Design of a Floating Point DSP for Full Precision MPEG-1 Layer II and III Decoding

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

Abstract— **The purpose of this work has been to design and evaluate a floating point DSP for MPEG-1 layer II and III audio decoding. The intention has been to minimize the word length and memory usage while still satisfying the *full precision* requirements for both layer II and III as specified in the standard. This is compared to one of our previous works where we have only aimed for a specific word length and the audio quality was not considered.**

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. Further, using a floating point representation means that smaller memories can be used which leads to smaller chip area and lower power consumption.

The result is a very area and MIPS effective DSP for MPEG-1 audio decoding.

I. Introduction

The MPEG-1 standard for audio coding [1] describes three implementation layers called layer I, II, and III, with corresponding increase of complexity and reduction of bit rate. A decoder that is capable of decoding a certain layer needs very little modification in order to be able to decode the lower complexity layers.

In this work we have focused on the last two layers. Layer II is today used for audio coding in DAB, European digital radio broadcasting, and layer III has become extremely popular for storing music on both personal computers and portable music players due to its high sound quality at low bit rates.

Embedded decoders usually have to use two 16-bit memory words for each intermediate value to achieve the required dynamic range with fixed point arithmetic. We have investigated the feasibility to use a short word length *floating point* representation to reduce the memory cost. This would reduce the memory usage which would have a positive effect on power consumption and chip area. Another advantage with floating point arithmetic is that the hardware eliminates virtually all scaling operations generally associated with fixed point arithmetic which leads to shorter firmware development time.

The rest of this paper is organized as follows: The introduction continues with more background information. Then follows section II with the main motivation behind our work, and section III with the prestudy phase where we determine the minimum precision that fulfills the *full precision* requirement. In section IV and V the DSP is designed. Finally, section VI and VII gives our results and conclusions.

A. Decoder Compliance

An important issue is how the quality of an MPEG-1 decoder is tested. This is done by decoding a special bit stream supplied with the standard and comparing the output to a reference file also supplied with the standard. The root mean square of the difference is calculated and the maximum absolute sample difference is found. From these data the decoder can be classified as *full precision*, *limited accuracy* or *not compliant*, depending on whether or not the values are below certain specified limits.

Naturally, this only tests the decoder for this specific file, and there is no formal way of evaluating the quality of a decoder for an arbitrary bit stream.

B. Earlier Work

We have already built a floating point DSP with 16-bit floating point precision in memory [2]. We have also implemented a layer III decoder for it. It is very data memory efficient, but it cannot satisfy the *full precision* requirement because of the limited precision in memory and registers. It is also not very program memory efficient since it was based on the RISC philosophy with as few instructions and addressing modes as possible in an attempt minimize hardware development and verification time.

II. Motivation

Our previous work has shown that layer III decoding with a floating point DSP and 16 bit data words is entirely feasible. The motivation behind the work presented in this paper was that we wanted to see if it was possible to extend or improve this approach on a number of ways.

- The word length should be increased so that the DSP could achieve *full precision* on both layer II and III. The previous work reached only *limited accuracy* due to its 16-bit memory design restriction.
- The instruction set functional coverage should be increased to allow an efficient implementation of a layer II decoder, but also other audio decoding algorithms.
- The instruction set and addressing modes should be improved in general to reduce the program memory size as well as the MIPS cost.

III. Prestudy

As a first step in the design process, we had to find the required (mantissa) word length necessary to satisfy the *full precision* requirement. Secondly, we studied some other audio coding standards for ideas of new instructions and addressing modes.

A. Word Length for Full Precision

In our previous work we have analyzed the mantissa and exponent word length requirements for layer III. A similar approach has been used for determining the required mantissa and exponent sizes for layer II.

Our own behavior model in C++ for layer III was extended to handle layer II in order to get a deep understanding of this part of the standard. It was also changed to use two special floating point data types for memory and register, respectively, with bit accurate simulations of the intended hardware for arithmetic operations. With this it was possible to study the impact of different mantissa sizes on the decoder output.

The required mantissa for layer III was 18 bits in general purpose registers and 16 bits in memory. The register width is larger because the registers are also used as accumulators. Layer II on the other hand required 22 bit mantissa in registers.

It should be noted that the layer II compliance test is much tougher on the decoder than the layer III compliance test.

The exponent range was not changed from our earlier work since only the mantissa affects the precision of the computations. It is still five bits for memory values and six bits for register values. Layer II in fact uses a smaller number range than layer III.

B. Other Audio Coding Standards

In order to improve the flexibility and the generality of the DSP, a number of other standards for audio coding were studied to get ideas for new instructions and addressing modes.

- The source code of an AAC decoder [3] was studied.
- The AC-3 standard [4] was studied.
- The SBC standard [5] was studied.

The conclusion was that it would be necessary to support large MDCT (modified discrete cosine transform). This can be implemented using FFT and consequently a bit-reversed addressing mode was added to speed up this operation. SBC is very control intensive, but overall not very computationally expensive.

