

Using Software Techniques when Developing Hardware Applications

Grant B. Wigley, John P. Hopf and David A. Kearney
Uninhabited Aerial Vehicle Laboratory (UAVL)
Advanced Computing Research Centre
University of South Australia
Mawson Lakes SA 5095

Grant.Wigley@unisa.edu.au John.Hopf@cs.unisa.edu.au David.Kearney@unisa.edu.au

Key words: FPGA Application, MP3, Design Technique

Abstract

This paper presents an MP3 decoder with portions of the algorithm implemented totally in hardware. We are not claiming that this is the first MP3 algorithm implemented in hardware, but what makes this project different from previous works, is that our project is a hardware/software implementation and we used traditional software techniques in the development of it. With the use of software profiling, we determined what parts of the algorithm consumed large amounts of CPU execution time and knowing the target was an FPGA, particular parts of the algorithm were chosen to be transferred from software to hardware based on their CPU usage and suitability for an FPGA implementation. The system was written with a client/server model architecture, so the parts of the algorithm written in hardware would be abstracted away behind a standard software interface. CPU usage tests were conducted on the two versions of the MP3 player, one with the complete algorithm in software, and two with the part of the algorithm in hardware. Our test results show that the part hardware/software MP3 implementation reduced the total CPU usage by 7.5% on average with a burst improvement of up to 15%.

1. Introduction

The rapid development of FPGA technology indicates that reconfigurable computing will definitely become part of mainstream workstation platforms in the future. The reasons for this are that multimedia applications are becoming more and more important and a diverse range of these applications can be more readily handled by reconfigurable hardware than just full custom hardware or just software. However reconfigurable computing application development has a reputation at present as being more difficult than software-only development, due to the perceived need for both hardware and software development skills. In

this paper we show how part of a multimedia application can be implemented in reconfigurable hardware using a very traditional software development process involving profiling and making use of hardware synthesis based on an HDL but viewed as a software development language.

The experience reported in this paper illustrates that there may be other advantages of using software development thinking in the development of reconfigurable applications. The application reported here makes use of software encapsulation of hardware and a hardware architecture that is a direct result of the software orientated approach. We will show that the traditional approach to hardware design that emphasises high performance interfaces between modules makes these hardware modules difficult to use in a software environment and makes the comprehension of hardware by software professionals difficult. A change in approach encourages all hardware to be wrapped in interfaces that are software friendly thus facilitating its integration into software-based systems. The influence of this encapsulation in the development process for reconfigurable hardware reported in this paper has lead to what we believe is a new architecture for hardware MP3 reordering which addresses the well known limitation of many reconfigurable platforms used in streaming applications where the copying of data between memory locations makes up the majority of the hardware execution leading to an uncompetitive hardware accelerator. Our application involves minimal transfer of data and embeds the processing in the data transfer thus hiding this overhead.

The paper is organized as follows. Section 2 describes previous works on MP3 software and hardware implementations, as well as FPGA MP3 implementations. We then describe the important details of the MP3 decoding algorithm, in particular the reordering phase, the part of the algorithm we chose to implement in hardware. Section 4 describes the design process, including profiling and why the reordering

phase of the algorithm was chosen to be implemented ahead of other more CPU intensive tasks. Section 5 discusses the prototype implementation, software and hardware architecture, followed by the results we obtained. Finally we look at future developments of the current prototype and conclude our research paper.

2. Previous Work

Since the introduction of the MP3 standard and the ability to download MP3's from the Internet and play them on a personal computer, there have been a number of MP3 hardware implementations.

MP3 hardware decoders have been produced by a number of companies for use in portable devices and home entertainment devices. Siemens have embedded an MP3 decoder into their SL45 range of mobile phones. Other portable devices include various MP3 walkmans such as the Diamond Rio and PortalPlayer [1]. These players can interface with your PC via the USB/Parallel port download and store MP3s into flash memory or play MP3s stored on a CD. These devices implement a pure hardware decoding architecture using small microprocessors and specially designed MP3 decoder chips. One of the first Diamond Rio player's uses the Cirrus Logic EP7209 chip and the PortalPlayer uses the Oki Semiconductor Tango chip, both of which are based on the ARM7TDMI core developed by ARM in the UK [2].

