REDUCED FLOATING POINT FOR MPEG1/2 LAYER III DECODING

Mikael Olausson, Andreas Ehliar, Johan Eilert and Dake Liu

Computer Engineering, Department of Electrical Engineering
Linköpings universitet SE-581 83 Linköping, Sweden
{mikol, ehliar, je, dake}@isy.liu.se

ABSTRACT

A new approach to decode MPEG1/2-Layer III, mp3, is presented. Instead of converting the algorithm to fixed point we propose a 16-bit floating point implementation. These 16 bits include 1 sign bit and 15 bits of both mantissa and exponent. The dynamic range is increased by using this 16-bit floating point as compared to both 24 and 32-bit fixed point. The 16-bit floating point is also suitable for fast prototyping. Usually new algorithms are developed in 64-bit floating point. Instead of using scaling and double precision as in fixed point implementation we can use this 16-bit floating point easily. In addition this format works well even for memory compiling. The intention of this approach is a fast, simple, low power, and low silicon area implementation for consumer products like cellular phones and PDAs. Both listening tests and tests versus the psychoacoustic model has been completed.

1. INTRODUCTION

Entertainment in small handheld devices like cellular phones and PDAs are getting more and more popular. One of these extra features is audio playback. MPEG-1/2 layer III, often known as MP3, is an audio coding standard that provides high audio quality at low bit rates [1]. Since a lot of these consumer product are portable, it is important to use low power implementations. The idea is to use small arithmetic units and still achieve high computional dynamic range. The standard for MPEG includes both encoder and decoder, but for the applications discussed here, the only interesting part is the decoder. Usually the decoder is implemented on a 24 or 32-bit fixed point processor. The bit size is chosen to give reasonable quality in the decoded music. When using a standard 16-bit processor, i.e., a DSP, double precision must be used in parts of the computations. Otherwise, a quality degradation can be heard. Here we will present a new approach to a fast, simple, low power, and low silicon area implementation using 16-bit floating point. The target is portable devices without extreme audio quality requirements; a cellular phone or a PDA. The headphones or the loud speakers are usually of quite poor quality. Therefore there is no need for high demands on the output music.

2. GENERAL DESCRIPTION OF THE MP3 FORMAT

The ISO/IEC 11172-3 and ISO/IEC 13818-3 are coding standards that provide high quality audio at low bit rates. There are three layers associated with the standard, layer I, II and III. They offer both increasing compression ratios, and increasing computing complexity. Layer III is more known as “mp3” based on the file extension it uses. The encoded bitrates ranges from 32 kbit/s up to 320 kbit/s. There are three main sampling frequencies associated with the standard 32, 44.1 and 48 kHz. There are also half frequencies which are just the main frequencies divided by 2. For a more complete description of the standard, see [1].

3. THE WORK

The work began with the reference code in C for the MPEG 2 layer III decoder. First the arithmetic instructions were exchanged against functions calls. This made it easier to perform profiling of the code and to elaborate with the precision and the dynamic range. The first approach was to use one format for all calculations within the decoder. While memory sizes usually are limited to byte lengths, we tried to use a floating point format while only using 16 bits. One bit is allocated for the sign bit and the remainder is split between the mantissa and the exponent. This approach turned out to be insufficient for the compliance testing [2]. To be called a fully compliant audio decoder, the rms level of the difference signal between the reference decoder and the decoder under test should be less than $2^{-15}/\sqrt{12}$ for a sine sweep signal 20Hz - 10 kHz with an amplitude of -20 dB relative to full scale. In addition to this, the difference shall have a maximum absolute value of no more than 2^{-14} relative to full scale. To be referred to as a limited accuracy decoder, the rms level of the difference signal between the reference decoder and the decoder under test should be less than $2^{-11}/\sqrt{12}$ for a sine sweep signal 20Hz - 10 kHz with an amplitude of -20 dB relative to full scale. There are no requirements on the maximum absolute difference. We were unable to hear any degradation in quality when we listened to the decoded files. The listening tests are described more in detail in section 5. We then decided to increase the internal precision, keeping the external precision to 16 bits. The distinction between internal and external precision lies in where the data is stored. While data is in the datapath the format of the data can be higher, but as soon as it is stored in memory it must be converted to 16 bits. Figure 1 shows the result of the compliance test for different sizes of the internal and external mantissa. To be fully compliant we needed a mantissa of 19 bits internally and 15 bits externally, alternatively 18 bits internally and 16 externally. The IEEE single precision floating point format consists of a 23 bits mantissa with an imlpicit one at the beginning. Our goal of this approach was not to aim for full compliance, instead the intention was a descent quality for low end equipment. Therefor, the limited accuracy test was acceptable. According to figure 1 the demands of the mantissa is only 13 bits internally and 9 bits externally, alternatively 12 bits internally and 10 bits externally for limited accuracy. This is a reduction of 6 bits from the fully compliant requirements and half of the mantissa size compared to the IEEE floating point format. The compliance test

