_int32_init
  0.00      1.60     0.00        1     0.00     0.00  init_deblock
  0.00      1.60     0.00        1     0.00     0.00  init_mpeg_matrix
  0.00      1.60     0.00        1     0.00     1.95  init_postproc
  0.00      1.60     0.00        1     0.00     0.00  init_vlc_tables
  0.00      1.60     0.00        1     0.00     0.00  xvid_Init_QP
  0.00      1.60     0.00        1     0.00     0.00  xvid_gbl_init
  0.00      1.60     0.00        1     0.00     0.00  xvid_global

 %         the percentage of the total running time of the
time       program used by this function.

cumulative a running sum of the number of seconds accounted
 seconds   for by this function and those listed above it.

 self      the number of seconds accounted for by this
seconds    function alone.  This is the major sort for this
           listing.

calls      the number of times this function was invoked, if
           this function is profiled, else blank.

 self      the average number of milliseconds spent in this
ms/call    function per call, if this function is profiled,
   else blank.

 total     the average number of milliseconds spent in this
```

```
ms/call    function and its descendents per call, if this
   function is profiled, else blank.

name       the name of the function.  This is the minor sort
           for this listing. The index shows the location of
   the function in the gprof listing. If the index is
   in parenthesis it shows where it would appear in
   the gprof listing if it were to be printed.


     Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) for 0.12% of 1.60 seconds

index % time    self  children    called     name
                                               <spontaneous>
[1]     99.6    0.00    1.60                 main [1]
                0.00    1.56      82/82          dec_main [4]
                0.00    0.02       1/1           dec_init [29]
                0.00    0.02       1/1           dec_stop [30]
                0.00    0.00      67/67          type2str [40]
                0.00    0.00     164/164         msecond [52]
                0.00    0.00      66/66          write_image [55]
-----------------------------------------------
                0.00    0.02       1/84          dec_init [29]
                0.00    0.02       1/84          dec_stop [30]
                0.00    1.56      82/84          dec_main [4]
[2]     99.5    0.00    1.60      84           xvid_decore [2]
                0.01    1.58      82/82          decoder_decode [3]
                0.00    0.00       1/1           decoder_create [35]
                0.00    0.00       1/1           decoder_destroy [62]
-----------------------------------------------
                0.01    1.58      82/82          xvid_decore [2]
[3]     99.3    0.01    1.58      82           decoder_decode [3]
                0.01    0.48      30/30          decoder_pframe [5]
                0.03    0.40      25/25          decoder_bframe [8]
                0.00    0.43      67/67          image_output [9]
                0.00    0.22      12/12          decoder_iframe [14]
                0.00    0.00      80/80          BitstreamReadHeaders [39]
                0.00    0.00      84/84          image_swap [53]
                0.00    0.00      82/83          emms_c [54]
                0.00    0.00      80/184051      DPRINTF [46]
                0.00    0.00       1/1           decoder_resize [63]
-----------------------------------------------
                0.00    1.56      82/82          main [1]
[4]     97.1    0.00    1.56      82           dec_main [4]
                0.00    1.56      82/84          xvid_decore [2]
-----------------------------------------------
                0.01    0.48      30/30          decoder_decode [3]
[5]     30.9    0.01    0.48      30           decoder_pframe [5]
                0.01    0.32   15977/22662       decoder_mbinter [6]
                0.00    0.12    4543/12751       decoder_mbintra [11]
                0.01    0.00   20021/20021       get_pmv2 [32]
                0.01    0.00      30/80          image_setedges [24]
```

```
                0.00    0.00   40042/70532       get_mv [33]
                0.00    0.00   19572/19572       get_mcbpc_inter [38]
                0.00    0.00   20520/45828       check_resync_marker [37]
                0.00    0.00   99257/184051      DPRINTF [46]
                0.00    0.00   19572/27780       get_cbpy [48]
-----------------------------------------------
                0.00    0.14    6685/22662       decoder_bframe [8]
                0.01    0.32   15977/22662       decoder_pframe [5]
[6]     29.4    0.01    0.46   22662         decoder_mbinter [6]
                0.01    0.34   16925/21721       decoder_mb_decode [7]
                0.04    0.00   40660/55797       interpolate8x8_halfpel_hv_c [17]
                0.04    0.00   56014/136681      transfer8x8_copy_c [16]
                0.01    0.00   17837/32096       interpolate8x8_halfpel_h_c [27]
                0.01    0.00   21461/36378       interpolate8x8_halfpel_v_c [28]
-----------------------------------------------
                0.00    0.10    4796/21721       decoder_bf_interpolate_mbinter [13]
                0.01    0.34   16925/21721       decoder_mbinter [6]
[7]     27.8    0.02    0.43   21721         decoder_mb_decode [7]
                0.17    0.00   90615/167121      idct_int32 [12]
                0.13    0.01   90615/90615       get_inter_block [15]
                0.06    0.00   90615/90615       dequant_h263_inter_c [18]
                0.06    0.00   90615/90615       transfer_16to8add_c [19]
-----------------------------------------------
                0.03    0.40      25/25          decoder_decode [3]
[8]     27.0    0.03    0.40      25          decoder_bframe [8]
                0.01    0.23   10415/10415       decoder_bf_interpolate_mbinter [13]
                0.00    0.14    6685/22662       decoder_mbinter [6]
                0.02    0.00      50/80          image_setedges [24]
                0.00    0.00   30490/70532       get_mv [33]
                0.00    0.00   17100/45828       check_resync_marker [37]
-----------------------------------------------
                0.00    0.43      67/67          decoder_decode [3]
[9]     26.9    0.00    0.43      67          image_output [9]
                0.43    0.00      67/67          yv12_to_bgr_c [10]
-----------------------------------------------
                0.43    0.00      67/67          image_output [9]
[10]    26.9    0.43    0.00      67          yv12_to_bgr_c [10]
-----------------------------------------------
                0.00    0.12    4543/12751       decoder_pframe [5]
                0.00    0.22    8208/12751       decoder_iframe [14]
[11]    21.4    0.01    0.34   12751         decoder_mbintra [11]
                0.14    0.00   76506/167121      idct_int32 [12]
                0.05    0.00   76506/76506       transfer_16to8copy_c [20]
                0.04    0.00   32110/32110       get_intra_block [21]
                0.04    0.00   76506/76506       dequant_h263_intra_c [22]
                0.03    0.00   76506/76506       predict_acdc [25]
                0.02    0.00   76506/76506       add_acdc [26]
                0.00    0.00   51004/51004       get_dc_size_lum [34]
                0.00    0.00   76506/184051      DPRINTF [46]
                0.00    0.00   33610/33610       get_dc_dif [47]
                0.00    0.00   25502/25502       get_dc_size_chrom [49]
-----------------------------------------------
                0.14    0.00   76506/167121      decoder_mbintra [11]
                0.17    0.00   90615/167121      decoder_mb_decode [7]
[12]    19.7    0.32    0.00  167121         idct_int32 [12]
```

```
-----------------------------------------------
                0.01    0.23   10415/10415       decoder_bframe [8]
[13]    15.0    0.01    0.23   10415         decoder_bf_interpolate_mbinter [13]
                0.00    0.10    4796/21721       decoder_mb_decode [7]
                0.06    0.00   80667/136681      transfer8x8_copy_c [16]
                0.03    0.00   62490/62490       interpolate8x8_avg2_c [23]
                0.02    0.00   15137/55797       interpolate8x8_halfpel_hv_c [17]
                0.01    0.00   14259/32096       interpolate8x8_halfpel_h_c [27]
                0.01    0.00   14917/36378       interpolate8x8_halfpel_v_c [28]
-----------------------------------------------
                0.00    0.22      12/12          decoder_decode [3]
[14]    13.8    0.00    0.22      12         decoder_iframe [14]
                0.00    0.22    8208/12751       decoder_mbintra [11]
                0.00    0.00    8208/45828       check_resync_marker [37]
                0.00    0.00    8208/184051      DPRINTF [46]
                0.00    0.00    8208/8208        get_mcbpc_intra [50]
                0.00    0.00    8208/27780       get_cbpy [48]
-----------------------------------------------
                0.13    0.01   90615/90615       decoder_mb_decode [7]
[15]     9.1    0.13    0.01   90615         get_inter_block [15]
                0.01    0.00 1139150/1603803     DPRINTF [31]
-----------------------------------------------
                0.04    0.00   56014/136681      decoder_mbinter [6]
                0.06    0.00   80667/136681      decoder_bf_interpolate_mbinter [13]
[16]     6.5    0.10    0.00  136681         transfer8x8_copy_c [16]
-----------------------------------------------
                0.02    0.00   15137/55797       decoder_bf_interpolate_mbinter [13]
                0.04    0.00   40660/55797       decoder_mbinter [6]
[17]     3.8    0.06    0.00   55797         interpolate8x8_halfpel_hv_c [17]
-----------------------------------------------
                0.06    0.00   90615/90615       decoder_mb_decode [7]
[18]     3.5    0.06    0.00   90615         dequant_h263_inter_c [18]
-----------------------------------------------
                0.06    0.00   90615/90615       decoder_mb_decode [7]
[19]     3.5    0.06    0.00   90615         transfer_16to8add_c [19]
-----------------------------------------------
                0.05    0.00   76506/76506       decoder_mbintra [11]
[20]     3.0    0.05    0.00   76506         transfer_16to8copy_c [20]
-----------------------------------------------
                0.04    0.00   32110/32110       decoder_mbintra [11]
[21]     2.8    0.04    0.00   32110         get_intra_block [21]
                0.00    0.00  464653/1603803     DPRINTF [31]
-----------------------------------------------
                0.04    0.00   76506/76506       decoder_mbintra [11]
[22]     2.8    0.04    0.00   76506         dequant_h263_intra_c [22]
-----------------------------------------------
                0.03    0.00   62490/62490       decoder_bf_interpolate_mbinter [13]
[23]     2.1    0.03    0.00   62490         interpolate8x8_avg2_c [23]
-----------------------------------------------
                0.01    0.00      30/80          decoder_pframe [5]
                0.02    0.00      50/80          decoder_bframe [8]
[24]     1.9    0.03    0.00      80         image_setedges [24]
-----------------------------------------------
                0.03    0.00   76506/76506       decoder_mbintra [11]
[25]     1.7    0.03    0.00   76506         predict_acdc [25]
```

```
-----------------------------------------------
                0.02    0.00   76506/76506       decoder_mbintra [11]
[26]     1.3    0.02    0.00   76506         add_acdc [26]
                0.00    0.00  136608/136608      DPRINTF [36]
-----------------------------------------------
                0.01    0.00   14259/32096       decoder_bf_interpolate_mbinter [13]
                0.01    0.00   17837/32096       decoder_mbinter [6]
[27]     1.3    0.02    0.00   32096         interpolate8x8_halfpel_h_c [27]
-----------------------------------------------
                0.01    0.00   14917/36378       decoder_bf_interpolate_mbinter [13]
                0.01    0.00   21461/36378       decoder_mbinter [6]
[28]     1.2    0.02    0.00   36378         interpolate8x8_halfpel_v_c [28]
-----------------------------------------------
                0.00    0.02       1/1           main [1]
[29]     1.2    0.00    0.02       1         dec_init [29]
                0.00    0.02       1/84          xvid_decore [2]
                0.00    0.00       1/1           xvid_global [70]
-----------------------------------------------
                0.00    0.02       1/1           main [1]
[30]     1.2    0.00    0.02       1         dec_stop [30]
                0.00    0.02       1/84          xvid_decore [2]
-----------------------------------------------
                0.00    0.00  464653/1603803     get_intra_block [21]
                0.01    0.00 1139150/1603803     get_inter_block [15]
[31]     1.0    0.02    0.00 1603803         DPRINTF [31]
-----------------------------------------------
                0.01    0.00   20021/20021       decoder_pframe [5]
[32]     0.7    0.01    0.00   20021         get_pmv2 [32]
-----------------------------------------------
                0.00    0.00   30490/70532       decoder_bframe [8]
                0.00    0.00   40042/70532       decoder_pframe [5]
[33]     0.2    0.00    0.00   70532         get_mv [33]
-----------------------------------------------
                0.00    0.00   51004/51004       decoder_mbintra [11]
[34]     0.2    0.00    0.00   51004         get_dc_size_lum [34]
-----------------------------------------------
                0.00    0.00       1/1           xvid_decore [2]
[35]     0.2    0.00    0.00       1         decoder_create [35]
                0.00    0.00       1/1           init_postproc [42]
                0.00    0.00       2/23          xvid_malloc [58]
                0.00    0.00       1/1           init_mpeg_matrix [66]
-----------------------------------------------
                0.00    0.00  136608/136608      add_acdc [26]
[36]     0.1    0.00    0.00  136608         DPRINTF [36]
-----------------------------------------------
                0.00    0.00    8208/45828       decoder_iframe [14]
                0.00    0.00   17100/45828       decoder_bframe [8]
                0.00    0.00   20520/45828       decoder_pframe [5]
[37]     0.1    0.00    0.00   45828         check_resync_marker [37]
-----------------------------------------------
                0.00    0.00   19572/19572       decoder_pframe [5]
[38]     0.1    0.00    0.00   19572         get_mcbpc_inter [38]
-----------------------------------------------
                0.00    0.00      80/80          decoder_decode [3]
[39]     0.1    0.00    0.00      80         BitstreamReadHeaders [39]
```