STMicroElectronics [3] has also developed an MP3 decoder chip, the STA013. The chip is based on ST's Very Long Instruction Word DSP core, which is already used in a wide range of audio applications such as Dolby Digital processing and audio enhancement. The STA013 decoder is capable of performing demultiplexing and error correction, Huffman decoding, inverse quantization, inverse discrete co-sine transform and filter bank reconstruction from 32 sub-bands into a single stereo output.

Salomonsen, Seggaard and Larsen from the Aalborg University have successfully implemented an MP3 bitstream processor based on a Von Neumann architecture [4]. It was hoped that if they could produce a processor to handle bitstream processing, that only a small Digital Signal Processor (DSP) would be needed for MP3 decoding.

3. MPEG Layer 3

Motion Pictures Expert Group (MPEG) provides 3 standards, better known as layers for audio compression, each with an increased level of complexity. Layer 1 uses the simplest techniques and consequently provides

the least compression out of the 3 layers with audio encoded at 192KBits/s per channel. It therefore needs high transmission bandwidth to achieve HI-FI quality. Layer II has a moderate level of complexity, and is encoded at 128KBit/s per channel. This reduced bit-rate requires less transmission bandwidth to achieve HI-FI quality. Layer 3 is the most complex and uses sophisticated compression techniques to achieve HI-FI quality at 64Kbits/s. Audio files adopting the MPEG Layer 3 technology are denoted an mp3 extension and these are the most common MPEG audio used today.

During encoding of MP3 audio, the majority of compression is achieved through the use of psycho-acoustic theory. By only encoding sounds that the human ear can perceive and omitting those that it can't, MP3 can reduce raw wave audio files by ratios exceeding 10:1, with minimal audible loss in sound quality.

3.1. MP3 Internal Structure

During the encoding of an MP3 file, data is placed into frames with each frame containing a header, cyclic redundancy check (CRC), side information, main data and ancillary data [5]. The size in bytes of the frames is dependent on the frequency rate and bit rate used for encoding. This can be mathematically expressed as:

$$Frame_Size = \frac{8 * (144 * Bitrate)}{FrequencyRate}$$

The frequency rate and bit rates are extracted from the header information, which has a consistent size of 32 bits across all MP3s [6]. The header additionally contains information about the MP3 file stereo characteristics and the presence of a CRC that will follow the header.

Side information follows the CRC and provides the necessary information for decoding the main data that ends each frame. Information about the Huffman lookup tables, which will be used during Huffman decoding and information relating to the reconstruction of scale-factors. If the bit reservoir was used during encoding, information relating to this is also included.

Coded scale factors and Huffman coded data are kept in the main data section that follows the side information. If the main data doesn't occupy the space reserved for it, main data from other adjoining frames can occupy the free space. This is the reservoir of bits technique that MP3 adopts and assists with compression. Finally, some ancillary data can be added to the end of the frame. This data is usually dependent on the encoder used and can contain the title of the song, or other information specific to the MP3.