has been performed on fixed point arithmetic in [3]. The requirements from their test is a 20 bits implementation.

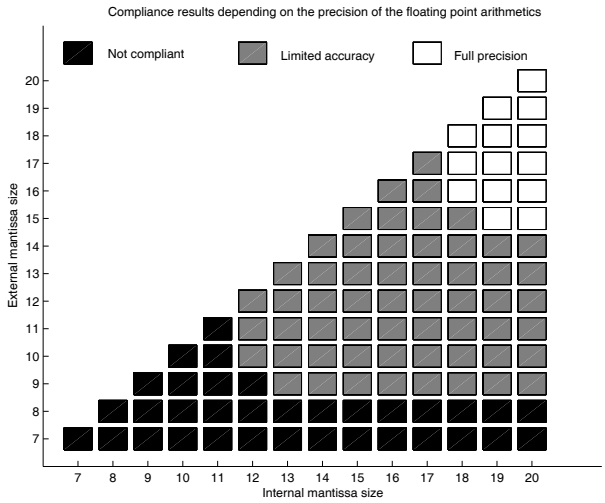

Fig. 1. Compliance test for different sizes of the internal and external mantissa.

While the mantissa is responsible for the precision, the exponent determines the dynamic range. One of the reasons for using this floating point approach was to avoid the problem of scaling variables that had to be completed in integer representation. Since we did not want variables to overflow, the upper limits of the exponent was set by the dynamic range of the variables. This was done by profiling the code and storing the maximum values of the variables. As a result, we could distinguish a difference in the upper limit for variables in the data path and the ones stored in memory. We needed a higher range for internal variable in the data path.

4. MOTIVATION

By using this reduced floating point representation we can achieve a high dynamic range and a reasonable precision, with fewer bits. We use 6 bits for the exponent in the internal representation. In order to get the same dynamic range using fixed point, it is necessary to use 64 bits. If this amount of bits is not available, double precision would be used instead. This results in a performance degrading. Another approach is to use appropriate scaling in order to reach the required dynamic range. Unfortunately, this process is tricky and time consuming. In addition, the code size would become bigger, the firmware would be more complex, and the debugging harder. However, floating point is a more straight forward implementation. In this floating point approach we can attack the problem of dynamic range and precision independently. If our goal is a high dynamic range we would allocate more bits for the exponent and if there is high demands on the precision we allocate more bits for the mantissa. In the fixed point approach we cannot separate these issues. By increasing the dynamic range the precision would also increase and vice versa. In the case of fixed point representation, an extra bit will increase the dynamic range by $2^{n+1}/2^n = 2$. In the floating point alternative, an extra bit in the exponent will give a $2^{2n}/2^n = 2^n$ increase in the dynamic range. From the calculations above it is clear that the floating point is superior when one wants to have an high dynamic range. For the precision aspect the floating point representation is also favorable. Due to the normalization of all the values, the same number of precision bits will always occur. In a fixed point representation we have to trade precision for dynamic range. By shifting the binary point to the right we can represent larger numbers on the expense of lower precision. The other extreme is when there are too many bits allocated for the precision resulting in an overflow. The IEEE standard[4] specifies a number of floating point formats. The first two are single precision and double precision formats that use a total of 32 and 64 bits respectively for their implementation. The reference decoder for the mp3 format uses the double precision format in their calculations. While our target for this paper is low-end applications, we will use the same concept as for the floating point standard but reduce both the mantissa and the exponent to fit our purposes. To find a balance between the fixed point and the floating point, the block floating point representation can be used. Again, the mantissa must be separated from the exponent, but the exponent stays the same for a whole block of values. As in the fixed point case, we will need to search through the whole block of data to find the right exponent and then scale the values appropriately. As a result both high dynamic range and high precision are possible. The drawback is when the values within a block differ significantly. The exponent is then chosen from the largest value and the small values will be shifted down and in turn lose most of their precision. The three representations are shown in figure 2.

