Using Low Precision Floating Point Numbers to Reduce Memory Cost for MP3 Decoding

Johan Eilert, Andreas Ehliar, Dake Liu
Dept. of Electrical Engineering
Linköping University
S-581 83 Linköping, Sweden
Email: {je,ehliar,dake}@isy.liu.se

***Abstract*— The purpose of our work has been to evaluate if it is practical to use a 16-bit floating point representation to store the intermediate sample values and other data in memory during the decoding of MP3 bit streams. A floating point number representation offers a better trade-off between dynamic range and precision than a fixed point representation for a given word length. Using a floating point representation means that smaller memories can be used which leads to smaller chip area and lower power consumption without reducing sound quality. We have designed and implemented a DSP processor based on 16-bit floating point intermediate storage. The DSP processor is capable of decoding all MP3 bit streams at 20 MHz and this has been demonstrated on an FPGA prototype.**

I. Introduction

MPEG-1 layer III [1], commonly referred to as MP3, is well understood, both on desktop systems and in embedded systems. Decoders for desktop systems can be implemented using either fixed point or floating point arithmetic, whereas embedded systems typically use fixed point arithmetic.

Embedded MP3 decoders usually have to use two 16-bit memory words for each intermediate value to achieve the required dynamic range and precision with fixed point arithmetic. We have investigated the feasibility of using a *16-bit floating point* representation to reduce the memory cost without sacrificing sound quality. This would halve the data memory usage which would have a significant impact on power consumption and chip area. Another advantage with floating point arithmetic is that the hardware eliminates all scaling operations associated with fixed point arithmetic which leads to shorter firmware development time.

One drawback of floating point is the complexity of the arithmetic units. However, for a given dynamic range, the multiplier in a floating point data path is smaller than the corresponding multiplier in a fixed point data path.

In order to evaluate our floating point approach, we have used the MPEG audio compliance test [3]. In short, a decoder can be classified as *full precision*, *limited accuracy*, or *not compliant* depending on the difference between the provided reference output and the decoded output. We have also conducted informal listening tests since there are no formal criteria for evaluating the quality of an MP3 decoder for an arbitrary bit stream.

II. Floating Point Requirements

In order to design a system with floating point arithmetic, two important design decisions of the system have to be made. One is the floating point format which decides the range and precision of all values that can be handled by the system. The other decision is the arithmetic operations that should be supported in hardware for a given target application.

A. The Floating Point Format

Although it is possible to analytically determine the maximum values encountered in an MP3 decoder, this information is not really useful. For example, by setting the gain and scale factors to their maximum values, it is possible to create a synthetic MP3 bit stream where the final output samples are magnitudes larger than the allowed output range. Because of this, we did not try to perform any formal analysis of the possible number ranges occurring in MP3 decoding.

Instead, we instrumented the ISO MP3 decoder [2] to use our own custom floating point arithmetic library with configurable mantissa and exponent widths. The library also supported arithmetic with mixed precision in order to mimic a processor with high precision data path but lower precision memory. By keeping track of the smallest and largest values encountered in the decoder, the library was used for determining the required dynamic range.

Our goal was to find an exponent configuration where all MP3 bit streams could be decoded without having to saturate any intermediate value. We did not consider hand-crafted bit streams with extreme values but we tested more than 200 different music and speech bit streams.

We concluded that all normal bit streams could be decoded successfully with an exponent size of 5 bits in data memory. The exponent bias was selected to give a number range of approximately 2^{-26} to 2^{5} which would correspond to the dynamic range of a 32-bit fixed point processor.

In order to simplify the hardware, we used the same bias for register values, but we had to increase the exponent to 6 bits to accommodate larger intermediate values. The register number range is 2^{-42} to 2^{21}. The larger exponent of the registers simplified software development.