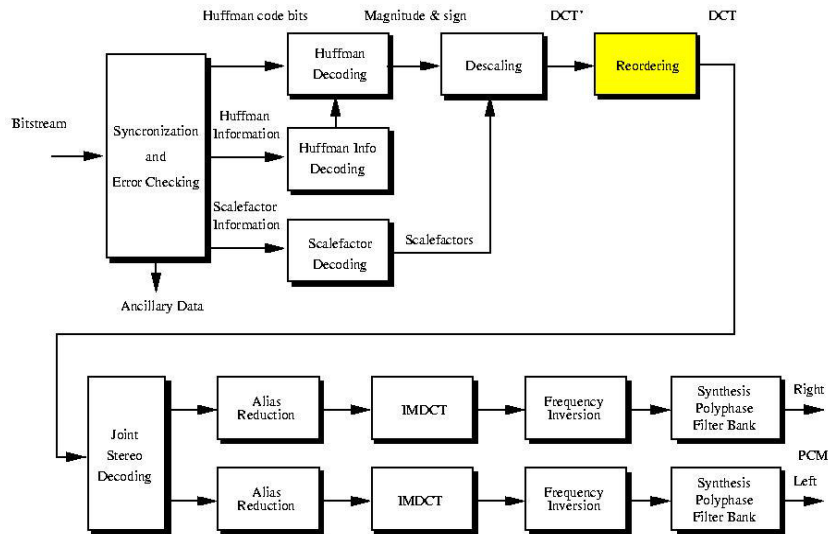


Figure 1: Flow chart of mp3 decoder

3.2. The decoding algorithm

The decoding of MP3 is a structured process and Figure 1 describes the algorithm as a flow chart. The re-ordering block is shaded, as it was the part implemented in hardware. Section 3.3 discusses reordering in detail.

As outlined in section 3.1, MP3 consists of frames containing a header, scale factor information and Huffman data. The first block of the decoding process involves synchronisation and error checking and here the frames are broken up into these parts. Next, the Huffman information is decoded to provide the necessary parameters to decode the Huffman code bits. This information is then sent to the Huffman decoding block with the Huffman code bits to generate the necessary frequency lines for the audio. At the same time, scale-factor information is decoded to produce the scale-factors needed in the de-scaling block. De-scaling then scales the frequency lines according to the scale-factors so they are returned to their original levels. Reordering then follows de-scaling.

After reordering, joint stereo decoding is done and it is here that the digital data is transformed into the original audio signals according to the stereo characteristics used during encoding. The same process is applied to both the left and right channels to achieve the two outputs. The function on the anti-aliasing block, is to reverse the alias reduction that was done during encoding and is accomplished by adding an alias artefact to the signal. The Inverse Modified Discrete Cosine Transform (IMDCT) block then takes this signal and converts it to a wave form which the initial raw

audio was in. This is then sent to the frequency inversion block where some of the frequencies are reversed to compensate for the inversions that will take place in the synthesis polyphase filter bank block.

3.3. Reorder in more detail

During decoding, de-scaling produces frequency lines, but these lines aren't always ordered in the same way [4]. If long windows are used prior to transformation in the discrete cosine transform (DCT) block, then the frequency lines are ordered first by sub-band and then by frequency. If short windows are used, then frequency lines are ordered first by sub-band, then by window and then by frequency. To increase the efficiency of Huffman encoding, the short windows are ordered by sub-band, frequency and then by window. It is the function of the reorder block to test if short windows have been used and to reorder them into their correct order if required. This can be checked by using flags provided in the side information portion of the MP3 frame.

4. Design Techniques

One of the main differences in the development of this application was the procedure we used. Firstly, the students that were involved in developing it had little knowledge of hardware. We had a strong background in software development and applied these skills in the development of hardware system. We used profiling to determine what parts to implement in hardware and then developed a software like architecture to data movement.

4.1. Profiling

To ensure the maximum benefits (greatest reduction of CPU usage) of implementing only a fraction of the MP3 decoder in hardware was achieved, the software MP3 decoder package was profiled. Profiling of software produces data relating to the amount of CPU usage used by each of the functions within an executable program during runtime.

The software MP3 package chosen to be the development platform was the MADplay MP3 [7]. It was chosen for a number of reasons. One, it is an integer-based implementation, as compared to the more common float/double implementation, and two it compiles under the Linux operating system and is distributed with the open source license arrangement. The MADplay package has an option for profiling during compilation and by providing an input file, the profile data was produced. Observation of the data showed that the 'III_imdct_1' function consumed the most resource during execution, but for reasons that will now be discussed, the 'III_reorder' function was selected to be implemented in hardware.

```
unsigned int tmp[32][3][6];
unsigned int sb, l, sfb, f, w, sbw[3], sw[3];

if (channel->flags & mixed_block_flag)
    sb = 2, sfb = 3;
else
    sb = 0, sfb = 0;

for (w = 0; w < 3; ++w) {
    sbw[w] = sb;
    sw[w] = 0;
}

f = sfwidth[sfb];
w = 0;

for (l = 18 * sb; l < 576; ++l) {
    tmp[sbw[w]][w][sw[w]++] = xr[l];

    if (sw[w] == 6) {
        sw[w] = 0;
        ++sbw[w];
    }
    if (--f == 0) {
        if (w++ == 2) {
            w = 0;
            ++sfb;
        }
        f = sfwidth[sfb];
    }
}
}
```