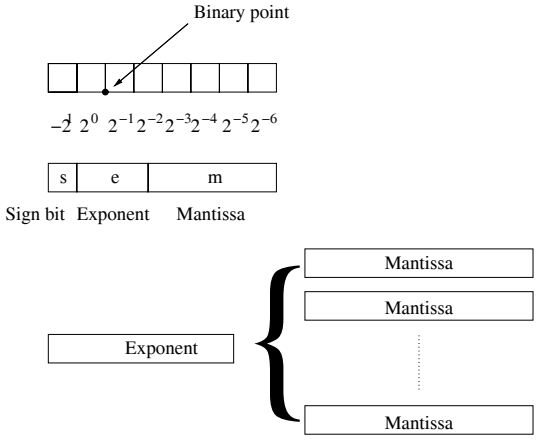

Fig. 2. Fixed point, floating point and block floating point representation. In fixed point, the binary point is imaginary.

5. TESTING THE QUALITY

In addition to the compliance test described in chapter3, we conducted objective measurement tests. The idea was to use the psychoacoustic model from an mp3 encoder to see if the error introduced by thereduced precision was inaudiable. for this purpose, the LAME version 3.92 was used. First, the masking levels from each subband were extracted. The decoder under test was then compared to the reference decoder using double floating point precision. From this, two different measurement could be made; one where we calculated the Signal-to-"hearable" noise ratio (SHNR) and one where we measured the masking flag rate. The SHNR is calculated in the same way as the more traditional Signal-to-Noise ratio (SNR), except that the noise is exchanged with the "hearable

noise". All the noise that is introduced by the decoder under test is compared to the reference decoder that exceeds the masking levels. The masking levels are calculated from the reference decoder output signal.

$$SHNR[dB] = 10 \times log_{10} \frac{\sum P_{dut}[i]}{\sum P_{hr}} \quad (1)$$

where $P_{dut}[i]$ is the output samples from the decoder under test squared and

$$P_{hr}[i] = \begin{cases} diff[i] - Mask[i] & diff[i] > Mask[i] \\ 0 & diff[i] <= Mask[i] \end{cases}$$

Mask[i] is the masking levels from the psychoacoustic model. diff[i] is the difference between the output signal from the reference decoder and the decoder under test squared; diff[i] = $(pcm_{ref} - pcm_{dut})^2$ To make it more convenient are all the calculations made on a midchannel. $\frac{(l+r)}{\sqrt{2}}$, where l is the left channel and r is the right channel. The second measurement is from [5]. The flag rate gives the percentage of frames with audible distortion, i.e., frames where the introduced noise is above the masking level. In table 1 we used the sine wave file from the compliance test and changed the number of bits for the mantissa and the exponent. The first test was the single precision floating point. The result was very good performance, but we used 32 bits for the representation. In order to be called a full precision decoder we had to use a 19 bit mantissa internally and a 15 bit mantissa externally. See figure 1. The degradation from the single floating point is only 7 dB. For limited accuracy the degradation is much bigger, especially in the case hearable noise. Here, a 13 bits internal mantissa and 9 bits externally were used. Further, a test was conducted where the internal and external representation were the same. Namely, 10 bits mantissa, 5 bits exponent and 1 sign bit. This is interesting since it fits into a 16 bit memory and a 16 bit datapath. As a conclusion of this test the difference between the SNR and SHNR decreases as you decrease the precision, i.e., the noise introduced becomes more and more "hearable".

Precision	**SNR**	**SHNR**	**Flag Rate**
Single float	89.1	115.8	$2.5 * 10^{-7}$
Full precision	82.9	108.6	$6.0 * 10^{-4}$
Limited Accuracy	53.9	60.9	0.16
16-bits	46.2	47.3	0.13

Table 1. The result of the objective measurement.

