THE JOURNAL OF THE ACOUSTICAL SOCIETY OF KOREA VOL.25, NO.4E 2006. 12 pp. 137~141

# Optimization of HE-AAC for Korean S-DMB Using TMS320C55x DSP Core

Hyung-Jung Kim*, Deock-Gu Jee*
*Electronics and Telecommunications Research Institute
(Received October 10; Revised December 4 2006; accepted December 18 2006)

**Abstract**

This paper presents HE-AAC decoder optimization on TMS320C55x fixed-point DSP core using a DSP-C like FFR code, which provides fast and flexible porting to a DSP core. Our optimization efforts are focused on methodologies that include general optimization methods of FFR code suitable for general DSP or RISC platform in high-level language and software optimization methods in assembly language level. The implementation result requires 48 MIPS and 135 Kbytes memory space to decode 48 Kbps stereo using real Korean S-DMB data.

***Keywords***: *DSP optimization, Real-time implementatiom Korean S-DMB, MPEG-AAC*

## I. Introduction

In this paper, we optimize and implement MPEG-4 High-Efficiency Advanced Audio Coding (HE-AAC) decoder on a fixed-point DSP. HE-AAC algorithm is a MPEG standard extending MPEG-4 AAC algorithm by adding spectral band replication (SBR) algorithm [1, 2]. Using low-pass signal and a small amount of control data for SBR, HE-AAC decoder generates full-band signal and improves coding efficiency by more than 30%.

A fixed-point reference code is essential in development of embedded portable systems using fixed-point processor. Because the usual reference code such as MPEG standard audio is based on floating-point arithmetic, the fixed-point simulation must be carried out accounting to performance and complexity of application algorithms. In generally the fixed point simulation requires much effort and time. Therefore, we refer to a DSP-C like Fixed-point Firmware Reference (FFR) code. FFR code provides fast and flexible porting to a DSP or MCU platforms and a bit exact model of the coding scheme according to the arithmetic behavior of the target processor [3].

Corresponding author: Hyung-Jung Kim (acekim@etri.re.kr)
ETRI, 161 Gajeong-Dong Yuseong-Gu, Daejeon 305-350, Korea.

The assembly level optimization in the fixed-point DSP is still very important in development of embedded systems. It is because of low efficiency of C-compiler, occurrence of more and more complex application algorithms and the necessity of the integrated implementation of many application algorithms on a single platform.

In this paper, we describe general FFR code optimization methodologies, which include general optimization methods for target processor. And we focused on TMS320C55x DSP core specific optimizations. The organization of this paper is as follows. The FFR code optimization techniques are presented in section II. And then DSP implementation issues and results are described in sections III and IV, respectively. Finally, we remark conclusions in section V.

## II. FFR code optimization techniques

FFR code is close to DSP-C, the upcoming ANSI-Extensions to ANSI-C. FFR code shows best performance among fixed-point C-codes for HE-AAC algorithm. Instead of superior audio quality, FFR code requires more computational complexity.

The major characteristic of FFR code is the introduction of a fractional data type representing the typical fixed-point data format and the corresponding operator. The fractional data types sfract, fract and dfract. For a typical 16-bit DSP sfract is 16-bit wide, fract is 32-bit wide and afract is 40-bit wide. With the use of the technique of operand overloading and the fractional data type, FFR code for HE-AAC decoder could provide fast and flexible portability to a various DSP or MCU platforms [4]. The main features of FFR code are as follows:

√ The fixed-point reference code stays as close as possible to the floating-point reference code.

√ No need for specific functions for different operands and different return values in the code.

√ Coefficient tables stay in floating point notation. At runtime, the values are casted to the corresponding fixed point representation.

FFR code itself has many advantages, but optimization in FFR code level is necessary to fully utilize the performance of target processor and to generate suitable fixed-point code for target processor. One major task of optimization in FFR code level is the replacement of overloaded operators by intrinsic.

The porting of the FFR code to the target platform means replacing all fractional operators defined by the sfract, fract and dfract classes by the corresponding intrinsic. Figure 1 show our proposed porting methodology that is consisting of 5 design phases and test phase.