```
                0.00    0.00     862/862         DPRINTF [51]
-----------------------------------------------
                0.00    0.00      67/67          main [1]
[40]     0.1    0.00    0.00      67          type2str [40]
-----------------------------------------------
                0.00    0.00       1/1           init_postproc [42]
[41]     0.1    0.00    0.00       1          init_noise [41]
                0.00    0.00       1/83          emms_c [54]
-----------------------------------------------
                0.00    0.00       1/1           decoder_create [35]
[42]     0.1    0.00    0.00       1          init_postproc [42]
                0.00    0.00       1/1           init_noise [41]
                0.00    0.00       1/1           init_deblock [65]
-----------------------------------------------
                                                 <spontaneous>
[43]     0.1    0.00    0.00                  get_qpmv2 [43]
-----------------------------------------------
                                                 <spontaneous>
[44]     0.1    0.00    0.00                  output_slice [44]
-----------------------------------------------
                                                 <spontaneous>
[45]     0.1    0.00    0.00                  transfer_8to16copy_c [45]
-----------------------------------------------
                0.00    0.00      80/184051      decoder_decode [3]
                0.00    0.00    8208/184051      decoder_iframe [14]
                0.00    0.00   76506/184051      decoder_mbintra [11]
                0.00    0.00   99257/184051      decoder_pframe [5]
[46]     0.0    0.00    0.00  184051          DPRINTF [46]
-----------------------------------------------
                0.00    0.00   33610/33610       decoder_mbintra [11]
[47]     0.0    0.00    0.00   33610          get_dc_dif [47]
-----------------------------------------------
                0.00    0.00    8208/27780       decoder_iframe [14]
                0.00    0.00   19572/27780       decoder_pframe [5]
[48]     0.0    0.00    0.00   27780          get_cbpy [48]
-----------------------------------------------
                0.00    0.00   25502/25502       decoder_mbintra [11]
[49]     0.0    0.00    0.00   25502          get_dc_size_chrom [49]
-----------------------------------------------
                0.00    0.00    8208/8208        decoder_iframe [14]
[50]     0.0    0.00    0.00    8208          get_mcbpc_intra [50]
-----------------------------------------------
                0.00    0.00     862/862         BitstreamReadHeaders [39]
[51]     0.0    0.00    0.00     862          DPRINTF [51]
-----------------------------------------------
                0.00    0.00     164/164         main [1]
[52]     0.0    0.00    0.00     164          msecond [52]
-----------------------------------------------
                0.00    0.00      84/84          decoder_decode [3]
[53]     0.0    0.00    0.00      84          image_swap [53]
-----------------------------------------------
                0.00    0.00       1/83          init_noise [41]
                0.00    0.00      82/83          decoder_decode [3]
[54]     0.0    0.00    0.00      83          emms_c [54]
-----------------------------------------------
```

```
                0.00    0.00      66/66          main [1]
[55]     0.0    0.00    0.00      66         write_image [55]
                0.00    0.00      66/66          write_tga [56]
-----------------------------------------------
                0.00    0.00      66/66          write_image [55]
[56]     0.0    0.00    0.00      66         write_tga [56]
-----------------------------------------------
                0.00    0.00       5/23          decoder_destroy [62]
                0.00    0.00      18/23          image_destroy [59]
[57]     0.0    0.00    0.00      23         xvid_free [57]
-----------------------------------------------
                0.00    0.00       2/23          decoder_create [35]
                0.00    0.00       3/23          decoder_resize [63]
                0.00    0.00      18/23          image_create [60]
[58]     0.0    0.00    0.00      23         xvid_malloc [58]
-----------------------------------------------
                0.00    0.00       6/12          decoder_resize [63]
                0.00    0.00       6/12          decoder_destroy [62]
[59]     0.0    0.00    0.00      12         image_destroy [59]
                0.00    0.00      18/23          xvid_free [57]
-----------------------------------------------
                0.00    0.00       6/6           decoder_resize [63]
[60]     0.0    0.00    0.00       6         image_create [60]
                0.00    0.00      18/23          xvid_malloc [58]
-----------------------------------------------
                0.00    0.00       1/1           xvid_gbl_init [69]
[61]     0.0    0.00    0.00       1         colorspace_init [61]
-----------------------------------------------
                0.00    0.00       1/1           xvid_decore [2]
[62]     0.0    0.00    0.00       1         decoder_destroy [62]
                0.00    0.00       6/12          image_destroy [59]
                0.00    0.00       5/23          xvid_free [57]
-----------------------------------------------
                0.00    0.00       1/1           decoder_decode [3]
[63]     0.0    0.00    0.00       1         decoder_resize [63]
                0.00    0.00       6/12          image_destroy [59]
                0.00    0.00       6/6           image_create [60]
                0.00    0.00       3/23          xvid_malloc [58]
-----------------------------------------------
                0.00    0.00       1/1           xvid_gbl_init [69]
[64]     0.0    0.00    0.00       1         idct_int32_init [64]
-----------------------------------------------
                0.00    0.00       1/1           init_postproc [42]
[65]     0.0    0.00    0.00       1         init_deblock [65]
-----------------------------------------------
                0.00    0.00       1/1           decoder_create [35]
[66]     0.0    0.00    0.00       1         init_mpeg_matrix [66]
-----------------------------------------------
                0.00    0.00       1/1           xvid_gbl_init [69]
[67]     0.0    0.00    0.00       1         init_vlc_tables [67]
-----------------------------------------------
                0.00    0.00       1/1           xvid_gbl_init [69]
[68]     0.0    0.00    0.00       1         xvid_Init_QP [68]
-----------------------------------------------
                0.00    0.00       1/1           xvid_global [70]
```

```
[69]     0.0    0.00    0.00       1         xvid_gbl_init [69]
                0.00    0.00       1/1           idct_int32_init [64]
                0.00    0.00       1/1           init_vlc_tables [67]
                0.00    0.00       1/1           xvid_Init_QP [68]
                0.00    0.00       1/1           colorspace_init [61]
-----------------------------------------------
                0.00    0.00       1/1           dec_init [29]
[70]     0.0    0.00    0.00       1         xvid_global [70]
                0.00    0.00       1/1           xvid_gbl_init [69]
-----------------------------------------------
```

This table describes the call tree of the program, and was sorted by the total amount of time spent in each function and its children.

Each entry in this table consists of several lines. The line with the index number at the left hand margin lists the current function. The lines above it list the functions that called this function, and the lines below it list the functions this one called. This line lists:

index A unique number given to each element of the table. Index numbers are sorted numerically. The index number is printed next to every function name so it is easier to look up where the function in the table.

% time This is the percentage of the 'total' time that was spent in this function and its children. Note that due to different viewpoints, functions excluded by options, etc, these numbers will NOT add up to 100%.

self This is the total amount of time spent in this function.

children This is the total amount of time propagated into this function by its children.

called This is the number of times the function was called. If the function called itself recursively, the number only includes non-recursive calls, and is followed by a '+' and the number of recursive calls.

name The name of the current function. The index number is printed after it. If the function is a member of a cycle, the cycle number is printed between the function's name and the index number.

For the function's parents, the fields have the following meanings:

self This is the amount of time that was propagated directly from the function into this parent.

children This is the amount of time that was propagated from the function's children into this parent.

called This is the number of times this parent called the function '/' the total number of times the function

```
was called.  Recursive calls to the function are not
included in the number after the '/'.

     name This is the name of the parent.  The parent's index
number is printed after it.  If the parent is a
member of a cycle, the cycle number is printed between
the name and the index number.

 If the parents of the function cannot be determined, the word
 '<spontaneous>' is printed in the 'name' field, and all the other
 fields are blank.

 For the function's children, the fields have the following meanings:

     self This is the amount of time that was propagated directly
from the child into the function.

     children This is the amount of time that was propagated from the
child's children to the function.

     called This is the number of times the function called
this child '/' the total number of times the child
was called.  Recursive calls by the child are not
listed in the number after the '/'.

     name This is the name of the child.  The child's index
number is printed after it.  If the child is a
member of a cycle, the cycle number is printed
between the name and the index number.

 If there are any cycles (circles) in the call graph, there is an
 entry for the cycle-as-a-whole.  This entry shows who called the
 cycle (as parents) and the members of the cycle (as children.)
 The '+' recursive calls entry shows the number of function calls that
 were internal to the cycle, and the calls entry for each member shows,
 for that member, how many times it was called from other members of
 the cycle.



Index by function name

  [39] BitstreamReadHeaders   [22] dequant_h263_intra_c   [42] init_postproc
  [31] DPRINTF (xvid_decraw.c) [54] emms_c                [67] init_vlc_tables
  [46] DPRINTF (xvid_decraw.c) [48] get_cbpy              [23] interpolate8x8_avg2_c
  [51] DPRINTF (xvid_decraw.c) [47] get_dc_dif            [27] interpolate8x8_halfpel_h_c
  [36] DPRINTF (xvid_decraw.c) [49] get_dc_size_chrom     [17] interpolate8x8_halfpel_hv_c
  [26] add_acdc               [34] get_dc_size_lum        [28] interpolate8x8_halfpel_v_c
  [37] check_resync_marker    [15] get_inter_block        [52] msecond (xvid_decraw.c)
  [61] colorspace_init        [21] get_intra_block        [44] output_slice
  [29] dec_init (xvid_decraw.c) [38] get_mcbpc_inter      [25] predict_acdc
   [4] dec_main (xvid_decraw.c) [50] get_mcbpc_intra      [16] transfer8x8_copy_c
  [30] dec_stop (xvid_decraw.c) [33] get_mv               [19] transfer_16to8add_c
  [13] decoder_bf_interpolate_mbinter (xvid_decraw.c) [32] get_pmv2 [20] transfer_16to8copy_c
   [8] decoder_bframe (xvid_decraw.c) [43] get_qpmv2      [45] transfer_8to16copy_c
```