The second test was the sound files from the SQAM disc [6], Sound Quality Assessment Material, a mixture of both male and female speech in English, French and German. It also includes electronic tune in file frer07_1. The remaining sounds are purely instrumental. These sounds are known to be revealing for MPEG coding. As we can see in table 2 the SNR remains rather constant. It is approximately 54 dB and similar to the compliance test in table 1. It seems as if the noise introduced is constant but the amount of "hearable" noise differ. This is one reason why the SNR measure does not suit well for the purpose of audio quality measurement. All the noise introduced will not effect the listening quality.

Test file	**SNR**	**SHNR**	**Flag Rate**
bass47_1	54.9 dB	69.0 dB	0.011
frer07_1	54.6	64.3	0.084
gspi35_1	54.4	69.4	0.012
gspi35_2	54.2	72.9	0.0086
harp40_1	54.1	107.0	$3.7 * 10^{-5}$
horn23_2	54.1	64.5	0.18
quar48_1	53.6	72.2	0.047
sopr44_1	54.8	65.0	0.020
spfe49_1	55.0	70.5	0.0050
spff51_1	55.2	70.4	0.0048
spfg53_1	54.3	74.1	0.0024
spme50_1	55.3	71.2	0.0067
spmf52_1	55.2	74.6	0.0034
spmg54_1	54.8	75.4	0.0024
trpt21_2	53.9	73.3	0.0076
vioo10_2	54.8	73.5	0.0013

Table 2. The result of the objective measurement.

In figure 3 and figure 4 the extremes from table 2 harp40_1 and horn23_2 can be seen. Harp40_1 has the highest SHNR and the lowest flag ratio, which indicates the best sound quality. The opposite is the file horn23_2, which has one of the lowest SHNR and the highest flag ratio. In figure 3 we can see the difference between the reference decoded file and the file decoded with our proposal; 1 sign bit, 9 bits mantissa and 5 exponent bits externally and 13 bits mantissa and 6 bits exponent internally. The difference between the two test files is not substantial. The SNR is very similar, around 54 dB. Finally, the error signal after the masking level is taken into account in figure 4. Here we can see that there is a significant difference. The noise introduced from our reduced floating point format is not masked away as well in the test file horn23_2 as in the test file harp40_1. This is not clear from the SNR, but with the measurements SHNR and flag ratio it becomes much clearer.

6. SAVINGS IN POWER AND AREA

The high dynamic range from this reduced floating point has more advantages as compared to an ordinary fixed point implementation. First, a mantissa of just 13 bits reduces the multiplier. In the comparing fixed point implementation a multiplier of 20-24 bits is needed. Here it is alright with just 13 bits. By using a reduced floating point format, one can also reduce the size of the accumulator register. There is no need for double precision and guard bits. In addition, the smaller size of the variables gives smaller adder units. In fixed point arithmetic you have to keep track of the scaling of the variables, otherwise you will run into precision or overflow problems. Since the scaling takes place within the arithmetic unit, there is no need for an barrel shifter, just a small one for the six bit exponent. The absence of need for dynamic scaling results in a decrease of the amount of code and the programming task becomes easier. The reduced format of the external variables, i.e. the variables that are stored in memory, reduces the size of the data memory. If you customize your own memory, it is enough with 15 bits for external storage. The internal variables within registers are still larger; 20 bits.

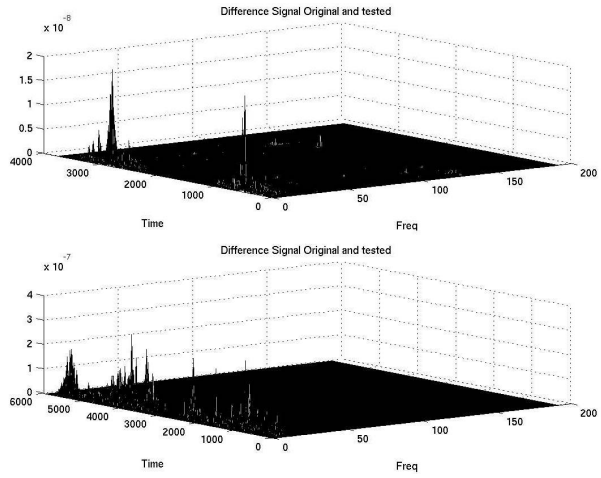

Fig. 3. The difference between the original and the decoder under test. Above for the test file harp40_1 and below for horn23_2. The frequency axle is the output from the fft and the sampling frequency is 44.1 kHz.