First phase is the preparation stage in which the main task is to analyze all kinds of constraint in the embedded systems, for example, memory sizes, power consumption and the ability of the target processor. The landmark of porting task and the confined conditions shall be cleared in this phase. The second phase is to analyze the features of specific processor, especially on the micro-architecture and the pipeline architecture of target processor that shall guide the embedded software optimization work. The third phase is to decide data type width such as sfract, fract and dfract.

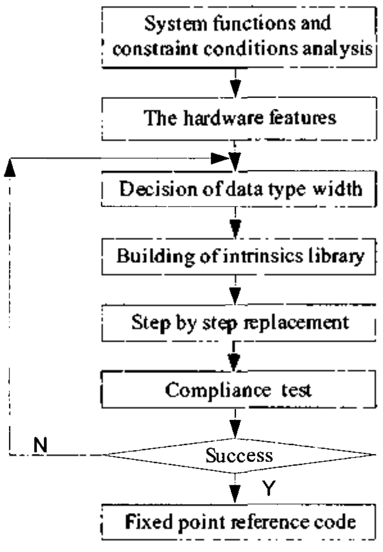
System functions and constraint conditions analysis
The hardware features
Decision of data type width
Building of intrinsics library
Step by step replacement
Compliance test
N
Success
Y
Fixed point reference code

Figure 1. Our FFR Poritng Methodology.

The forth phase is to build intrinsic functions corresponding fractional operators i.e. FFR arithmetic library. Since a standard C/C++ compiler does not know about the TMS320C55x intrinsic, a C implementation of these functions is required. We have built our intrinsic library for TMS320C55x core. To speed up software development, it is desirable that the same code can be compiled either for the DSP target or directly for the host.

The fifth phase is to replace all fractional operators by the corresponding intrinsic functions. Then, we must verify that the output fixed point reference code for HE-AAC decoder ensure full compliance with ISO 14496-4 audio standard. Finally we can get fixed-point reference code of HE-AAC decoder for TMS320C55x DSP core.

Considering the architecture of TMS320C55x DSPcore and output audio quality of HE-AAC decoder, 16-bit, 32-bit and 40-bit data width were selected for sfract, fract and dfract data type respectively and dfract was allocated in only 40-bit accumulators. In next step, in order to reduce computational load, implemented intrinsic functions are bit-exact with TMS320C55x intrinsic but not with FFR arithmetic library.

## III. DSP implementation

The overall HE-AAC decoding algorithm can be partitioned into two categories, which are control-intensive part and computation-intensive part. Control-intensive part is not as time critical as computation-intensive part. Thus, we can perform distinct optimization for each part of the algorithm.

TMSC55x DSP core is suitable for mobile terminal platform and supports a variety of parallel instructions by the use of the process's multiple−bus architecture, dual MAC units, separated program unit, address unit, and data computation unit. The key feature of it is dual MAC units that are capable of parallel MAC computations per one cycle. Thus, the C55x DSP can provide more powerful MAC operations than other DSP and have large internal memory space. But, C−compiler can not efficiently utilize these characteristics. Therefore, hand assembly coding is required.

## 3.1. C-compiler based Implementation

The assembly code produced from C−compiler has much redundancy due to the inefficiency of it. So, in order to get more efficient assembly code, further optimization is required in assembly level. But hand assembly coding is time−consuming work, and therefore development cost is increased. As a result, we must tradeoff between performance and cost.

Before the main optimization, in order to decide whether hand assembly coding is needed for overall part of HE−AAC algorithm or especially complex parts, analysis of computational complexity is needed. Thus, we substituted arithmetic functions with C compiler intrinsics which can be executed with no function call overhead in one cycle, and compiled C code. And then, we analyzed the computational complexities of cross−compiled decoder and C−compiler efficiencies of each functional block.

Table 1 gives the result of our C−compiler based implementation with C55x−intrinscs before optimization. As shown in Table I, the control−intensive part is not as complex as computation−intensive part. But, this part has relatively high computational loads. Therefore, hand assembly coding must be performed for all part of HE−AAC algorithm.