```
[35] decoder_create         [12] idct_int32             [40] type2str
 [3] decoder_decode         [64] idct_int32_init        [55] write_image (xvid_decraw.c)
[62] decoder_destroy        [60] image_create           [56] write_tga (xvid_decraw.c)
[14] decoder_iframe (xvid_decraw.c) [59] image_destroy  [68] xvid_Init_QP
 [7] decoder_mb_decode (xvid_decraw.c) [9] image_output  [2] xvid_decore
 [6] decoder_mbinter (xvid_decraw.c) [24] image_setedges [57] xvid_free
[11] decoder_mbintra (xvid_decraw.c) [53] image_swap    [69] xvid_gbl_init (xvid_decraw.c)
 [5] decoder_pframe (xvid_decraw.c) [65] init_deblock   [70] xvid_global
[63] decoder_resize (xvid_decraw.c) [66] init_mpeg_matrix [58] xvid_malloc
[18] dequant_h263_inter_c   [41] init_noise             [10] yv12_to_bgr_c
```

IDCT datapath and controlling FSM

D

This appendix contains the Quartus designs of the row/col unit, which is part of the hardware IDCT, the IDCT datapath and the FSM controlling the IDCT datapath. It performs the 1-dimensional IDCT on a row or column. Figure D.1 shows the design of the row/col unit. Figure D.2 shows the details of the part "IDCT_DEEL" which is visible in figure D.1. For clarity the figures are shown on separate pages.

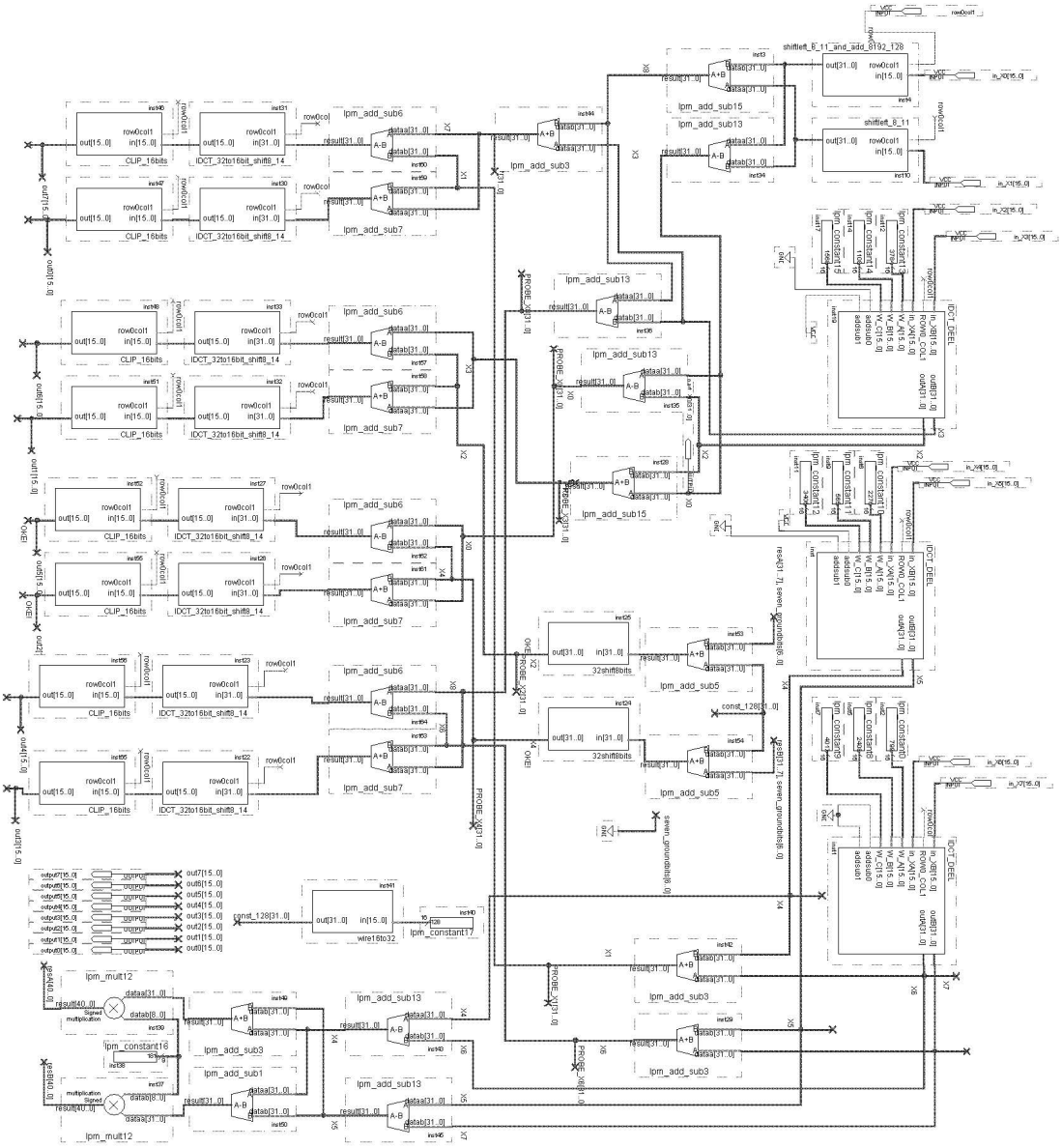


Figure D.1: Quartus design of the row/col unit.

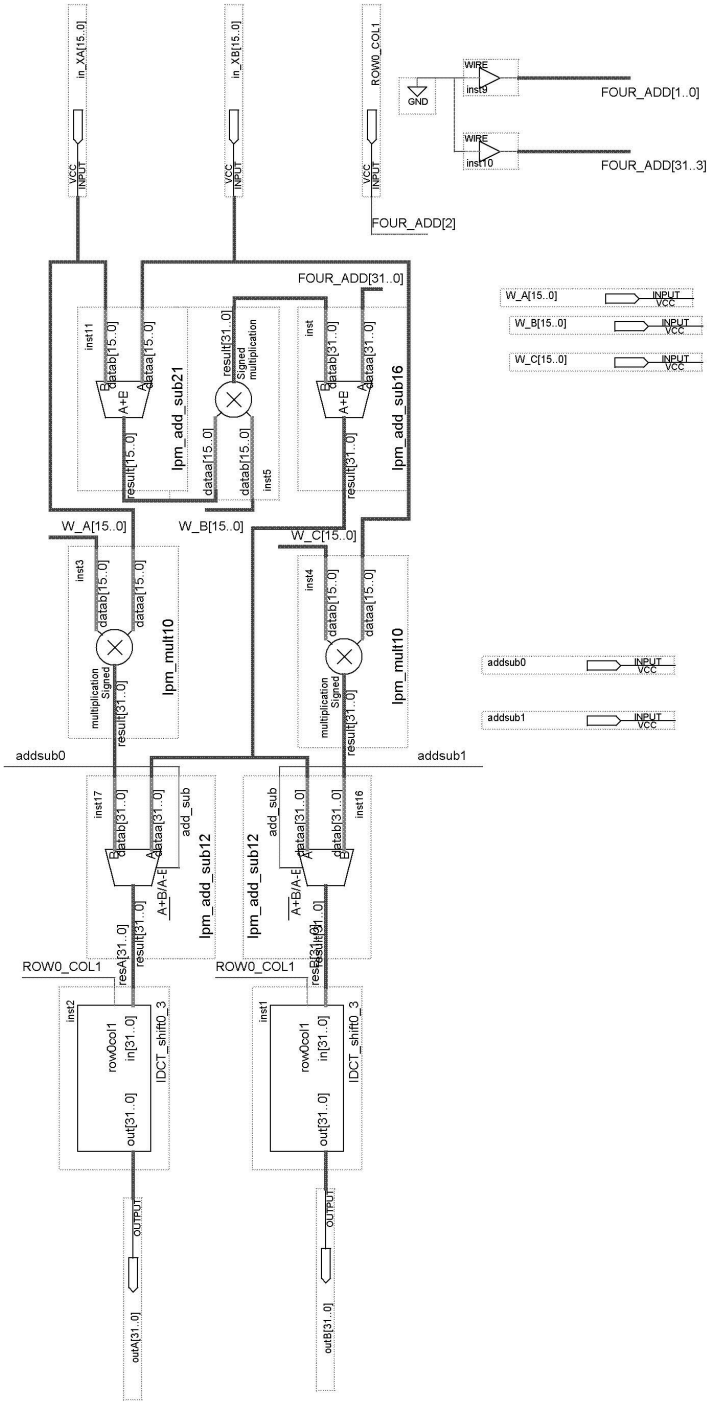


Figure D.2: Quartus design of "IDCT_DEEL" block which is part of the row/col unit.

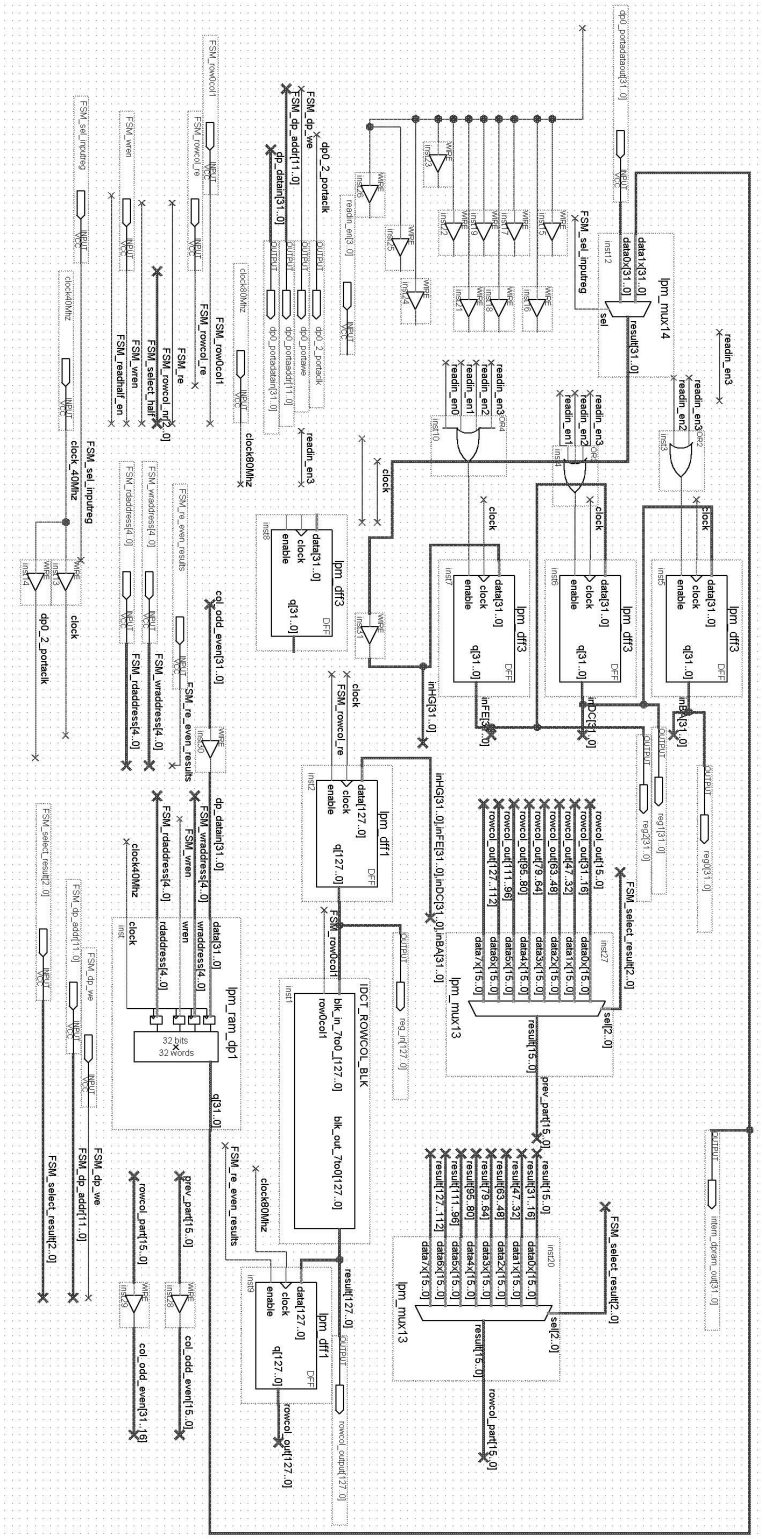


Figure D.3: Quartus design of the IDCT datapath.

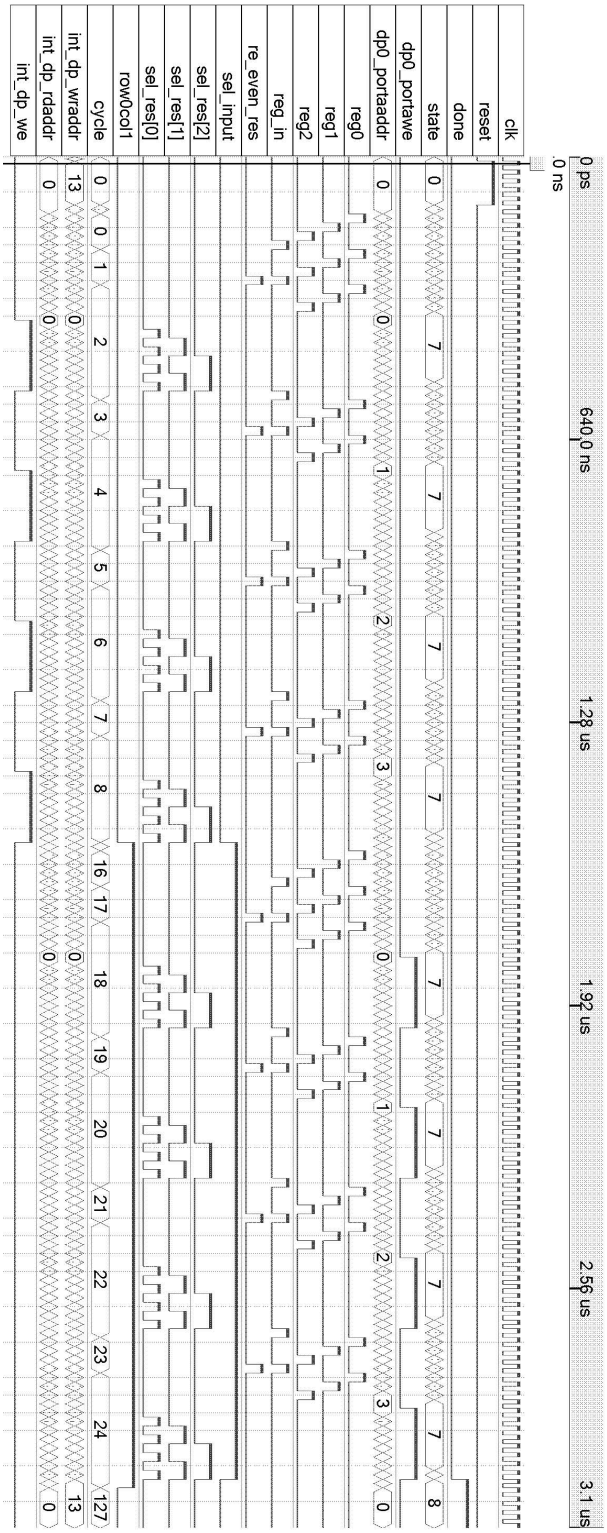


Figure D.4: Simulation of all the signals originating from the FSM controlling the IDCT datapath.

Source code of XviD decoder: decoder.c

E

This appendix contains part of the XviD source code containing the decoding functions. The file listed is `decoder.c` from the `src` directory. It is available under the GNU General Public License, which is shown in appendix F.

```
/*****************************************************************************
 *
 *  XVID MPEG-4 VIDEO CODEC
 *  - Decoder Module -
 *
 *  Copyright(C) 2002      MinChen <chenm001@163.com>
 *               2002-2003 Peter Ross <pross@xvid.org>
 *
 *  This program is free software ; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation ; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY ; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program ; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307 USA
 *
 * $Id$
 *
 ****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifdef BFRAMES_DEC_DEBUG
#define BFRAMES_DEC
#endif

#include "xvid.h"
#include "portab.h"
#include "global.h"

#include "decoder.h"
#include "bitstream/bitstream.h"
#include "bitstream/mbcoding.h"

#include "quant/quant.h"
#include "quant/quant_matrix.h"
#include "dct/idct.h"
#include "dct/fdct.h"
#include "utils/mem_transfer.h"
#include "image/interpolate8x8.h"
#include "image/reduced.h"
#include "image/font.h"

#include "bitstream/mbcoding.h"
#include "prediction/mbprediction.h"
#include "utils/timer.h"
#include "utils/emms.h"
#include "motion/motion.h"
#include "motion/gmc.h"

#include "image/image.h"
#include "image/colorspace.h"
#include "image/postprocessing.h"
#include "utils/mem_align.h"

static int
decoder_resize(DECODER * dec)
{
/* free existing */
image_destroy(&dec->cur, dec->edged_width, dec->edged_height);
image_destroy(&dec->refn[0], dec->edged_width, dec->edged_height);
```

```
image_destroy(&dec->refn[1], dec->edged_width, dec->edged_height);
image_destroy(&dec->tmp, dec->edged_width, dec->edged_height);
image_destroy(&dec->qtmp, dec->edged_width, dec->edged_height);

image_destroy(&dec->gmc, dec->edged_width, dec->edged_height);

if (dec->last_mbs)
xvid_free(dec->last_mbs);
if (dec->mbs)
xvid_free(dec->mbs);
if (dec->qscale)
xvid_free(dec->qscale);

/* realloc */
dec->mb_width = (dec->width + 15) / 16;
dec->mb_height = (dec->height + 15) / 16;

dec->edged_width = 16 * dec->mb_width + 2 * EDGE_SIZE;
dec->edged_height = 16 * dec->mb_height + 2 * EDGE_SIZE;

if (image_create(&dec->cur, dec->edged_width, dec->edged_height)) {
xvid_free(dec);
return XVID_ERR_MEMORY;
}

if (image_create(&dec->refn[0], dec->edged_width, dec->edged_height)) {
image_destroy(&dec->cur, dec->edged_width, dec->edged_height);
xvid_free(dec);
return XVID_ERR_MEMORY;
}

/* Support B-frame to reference last 2 frame */
if (image_create(&dec->refn[1], dec->edged_width, dec->edged_height)) {
image_destroy(&dec->cur, dec->edged_width, dec->edged_height);
image_destroy(&dec->refn[0], dec->edged_width, dec->edged_height);
xvid_free(dec);
return XVID_ERR_MEMORY;
}
if (image_create(&dec->tmp, dec->edged_width, dec->edged_height)) {
image_destroy(&dec->cur, dec->edged_width, dec->edged_height);
image_destroy(&dec->refn[0], dec->edged_width, dec->edged_height);
image_destroy(&dec->refn[1], dec->edged_width, dec->edged_height);
xvid_free(dec);
return XVID_ERR_MEMORY;
}

if (image_create(&dec->qtmp, dec->edged_width, dec->edged_height)) {
image_destroy(&dec->cur, dec->edged_width, dec->edged_height);
image_destroy(&dec->refn[0], dec->edged_width, dec->edged_height);
image_destroy(&dec->refn[1], dec->edged_width, dec->edged_height);
image_destroy(&dec->tmp, dec->edged_width, dec->edged_height);
xvid_free(dec);
return XVID_ERR_MEMORY;
}

if (image_create(&dec->gmc, dec->edged_width, dec->edged_height)) {
image_destroy(&dec->qtmp, dec->edged_width, dec->edged_height);
image_destroy(&dec->cur, dec->edged_width, dec->edged_height);
image_destroy(&dec->refn[0], dec->edged_width, dec->edged_height);
image_destroy(&dec->refn[1], dec->edged_width, dec->edged_height);
image_destroy(&dec->tmp, dec->edged_width, dec->edged_height);
xvid_free(dec);
return XVID_ERR_MEMORY;
}

dec->mbs =
xvid_malloc(sizeof(MACROBLOCK) * dec->mb_width * dec->mb_height,
CACHE_LINE);
if (dec->mbs == NULL) {
image_destroy(&dec->cur, dec->edged_width, dec->edged_height);
image_destroy(&dec->refn[0], dec->edged_width, dec->edged_height);
image_destroy(&dec->refn[1], dec->edged_width, dec->edged_height);
image_destroy(&dec->tmp, dec->edged_width, dec->edged_height);
image_destroy(&dec->qtmp, dec->edged_width, dec->edged_height);
xvid_free(dec);
return XVID_ERR_MEMORY;
}
memset(dec->mbs, 0, sizeof(MACROBLOCK) * dec->mb_width * dec->mb_height);

