

The design of a very low power MP3 decoder accelerator

P. Dytrych, M. Adé, J. Coninx, J. David, P. Vandebroek

{peter.dytrych, marleen.ade, jeroen.coninx, j.david, patrick.vandebroek}@philips.com
Philips PDSL Leuven
Interleuvenlaan 74-82
B-3001 Leuven
Belgium

Abstract

Today's portable multi-media market continues to apply increasing pressure on product power consumption, computational throughput, flexibility, product lifetimes and cost. These, sometimes conflicting, trends result in development methodologies that favor sub-system developments that use aggressive power reduction techniques within an architecture that is efficiently programmable whilst offering itself in a soft easily integrated format for inclusion in complete systems on a chip.

This paper will present the development of an ultra low power, low cost audio sub-system initially driven by an MP3 decoder application. Some of the main hardware and software techniques, which were used for power reduction will be outlined. Key to the development was the use of a re-targetable compiler from Target Compiler Technologies which allowed tight co-development in the important area of source code optimizations, instruction set architecture design and micro-architecture.

Preliminary results indicate that the system which has been developed offers very good power efficiency whilst still maintaining a large degree of programming flexibility in C and can be delivered in a soft package comprising VHDL RT level code.

1. Introduction

Power consumption continues to be a very important issue in applications. This is due to the increase in portable products on the one hand and their higher CPU needs on the other hand. The portable MP3 players that are popping up in the market are a good example. The CPU requirements for these devices increase rapidly. The algorithms to be executed become increasingly computationally intensive at the one hand. On the other hand a larger and larger number of functions needs to be executed. At the same time product lifetimes are decreasing and this puts increasing pressure on product development cycle times, particularly the software component of these projects. Yet another factor is the ability to easily integrate the CPU within a larger system on a chip, this tends to favor a soft core methodology.

To be able to play a role in this market segment, it is necessary to have a good, low power methodology. Many cores in the open market are optimized to achieve high computational throughputs but have poor power efficiencies or are difficult to integrate into products due to their full custom designs relying on specific processes.

The goal of our project was to design a minimal system which exhibited very high power efficiency whilst offering a quick development route for new applications, via C compiler technology, in a soft core package.

In this paper we describe our work on an ultra low power MP3 accelerator. The goal is to have a main MP3 decoder running at less than 20MIPS (without IOs) and a power consumption of a few mW. On the other hand limited extra flexibility was needed such that it can also handle other audio decoder algorithms.

Two philosophies that were present for the entire project were minimalism and a holistic approach to system design. The aspect of minimalism was particularly dominant in the definition phase of the project and resulted in a feature set which was carefully managed and started with capabilities which were sufficient for the target applications. Extensions to this set were only permitted if significant benefits were found which incurred relatively small costs, 'creeping specifications' were particularly avoided.

The holistic approach forced the system architect to keep all aspects of the design flows from 'sea of C' to 'GDSII and object code' in mind as much as possible. This resulted in the early development of a fully functional VLSI flow, from RTL to laid out database, as well as producing early processor nML and VHDL descriptions that enabled the quick adoption of working compilers and instruction set simulators.

With this environment in place and an early version of the processor all difficult tradeoffs could be explored in an incremental fashion whilst progressing the design. Past experience also greatly aided this process by concentrating on known problem areas, an example of this being VLSI routing efficiency versus various micro-architectural aspects such as pipeline bypass strategies.

The implementation of the MP3 algorithm started from a C description. To ease the development of the accelerator, the development environment of Target Compiler Technologies is used. The Target Compiler Technology tools allow to define a specific processor architecture and generate a dedicated C-compiler for it.

The introduction will give the background to the work as well as developing some design philosophies that were dominant throughout the project. The hardware architecture will be covered next with specific focus on mechanisms and methodologies for power reduction. This is followed by a section on software optimizations, again the main driver here was power reduction through algorithmic optimizations which leveraged the underlying hardware resources. Finally some preliminary results and conclusions will be outlined.

2. The hardware core

In this section the hardware of the MP3 decoder accelerator is described. The aim of the design was to have it as small as possible, but also such that the compiler can easily optimize the compiled C code. An overview of the datapath is shown in figure 1. There are two multipliers, three ALU's and four register files. These four register files are very small, but are needed to achieve low power consumption where it is better to have more, small register files than just one, big one.