IV. DSP Design

When the exponent and mantissa word length requirements were known, the actual DSP could be designed.

A. Data Types and Registers

Similar to our previous work, the DSP has three native data types.

- Integer and pointers (22 bits).
- Floating point value in register (29 bits).

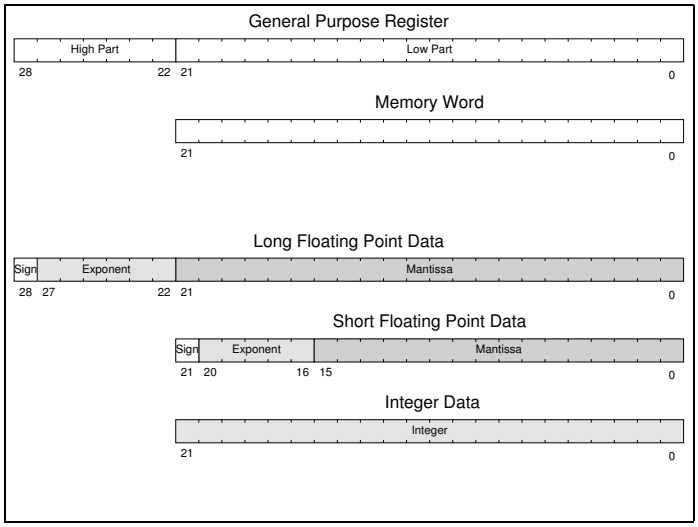

Fig. 1. The native data types of the DSP. The widths of the general purpose registers and a memory word are also shown for comparison.

- Floating point value in memory (22 bits).

The data types are represented graphically in Fig. 1. Note that the memory is 22 bits wide.

All signal processing operations work with the longer floating point data type, and there is a special instruction to round and truncate a value down to the shorter floating point format. The shorter form is used for values in memory and there is no direct hardware support for storing a long floating point value in memory.

The entire programmer accessible register set is shown in Fig. 2. The general purpose registers are used for storing intermediate values during calculations. Integers and pointers use only the low part and the upper part is generally not affected by integer instructions. Note that there are no accumulator registers. The general purpose registers are used for this purpose.

Further, there are loop registers for zero overhead loops and address generator registers.

B. Address Generator

An important part of a DSP is the address generator. Specialized addressing modes can often give a huge performance increase since they can potentially eliminate many instructions related to address calculations from the kernel loops.

After careful study of our C++ behavioral layer II and layer III code, it was concluded that the algorithms chosen in our decoder would need no more than four dedicated address registers.

The following addressing modes are available:

- *Address register indirect*, with optional post-increment or decrement, or increment by other amount specified in step register. There is also an option for modulo addressing with the top and bottom registers that specify the last and the first address of a circular buffer, respectively.

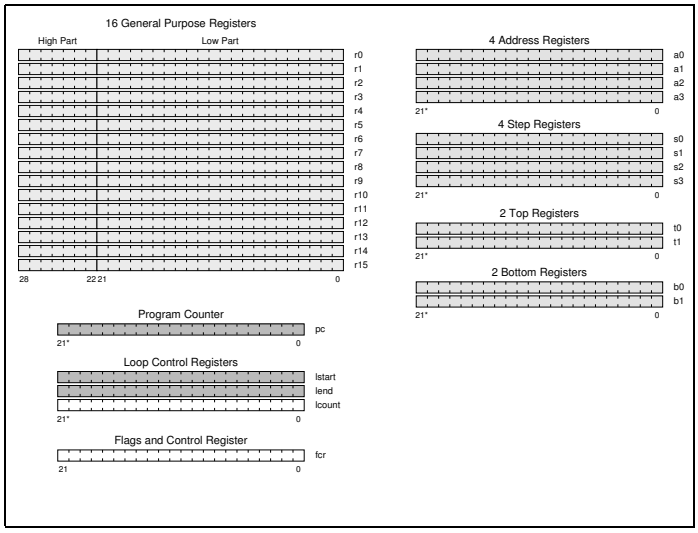

Fig. 2. Programmer accessible registers.

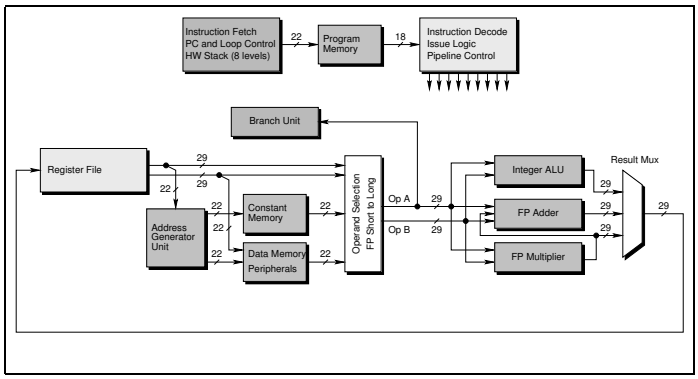

Fig. 3. The principal data flow in the DSP.