/* For skip MB flag */
dec->last_mbs =
xvid_malloc(sizeof(MACROBLOCK) * dec->mb_width * dec->mb_height,
CACHE_LINE);
if (dec->last_mbs == NULL) {
xvid_free(dec->mbs);
image_destroy(&dec->cur, dec->edged_width, dec->edged_height);
image_destroy(&dec->refn[0], dec->edged_width, dec->edged_height);
```

```
image_destroy(&dec->refn[1], dec->edged_width, dec->edged_height);
image_destroy(&dec->tmp, dec->edged_width, dec->edged_height);
image_destroy(&dec->qtmp, dec->edged_width, dec->edged_height);
xvid_free(dec);
return XVID_ERR_MEMORY;
}

memset(dec->last_mbs, 0, sizeof(MACROBLOCK) * dec->mb_width * dec->mb_height);

/* nothing happens if that fails */
dec->qscale =
xvid_malloc(sizeof(int) * dec->mb_width * dec->mb_height, CACHE_LINE);

if (dec->qscale)
memset(dec->qscale, 0, sizeof(int) * dec->mb_width * dec->mb_height);

return 0;
}


int
decoder_create(xvid_dec_create_t * create)
{
DECODER *dec;

DAMP_DEBUG(printf("decoder_create\r\n");)
if (XVID_VERSION_MAJOR(create->version) != 1) /* v1.x.x */
return XVID_ERR_VERSION;

DAMP_DEBUG(printf("decoder_create: malloc\r\n");)
dec = xvid_malloc(sizeof(DECODER), CACHE_LINE);
if (dec == NULL) {
return XVID_ERR_MEMORY;
}

DAMP_DEBUG(printf("decoder_create: memset\r\n");)
memset(dec, 0, sizeof(DECODER));

DAMP_DEBUG(printf("decoder_create: xvid_malloc\r\n");)
dec->mpeg_quant_matrices = xvid_malloc(sizeof(uint16_t) * 64 * 8, CACHE_LINE);
if (dec->mpeg_quant_matrices == NULL) {
xvid_free(dec);
return XVID_ERR_MEMORY;
}

DAMP_DEBUG(printf("decoder_create: handle\r\n");)
create->handle = dec;

dec->width = create->width;
dec->height = create->height;

image_null(&dec->cur);
image_null(&dec->refn[0]);
image_null(&dec->refn[1]);
image_null(&dec->tmp);
image_null(&dec->qtmp);

/* image based GMC */
image_null(&dec->gmc);

DAMP_DEBUG(printf("decoder_create: mbs\r\n");)
dec->mbs = NULL;
dec->last_mbs = NULL;
dec->qscale = NULL;

DAMP_DEBUG(printf("decoder_create: init_timer\r\n");)
init_timer();
DAMP_DEBUG(printf("decoder_create: postproc\r\n");)
init_postproc(&dec->postproc);
DAMP_DEBUG(printf("decoder_create: init_mpeg_matrix\r\n");)
init_mpeg_matrix(dec->mpeg_quant_matrices);

DAMP_DEBUG(printf("decoder_create: bframe\r\n");)
/* For B-frame support (used to save reference frame's time */
dec->frames = 0;
dec->time = dec->time_base = dec->last_time_base = 0;
dec->low_delay = 0;
dec->packed_mode = 0;
dec->time_inc_resolution = 1; /* until VOL header says otherwise */

DAMP_DEBUG(printf("decoder_create: dim\r\n");)
dec->fixed_dimensions = (dec->width > 0 && dec->height > 0);


DAMP_DEBUG(printf("decoder_create: return\r\n");)

if (dec->fixed_dimensions)
```

```
return decoder_resize(dec);
else
return 0;
}


int
decoder_destroy(DECODER * dec)
{
xvid_free(dec->last_mbs);
xvid_free(dec->mbs);
xvid_free(dec->qscale);

/* image based GMC */
image_destroy(&dec->gmc, dec->edged_width, dec->edged_height);

image_destroy(&dec->refn[0], dec->edged_width, dec->edged_height);
image_destroy(&dec->refn[1], dec->edged_width, dec->edged_height);
image_destroy(&dec->tmp, dec->edged_width, dec->edged_height);
image_destroy(&dec->qtmp, dec->edged_width, dec->edged_height);
image_destroy(&dec->cur, dec->edged_width, dec->edged_height);
xvid_free(dec->mpeg_quant_matrices);
xvid_free(dec);

write_timer();
return 0;
}

static const int32_t dquant_table[4] = {
-1, -2, 1, 2
};

/* decode an intra macroblock */
static void
decoder_mbintra(DECODER * dec,
MACROBLOCK * pMB,
const uint32_t x_pos,
const uint32_t y_pos,
const uint32_t acpred_flag,
const uint32_t cbp,
Bitstream * bs,
const uint32_t quant,
const uint32_t intra_dc_threshold,
const unsigned int bound,
const int reduced_resolution)
{

DECLARE_ALIGNED_MATRIX(block, 6, 64, int16_t, CACHE_LINE);
DECLARE_ALIGNED_MATRIX(data, 6, 64, int16_t, CACHE_LINE);

uint32_t stride = dec->edged_width;
uint32_t stride2 = stride / 2;
uint32_t next_block = stride * 8;
uint32_t i;
uint32_t iQuant = pMB->quant;
uint8_t *pY_Cur, *pU_Cur, *pV_Cur;

if (reduced_resolution) {
pY_Cur = dec->cur.y + (y_pos << 5) * stride + (x_pos << 5);
pU_Cur = dec->cur.u + (y_pos << 4) * stride2 + (x_pos << 4);
pV_Cur = dec->cur.v + (y_pos << 4) * stride2 + (x_pos << 4);
}else{
pY_Cur = dec->cur.y + (y_pos << 4) * stride + (x_pos << 4);
pU_Cur = dec->cur.u + (y_pos << 3) * stride2 + (x_pos << 3);
pV_Cur = dec->cur.v + (y_pos << 3) * stride2 + (x_pos << 3);
}

memset(block, 0, 6 * 64 * sizeof(int16_t)); /* clear */

for (i = 0; i < 6; i++) {
uint32_t iDcScaler = get_dc_scaler(iQuant, i < 4);
int16_t predictors[8];
int start_coeff;

start_timer();
predict_acdc(dec->mbs, x_pos, y_pos, dec->mb_width, i, &block[i * 64],
 iQuant, iDcScaler, predictors, bound);
if (!acpred_flag) {
pMB->acpred_directions[i] = 0;
}
stop_prediction_timer();

if (quant < intra_dc_threshold) {
int dc_size;
int dc_dif;

dc_size = i < 4 ? get_dc_size_lum(bs) : get_dc_size_chrom(bs);
```

```
dc_dif = dc_size ? get_dc_dif(bs, dc_size) : 0;

if (dc_size > 8) {
BitstreamSkip(bs, 1); /* marker */
}

block[i * 64 + 0] = dc_dif;
start_coeff = 1;

DPRINTF(XVID_DEBUG_COEFF,"block[0] %i\n", dc_dif);
} else {
start_coeff = 0;
}

start_timer();
if (cbp & (1 << (5 - i))) /* coded */
{
int direction = dec->alternate_vertical_scan ?
2 : pMB->acpred_directions[i];

get_intra_block(bs, &block[i * 64], direction, start_coeff);
}
stop_coding_timer();

start_timer();
add_acdc(pMB, i, &block[i * 64], iDcScaler, predictors, dec->bs_version);
stop_prediction_timer();

start_timer();
if (dec->quant_type == 0) {
dequant_h263_intra(&data[i * 64], &block[i * 64], iQuant, iDcScaler, dec->mpeg_quant_matrices);
} else {
dequant_mpeg_intra(&data[i * 64], &block[i * 64], iQuant, iDcScaler, dec->mpeg_quant_matrices);
}
stop_iquant_timer();

start_timer();
//DAMP
if (USE_HARDWARE) {
    #if defined(STANDALONE)
        //us_hardware_idct(data,data);
        us_hardware_idct(&data[i * 64],&data[i * 64]);
    #elif defined(LINUX_USER)
        us_hardware_idct_ioctl(&data[i * 64]);
    #else
        idct(&data[i * 64]);
    #endif
        } else {
          idct(&data[i * 64]);
        }
//DAMP idct(&data[i * 64]);
stop_idct_timer();

}

if (dec->interlacing && pMB->field_dct) {
next_block = stride;
stride *= 2;
}

start_timer();

if (reduced_resolution)
{
next_block*=2;
copy_upsampled_8x8_16to8(pY_Cur, &data[0 * 64], stride);
copy_upsampled_8x8_16to8(pY_Cur + 16, &data[1 * 64], stride);
copy_upsampled_8x8_16to8(pY_Cur + next_block, &data[2 * 64], stride);
copy_upsampled_8x8_16to8(pY_Cur + 16 + next_block, &data[3 * 64], stride);
copy_upsampled_8x8_16to8(pU_Cur, &data[4 * 64], stride2);
copy_upsampled_8x8_16to8(pV_Cur, &data[5 * 64], stride2);
}else{
transfer_16to8copy(pY_Cur, &data[0 * 64], stride);
transfer_16to8copy(pY_Cur + 8, &data[1 * 64], stride);
transfer_16to8copy(pY_Cur + next_block, &data[2 * 64], stride);
transfer_16to8copy(pY_Cur + 8 + next_block, &data[3 * 64], stride);
transfer_16to8copy(pU_Cur, &data[4 * 64], stride2);
transfer_16to8copy(pV_Cur, &data[5 * 64], stride2);
}
stop_transfer_timer();
}


/*DAMP DAMP DAMP DAMP */
#define EX_GPIO      (*(volatile unsigned long int *)(EX_REG_BASE))
#include "dct/hardware_idct_user.h"
#include "dct/hardware_idct_user_ioctl.h"
```

```
static void
put_into_damp(Bitstream * bs, int direction, const quant_interFuncPtr dequant,
int16_t * data, const uint32_t quant, const uint16_t * mpeg_quant_matrices)
{
short array[64];
int timeout=350000;
int number_of_times = 10000;
DECLARE_ALIGNED_MATRIX(block, 1, 64, int16_t, CACHE_LINE);
memset(block, 0, 64 * sizeof(int16_t)); /* clear */

start_timer();
get_inter_block(bs, block, direction);
stop_coding_timer();

start_timer();
dequant(data, block, quant, mpeg_quant_matrices);
stop_iquant_timer();

start_timer();
  #if 0
/*   { static FILE *array_out;
  static int file_counter;
  int i;
*/   /* write number_of_times IDCT results to file */
/*     if (file_counter == 0)
     { if ((array_out = fopen("array.h", "w")) == 0)
     {
     printf("Error opening output file (fatal)");
file_counter = number_of_times + 1;
     }
    else
    {
    fprintf(array_out, "unsigned long array[32*%i+1] = {", number_of_times); *//* 1 extra for last 0 value */
/*     }
  }
     if (file_counter<number_of_times)
     {

       for (i=0;i<32;i++) fprintf(array_out, "0x%.8lx, ",((unsigned long*)data)[i]);
       file_counter++;
  }
  if (file_counter == number_of_times)
  */{
  //   fprintf(array_out, "0 }\n *//*end of array with %i times 64 input values (2 values per 32 bit word) for IDCT*/", number_of_times);
/*      fclose(array_out);
        file_counter++;
     }
}
*/
  #endif 0
if (USE_HARDWARE) {
    #if defined(STANDALONE)
        //us_hardware_idct(data,data);
        us_hardware_idct(data,data);
    #elif defined(LINUX_USER)
        us_hardware_idct_ioctl(data);
    #else
        idct(data);
    #endif

      #if 0
/*hardware part*/
/*   unsigned long temp;
  int i;
  timeout=350000;


      //memcpy(DPRAM, data, 128); /*copy 128 bytes to DPRAM*/
/*   for (i=0;i<32;i++) DPRAM[i]=((unsigned long*)data)[i];
  EX_GPIO = 0x01; /* reset IDCT */
/*   temp = EX_GPIO;
     while (temp & 0x00010000)  temp = EX_GPIO;
  //while (EX_GPIO & 0x00010000) /*wait to be sure circuit has been reset.*/
  // {};
/*   EX_GPIO = 0x00; /*start the IDCT by releasing reset*/
/*   temp = EX_GPIO;
     while ( !(temp & 0x00010000)) temp = EX_GPIO;
  for(i=0;i<350000;i++)
//     { /*wait until done signal received*/
/*        temp = EX_GPIO;
       if (!temp) temp = EX_GPIO;
//        if(!(temp & 0x00010000)) break;
  //    timeout--;
//     }
  //memcpy(data, DPRAM, 128); /*copy 128 bytes back from DPRAM*/
/*   for (i=0;i<32;i++) ((unsigned long*)array)[i]=DPRAM[i];
```

```
  idct(data);
  timeout = 0;
  for (i=0;i<64;i++) if (array[i] != data[i]) timeout++;
  if (timeout) printf("FOUT ");
  EX_GPIO = 0x01; /* reset IDCT */
/* End of hardware part -> idct(data);*/
   #endif 0
        } else {
          idct(data);
        }
stop_idct_timer();
}



static void
decoder_mb_decode(DECODER * dec,
const uint32_t cbp,
Bitstream * bs,
uint8_t * pY_Cur,
uint8_t * pU_Cur,
uint8_t * pV_Cur,
const int reduced_resolution,
const MACROBLOCK * pMB)
{
/*DECLARE_ALIGNED_MATRIX(block, 1, 64, int16_t, CACHE_LINE);*/
DECLARE_ALIGNED_MATRIX(data, 6, 64, int16_t, CACHE_LINE);

int stride = dec->edged_width;
int next_block = stride * (reduced_resolution ? 16 : 8);
const int stride2 = stride/2;
int i;
const uint32_t iQuant = pMB->quant;
const int direction = dec->alternate_vertical_scan ? 2 : 0;
const quant_interFuncPtr dequant = dec->quant_type == 0 ? dequant_h263_inter : dequant_mpeg_inter;