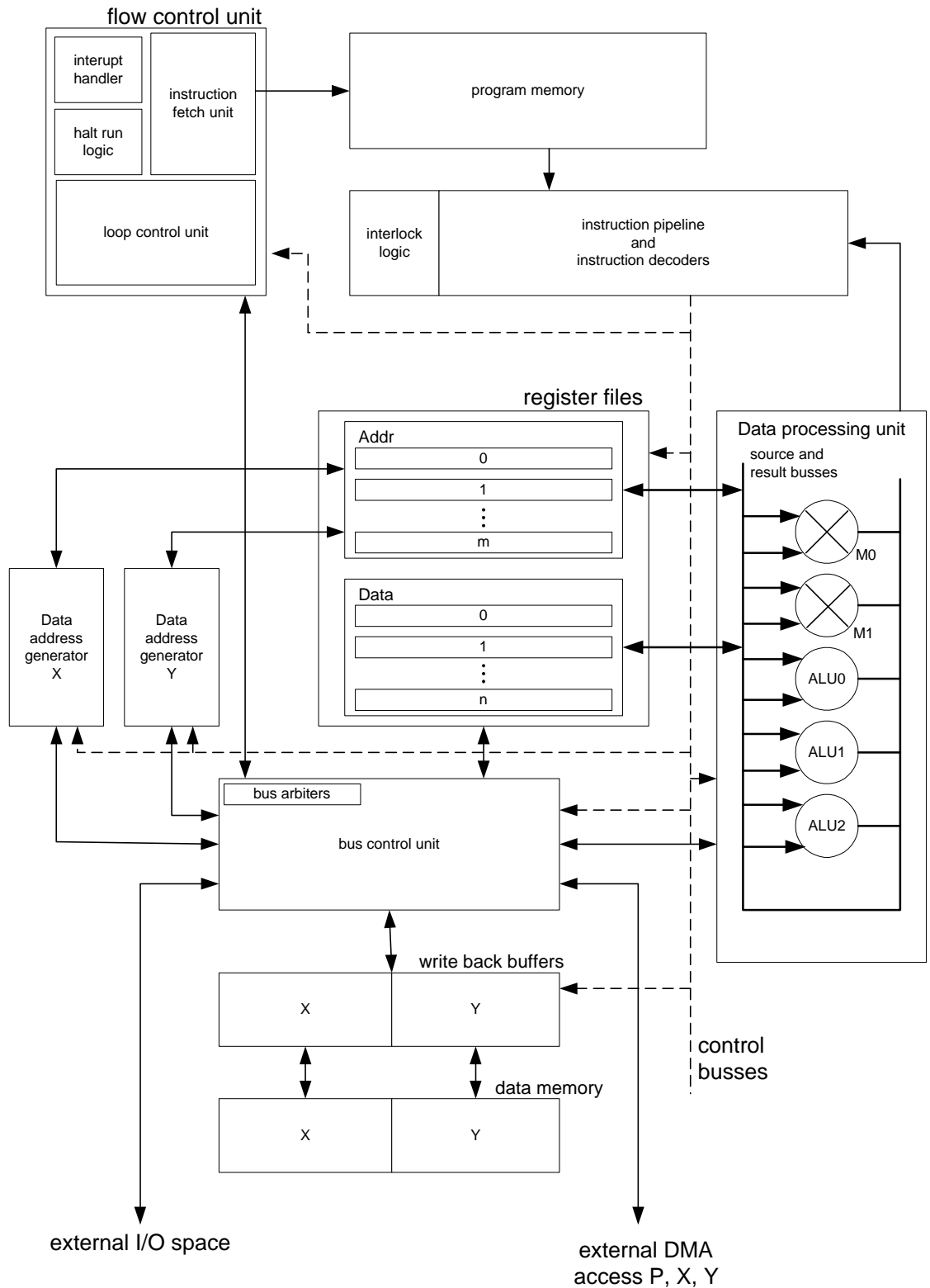


Figure 1: A schematic view of the hardware of the MP3 accelerator.

Next to the datapath, there are several other modules in the core. The most important ones are the two write back buffers, an instruction decoder, a loop control unit, an interrupt controller, a flow control unit and a data address generator. Of course there are, since it is a dual Harvard architecture, the three major memory units, the X data memory, the Y data memory and the program memory.

2.1 ISA (instruction set architecture)

The processor has a carefully optimized ISA which attempts to make a fine balance amongst code size, instruction decoder size and compiler efficiency in scheduling ILP (instruction level parallelism). This was a difficult balance to achieve due to the need to maintain orthogonality in much of the ISA to keep compilation efficient whilst maintaining a short instruction word.