- *Absolute address*.
- *General purpose register indirect* which is useful if the address registers are already allocated for other purposes.
- *Indexed*, address register plus general purpose register. This is used for implementing offset adressing. The offset is loaded into the general purpose register.
- Indexed, with *bit-reversed addressing* for FFT which is used for implementing (large) MDCTs.

The first two modes given above can be used for selecting memory operands for most arithmetic instructions. The remaining modes are restricted to only load and store instructions.

V. Micro-architecture

A. Data Flow

The main data flow in the DSP can be seen in Fig. 3. The output from the floating point multiplier is fed back into the floating point adder for efficient implementation of the common multiply-and-accumulate operation.

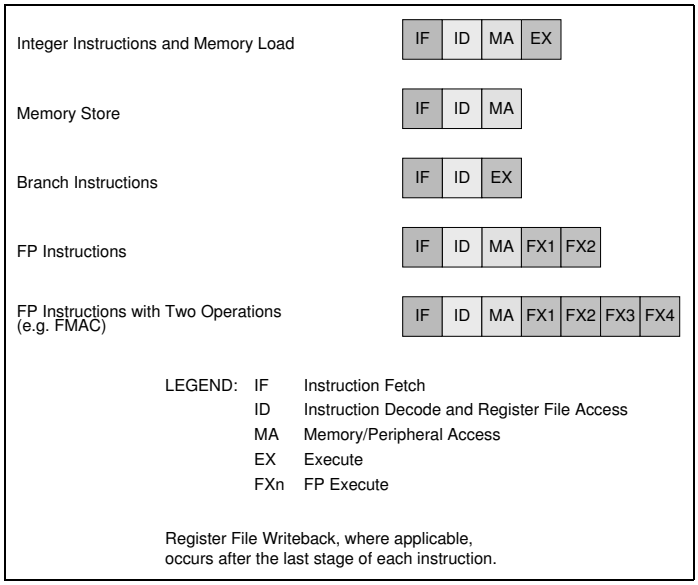

Fig. 4. Pipeline stages for different types of instructions.

The memories are located before the arithmetic units in order to enable memory operands without explicit load instructions which had proved to be one of the bottlenecks of our old design. The alternative approach with load instructions executed in parallel with other instructions could not be used since our register file only has one write port.

B. Data Path Pipeline

The DSP pipeline is shown in Fig. 4.

Our earlier work had used five pipeline stages for the floating point part, partly to make sure that the design could run at full speed on the FPGA board we had available at that time. This put high pressure on the programmer to schedule floating point code properly to gain maximum performance.

The number of pipeline stages was now reduced to two. The integer data path has full result forwarding in order to eliminate all pipeline latencies.

VI. Results

A. Performance Evaluation

Some of the most computationally heavy decoder functions in layer III have been implemented. The new DSP uses fewer executed instructions for all implemented functions, and in all cases significantly shorter program code since the code for the previous DSP often had to be completely unrolled for maximum performance.

The main reason for the improvements are the improved addressing modes and a few instructions that did not exist on the old architecture.

MIPS cost estimations are less than 12 MIPS for decoding of a typical layer III audio stream (stereo, 128 kbps) at full audio quality. This compares well to our earlier work which used 15 MIPS and did not achieve *full precision*.

B. Memory Usage

A conservative over-estimation of the program and constant memory size is 80 kbits, and the data memory size is less than 130 kbits. On top of this we have some input and output buffering memories of approximately 32 kbits.

VII. Conclusions and Future Work

We have designed a floating point DSP capable of decoding layer II and III that conforms with the *full precision* classification specified in the MPEG-1 standard. It performs very well compared to our earlier layer III implemention, both regarding MIPS and memory usage.

It also compares well to other DSPs, for example one commercial solution with an ADSP-218x uses nearly twice as much memory and twice as many MIPS [6].

Memories are the major contributors to silicon area for this application, and we have shown that floating point can be used to reduce the size of the memories.

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] Eilert, J. and Ehliar, A., "Using Low Precision Floating Point Numbers to Reduce Memory Cost for MP3 Decoding", *Proc of the IEEE Int'l Workshop on Multimedia Signal Processing (MMSP), Siena, Italy, Sep. 2004*

[3] Source code of the "Freeware Advanced Audio (AAC) Decoder", available at, http://www.audiocoding.com

[4] Advanced Television Systems Committee, "Digital Audio Compression (AC-3), Revision A", *2001*

[5] Bluetooth Audio Video Working Group, "Advanced Audio Distribution Profile Specification", *2003*

[6] 16-bit Fixed Point MP3 Codec IP, http://www.digiwell.com.tw/image/Brochure-MP3Codec.pdf