Figure 2 – MADplay Software Reorder Code

The 'III_reorder' function ranked 12th in resource usage, and therefore a substantial amount of CPU execution time could be saved if it is implemented in

hardware. Reasons this function was chosen above others included, one, the function didn't depend on large amounts of data being transferred between the software/hardware interface, as the bandwidth between the reconfigurable computer and the host computer is only a 33Mhz bus. Two, it didn't require large numbers of multiplications [8], which is a well-known problem for an FPGA to do efficiently in area, and three the function could easily be removed from the current software implementation and create a nice abstraction interface, one of the major goals of the project.

In figure 2, a code listing of the reordering process from the MADplay MP3 play is given and this code is what was converted into hardware.

4.2. Memory Architecture

The design avoids having to firstly transfer the data into memory, then retrieve and manipulate it. The hardware processes the data as it arrives, by manipulating the address to its corresponding reordered address location, for that particular piece of data. By the time all the data has been passed into the hardware design, and stored in the on-board memory, the design has completed the reordering process and the reordered data is now ready to be read back to the software part of the algorithm. This saves a considerable amount of internal I/O that normally would be necessary, therefore making the reordering process faster.

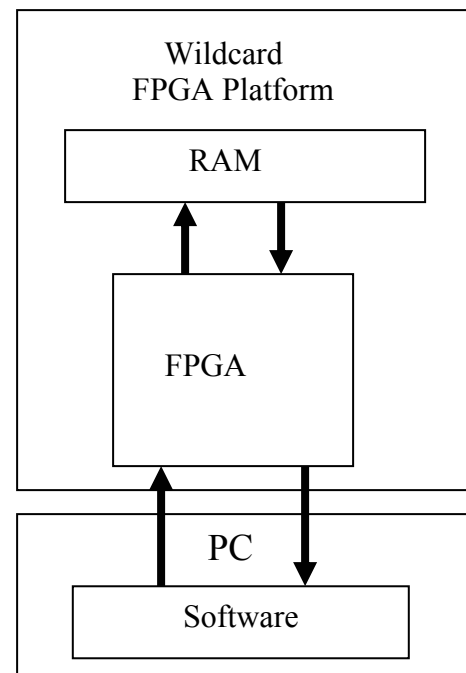


Figure 3 – Hardware Architecture

5. Prototype Implementation

The prototype was developed under the Linux operating system (kernel 2.2.14), running on an Intel Celeron 466MHz PC with 64MB of system memory and a PCI to PCMCIA adapter card. Our prototype makes use of the open source MADplay MP3 decoder package, freely available from the Internet.

5.1. Hardware Platform

The target reconfigurable computing architecture chosen for this project was the Annapolis Wildcard Development Board [9]. It has a Virtex series (XCV300E) FPGA as well as 2 banks of 256kB of SDRAM memory, on board. The development board complies with PCMCIA specifications and can operate at a 33Mhz bus speed. This board was chosen as the target architecture for many reasons. One, it is one of the very few PCMCIA based FPGA development cards and since we wanted the hardware/software MP3 play to be portable, this was a necessary specification. Two, the development board was shipped with Linux device drivers. The MADplay MP3 software implementation was developed under Linux and we needed a compact FPGA development board that would compile and run smoothly under such an operating system. Three, we wanted a board that had off-chip memory as we wanted to use the FPGA logic for the algorithm and not use it for memory.