Much time was spent in architectural exploration within the design environment for this part of the work, source code transformations were also considered within these co-developments. Another factor which confounds this process is the very mixed nature of much application code which is typically composed of both intense DSP based kernels and much irregular 'control code'.

Such code is usually described by the 20/80 rule – 80% of the cycles are spent in 20% of the code. The difficulty here is to design an ISA which allows optimum use of ILP within the highly parallel DSP kernels (20% of the code) which are typified by nested arithmetic and memory intensive loops whilst giving good code density and cycle efficiency for the remaining, largely unparallel, 'control code'.

To support such diverse needs the ISA of the CPU can be thought of as having several distinct facets, in the form of instruction classes, that implement various styles of computation. So for example there is a relatively non-parallel 'RISC like' facet as well as one that codes for maximum parallelism in DSP kernels.

The resulting instruction set is a fine balance between the needs described as well as the pipeline structure of the CPU, the register structure and many other factors of the micro-architecture. We currently measure an average instruction width of approximately 25bits/instruction across full applications like MP3 decode.

2.2 Power reduction techniques

2.2.1 Standard

The CPU uses all of the standard techniques for low power design, mainly based around micro-clock gating, operand isolation and general unnecessary toggle reduction techniques, these will not be elaborated on. We also use a technology library, including SRAMs, which allows use of aggressive voltage scaling to below 1V.

2.2.2 Pipeline

A pipeline structure was developed that allowed use of a single edge clock whilst giving memory access a full clock cycle. This will tend to maximize execution clock rate thus allowing most scope for voltage scaling which gives approx quadratic power reduction.

The pipeline structure is also finely balanced with the rest of the micro-architecture design resulting in minimal and simple interlocks, minimal bypassing and minimal length control transfers amongst pipeline segments.

The pipeline also leverages the ISA structure by utilizing a small number of distributed instruction decoders as opposed to one large one thus reducing unnecessary logic toggling.

2.2.3 Locality of reference

The CPU uses distributed register files, local to their respective computational resources, to reduce power consumption. The ISA also includes some coupling mechanisms that attempt to reduce the need for copies amongst register files. Another factor is that the pipeline structure of the CPU was designed so as to minimize the need for bypassing mechanisms, these advantages all add up when compared to using a single multi-ported central register file. For a machine with the parallelism that is available within our CPU this is a large advantage in both power and maximum clock frequency.

2.2.4 Memories

Significant area (cost) and power consumption is now evident in the memory sub-systems of CPUs. Several techniques have been used to reduce both the size and power consumption of these memories. Globally all memory spaces are made up of smaller physical segments such that only one is active in any space at any cycle.

2.2.5 Data memory

Data memory utilization is optimized by allowing use of efficient data structures that are supported by powerful addressing modes within the CPU. Both cyclic and bit reversed addressing are supported as well as other common DSP addressing modes. Particularly efficient stack support is provided allowing efficient linkage and local storage for functions. These techniques ensure that data memory requirements are minimized as well as execution cycles with the provision of efficient addressing modes. On the architectural front a good balance has been sought between memory sub-system costs and the ability to provide sufficient memory bandwidth for the parallel computational units.

2.2.6 Program memory (Compression)

To increase the efficiency of the program memory, innovative techniques for the code size reduction have been included, on top of the very efficient code that is generated by the TCT C-compiler anyway. E.g. an efficient method for code compression has been included. A major tradeoff in designing the ISA was the width of the instruction word against the amount of encoding used whilst still leaving enough degrees of freedom for the compiler to perform code selection well. An ISA with key areas that were orthogonal was developed and this also allowed a form of NOP compression to be used. This has been leveraged by the use of scheduling techniques within the compiler, which favor the generation of instruction sequences, which allow maximal compression to be used. We have measured savings between 20% and 25% for typical applications.

2.2.7 General

The CPU supports extensive IO facilities allowing easy, efficient interfacing to other systems. Particularly, interrupts are implemented in a complete and robust way. The interrupt system is characterized by very low latency so that even single instruction hardware loops are fully interruptible. This allows a minimal amount of specific buffering to be implemented thus keeping system costs low.

The CPU is implemented in VHDL at RT level and a typical semi-custom ASIC VLSI design flow is used. The requirements for low power are carefully considered in all stages of the VLSI design flow as well, particularly in the synthesis and layout areas.