/* PUT INTO DAMP */

for (i = 0; i < 6; i++) {

if (cbp & (1 << (5 - i))) { /* coded */

put_into_damp(bs, direction, dequant, &data[i * 64],
 iQuant, dec->mpeg_quant_matrices);

/*memset(block, 0, 64 * sizeof(int16_t));*/ /* clear */
/*
start_timer();
get_inter_block(bs, block, direction);
stop_coding_timer();

start_timer();
dequant(&data[i * 64], block, iQuant, dec->mpeg_quant_matrices);
stop_iquant_timer();

start_timer();
idct(&data[i * 64]);
stop_idct_timer(); */
}
}

/* END OF DAMP */

if (dec->interlacing && pMB->field_dct) {
next_block = stride;
stride *= 2;
}

start_timer();
if (reduced_resolution) {
if (cbp & 32)
add_upsampled_8x8_16to8(pY_Cur, &data[0 * 64], stride);
if (cbp & 16)
add_upsampled_8x8_16to8(pY_Cur + 16, &data[1 * 64], stride);
if (cbp & 8)
add_upsampled_8x8_16to8(pY_Cur + next_block, &data[2 * 64], stride);
if (cbp & 4)
add_upsampled_8x8_16to8(pY_Cur + 16 + next_block, &data[3 * 64], stride);
if (cbp & 2)
add_upsampled_8x8_16to8(pU_Cur, &data[4 * 64], stride2);
if (cbp & 1)
add_upsampled_8x8_16to8(pV_Cur, &data[5 * 64], stride2);
} else {
if (cbp & 32)
transfer_16to8add(pY_Cur, &data[0 * 64], stride);
if (cbp & 16)
transfer_16to8add(pY_Cur + 8, &data[1 * 64], stride);
```

```
if (cbp & 8)
transfer_16to8add(pY_Cur + next_block, &data[2 * 64], stride);
if (cbp & 4)
transfer_16to8add(pY_Cur + 8 + next_block, &data[3 * 64], stride);
if (cbp & 2)
transfer_16to8add(pU_Cur, &data[4 * 64], stride2);
if (cbp & 1)
transfer_16to8add(pV_Cur, &data[5 * 64], stride2);
}
stop_transfer_timer();
}

/* decode an inter macroblock */
static void
decoder_mbinter(DECODER * dec,
const MACROBLOCK * pMB,
const uint32_t x_pos,
const uint32_t y_pos,
const uint32_t cbp,
Bitstream * bs,
const uint32_t rounding,
const int reduced_resolution,
const int ref)
{
uint32_t stride = dec->edged_width;
uint32_t stride2 = stride / 2;
uint32_t i;

uint8_t *pY_Cur, *pU_Cur, *pV_Cur;

int uv_dx, uv_dy;
VECTOR mv[4]; /* local copy of mvs */

if (reduced_resolution) {
pY_Cur = dec->cur.y + (y_pos << 5) * stride + (x_pos << 5);
pU_Cur = dec->cur.u + (y_pos << 4) * stride2 + (x_pos << 4);
pV_Cur = dec->cur.v + (y_pos << 4) * stride2 + (x_pos << 4);
for (i = 0; i < 4; i++) {
mv[i].x = RRV_MV_SCALEUP(pMB->mvs[i].x);
mv[i].y = RRV_MV_SCALEUP(pMB->mvs[i].y);
}
} else {
pY_Cur = dec->cur.y + (y_pos << 4) * stride + (x_pos << 4);
pU_Cur = dec->cur.u + (y_pos << 3) * stride2 + (x_pos << 3);
pV_Cur = dec->cur.v + (y_pos << 3) * stride2 + (x_pos << 3);
for (i = 0; i < 4; i++)
mv[i] = pMB->mvs[i];
}

for (i = 0; i < 4; i++) {
/* clip to valid range */
int border = (int)(dec->mb_width - x_pos) << (5 + dec->quarterpel);
if (mv[i].x > border) {
DPRINTF(XVID_DEBUG_MV, "mv.x > max -- %d > %d, MB %d, %d", mv[i].x, border, x_pos, y_pos);
mv[i].x = border;
} else {
border = (-(int)x_pos-1) << (5 + dec->quarterpel);
if (mv[i].x < border) {
DPRINTF(XVID_DEBUG_MV, "mv.x < min -- %d < %d, MB %d, %d", mv[i].x, border, x_pos, y_pos);
mv[i].x = border;
}
}

border = (int)(dec->mb_height - y_pos) << (5 + dec->quarterpel);
if (mv[i].y >  border) {
DPRINTF(XVID_DEBUG_MV, "mv.y > max -- %d > %d, MB %d, %d", mv[i].y, border, x_pos, y_pos);
mv[i].y = border;
} else {
border = (-(int)y_pos-1) << (5 + dec->quarterpel);
if (mv[i].y < border) {
DPRINTF(XVID_DEBUG_MV, "mv.y < min -- %d < %d, MB %d, %d", mv[i].y, border, x_pos, y_pos);
mv[i].y = border;
}
}
}

start_timer();

if (pMB->mode != MODE_INTER4V) { /* INTER, INTER_Q, NOT_CODED, FORWARD, BACKWARD */

uv_dx = mv[0].x;
uv_dy = mv[0].y;
if (dec->quarterpel) {
uv_dx /= 2;
uv_dy /= 2;
}
uv_dx = (uv_dx >> 1) + roundtab_79[uv_dx & 0x3];
```

```
uv_dy = (uv_dy >> 1) + roundtab_79[uv_dy & 0x3];

if (reduced_resolution)
interpolate32x32_switch(dec->cur.y, dec->refn[0].y, 32*x_pos, 32*y_pos,
mv[0].x, mv[0].y, stride, rounding);
else if (dec->quarterpel)
interpolate16x16_quarterpel(dec->cur.y, dec->refn[ref].y, dec->qtmp.y, dec->qtmp.y + 64,
  dec->qtmp.y + 128, 16*x_pos, 16*y_pos,
mv[0].x, mv[0].y, stride, rounding);
else
interpolate16x16_switch(dec->cur.y, dec->refn[ref].y, 16*x_pos, 16*y_pos,
mv[0].x, mv[0].y, stride, rounding);

} else { /* MODE_INTER4V */

if(dec->quarterpel) {
uv_dx = (mv[0].x / 2) + (mv[1].x / 2) + (mv[2].x / 2) + (mv[3].x / 2);
uv_dy = (mv[0].y / 2) + (mv[1].y / 2) + (mv[2].y / 2) + (mv[3].y / 2);
} else {
uv_dx = mv[0].x + mv[1].x + mv[2].x + mv[3].x;
uv_dy = mv[0].y + mv[1].y + mv[2].y + mv[3].y;
}

uv_dx = (uv_dx >> 3) + roundtab_76[uv_dx & 0xf];
uv_dy = (uv_dy >> 3) + roundtab_76[uv_dy & 0xf];

if (reduced_resolution) {
interpolate16x16_switch(dec->cur.y, dec->refn[0].y, 32*x_pos, 32*y_pos,
mv[0].x, mv[0].y, stride, rounding);
interpolate16x16_switch(dec->cur.y, dec->refn[0].y , 32*x_pos + 16, 32*y_pos,
mv[1].x, mv[1].y, stride, rounding);
interpolate16x16_switch(dec->cur.y, dec->refn[0].y , 32*x_pos, 32*y_pos + 16,
mv[2].x, mv[2].y, stride, rounding);
interpolate16x16_switch(dec->cur.y, dec->refn[0].y , 32*x_pos + 16, 32*y_pos + 16,
mv[3].x, mv[3].y, stride, rounding);
interpolate16x16_switch(dec->cur.u, dec->refn[0].u , 16 * x_pos, 16 * y_pos,
uv_dx, uv_dy, stride2, rounding);
interpolate16x16_switch(dec->cur.v, dec->refn[0].v , 16 * x_pos, 16 * y_pos,
uv_dx, uv_dy, stride2, rounding);

} else if (dec->quarterpel) {
interpolate8x8_quarterpel(dec->cur.y, dec->refn[0].y , dec->qtmp.y, dec->qtmp.y + 64,
dec->qtmp.y + 128, 16*x_pos, 16*y_pos,
mv[0].x, mv[0].y, stride, rounding);
interpolate8x8_quarterpel(dec->cur.y, dec->refn[0].y , dec->qtmp.y, dec->qtmp.y + 64,
dec->qtmp.y + 128, 16*x_pos + 8, 16*y_pos,
mv[1].x, mv[1].y, stride, rounding);
interpolate8x8_quarterpel(dec->cur.y, dec->refn[0].y , dec->qtmp.y, dec->qtmp.y + 64,
dec->qtmp.y + 128, 16*x_pos, 16*y_pos + 8,
mv[2].x, mv[2].y, stride, rounding);
interpolate8x8_quarterpel(dec->cur.y, dec->refn[0].y , dec->qtmp.y, dec->qtmp.y + 64,
dec->qtmp.y + 128, 16*x_pos + 8, 16*y_pos + 8,
mv[3].x, mv[3].y, stride, rounding);
} else {
interpolate8x8_switch(dec->cur.y, dec->refn[0].y , 16*x_pos, 16*y_pos,
mv[0].x, mv[0].y, stride, rounding);
interpolate8x8_switch(dec->cur.y, dec->refn[0].y , 16*x_pos + 8, 16*y_pos,
mv[1].x, mv[1].y, stride, rounding);
interpolate8x8_switch(dec->cur.y, dec->refn[0].y , 16*x_pos, 16*y_pos + 8,
mv[2].x, mv[2].y, stride, rounding);
interpolate8x8_switch(dec->cur.y, dec->refn[0].y , 16*x_pos + 8, 16*y_pos + 8,
mv[3].x, mv[3].y, stride, rounding);
}
}

/* chroma */
if (reduced_resolution) {
interpolate16x16_switch(dec->cur.u, dec->refn[0].u, 16 * x_pos, 16 * y_pos,
uv_dx, uv_dy, stride2, rounding);
interpolate16x16_switch(dec->cur.v, dec->refn[0].v, 16 * x_pos, 16 * y_pos,
uv_dx, uv_dy, stride2, rounding);
} else {
interpolate8x8_switch(dec->cur.u, dec->refn[ref].u, 8 * x_pos, 8 * y_pos,
uv_dx, uv_dy, stride2, rounding);
interpolate8x8_switch(dec->cur.v, dec->refn[ref].v, 8 * x_pos, 8 * y_pos,
uv_dx, uv_dy, stride2, rounding);
}

stop_comp_timer();

if (cbp)
decoder_mb_decode(dec, cbp, bs, pY_Cur, pU_Cur, pV_Cur,
reduced_resolution, pMB);
}

static void
decoder_mbgmc(DECODER * dec,
```

```
MACROBLOCK * const pMB,
const uint32_t x_pos,
const uint32_t y_pos,
const uint32_t fcode,
const uint32_t cbp,
Bitstream * bs,
const uint32_t rounding)
{
const uint32_t stride = dec->edged_width;
const uint32_t stride2 = stride / 2;

uint8_t *const pY_Cur=dec->cur.y + (y_pos << 4) * stride + (x_pos << 4);
uint8_t *const pU_Cur=dec->cur.u + (y_pos << 3) * stride2 + (x_pos << 3);
uint8_t *const pV_Cur=dec->cur.v + (y_pos << 3) * stride2 + (x_pos << 3);

NEW_GMC_DATA * gmc_data = &dec->new_gmc_data;

pMB->mvs[0] = pMB->mvs[1] = pMB->mvs[2] = pMB->mvs[3] = pMB->amv;

start_timer();

/* this is where the calculations are done */

gmc_data->predict_16x16(gmc_data,
dec->cur.y + y_pos*16*stride + x_pos*16, dec->refn[0].y,
stride, stride, x_pos, y_pos, rounding);

gmc_data->predict_8x8(gmc_data,
dec->cur.u + y_pos*8*stride2 + x_pos*8, dec->refn[0].u,
dec->cur.v + y_pos*8*stride2 + x_pos*8, dec->refn[0].v,
stride2, stride2, x_pos, y_pos, rounding);

gmc_data->get_average_mv(gmc_data, &pMB->amv, x_pos, y_pos, dec->quarterpel);

pMB->amv.x = gmc_sanitize(pMB->amv.x, dec->quarterpel, fcode);
pMB->amv.y = gmc_sanitize(pMB->amv.y, dec->quarterpel, fcode);

pMB->mvs[0] = pMB->mvs[1] = pMB->mvs[2] = pMB->mvs[3] = pMB->amv;

stop_transfer_timer();

if (cbp)
decoder_mb_decode(dec, cbp, bs, pY_Cur, pU_Cur, pV_Cur, 0, pMB);

}


static void
decoder_iframe(DECODER * dec,
Bitstream * bs,
int reduced_resolution,
int quant,
int intra_dc_threshold)
{
uint32_t bound;
uint32_t x, y;
uint32_t mb_width = dec->mb_width;
uint32_t mb_height = dec->mb_height;

if (reduced_resolution) {
mb_width = (dec->width + 31) / 32;
mb_height = (dec->height + 31) / 32;
}

bound = 0;

for (y = 0; y < mb_height; y++) {
for (x = 0; x < mb_width; x++) {
MACROBLOCK *mb;
uint32_t mcbpc;
uint32_t cbpc;
uint32_t acpred_flag;
uint32_t cbpy;
uint32_t cbp;

while (BitstreamShowBits(bs, 9) == 1)
BitstreamSkip(bs, 9);

if (check_resync_marker(bs, 0))
{
bound = read_video_packet_header(bs, dec, 0,
&quant, NULL, NULL, &intra_dc_threshold);
x = bound % mb_width;
y = bound / mb_width;
}
mb = &dec->mbs[y * dec->mb_width + x];
```

```
DPRINTF(XVID_DEBUG_MB, "macroblock (%i,%i) %08x\n", x, y, BitstreamShowBits(bs, 32));

mcbpc = get_mcbpc_intra(bs);
mb->mode = mcbpc & 7;
cbpc = (mcbpc >> 4);

acpred_flag = BitstreamGetBit(bs);

cbpy = get_cbpy(bs, 1);
cbp = (cbpy << 2) | cbpc;

if (mb->mode == MODE_INTRA_Q) {
quant += dquant_table[BitstreamGetBits(bs, 2)];
if (quant > 31) {
quant = 31;
} else if (quant < 1) {
quant = 1;
}
}
mb->quant = quant;
mb->mvs[0].x = mb->mvs[0].y =
mb->mvs[1].x = mb->mvs[1].y =
mb->mvs[2].x = mb->mvs[2].y =
mb->mvs[3].x = mb->mvs[3].y =0;

if (dec->interlacing) {
mb->field_dct = BitstreamGetBit(bs);
DPRINTF(XVID_DEBUG_MB,"deci: field_dct: %i\n", mb->field_dct);
}

decoder_mbintra(dec, mb, x, y, acpred_flag, cbp, bs, quant,
intra_dc_threshold, bound, reduced_resolution);