5.2. Software Architecture

It was decided that it be appropriate for a client/server architecture to be constructed for the MP3 decoder. This methodology was adopted for several reasons. One, by embedding the calls to the reconfigurable computer in a server, no specific FPGA code had to be embedded into the client. This provides better maintainability and an abstraction of the hardware with accesses to the hardware via a software interface.

Two, the architecture also allows other MP3 decoder packages written in different languages supporting TCP sockets to use the reordering hardware implementation with little software modification. The use of industry standard TCP sockets also adds the ability for the hardware implementation to be accessed over high bandwidth networks.

The system works such that the server receives data from the client via TCP sockets. The server then stores this data, initialises communication with the reconfigurable computer and then passes the data to the FPGA via the PCI bus. Once the hardware has completed it's processing, the server then communicates

with the reconfigurable computer to read the data back from the FPGA. The data read from the FPGA is then passed back to the client for further processing.

5.3. Hardware Architecture

The hardware application was developed in VHDL, synthesized and Xilinx place and route tools were used to create the bitstream file. The reordering application used 30 CLBs in total area. The complete application used many more but these implemented the control of the FPGA as well as data I/O.

6. Results

The primary test undertaken on our implementation of the MP3 hardware reorder was to benchmark it against total CPU time usage. The test involved executing the complete software implementation (standard MADplay distribution) followed by the hardware/software implementation. Obviously we would be hoping the hardware/software implementation would reduce the overall CPU usage, thus providing more CPU time for other software tasks. The test involved using the standard Linux system tool 'top', which logs CPU usage every 5 seconds [11].

We graphed the two results (see figure 4) and compared the data. The graph clearly indicates the hardware reordering saved CPU time. From the raw data it was calculated that 7.5% of CPU time was saved overall on average. This value is however dependent of the MP3 used, as different MP3s use varying amounts of reordering during decoding. If we used an ideal MP3 input file, which used the reorder function more heavily, we estimate over a short period of time the CPU usage reduction would rise to at least 15%.

An overhead that wasn't taken into account in the results above was the extra CPU usage introduced due to the interface between the operating system and Wildcard platform. We used the standard device driver that was shipped with the platform and as the source code for the driver wasn't available we were unable to optimise the code to minimise this overhead. With access to this source code we would be able to modify the device driver in such a way that we could improve the results. At this stage we didn't perform a simulation of the hardware to determine the maximum speed up without using the Wildcard interface.

We believe any improvement using an FPGA development board is a great result and since only one of the many functions of MP3 algorithm was transferred to into hardware, if the more CPU intensive functions were moved we would expect an even greater result.

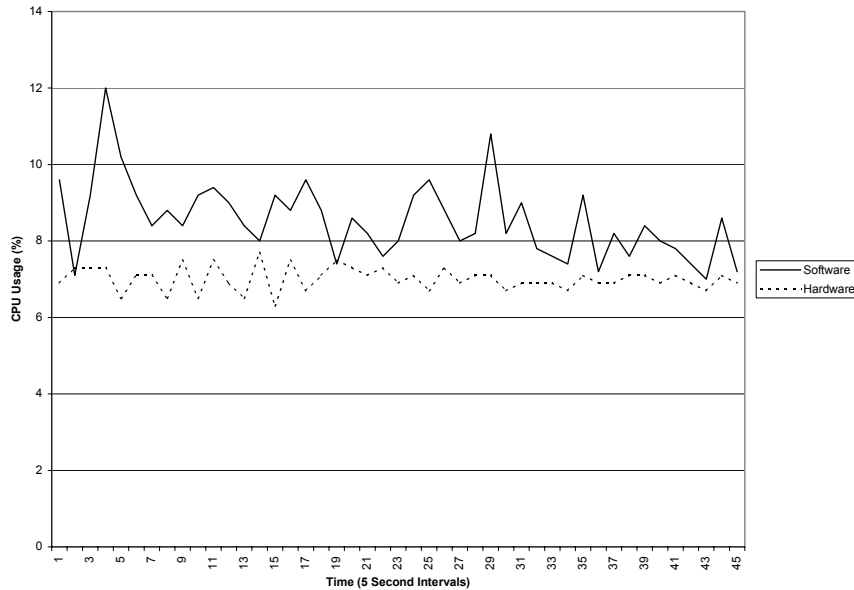


Figure 4 - Graph of CPU usage

7. Future work

As described in section 4 the only part of the MP3 algorithm implemented in hardware is the reordering. With the client/server model in place, and the abstraction of the hardware created because of this, there is a possibility to implement more of the algorithm and thus further saving CPU usage.