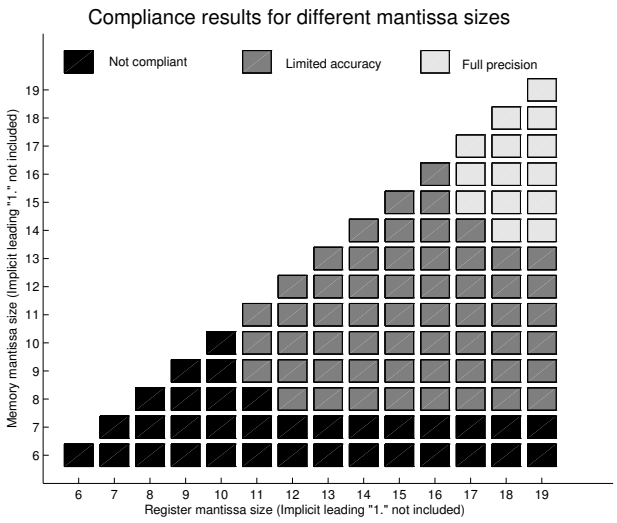

Fig. 1. A comparison of the compliance results for different mantissa sizes in the ISO decoder.

While the choice of exponent influences the magnitude of the floating point values, the size of the mantissa corresponds to the number of significant digits in the calculations. A larger mantissa leads to higher precision, but also larger memories for intermediate storage. It is therefore important to determine the minimal size that gives acceptable results. This can be determined through listening tests, or numerical methods, such as the one used for MP3 decoder compliance testing.

An MP3 decoder is tested by decoding a bit stream supplied in the compliance test and comparing the output with a supplied reference output. If the rms of the difference is less than $8.8 \cdot 10^{-6}$ and the absolute difference is less than 2^{-14} relative to full scale for all samples, the decoder is classified as a full precision decoder. Otherwise, if the rms of the difference is less than $1.4 \cdot 10^{-4}$ regardless of the maximum absolute difference, the decoder is classified as a limited accuracy decoder. If the decoder fails to meet these criteria, the decoder is not compliant.

The compliance level for different sizes of the mantissa was investigated and the result is given in Fig. 1. The exponent sizes used was 6 and 5 in registers and memory respectively.

B. Operations

An analysis of the ISO MP3 decoder shows that the following floating point operations should be supported in hardware to implement an efficient MP3 decoder.

- Add
- Subtract
- Multiply
- Round (Before saving to memory)
- Load floating point value
- Floating point to integer conversion

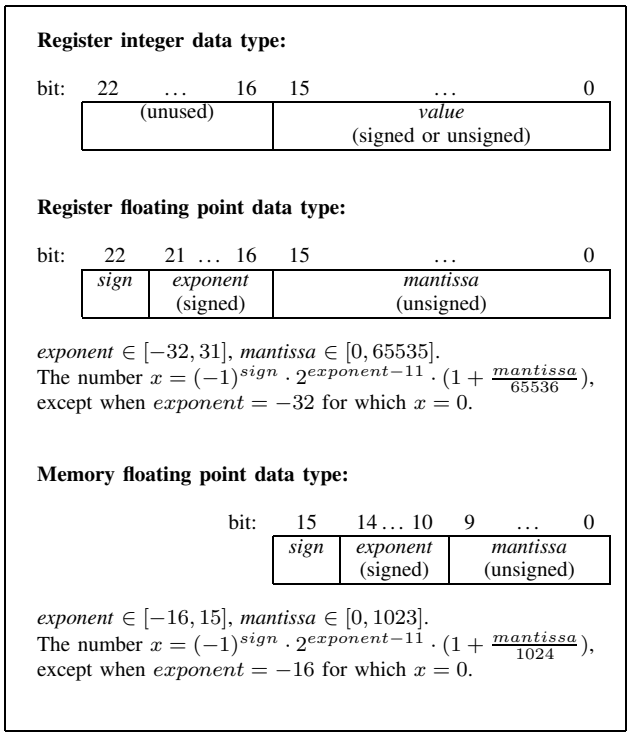

Fig. 2. The main data types in our DSP.

These operations can be mapped to a floating point adder and a floating point multiplier. All remaining operations can be reduced to these primitives or implemented as table look-ups. Because the memory and registers have different word lengths it is necessary to convert between different floating point formats. The round operation converts from the register word length to the memory word length, and the floating point load operation expands a memory word to a register word.

III. Hardware Implementation

As a proof of concept, we developed a simple pipelined DSP core to prove the feasibility of the approach outlined above. The DSP core is a load-store architecture with separate program, data, and constant memories. The general idea was to keep the hardware reasonably simple without making the software unreasonably complex. In our experience, software is generally easier to debug than hardware. The instruction set was kept at a minimum and the hardware had no inter-instruction dependency checking.