}
if(dec->out_frm)
output_slice(&dec->cur, dec->edged_width,dec->width,dec->out_frm,0,y,mb_width);
}

}


static void
get_motion_vector(DECODER * dec,
Bitstream * bs,
int x,
int y,
int k,
VECTOR * ret_mv,
int fcode,
const int bound)
{

const int scale_fac = 1 << (fcode - 1);
const int high = (32 * scale_fac) - 1;
const int low = ((-32) * scale_fac);
const int range = (64 * scale_fac);

const VECTOR pmv = get_pmv2(dec->mbs, dec->mb_width, bound, x, y, k);
VECTOR mv;

mv.x = get_mv(bs, fcode);
mv.y = get_mv(bs, fcode);

DPRINTF(XVID_DEBUG_MV,"mv_diff (%i,%i) pred (%i,%i) result (%i,%i)\n", mv.x, mv.y, pmv.x, pmv.y, mv.x+pmv.x, mv.y+pmv.y);

mv.x += pmv.x;
mv.y += pmv.y;

if (mv.x < low) {
mv.x += range;
} else if (mv.x > high) {
mv.x -= range;
}

if (mv.y < low) {
mv.y += range;
} else if (mv.y > high) {
mv.y -= range;
}

ret_mv->x = mv.x;
ret_mv->y = mv.y;
}

/* for P_VOP set gmc_warp to NULL */
static void
decoder_pframe(DECODER * dec,
```

```
Bitstream * bs,
int rounding,
int reduced_resolution,
int quant,
int fcode,
int intra_dc_threshold,
const WARPPOINTS *const gmc_warp)
{
uint32_t x, y;
uint32_t bound;
int cp_mb, st_mb;
uint32_t mb_width = dec->mb_width;
uint32_t mb_height = dec->mb_height;

if (reduced_resolution) {
mb_width = (dec->width + 31) / 32;
mb_height = (dec->height + 31) / 32;
}

start_timer();
image_setedges(&dec->refn[0], dec->edged_width, dec->edged_height,
dec->width, dec->height, dec->bs_version);
stop_edges_timer();

if (gmc_warp) {
/* accuracy: 0==1/2, 1=1/4, 2=1/8, 3=1/16 */
generate_GMCparameters( dec->sprite_warping_points,
dec->sprite_warping_accuracy, gmc_warp,
dec->width, dec->height, &dec->new_gmc_data);

/* image warping is done block-based in decoder_mbgmc(), now */
}

bound = 0;

for (y = 0; y < mb_height; y++) {
cp_mb = st_mb = 0;
for (x = 0; x < mb_width; x++) {
MACROBLOCK *mb;

/* skip stuffing */
while (BitstreamShowBits(bs, 10) == 1)
BitstreamSkip(bs, 10);

if (check_resync_marker(bs, fcode - 1)) {
bound = read_video_packet_header(bs, dec, fcode - 1,
&quant, &fcode, NULL, &intra_dc_threshold);
x = bound % mb_width;
y = bound / mb_width;
}
mb = &dec->mbs[y * dec->mb_width + x];

DPRINTF(XVID_DEBUG_MB, "macroblock (%i,%i) %08x\n", x, y, BitstreamShowBits(bs, 32));

if (!(BitstreamGetBit(bs))) { /* block _is_ coded */
uint32_t mcbpc, cbpc, cbpy, cbp;
uint32_t intra, acpred_flag = 0;
int mcsel = 0; /* mcsel: '0'=local motion, '1'=GMC */

cp_mb++;
mcbpc = get_mcbpc_inter(bs);
mb->mode = mcbpc & 7;
cbpc = (mcbpc >> 4);

DPRINTF(XVID_DEBUG_MB, "mode %i\n", mb->mode);
DPRINTF(XVID_DEBUG_MB, "cbpc %i\n", cbpc);

intra = (mb->mode == MODE_INTRA || mb->mode == MODE_INTRA_Q);

if (gmc_warp && (mb->mode == MODE_INTER || mb->mode == MODE_INTER_Q))
mcsel = BitstreamGetBit(bs);
else if (intra)
acpred_flag = BitstreamGetBit(bs);

cbpy = get_cbpy(bs, intra);
DPRINTF(XVID_DEBUG_MB, "cbpy %i mcsel %i \n", cbpy,mcsel);

cbp = (cbpy << 2) | cbpc;

if (mb->mode == MODE_INTER_Q || mb->mode == MODE_INTRA_Q) {
int dquant = dquant_table[BitstreamGetBits(bs, 2)];
DPRINTF(XVID_DEBUG_MB, "dquant %i\n", dquant);
quant += dquant;
if (quant > 31) {
quant = 31;
} else if (quant < 1) {
quant = 1;
```

```
}
DPRINTF(XVID_DEBUG_MB, "quant %i\n", quant);
}
mb->quant = quant;

if (dec->interlacing) {
if (cbp || intra) {
mb->field_dct = BitstreamGetBit(bs);
DPRINTF(XVID_DEBUG_MB,"decp: field_dct: %i\n", mb->field_dct);
}

if ((mb->mode == MODE_INTER || mb->mode == MODE_INTER_Q) && !mcsel) {
mb->field_pred = BitstreamGetBit(bs);
DPRINTF(XVID_DEBUG_MB, "decp: field_pred: %i\n", mb->field_pred);

if (mb->field_pred) {
mb->field_for_top = BitstreamGetBit(bs);
DPRINTF(XVID_DEBUG_MB,"decp: field_for_top: %i\n", mb->field_for_top);
mb->field_for_bot = BitstreamGetBit(bs);
DPRINTF(XVID_DEBUG_MB,"decp: field_for_bot: %i\n", mb->field_for_bot);
}
}
}

if (mcsel) {
decoder_mbgmc(dec, mb, x, y, fcode, cbp, bs, rounding);
continue;

} else if (mb->mode == MODE_INTER || mb->mode == MODE_INTER_Q) {

if (dec->interlacing && mb->field_pred) {
get_motion_vector(dec, bs, x, y, 0, &mb->mvs[0], fcode, bound);
get_motion_vector(dec, bs, x, y, 0, &mb->mvs[1], fcode, bound);
} else {
get_motion_vector(dec, bs, x, y, 0, &mb->mvs[0], fcode, bound);
mb->mvs[1] = mb->mvs[2] = mb->mvs[3] = mb->mvs[0];
}
} else if (mb->mode == MODE_INTER4V ) {
get_motion_vector(dec, bs, x, y, 0, &mb->mvs[0], fcode, bound);
get_motion_vector(dec, bs, x, y, 1, &mb->mvs[1], fcode, bound);
get_motion_vector(dec, bs, x, y, 2, &mb->mvs[2], fcode, bound);
get_motion_vector(dec, bs, x, y, 3, &mb->mvs[3], fcode, bound);
} else { /* MODE_INTRA, MODE_INTRA_Q */
mb->mvs[0].x = mb->mvs[1].x = mb->mvs[2].x = mb->mvs[3].x = 0;
mb->mvs[0].y = mb->mvs[1].y = mb->mvs[2].y = mb->mvs[3].y = 0;
decoder_mbintra(dec, mb, x, y, acpred_flag, cbp, bs, quant,
intra_dc_threshold, bound, reduced_resolution);
continue;
}

decoder_mbinter(dec, mb, x, y, cbp, bs,
rounding, reduced_resolution, 0);

} else if (gmc_warp) { /* a not coded S(GMC)-VOP macroblock */
mb->mode = MODE_NOT_CODED_GMC;
mb->quant = quant;
decoder_mbgmc(dec, mb, x, y, fcode, 0x00, bs, rounding);

if(dec->out_frm && cp_mb > 0) {
output_slice(&dec->cur, dec->edged_width,dec->width,dec->out_frm,st_mb,y,cp_mb);
cp_mb = 0;
}
st_mb = x+1;
} else { /* not coded P_VOP macroblock */
mb->mode = MODE_NOT_CODED;
mb->quant = quant;

mb->mvs[0].x = mb->mvs[1].x = mb->mvs[2].x = mb->mvs[3].x = 0;
mb->mvs[0].y = mb->mvs[1].y = mb->mvs[2].y = mb->mvs[3].y = 0;

decoder_mbinter(dec, mb, x, y, 0, bs,
rounding, reduced_resolution, 0);

if(dec->out_frm && cp_mb > 0) {
output_slice(&dec->cur, dec->edged_width,dec->width,dec->out_frm,st_mb,y,cp_mb);
cp_mb = 0;
}
st_mb = x+1;
}
}

if(dec->out_frm && cp_mb > 0)
output_slice(&dec->cur, dec->edged_width,dec->width,dec->out_frm,st_mb,y,cp_mb);
}
}
```

```
/* decode B-frame motion vector */
static void
get_b_motion_vector(Bitstream * bs,
VECTOR * mv,
int fcode,
const VECTOR pmv,
const DECODER * const dec,
const int x, const int y)
{
const int scale_fac = 1 << (fcode - 1);
const int high = (32 * scale_fac) - 1;
const int low = ((-32) * scale_fac);
const int range = (64 * scale_fac);

int mv_x = get_mv(bs, fcode);
int mv_y = get_mv(bs, fcode);

mv_x += pmv.x;
mv_y += pmv.y;

if (mv_x < low)
mv_x += range;
else if (mv_x > high)
mv_x -= range;

if (mv_y < low)
mv_y += range;
else if (mv_y > high)
mv_y -= range;

mv->x = mv_x;
mv->y = mv_y;
}

/* decode an B-frame direct & interpolate macroblock */
static void
decoder_bf_interpolate_mbinter(DECODER * dec,
IMAGE forward,
IMAGE backward,
const MACROBLOCK * pMB,
const uint32_t x_pos,
const uint32_t y_pos,
Bitstream * bs,
const int direct)
{
uint32_t stride = dec->edged_width;
uint32_t stride2 = stride / 2;
int uv_dx, uv_dy;
int b_uv_dx, b_uv_dy;
uint8_t *pY_Cur, *pU_Cur, *pV_Cur;
const uint32_t cbp = pMB->cbp;

pY_Cur = dec->cur.y + (y_pos << 4) * stride + (x_pos << 4);
pU_Cur = dec->cur.u + (y_pos << 3) * stride2 + (x_pos << 3);
pV_Cur = dec->cur.v + (y_pos << 3) * stride2 + (x_pos << 3);

if (!direct) {
uv_dx = pMB->mvs[0].x;
uv_dy = pMB->mvs[0].y;

b_uv_dx = pMB->b_mvs[0].x;
b_uv_dy = pMB->b_mvs[0].y;

if (dec->quarterpel) {
uv_dx /= 2;
uv_dy /= 2;
b_uv_dx /= 2;
b_uv_dy /= 2;
}

uv_dx = (uv_dx >> 1) + roundtab_79[uv_dx & 0x3];
uv_dy = (uv_dy >> 1) + roundtab_79[uv_dy & 0x3];

b_uv_dx = (b_uv_dx >> 1) + roundtab_79[b_uv_dx & 0x3];
b_uv_dy = (b_uv_dy >> 1) + roundtab_79[b_uv_dy & 0x3];

} else {
if(dec->quarterpel) {
uv_dx = (pMB->mvs[0].x / 2) + (pMB->mvs[1].x / 2) + (pMB->mvs[2].x / 2) + (pMB->mvs[3].x / 2);
uv_dy = (pMB->mvs[0].y / 2) + (pMB->mvs[1].y / 2) + (pMB->mvs[2].y / 2) + (pMB->mvs[3].y / 2);
b_uv_dx = (pMB->b_mvs[0].x / 2) + (pMB->b_mvs[1].x / 2) + (pMB->b_mvs[2].x / 2) + (pMB->b_mvs[3].x / 2);
b_uv_dy = (pMB->b_mvs[0].y / 2) + (pMB->b_mvs[1].y / 2) + (pMB->b_mvs[2].y / 2) + (pMB->b_mvs[3].y / 2);
} else {
uv_dx = pMB->mvs[0].x + pMB->mvs[1].x + pMB->mvs[2].x + pMB->mvs[3].x;
uv_dy = pMB->mvs[0].y + pMB->mvs[1].y + pMB->mvs[2].y + pMB->mvs[3].y;
b_uv_dx = pMB->b_mvs[0].x + pMB->b_mvs[1].x + pMB->b_mvs[2].x + pMB->b_mvs[3].x;
b_uv_dy = pMB->b_mvs[0].y + pMB->b_mvs[1].y + pMB->b_mvs[2].y + pMB->b_mvs[3].y;
```