3. The MP3 decoder: software optimizations

3.1 The MP3 decoding algorithm

MP3 started as a synonym for the audio MPEG-1 standard, implementing its most complex layer 3 compression methodology. In the past few years, it was extended to lower sampling frequencies in MPEG-2 LSF, which gives better performance with lower sample frequencies at lower bit rates. The (still unofficial) MPEG-2.5 has decreased the sample rate even further. Our decoder supports all three standards, at all defined fixed and free-format bit rates. It is an ISO/IEC compliant full layer3 audio decoder.

The functional block diagram of an MP3 decoder in general is depicted in Figure 2. The bitstream is organized in frames. One frame contains the data for 1152 samples per channel for MPEG-1, or 576 for MPEG-2 and MPEG2.5. The first step is to parse the bitstream for the data for one frame. Then scalefactors are reconstructed, and the Huffman symbols are decoded. With the scalefactors, the symbols are requantized. In stereo mode, techniques to reduce the overall bit requirements for the two channels may have been applied, e.g. based on correlations between the two channels. Stereo processing will reconstruct the two channels separately. The IMDCT (inverse modified discrete cosine transform) will reduce the spectral resolution to the 32 frequency subbands that are input for the polyphase filter, which will then convert these subbands into the time domain to produce the pcm samples.

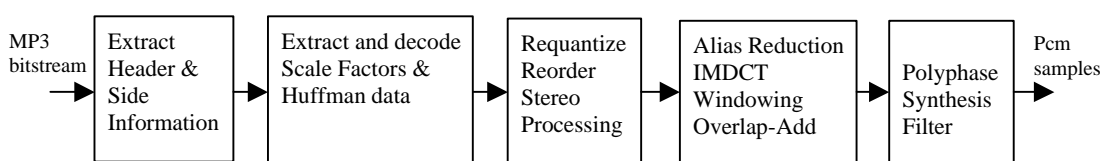


Figure 2: MP3 decoder block diagram

3.2 Mapping software onto the new platform

The starting point was a PC-functional 24-bit fixed-point C-implementation. In a first phase, this code had to be compiled onto the new CPU to obtain exactly the same results as on the PC. The integer C built-in data types are supported as such by the TCT tools, and are mapped in a straightforward fashion onto the processor's architecture single and double word provisions. The code also provides a user defined double precision fractional type that mimics the behavior of an accumulator with overflow bits. To support

this type, a new primitive data type along with its operations had to be defined in the processor model, allowing the compiler to do an efficient mapping of this type.

To obtain the same finite-word-length effects on the platform as with the PC application, sometimes extra explicit conversions needed to be introduced in the C-code, or new rules were defined for the compiler to support some constructs. This is a quite tedious phase in the process. The result of this phase was functionally correct object code for the new device, taking approximately 5K program memory words.

3.3 Software optimization strategies

In the second phase, the software is being optimized on the new platform. As saving on cycle count will allow almost quadratic power savings (since less cycles allow for a lower clock frequency, and thus for a downscaling of the voltage) this becomes the primary issue. Of course, program and data memory sizes will not be neglected in doing so. Thanks to the presence of a C-compiler, many changes can be made and evaluated on the platform in a relatively short amount of time. This would be hardly possible if one was to write the code in an assembler language. High-level optimizations like complete rewrites of some functions, changing data structures or modifying loop-nesting order have a major impact on the code, and would imply great development times, without the a priori guarantee of substantial savings.

Three basic types of software optimizations are executed. First processor independent optimizations of the code are introduced. Secondly processor dependent optimizations are performed, based on knowledge of the processor architecture, and feedback from the compiled code and profiling. Thirdly, the code is also modified such that the compiler can better exploit the parallelism in the hardware architecture.

3.3.1 Processor independent optimizations

The first optimization consists of careful checking of the code for processor independent optimizations. Key issues here are:

- Reduce computational complexity
E.g. for the core of the Huffman decoding, another decoding strategy can be selected that almost halves the average time for decoding one symbol, with a penalty of larger table sizes.
- Keep the amount of memory accesses as small as possible
This is in general a good practice. For our new platform, as for any other similar DSP architecture, it is even extremely important. It can be achieved by selecting alternative algorithms on one hand, or by applying techniques for data-reuse on the other hand. The latter can also lead to major reworks of the code, to change the order of the computations, but it will not alter the computations themselves. It may have repercussions on address calculations.
- Perform function calls only when strictly needed
E.g. for some specific parameter values, a function could actually perform no useful actions. These cases should be intercepted.
- Replace rather general applicable functions with specialized more efficient versions
- Remove false paths as much as possible based on compile-time analysis
- Reduce the number of arguments of a function where possible

E.g. modify data structures to combine related data, or create more specific functions that take less arguments.