This application will also be ported to an operating system (OS) for reconfigurable computing that is being developed by the authors. Having part of the algorithm in hardware and part of the algorithm in software is an ideal application to test the performance of the OS. It will involve the OS creating a memory controller which will be able to load the streaming data to the FPGA implemented hardware, the application will then modify the data according to the algorithm and the OS will then pass the data back to the software part of the algorithm to finish the process.

Possible future outcomes of our project could span out into other applications such as DVD decoding and picture and movie compression.

8. Conclusion

In conclusion the results obtained prove that there are gains to be made in considering a hardware/software implementation to decrease CPU consumption for CPU intensive tasks. Although the gains may seem small, we consider this to be an exciting discovery. These gains

can be attributed to three main features of our prototype. Firstly our design avoids having to move/copy data internally between the chip and on-board memory, by processing the data as it arrives and storing at its calculated position, using address manipulation.

Secondly, the use of a server/client architecture. The client/server architecture effectively hid the hardware from the software so that changes to the software or hardware were independent and seamless. This architecture also makes it possible for our prototype to be used over a network and makes it easier to maintain and port to other operating systems.

Thirdly, the use of traditional software development techniques and profiling to determine which operations would benefit from being implemented in hardware. Although the function chosen here was not the highest ranked in terms of CPU usage, we still were able to achieve a good decrease in performance and from this we feel we have proven that the use of software techniques in the development of hardware/software multimedia applications have been ratified.

9. References

- [1] M. Hachman, "Startup Portalplayer takes on chip giants in MP3 play," EDTN Network. 31 October 2001.
<http://www.ebnonline.com/story/OEG20000605S0054>
- [2] M. Hachman, "The Audio Version Of Pokemon? MP3 Spells Another Big Chip Market," BYTE

- Magazine. 31 October 2001.
<http://www.byte.com/documents/s=394/byt20000607s0014/index2.htm>
- [3] STMicroElectronics, "MP3: The Sound of Music," 31 October 2001.
http://www.angliac.co.uk/extras/articles_mp3.asp
 - [4] K. Salomonsen, S. Sogaard, and E. Larsen, "Design and Implementation of an MPEG/Audio Layer III Bitstream Processor," Aalborg University, Aalborg 1997.
 - [5] K. Brandenburg, "MP3 and AAC explained," In 17th International Conference on High Quality Audio Coding, Florence, Italy, 1999.
 - [6] K. Brandenburg and H. Popp, "An Introduction to MPEG Layer-3," Fraunhofer Institute for Integrated Circuits, Erlangen, Germany 2000.
 - [7] R. Leslie, "MAD: MPEG Audio Decoder," 1st November 2001.
<http://www.mars.org/home/rob/proj/mpeg>
 - [8] S. Haynes, A. Ferrari, and P. Cheung, "Flexible Reconfigurable Multiplier Blocks suitable for Enhancing the Architecture of FPGAs," presented at Custom Integrated Circuit Conference, CICC'99, 1999.
 - [9] Annapolis_Micro_Systems, "Wildcard Reference Manual V2.0," Annapolis, Maryland 1999.
 - [10] P. Ashenden, The VHDL Cookbook, 1st ed. Adelaide: University of South Australia, 1990.
 - [11] Z. Buckholz, "Monitoring System Performance with MRTG," Daemon News. 1st November 2001.
<http://www.daemonnews.org/20001/sysperf.html>
 - [12] G. Wigley and D. Kearney, "The Development of an Operating System for Reconfigurable Computing," presented at IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'01), Napa Valley, 2001.