A. Data types

Each general purpose register can contain a 16-bit integer or a 23-bit floating point value. In the former case, the upper 7 bits are unused. When a floating point value is loaded from memory it is expanded from 16 to 23 bits. Before storing a floating point value it is rounded to 16 bits. The data types are summarized in Fig. 2.

The most important reason for using these values is to avoid a configuration where the decoder barely meets the requirements for limited accuracy. Another reason is the convenience of having a 16-bit wide memory.

B. Instruction Set

The instruction set basically consisted of load and store from any of the general purpose registers, register to register integer and floating point operations, and I/O operations.

There are 16 general purpose registers. This number was decided upon after studying the algorithms used in MP3 decoding. It allowed us to keep all intermediate values in registers for the most important algorithms.

There is a hardware stack for saving the program counter during subroutine calls. Conditional branches are limited to branch-if-zero, and branch-if-not-zero.

There are a few application specific instructions. The Huffman decoder part is accelerated by bit access instructions, and some signal processing parts are accelerated with a MAC (multiply-and-accumulate) instruction. The address generation capabilities are in most cases limited to absolute or register indirect, but the bit access instructions and the floating point MAC instruction can use the single dedicated address register with auto-increment and modulo addressing.

The integer pipeline has five pipeline stages, and the floating point pipeline has eight stages. The pipelines share fetch, decode, and write-back stages. In hindsight, the pipeline could have been shorter.

RTL code for the DSP was written in VHDL and tested on an FPGA prototype board. The estimated gate count, excluding memories, is 32500 gates when synthesized for Leonardo Spectrum's sample SCL05u technology. There is room for improvement in the RTL code, especially in the instruction decoder.

IV. Software Implementation

We decided to implement a new MP3 decoder from scratch rather than building upon the ISO MP3 decoder. This was done partly to learn as much as possible about MP3 decoding and partly because we felt that the ISO MP3 decoder was too complex and inefficient. This new decoder was then used as our internal reference during the assembly code development for the DSP.

A. Algorithms

In order to achieve high performance with a deep pipeline and a limited instruction set, algorithms had to be carefully written. Since the integer part of a register is used as the mantissa in a floating point value, some operations can be accelerated by manipulating the mantissa directly. For example, integer shift and integer to floating point conversion can be implemented by using the floating point subtract instruction.

The Huffman decoder uses a simple, one bit at a time, tree traversal technique. This approach is memory inefficient but reasonably fast since each tree node is one instruction.

The $x^{4/3}$ calculation in the sample dequantization can be implemented with a large look-up table with more than 8000 entries. We used a fifth order polynomial approximation for the mantissa and a table look-up for the exponent. Finally, a look-up table was used for small values in the range $[-15, 15]$ to accelerate this common case.

The 36-point inverse modified DCT, IMDCT, was implemented using a fast IMDCT algorithm [4] and the 12-point IMDCT was implemented using 36 floating point multiply and accumulate instructions.

The 32-point DCT used in the subband synthesis part was implemented using Lee's fast DCT algorithm [5]. With careful scheduling, the 16-point kernel could be implemented in registers only, without loading or storing temporary values to memory.

B. Quality

According to the MP3 compliance test, our decoder is classified as a limited accuracy MPEG-1 Layer III decoder. The rms of the difference between our decoded output and the reference provided with the compliance test is $3.2 \cdot 10^{-5}$ which is well below the limit for limited accuracy, $1.4 \cdot 10^{-4}$.

Even though our decoder is not a full precision layer III decoder, informal listening tests could not discern files decoded with our decoder from files decoded with the full precision ISO MP3 decoder.

C. Memory Use

The final version of the decoder used approximately 6800 24-bit words for program memory, 900 23-bit words for the constant memory, and 6100 16-bit words for data memory. We have not spent any time trying to reduce the program memory size. More than 40% of the program memory is used for the Huffman tables.

D. Performance

In order to measure the performance of the decoder on a typical MP3 bit stream we used a 44.1 kHz music bit stream, with an average bit rate of 202 kbps. A profile of the decoder is shown in Fig. 3.