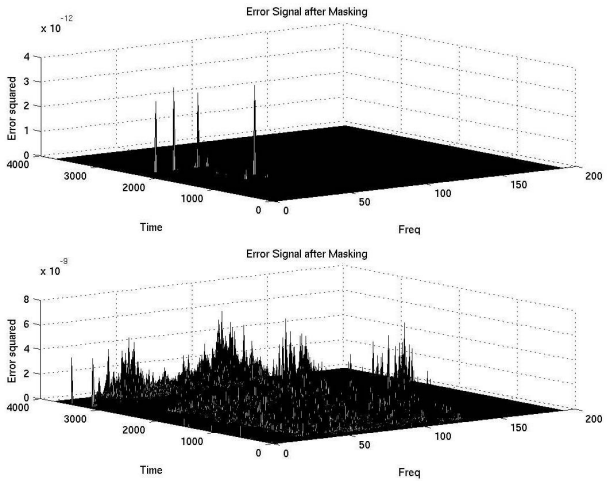

Fig. 4. The remaining error after the the noise below masking level is taken away. Above for the test file harp40_1 and below for horn23_1.The frequency axle is the output from the fft and the sampling frequency is 44.1 kHz.

Operation	**20-bit reduced floating point**	**20-bit fixed point**
Multiplier	13x13	20x20
Accumulator	20 bits	48 bits
Register	20 bits	20 bits
Memory	15 bits	20 bits

Table 3. Comparison between 20-bit reduced floating point and 20-bit fixed point

On the negative side is the more complex arithmetic unit, where variables need to be shifted before addition and subtraction. There is also a need for post scaling to make sure that the bits in the mantissa are aligned correctly. This hardware is expensive and power consuming. Finally, it might give a deeper pipeline.

7. FUTURE WORK

To prove this concept more accurately it has to be implemented in silicon and conduct additional measurements. The real numbers on silicon sizes, memory issues and implementation difficulties could then be obtained. It might be possible to exchange algorithm from the reference decoder to a more simple without any quality degradation. This new algorithm might suit better for this implementation.

An subjective listening test with proffesional listeners is also preferable. We have made less complicated listening tests ourselves, but we do not know what kind of artifacts to listen for. The aim for this implementation is not high performing audio devices, rather low end products. Consequentially, we might be able to shrink the mantissa and exponent sizes even further.

Another interesting aspect would be to have a reconfigurable architecture. The number of mantissa bits and exponent bits would then be programmable on the fly. In that case you can trade power consumption for audio quality.

8. CONCLUSION

We have proposed a floating point approach to implement a mp3 decoder. Instead of the usual fixed point we have used a floating point implementation with different number of bits for the internal and the external representation. By this approach we can reduce the size of the arithmetic units and still keep good quality sound. The firmware also becomes simpler. There will be no need of scaling of variables, this is done automatically within the arithmetic unit. We have also performed simpler listening tests and done some objective sound quality measurements.

9. ACKNOWLEDGMENT

This work was financially supported by the stringent of SSF and the Center for Industrial Information Technology at Linköping Institute of Technology (CENIIT).

10. REFERENCES

[1] Iso/iec-13818-3, information technology - generic coding of moving pictures and associated audio - part 3: Audio, 1998.

[2] Iso/iec-11172-4, information technology - generic coding of moving pictures and associated audio - part 4: Compliance testing, 1998.

[3] In-Cheol Park Yongseok Yi. A Fixed-Point MPEG Audio Processor Operating at Low Frequency. *IEEE Transactions on Circuits and Systems–II: Analog and Digital Signal Processing*, 47:779–786, November 2001.

[4] Ieee standard for binary floating-point arithmetic 1985, 1985.

[5] Itu-r 1387-1, method for objective measurements of perceived audio quality, 1998.

[6] SQAM Sound Quality Assessment Material. In *http://www.tnt.uni-hannover.de/project/mpeg/audio/sqam/*.