The reuse of data has to be optimized. By also taking into account the architecture of the processor, the number of memory accesses can be reduced.

3.3.2 Processor dependent optimizations

Also processor dependent optimizations are looked at. One important element is to avoid memory spills: due to the very limited number of registers, they are very likely to occur frequently even in the regular computation loops: absolute care has to be taken not to use more than the number of available accumulator registers. Otherwise, their content will be spilled to and reloaded from memory, taking 3 cycles for each movement, and thus deteriorating performance in a totally unacceptable way. This is particularly important when applying the general strategy for data reuse: it will limit the freedom drastically there. To a lesser extent, the same holds for the number of pointers into X and Y memory. Another issue is pointer increments: for steps larger than 2, no hardware provisions are made, and the step size has to be set explicitly, so it is important to keep the step sizes constant when possible.

The data type 'long' is mapped on a double precision word on the new CPU. If it can be shown that some data with this type actually never exceeds the boundary of a single word, then both from a memory size point of view and from access time, it is better to change its type to a single word type.

To allow the compiler to fully exploit the parallelism of the architecture, pragmas are introduced. These small statements on some selected data arrays tell the compiler in which data memory section to map the array. In this way data can explicitly be directed towards the X or Y memory. E.g. for computing filters, the coefficients could be redirected to Y memory, and with the input data in X memory, parallel loads can occur. With more data available, possibly the two MAC units can be active in parallel, gaining even more cycles.

3.3.3 Compiler co-optimizations

During the development also the compiler has been optimized such that it can detect structures in the code that can easily be supported by the hardware of the processor. E.g. the compiler could select the hardware support for modulo buffer addressing through the use of a special function in the C-code for updating the pointers. It has also been improved to make better use of the hardware do-loop provisions, which can result in gains of 20% in functions with large loop counts.

3.4 Examples

On a first example module, a sub-module of the polyphase filter, processor independent optimization focused on data-reuse. In theory a great amount of memory accesses could be saved. In reality, the limitations of the size of the register files and accumulators in the processor put a strict bound on this. The maximum saving that could be obtained by rearranging loops and calculations was 15%. On the other hand, making the modulo buffer accessing visible to the compiler resulted in over 50% saving. A few more % could

be saved by limiting the amount of pointers used to the amount of pointers available in the CPU and by carefully selecting the step sizes that modified the pointer indices.

Another example is the bitstream-parsing functions. Here, a new set of algorithms, combined with introducing more specific functions and some data structure modifications, reduced computational efforts as well as memory accesses drastically, to save up to 80% of the cycles spent. Processor dependent optimizations are limited since this part is dominated by memory accesses and bit manipulations. Introducing modulo buffer accessing again saved 20% more of the remaining cycles.

4. Preliminary results

The typical case in MP3 coded audio compresses stereo sound at 44.1kHz sampling rate to a bit-rate of 128 kbps. For this configuration, the unaltered code would have taken over 40 MIPS. After the compiler co-optimizations, the bit stream parsing optimizations, and introducing some computing parallelism, we were able to reduce this number to 25 MIPS. We expect to reduce this number further to well below 20 MIPS, with the range of techniques described above.

The implementation of the hardware core is finished and validation is progressing. The current target technology is CMOS18. The first simulations of the area, timing and the power consumption are available. The gate count is about 45 Kgates. Initial simulations show that the power consumption of an MP3 decoder will be below 5mW.

5. Conclusions

In this paper we have presented the design of a re-programmable, ultra low power, low cost MP3 core. To ease the development the Target Compiler Technology development environment has been used. It also allows a greater flexibility if changes or extensions have to be made in the future.

Due to the carefully chosen instruction set architecture, the efficient compiler and a careful design of the core, the estimated power consumption for MP3 decoding of a compressed stereo sound at 44.1kHz is below 5mW.

Finally it should be noted that due to the tight time scales for this project only limited time was spent on many of the optimization phases, it is believed that there is great scope for further improvement.

6. References

ISO/IEC 11172-3, Information technology - Coding of moving pictures and associated audio for digital storage media at up to about 1.5Mbit/s - Part 3: Audio, 1993.

ISO/IEC 13818-3, Information technology – Generic coding of moving pictures and associated audio information - Part 3: Audio, 1998.