The time spent in the Huffman decoding and sample dequantization is data dependent. A bit stream was constructed to trigger worst case execution time in the data dependent parts. In our case, this consisted of a 48 kHz bit stream using only short blocks and joint-stereo. By selecting the right Huffman table, a maximum number of big values could be fitted into a frame to stress the sample dequantization. The resulting worst case execution path requires 19.6 MIPS to sustain a real time decoding process. The worst case profile is shown in Fig. 4

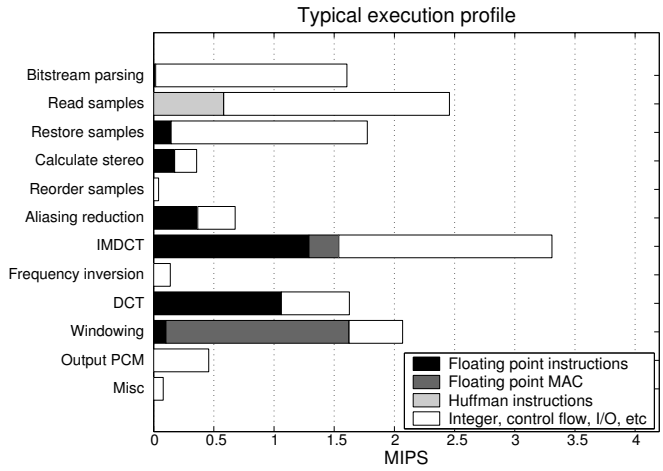

Fig. 3. Profiling of the decoder while decoding a typical MP3 bit stream. (14.6 MIPS in total.)

Fig. 4. Profiling of the decoder while decoding the worst case MP3 bit stream. (19.6 MIPS in total.)

V. Future Work

The focus of this work has so far been on the effects of using floating point arithmetic. Therefore, we have not put very much effort in optimizing the instruction set beyond what is needed to support the required floating point operations. Future improvements could include hardware assisted loops, and better address generation such as general support for pointer auto-increment. It would be relatively easy to implement a simple Huffman accelerator unit that would both significantly reduce the size of the Huffman tables as well as speed up the Huffman decoder.

We investigated the word lengths required for full precision, but only in the ISO MP3 decoder, as shown in Fig. 1. It would be interesting to verify that full precision can be achieved also in our MP3 decoder by increasing the width of the floating point data types.

Program memory	6800 words (24-bit)
Data memory	6100 words (16-bit)
Constant memory	900 words (23-bit)
Clock frequency	20 MHz
Gate count	32500
MIPS cost (worst case)	19.6 MIPS
MIPS cost (typical)	14.6 MIPS
Compliance	Limited accuracy (rms is $3.2 \cdot 10^{-5}$)

Fig. 5. Performance of our MP3 decoder.

Finally, it would be very interesting to know if anything could be gained by implementing an MP3 encoder or other audio coding standards such as Ogg Vorbis and AAC using a similar floating point scheme.

VI. Conclusions

Our MP3 decoder stores intermediate data in a 16-bit floating point format to limit memory usage. It is classified as a limited accuracy ISO/IEC 11172-3 MPEG-1 layer III decoder.

The hardware has been implemented in VHDL and it has been tested on an FPGA prototype board. The gate count, excluding memories, is 32500 gates when synthesized for Leonardo Spectrum's sample SCL05u technology. A clock frequency of 20 MHz is enough to decode all bit streams.

The performance of the decoder is summarized in Fig. 5. We see some possible improvements that could reduce the program memory size and increase the performance.

References

[1] ISO/IEC, "Information Technology — Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to About 1.5Mbit/s, Part 3: Audio," 1993

[2] "ISO MP3 sources (dist10)," ftp://ftp.tnt.uni-hannover.de/pub/MPEG/audio/mpeg2/software/technical_report/dist10.tar.gz

[3] ISO/IEC, "Information Technology — Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to About 1.5Mbit/s, Part 4: Compliance Testing," 1995

[4] Lee, S.-W., "Improved algorithm for efficient computation of the forward and backward MDCT in MPEG audio coder," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on, Vol. 48, Iss. 10, Oct 2001*

[5] Lee, B., "A new algorithm to compute the discrete cosine Transform," *IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. 32, Iss. 6, Dec 1984*