Table 1. Performance of our C-compiler based implementation with C55x-intrinsics before optimization.

| Functional Block | | Complexity(MHz) | % |
|---|---|---|---|
| AAC | Huffman decoding | 7 | 2.04 |
| | Stereo processing | 6 | 1.75 |
| | IMDCT | 77 | 22.45 |
| | Other | 19 | 5.54 |
| SBR | Analysis filter bank | 34 | 9.91 |
| | HF generation | 26 | 7.58 |
| | Envelope Adjuster | 51 | 14.87 |
| | Synthesis filter bank | 109 | 31.78 |
| | Other | 14 | 4.08 |
| Total | | 343 | 100 |

## 3.2. Assembly-language level optimization

In computation−intensive part such as IMDCT, analysis filter bank, HF generation, Envelope Adjuster, and synthesis filter bank, most of operations are memory access, addition, and multiplication. In case of multiplication, 16−bit x 16−bit, 32−bit x 16−bit, and 32−bit x 32−bit multiplication is needed to obtain good audio output quality in HE−AAC decoder [5]. Considering that MAC is the major burden of computation−intensive part, we have concentrated on C55x specific optimization to reduce the computational loads of MAC operation.

C55x DSP core support multiple memory access operation, which makes it possible to three 16−bit data road from memory in one cycle. And dual MAC units can

```
/*                    Original C-Code                         */
for (i = 1; i 〈 M; i++) {
    tmp[2*i] = add(mpy_32_16(Dat[L-i],sin_tb[L-i]),
                   mpy_32_16(Dat[i],sin_tb[i])))〉〉 1;
    tmp[2*i+1] = sub(mpy_32_16(Dat[L-i],sin_tb[L-i]),
                   mpy_32_16(Dat[i],sin_tb[i])))〉〉1;
}
```

```
;;                    Hand Assembly Code                      ;;
rptb      Loop
mpy        uns(*ar0-), *cdp, ac0
::mpy uns(*ar1-), *cdp, ac1

mac       *ar0+, *(cdp+T0), ac0〉〉#16
::mac *ar1+, *(cdp+T0), ac1〉〉#16
   ||neg  T0

mpy        uns(*ar0-), *cdp, ac2
::mpy uns(*ar1-), *cdp, ac3
   ||add            #1, T0

mac       *ar0+, *(cdp+T0), ac2〉〉#16
::mac *ar1+, *(cdp+T0), ac3〉〉#16
||neg      T0

add       ac1, ac2
sub       #1, T0
||sfts    ac2, #-1

mov       ac2, dbl(*ar2+)
||sub     ac0, ac3

sfts      ac3, #-1
mov       ac3, dbl(*ar2+)
||add     #2, ar0

Loop: sub #2, ar1
```

Figure 2. An example of optimization in assembly level.

perform two multiplications in one cycle. But, in general, two multiplications need four operands. And so, one operand must be commonly used in two multiplication operations for parallel MAC operations. Therefore, in order to use parallel MAC operations, we have used sophisticated data memory addressing and instruction scheduling.

Figure 2 has given an elaborate example about how to using parallel MAC operations to optimize assembly code. The codes in figure 2 are typical codes after optimization manually and they need only 10 cycles.

## IV. Implementation Results

DSP platform for the HE-AAC implementation are Spectrum Digital's C5509 EVM and TI OMAP1610 EVM. All HE-AAC algorithms arerunning on TMS320C55x DSP core and MPEG-2 demux module is running on ARM9 core of OMAP1610. The MPEG-2 demux module extract audio packet data form real Korean S-DMB data and transfer it to HE-AAC algorithm. In order to objectively measure the sound quality of developed HE-AAC decoder, ISO 14496-4 compliance test was performed [6]. HE-AAC decoder don't exceed the criteria of compliance test, and therefore satisfy ISO 14496-4 compliance.