```
}

uv_dx = (uv_dx >> 3) + roundtab_76[uv_dx & 0xf];
uv_dy = (uv_dy >> 3) + roundtab_76[uv_dy & 0xf];
b_uv_dx = (b_uv_dx >> 3) + roundtab_76[b_uv_dx & 0xf];
b_uv_dy = (b_uv_dy >> 3) + roundtab_76[b_uv_dy & 0xf];
}

start_timer();
if(dec->quarterpel) {
if(!direct) {
interpolate16x16_quarterpel(dec->cur.y, forward.y, dec->qtmp.y, dec->qtmp.y + 64,
  dec->qtmp.y + 128, 16*x_pos, 16*y_pos,
pMB->mvs[0].x, pMB->mvs[0].y, stride, 0);
} else {
interpolate8x8_quarterpel(dec->cur.y, forward.y, dec->qtmp.y, dec->qtmp.y + 64,
  dec->qtmp.y + 128, 16*x_pos, 16*y_pos,
pMB->mvs[0].x, pMB->mvs[0].y, stride, 0);
interpolate8x8_quarterpel(dec->cur.y, forward.y, dec->qtmp.y, dec->qtmp.y + 64,
  dec->qtmp.y + 128, 16*x_pos + 8, 16*y_pos,
pMB->mvs[1].x, pMB->mvs[1].y, stride, 0);
interpolate8x8_quarterpel(dec->cur.y, forward.y, dec->qtmp.y, dec->qtmp.y + 64,
  dec->qtmp.y + 128, 16*x_pos, 16*y_pos + 8,
pMB->mvs[2].x, pMB->mvs[2].y, stride, 0);
interpolate8x8_quarterpel(dec->cur.y, forward.y, dec->qtmp.y, dec->qtmp.y + 64,
  dec->qtmp.y + 128, 16*x_pos + 8, 16*y_pos + 8,
pMB->mvs[3].x, pMB->mvs[3].y, stride, 0);
}
} else {
interpolate8x8_switch(dec->cur.y, forward.y, 16 * x_pos, 16 * y_pos,
pMB->mvs[0].x, pMB->mvs[0].y, stride, 0);
interpolate8x8_switch(dec->cur.y, forward.y, 16 * x_pos + 8, 16 * y_pos,
pMB->mvs[1].x, pMB->mvs[1].y, stride, 0);
interpolate8x8_switch(dec->cur.y, forward.y, 16 * x_pos, 16 * y_pos + 8,
pMB->mvs[2].x, pMB->mvs[2].y, stride, 0);
interpolate8x8_switch(dec->cur.y, forward.y, 16 * x_pos + 8, 16 * y_pos + 8,
pMB->mvs[3].x, pMB->mvs[3].y, stride, 0);
}

interpolate8x8_switch(dec->cur.u, forward.u, 8 * x_pos, 8 * y_pos, uv_dx,
uv_dy, stride2, 0);
interpolate8x8_switch(dec->cur.v, forward.v, 8 * x_pos, 8 * y_pos, uv_dx,
uv_dy, stride2, 0);


if(dec->quarterpel) {
if(!direct) {
interpolate16x16_quarterpel(dec->tmp.y, backward.y, dec->qtmp.y, dec->qtmp.y + 64,
  dec->qtmp.y + 128, 16*x_pos, 16*y_pos,
pMB->b_mvs[0].x, pMB->b_mvs[0].y, stride, 0);
} else {
interpolate8x8_quarterpel(dec->tmp.y, backward.y, dec->qtmp.y, dec->qtmp.y + 64,
  dec->qtmp.y + 128, 16*x_pos, 16*y_pos,
pMB->b_mvs[0].x, pMB->b_mvs[0].y, stride, 0);
interpolate8x8_quarterpel(dec->tmp.y, backward.y, dec->qtmp.y, dec->qtmp.y + 64,
  dec->qtmp.y + 128, 16*x_pos + 8, 16*y_pos,
pMB->b_mvs[1].x, pMB->b_mvs[1].y, stride, 0);
interpolate8x8_quarterpel(dec->tmp.y, backward.y, dec->qtmp.y, dec->qtmp.y + 64,
  dec->qtmp.y + 128, 16*x_pos, 16*y_pos + 8,
pMB->b_mvs[2].x, pMB->b_mvs[2].y, stride, 0);
interpolate8x8_quarterpel(dec->tmp.y, backward.y, dec->qtmp.y, dec->qtmp.y + 64,
  dec->qtmp.y + 128, 16*x_pos + 8, 16*y_pos + 8,
pMB->b_mvs[3].x, pMB->b_mvs[3].y, stride, 0);
}
} else {
interpolate8x8_switch(dec->tmp.y, backward.y, 16 * x_pos, 16 * y_pos,
pMB->b_mvs[0].x, pMB->b_mvs[0].y, stride, 0);
interpolate8x8_switch(dec->tmp.y, backward.y, 16 * x_pos + 8,
16 * y_pos, pMB->b_mvs[1].x, pMB->b_mvs[1].y, stride, 0);
interpolate8x8_switch(dec->tmp.y, backward.y, 16 * x_pos,
16 * y_pos + 8, pMB->b_mvs[2].x, pMB->b_mvs[2].y, stride, 0);
interpolate8x8_switch(dec->tmp.y, backward.y, 16 * x_pos + 8,
16 * y_pos + 8, pMB->b_mvs[3].x, pMB->b_mvs[3].y, stride, 0);
}

interpolate8x8_switch(dec->tmp.u, backward.u, 8 * x_pos, 8 * y_pos,
b_uv_dx, b_uv_dy, stride2, 0);
interpolate8x8_switch(dec->tmp.v, backward.v, 8 * x_pos, 8 * y_pos,
b_uv_dx, b_uv_dy, stride2, 0);

interpolate8x8_avg2(dec->cur.y + (16 * y_pos * stride) + 16 * x_pos,
dec->cur.y + (16 * y_pos * stride) + 16 * x_pos,
dec->tmp.y + (16 * y_pos * stride) + 16 * x_pos,
stride, 1, 8);

interpolate8x8_avg2(dec->cur.y + (16 * y_pos * stride) + 16 * x_pos + 8,
dec->cur.y + (16 * y_pos * stride) + 16 * x_pos + 8,
```

```
dec->tmp.y + (16 * y_pos * stride) + 16 * x_pos + 8,
stride, 1, 8);

interpolate8x8_avg2(dec->cur.y + ((16 * y_pos + 8) * stride) + 16 * x_pos,
dec->cur.y + ((16 * y_pos + 8) * stride) + 16 * x_pos,
dec->tmp.y + ((16 * y_pos + 8) * stride) + 16 * x_pos,
stride, 1, 8);

interpolate8x8_avg2(dec->cur.y + ((16 * y_pos + 8) * stride) + 16 * x_pos + 8,
dec->cur.y + ((16 * y_pos + 8) * stride) + 16 * x_pos + 8,
dec->tmp.y + ((16 * y_pos + 8) * stride) + 16 * x_pos + 8,
stride, 1, 8);

interpolate8x8_avg2(dec->cur.u + (8 * y_pos * stride2) + 8 * x_pos,
dec->cur.u + (8 * y_pos * stride2) + 8 * x_pos,
dec->tmp.u + (8 * y_pos * stride2) + 8 * x_pos,
stride2, 1, 8);

interpolate8x8_avg2(dec->cur.v + (8 * y_pos * stride2) + 8 * x_pos,
dec->cur.v + (8 * y_pos * stride2) + 8 * x_pos,
dec->tmp.v + (8 * y_pos * stride2) + 8 * x_pos,
stride2, 1, 8);

stop_comp_timer();

if (cbp)
decoder_mb_decode(dec, cbp, bs, pY_Cur, pU_Cur, pV_Cur, 0, pMB);
}

/* for decode B-frame dbquant */
static __inline int32_t
get_dbquant(Bitstream * bs)
{
if (!BitstreamGetBit(bs)) /*  '0' */
return (0);
else if (!BitstreamGetBit(bs)) /* '10' */
return (-2);
else /* '11' */
return (2);
}

/*
 * decode B-frame mb_type
 * bit ret_value
 * 1 0
 * 01 1
 * 001 2
 * 0001 3
 */
static int32_t __inline
get_mbtype(Bitstream * bs)
{
int32_t mb_type;

for (mb_type = 0; mb_type <= 3; mb_type++)
if (BitstreamGetBit(bs))
return (mb_type);

return -1;
}

static void
decoder_bframe(DECODER * dec,
Bitstream * bs,
int quant,
int fcode_forward,
int fcode_backward)
{
uint32_t x, y;
VECTOR mv;
const VECTOR zeromv = {0,0};
int i;

start_timer();
image_setedges(&dec->refn[0], dec->edged_width, dec->edged_height,
dec->width, dec->height, dec->bs_version);
image_setedges(&dec->refn[1], dec->edged_width, dec->edged_height,
dec->width, dec->height, dec->bs_version);
stop_edges_timer();

for (y = 0; y < dec->mb_height; y++) {
/* Initialize Pred Motion Vector */
dec->p_fmv = dec->p_bmv = zeromv;
for (x = 0; x < dec->mb_width; x++) {
MACROBLOCK *mb = &dec->mbs[y * dec->mb_width + x];
MACROBLOCK *last_mb = &dec->last_mbs[y * dec->mb_width + x];
const int fcode_max = (fcode_forward>fcode_backward) ? fcode_forward : fcode_backward;
```

```
uint32_t intra_dc_threshold; /* fake variable */

if (check_resync_marker(bs, fcode_max  - 1)) {
int bound = read_video_packet_header(bs, dec, fcode_max - 1, &quant,
 &fcode_forward, &fcode_backward, &intra_dc_threshold);
x = bound % dec->mb_width;
y = bound / dec->mb_width;
/* reset predicted macroblocks */
dec->p_fmv = dec->p_bmv = zeromv;
}

mv =
mb->b_mvs[0] = mb->b_mvs[1] = mb->b_mvs[2] = mb->b_mvs[3] =
mb->mvs[0] = mb->mvs[1] = mb->mvs[2] = mb->mvs[3] = zeromv;
mb->quant = quant;

/*
 * skip if the co-located P_VOP macroblock is not coded
 * if not codec in co-located S_VOP macroblock is _not_
 * automatically skipped
 */

if (last_mb->mode == MODE_NOT_CODED) {
mb->cbp = 0;
mb->mode = MODE_FORWARD;
decoder_mbinter(dec, mb, x, y, mb->cbp, bs, 0, 0, 1);
continue;
}

if (!BitstreamGetBit(bs)) { /* modb=='0' */
const uint8_t modb2 = BitstreamGetBit(bs);

mb->mode = get_mbtype(bs);

if (!modb2) /* modb=='00' */
mb->cbp = BitstreamGetBits(bs, 6);
else
mb->cbp = 0;

if (mb->mode && mb->cbp) {
quant += get_dbquant(bs);
if (quant > 31)
quant = 31;
else if (quant < 1)
quant = 1;
}
mb->quant = quant;

if (dec->interlacing) {
if (mb->cbp) {
mb->field_dct = BitstreamGetBit(bs);
DPRINTF(XVID_DEBUG_MB,"decp: field_dct: %i\n", mb->field_dct);
}

if (mb->mode) {
mb->field_pred = BitstreamGetBit(bs);
DPRINTF(XVID_DEBUG_MB, "decp: field_pred: %i\n", mb->field_pred);

if (mb->field_pred) {
mb->field_for_top = BitstreamGetBit(bs);
DPRINTF(XVID_DEBUG_MB,"decp: field_for_top: %i\n", mb->field_for_top);
mb->field_for_bot = BitstreamGetBit(bs);
DPRINTF(XVID_DEBUG_MB,"decp: field_for_bot: %i\n", mb->field_for_bot);
}
}
}

} else {
mb->mode = MODE_DIRECT_NONE_MV;
mb->cbp = 0;
}

switch (mb->mode) {
case MODE_DIRECT:
get_b_motion_vector(bs, &mv, 1, zeromv, dec, x, y);

case MODE_DIRECT_NONE_MV:
for (i = 0; i < 4; i++) {
mb->mvs[i].x = last_mb->mvs[i].x*dec->time_bp/dec->time_pp + mv.x;
mb->mvs[i].y = last_mb->mvs[i].y*dec->time_bp/dec->time_pp + mv.y;

mb->b_mvs[i].x = (mv.x)
?  mb->mvs[i].x - last_mb->mvs[i].x
: last_mb->mvs[i].x*(dec->time_bp - dec->time_pp)/dec->time_pp;
mb->b_mvs[i].y = (mv.y)
? mb->mvs[i].y - last_mb->mvs[i].y
: last_mb->mvs[i].y*(dec->time_bp - dec->time_pp)/dec->time_pp;
```

```
}

decoder_bf_interpolate_mbinter(dec, dec->refn[1], dec->refn[0],
mb, x, y, bs, 1);
break;

case MODE_INTERPOLATE:
get_b_motion_vector(bs, &mb->mvs[0], fcode_forward, dec->p_fmv, dec, x, y);
dec->p_fmv = mb->mvs[1] = mb->mvs[2] = mb->mvs[3] = mb->mvs[0];

get_b_motion_vector(bs, &mb->b_mvs[0], fcode_backward, dec->p_bmv, dec, x, y);
dec->p_bmv = mb->b_mvs[1] = mb->b_mvs[2] = mb->b_mvs[3] = mb->b_mvs[0];

decoder_bf_interpolate_mbinter(dec, dec->refn[1], dec->refn[0],
mb, x, y, bs, 0);
break;

case MODE_BACKWARD:
get_b_motion_vector(bs, &mb->mvs[0], fcode_backward, dec->p_bmv, dec, x, y);
dec->p_bmv = mb->mvs[1] = mb->mvs[2] = mb->mvs[3] = mb->mvs[0];

decoder_mbinter(dec, mb, x, y, mb->cbp, bs, 0, 0, 0);
break;

case MODE_FORWARD:
get_b_motion_vector(bs, &mb->mvs[0], fcode_forward, dec->p_fmv, dec, x, y);
dec->p_fmv = mb->mvs[1] = mb->mvs[2] = mb->mvs[3] = mb->mvs[0];

decoder_mbinter(dec, mb, x, y, mb->cbp, bs, 0, 0, 1);
break;

default:
DPRINTF(XVID_DEBUG_ERROR,"Not supported B-frame mb_type = %i\n", mb->mode);
}
} /* End of for */
}
}

/* perform post processing if necessary, and output the image */
void decoder_output(DECODER * dec, IMAGE * img, MACROBLOCK * mbs,
xvid_dec_frame_t * frame, xvid_dec_stats_t * stats,
int coding_type, int quant)
{
if (dec->cartoon_mode)
frame->general &= ~XVID_FILMEFFECT;

if (frame->general & (XVID_DEBLOCKY|XVID_DEBLOCKUV|XVID_FILMEFFECT) && mbs != NULL) /* post process */
{
/* note: image is stored to tmp */
image_copy(&dec->tmp, img, dec->edged_width, dec->height);
image_postproc(&dec->postproc, &dec->tmp, dec->edged_width,
   mbs, dec->mb_width, dec->mb_height, dec->mb_width,
   frame->general, dec->frames, (coding_type == B_VOP));
img = &dec->tmp;
}

image_output(img, dec->width, dec->height,
 dec->edged_width, (uint8_t**)frame->output.plane, frame->output.stride,
 frame->output.csp, dec->interlacing);

if (stats) {
stats->type = coding2type(coding_type);
stats->data.vop.time_base = (int)dec->time_base;
stats->data.vop.time_increment = 0; /* XXX: todo */
stats->data.vop.qscale_stride = dec->mb_width;
stats->data.vop.qscale = dec->qscale;
if (stats->data.vop.qscale != NULL && mbs != NULL) {
int i;
for (i = 0; i < dec->mb_width*dec->mb_height; i++)
stats->data.vop.qscale[i] = mbs[i].quant;
} else
stats->data.vop.qscale = NULL;
}
}


int
decoder_decode(DECODER * dec,
xvid_dec_frame_t * frame, xvid_dec_stats_t * stats)
{

Bitstream bs;
uint32_t rounding;
uint32_t reduced_resolution;
uint32_t quant = 2;
uint32_t fcode_forward;
uint32_t fcode_backward;
```

```
uint32_t intra_dc_threshold;
WARPPOINTS gmc_warp;
int coding_type;
int success, output, seen_something;

if (XVID_VERSION_MAJOR(frame->version) != 1 || (stats && XVID_VERSION_MAJOR(stats->version) != 1)) /* v1.x.x */
return XVID_ERR_VERSION;

start_global_timer();

dec->low_delay_default = (frame->general & XVID_LOWDELAY);
if ((frame->general & XVID_DISCONTINUITY))
dec->frames = 0;
dec->out_frm = (frame->output.csp == XVID_CSP_SLICE) ? &frame->output : NULL;

if (frame->length < 0) { /* decoder flush */
int ret;
/* if not decoding "low_delay/packed", and this isn't low_delay and
we have a reference frame, then outout the reference frame */
if (!(dec->low_delay_default && dec->packed_mode) && !dec->low_delay && dec->frames>0) {
decoder_output(dec, &dec->refn[0], dec->last_mbs, frame, stats, dec->last_coding_type, quant);
dec->frames = 0;
ret = 0;
} else {
if (stats) stats->type = XVID_TYPE_NOTHING;
ret = XVID_ERR_END;
}

emms();
stop_global_timer();
return ret;
}

BitstreamInit(&bs, frame->bitstream, frame->length);

/* XXX: 0x7f is only valid whilst decoding vfw xvid/divx5 avi's */
if(dec->low_delay_default && frame->length == 1 && BitstreamShowBits(&bs, 8) == 0x7f)
{
image_output(&dec->refn[0], dec->width, dec->height, dec->edged_width,
 (uint8_t**)frame->output.plane, frame->output.stride, frame->output.csp, dec->interlacing);
if (stats) stats->type = XVID_TYPE_NOTHING;
emms();
return 1; /* one byte consumed */
}

success = 0;
output = 0;
seen_something = 0;

repeat:

coding_type = BitstreamReadHeaders(&bs, dec, &rounding, &reduced_resolution,
&quant, &fcode_forward, &fcode_backward, &intra_dc_threshold, &gmc_warp);

DPRINTF(XVID_DEBUG_HEADER, "coding_type=%i,  packed=%i,  time=%lli,  time_pp=%i,  time_bp=%i\n",
coding_type,dec->packed_mode, dec->time, dec->time_pp, dec->time_bp);

if (coding_type == -1) { /* nothing */
if (success) goto done;
if (stats) stats->type = XVID_TYPE_NOTHING;
emms();
return BitstreamPos(&bs)/8;
}

if (coding_type == -2 || coding_type == -3) { /* vol and/or resize */

if (coding_type == -3)
decoder_resize(dec);

if (stats) {
stats->type = XVID_TYPE_VOL;
stats->data.vol.general = 0;
/*XXX: if (dec->interlacing)
stats->data.vol.general |= ++INTERLACING; */
stats->data.vol.width = dec->width;
stats->data.vol.height = dec->height;
stats->data.vol.par = dec->aspect_ratio;
stats->data.vol.par_width = dec->par_width;
stats->data.vol.par_height = dec->par_height;
emms();
return BitstreamPos(&bs)/8; /* number of bytes consumed */
}
goto repeat;
}

if(dec->frames == 0 && coding_type != I_VOP) {
/* 1st frame is not an i-vop */
```

```
goto repeat;
}

dec->p_bmv.x = dec->p_bmv.y = dec->p_fmv.y = dec->p_fmv.y = 0; /* init pred vector to 0 */

/* packed_mode: special-N_VOP treament */
if (dec->packed_mode && coding_type == N_VOP) {
if (dec->low_delay_default && dec->frames > 0) {
decoder_output(dec, &dec->refn[0], dec->last_mbs, frame, stats, dec->last_coding_type, quant);
output = 1;
}
/* ignore otherwise */
} else if (coding_type != B_VOP) {
switch(coding_type) {
case I_VOP :
decoder_iframe(dec, &bs, reduced_resolution, quant, intra_dc_threshold);
break;
case P_VOP :
decoder_pframe(dec, &bs, rounding, reduced_resolution, quant,
fcode_forward, intra_dc_threshold, NULL);
break;
case S_VOP :
decoder_pframe(dec, &bs, rounding, reduced_resolution, quant,
fcode_forward, intra_dc_threshold, &gmc_warp);
break;
case N_VOP :
/* XXX: not_coded vops are not used for forward prediction */
/* we should not swap(last_mbs,mbs) */
image_copy(&dec->cur, &dec->refn[0], dec->edged_width, dec->height);
SWAP(MACROBLOCK *, dec->mbs, dec->last_mbs); /* it will be swapped back */
break;
}

if (reduced_resolution) {
image_deblock_rrv(&dec->cur, dec->edged_width, dec->mbs,
(dec->width + 31) / 32, (dec->height + 31) / 32, dec->mb_width,
16, 0);
}

/* note: for packed_mode, output is performed when the special-N_VOP is decoded */
if (!(dec->low_delay_default && dec->packed_mode)) {
if (dec->low_delay) {
decoder_output(dec, &dec->cur, dec->mbs, frame, stats, coding_type, quant);
output = 1;
} else if (dec->frames > 0) { /* is the reference frame valid? */
/* output the reference frame */
decoder_output(dec, &dec->refn[0], dec->last_mbs, frame, stats, dec->last_coding_type, quant);
output = 1;
}
}

image_swap(&dec->refn[0], &dec->refn[1]);
image_swap(&dec->cur, &dec->refn[0]);
SWAP(MACROBLOCK *, dec->mbs, dec->last_mbs);
dec->last_reduced_resolution = reduced_resolution;
dec->last_coding_type = coding_type;

dec->frames++;
seen_something = 1;

} else { /* B_VOP */

if (dec->low_delay) {
DPRINTF(XVID_DEBUG_ERROR, "warning: bvop found in low_delay==1 stream\n");
dec->low_delay = 1;
}

if (dec->frames < 2) {
/* attemping to decode a bvop without atleast 2 reference frames */
image_printf(&dec->cur, dec->edged_width, dec->height, 16, 16,
"broken b-frame, mising ref frames");
if (stats) stats->type = XVID_TYPE_NOTHING;
} else if (dec->time_pp <= dec->time_bp) {
/* this occurs when dx50_bvop_compatibility==0 sequences are
decoded in vfw. */
image_printf(&dec->cur, dec->edged_width, dec->height, 16, 16,
"broken b-frame, tpp=%i tbp=%i", dec->time_pp, dec->time_bp);
if (stats) stats->type = XVID_TYPE_NOTHING;
} else {
decoder_bframe(dec, &bs, quant, fcode_forward, fcode_backward);
decoder_output(dec, &dec->cur, dec->mbs, frame, stats, coding_type, quant);
}

output = 1;
dec->frames++;
}
```

```
#if 0 /* Avoids to read to much data because of 32bit reads in our BS functions */
 BitstreamByteAlign(&bs);
#endif

/* low_delay_default mode: repeat in packed_mode */
if (dec->low_delay_default && dec->packed_mode && output == 0 && success == 0) {
success = 1;
goto repeat;
}

done :

/* low_delay_default mode: if we've gotten here without outputting anything,
   then output the recently decoded frame, or print an error message  */
if (dec->low_delay_default && output == 0) {
if (dec->packed_mode && seen_something) {
/* output the recently decoded frame */
decoder_output(dec, &dec->refn[0], dec->last_mbs, frame, stats, dec->last_coding_type, quant);
} else {
image_clear(&dec->cur, dec->width, dec->height, dec->edged_width, 0, 128, 128);
image_printf(&dec->cur, dec->edged_width, dec->height, 16, 16,
"warning: nothing to output");
image_printf(&dec->cur, dec->edged_width, dec->height, 16, 64,
"bframe decoder lag");

decoder_output(dec, &dec->cur, NULL, frame, stats, P_VOP, quant);
if (stats) stats->type = XVID_TYPE_NOTHING;
}
}

emms();
stop_global_timer();

return (BitstreamPos(&bs) + 7) / 8; /* number of bytes consumed */
}
```

GNU General Public License

F

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain

that everyone understands that there is no warranty for this free
software. If the software is modified by someone else and passed on, we
want its recipients to know that what they have is not the original, so
that any problems introduced by others will not reflect on the original
authors' reputations.

Finally, any free program is threatened constantly by software
patents. We wish to avoid the danger that redistributors of a free
program will individually obtain patent licenses, in effect making the
program proprietary. To prevent this, we have made it clear that any
patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and
modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains
a notice placed by the copyright holder saying it may be distributed
under the terms of this General Public License. The "Program", below,
refers to any such program or work, and a "work based on the Program"
means either the Program or any derivative work under copyright law:
that is to say, a work containing the Program or a portion of it,
either verbatim or with modifications and/or translated into another
language. (Hereinafter, translation is included without limitation in
the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not
covered by this License; they are outside its scope. The act of
running the Program is not restricted, and the output from the Program
is covered only if its contents constitute a work based on the
Program (independent of having been made by running the Program).
Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's
source code as you receive it, in any medium, provided that you
conspicuously and appropriately publish on each copy an appropriate
copyright notice and disclaimer of warranty; keep intact all the
notices that refer to this License and to the absence of any warranty;
and give any other recipients of the Program a copy of this License
along with the Program.

You may charge a fee for the physical act of transferring a copy, and
you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion
of it, thus forming a work based on the Program, and copy and
distribute such modifications or work under the terms of Section 1
above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices
stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in
whole or in part contains or is derived from the Program or any
part thereof, to be licensed as a whole at no charge to all third
parties under the terms of this License.

c) If the modified program normally reads commands interactively
when run, you must cause it, when started running for such
interactive use in the most ordinary way, to print or display an
announcement including an appropriate copyright notice and a
notice that there is no warranty (or else, saying that you provide
a warranty) and that users may redistribute the program under
these conditions, and telling the user how to view a copy of this
License. (Exception: if the Program itself is interactive but
does not normally print such an announcement, your work based on
the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If
identifiable sections of that work are not derived from the Program,
and can be reasonably considered independent and separate works in
themselves, then this License, and its terms, do not apply to those
sections when you distribute them as separate works. But when you
distribute the same sections as part of a whole which is a work based
on the Program, the distribution of the whole must be on the terms of
this License, whose permissions for other licensees extend to the
entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest
your rights to work written entirely by you; rather, the intent is to
exercise the right to control the distribution of derivative or
collective works based on the Program.

In addition, mere aggregation of another work not based on the Program
with the Program (or with a work based on the Program) on a volume of
a storage or distribution medium does not bring the other work under
the scope of this License.

3. You may copy and distribute the Program (or a work based on it,
under Section 2) in object code or executable form under the terms of
Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable
source code, which must be distributed under the terms of Sections
1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three
years, to give any third party, for a charge no more than your

cost of physically performing source distribution, a complete
machine-readable copy of the corresponding source code, to be
distributed under the terms of Sections 1 and 2 above on a medium
customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer
to distribute corresponding source code. (This alternative is
allowed only for noncommercial distribution and only if you
received the program in object code or executable form with such
an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for
making modifications to it. For an executable work, complete source
code means all the source code for all modules it contains, plus any
associated interface definition files, plus the scripts used to
control compilation and installation of the executable. However, as a
special exception, the source code distributed need not include
anything that is normally distributed (in either source or binary
form) with the major components (compiler, kernel, and so on) of the
operating system on which the executable runs, unless that component
itself accompanies the executable.

If distribution of executable or object code is made by offering
access to copy from a designated place, then offering equivalent
access to copy the source code from the same place counts as
distribution of the source code, even though third parties are not
compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program
except as expressly provided under this License. Any attempt
otherwise to copy, modify, sublicense or distribute the Program is
void, and will automatically terminate your rights under this License.
However, parties who have received copies, or rights, from you under
this License will not have their licenses terminated so long as such
parties remain in full compliance.

5. You are not required to accept this License, since you have not
signed it. However, nothing else grants you permission to modify or
distribute the Program or its derivative works. These actions are
prohibited by law if you do not accept this License. Therefore, by
modifying or distributing the Program (or any work based on the
Program), you indicate your acceptance of this License to do so, and
all its terms and conditions for copying, distributing or modifying
the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the
Program), the recipient automatically receives a license from the
original licensor to copy, distribute or modify the Program subject to
these terms and conditions. You may not impose any further
restrictions on the recipients' exercise of the rights granted herein.

You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any

later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>

```
    Copyright (C) <year>  <name of author>

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
    Gnomovision version 69, Copyright (C) year name of author
    Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type ‘show c’ for details.
```

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than ‘show w’ and ‘show c’; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  ‘Gnomovision’ (which makes passes at compilers) written by James Hacker.

  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Curriculum Vitae

Guido de Goede was born on the 1st of July 1981 in Rotterdam and attended Primary School from 1987 to 1993. During these years he acquired his first programming experience on Z80-microcontroller computers (MSX) and designed videotext pages for regional TV channels (up to 2.5 million subscribers). In October 1992 he was chairman of the Children City-council (Dutch: "Kinder Gemeenteraad") of his home town Hendrik Ido Ambacht. In 1999 he graduated in 9 subjects from Secondary School the Marnix Gymnasium Rotterdam, with all science subjects grade 9, all modern languages grade 8 and grade 7.45 for Latin. From age 15 he worked in holidays as a graphical designer for the "KabelKoerier", the Cable News Channel in H.I.Ambacht. This function contained photo, video, and webpage designing and keeping up a national database of TV-programmes for videotext, called "Telezap". In 1996 he scored best of the school in the European Kangaroo Mathematics Contest, ranging best 4% of Holland. He graduated best of the school in Physics and recieved a "Ster beurs"-scholarship from the TU Delft for excellent students after graduation in 1999. In 2000 he acquired his Electrical engineering Propaedeuse-diploma Cum Laude in one year and was rewarded "Best student of the year" by dr.ir. JJ Gerbrands. During the first year of college he was actively involved in the Propaedeuse Response Committee and in the College Response Committee during the second and third year. In the third year he was an active member of the Faculty Student Counsil. He acquired his Bachelor degree without delay with an average grade 8 in three years. He has programming experience in BASIC, Scheme LISP, C, C++ and a good understanding of JAVA and Pascal. He is experienced in frequent private tuition of Second School students in Mathematics, Physics and Chemistry. Currently he is an M.Sc. student in Computer Engineering. As a Christian he believes in the resurrection of Jesus Christ and is actively involved in his home town church as a youth group leader. He likes group sports activities and gymnastics.