Table 2 shows the performance of the implemented decoder. The maximum numbers of operations for the 48 kbps stereo bitstream is 47.7 million cycles per second (MCPS). It can be seen from Table 2 that this decoder uses about 25% of 200-MHz DSP. AAC part uses less complexity and SBR part use more complexity. The size of program memory and data memory are 25.5 Kwords and 41 Kwords, respectively.

Table 2. Implementation result of HE-AAC.

| Functional Block | | Complexity(MHz) | % |
|---|---|---|---|
| AAC | Huffman decoding | 2.78 | 5.83 |
| | Stereo processing | 1.72 | 3.60 |
| | IMDCT | 8.89 | 18.63 |
| | Other | 3.61 | 7.56 |
| | Sub Total | 17.00 | 35.62 |
| SBR | Analysis filter bank | 5.65 | 11.84 |
| | HF generation | 3.76 | 7.88 |
| | Envelope Adjuster | 6.31 | 13.23 |
| | Synthesis filter bank | 14.7 | 30.81 |
| | Other | 0.3 | 0.63 |
| | Sub Total | 30.72 | 64.38 |
| Total | | 47.72 | 100 |

Table 3. The best optimization performance for typcial devices for stereo, 44.1Khz.

| Type | Vendor | Type | Data [bit] | Performance [Mclcyles] |
|---|---|---|---|---|
| C64 | TI | DSP | 16 | 24 |
| MMDSP+ | ST | DSP | 24 | 24 |
| ARM9E | ARM | MCU | 32 | 44 |
| ARM9 | ARM | MCU | 32 | 61 |

Table 3 presents real world optimal figures for typical devices.Typical devices means including caches of typical size, typical hardware accelerator, typical optimal function etc. And FFR code is based on the fractional data type, so it is more suitable 24-bit and 32-bit processor than 16-bit processor. TI C55x core is suitable to mobile terminal but has disadvantages than mentioned typical devices because of its structural limitation like only 16-bit processor and absence of hardware accelerator.

## V. Conclusions

In this paper, we have proposed optimization techniques for real-time implementation of HE-AAC algorithm on TMS320C55x DSP core using a DSP-C like fixed point firmware reference code. By the use of feature of DSP core, FFR code could be ported to target device easily and efficiently. For efficient real-time implementation, we optimized HE-AAC decoder in C-language level using our intrinsic library, and then performed optimization in assembly language.

We use the real Korean S-DMB data for conformation test. The implemented HE-AAC decoder requires the computational complexity of maximally 47.72 million cycles per second to decode 48 kbps stereo bit-stream. And compliance test results confirmed that the developed decoder ensured full compliance with ISO 14496-4 audio standard and good audio quality.

## References

1. ISO/IEC Interanl Standard 13818-7 *Generic Coding of Moving Pictures and Associated Audio Information -Part 7: Advanced Audio Coding,* 1997.

2. ISO/IEC 14496-3:2001/Amd 1:2003, *Bandwidth extension*
3. Thomas Ziegler, Martin Dietz , and Klaus Peichl, "Writing Firmware Reference Code for Fixed Point Processors", International Signal Processing(ISPC), Dallas, March 2003.
4. Thomas Ziegler, Klaus Peichl, Gustavo Hoffmann, and Martin Wolters, *Using Fixed Point Frimware Reference Code - A Case Study*
5. Keun-Sup Lee, Young Cheol Park and Dae Hee Youn, "Software Optimization of the MPEG-Audio decoder using a 32-bit MCU RISC Processor", 2002.
6. ISO/IEC 13818-4 *Generic coding of moving pictures and associated audio information(Part 4: Compliance testing)*

## [Profile]

• **Hyung-Jung Kim**
received the B.S. and the M.S, degrees from Hanyang University, Seoul, Korea, in 1993 and 1995. Since 1995, he has been working for mobile communication lab., ETRI, Daejeon, Korea. His research interests include multimedia real-time application and Software defined radio.

• **Deock-Gu Jee**
received the B.S. and the M.S, degrees from Chungbuk National University, CheongJu, Korea, in 1998 and 2000. Since 2000, he has been working for mobile communication lab., ETRI, Daejeon, Korea. His research interests include multimedia real-time application and Software defined radio.