Development of an Embedded System LSI for MPEG-2 AAC Decoder: a Case Study

Khaled, Jamaleddine
Department of Computer Science and Communication Engineering, Kyushu University

Eko, Fajar Nurprasetyo
Department of Computer Science and Communication Engineering, Kyushu University

Yamashita, Hajime
Department of Computer Science and Communication Engineering, Kyushu University

Yasuura, Hiroto
Department of Computer Science and Communication Engineering, Kyushu University

Development of an Embedded System LSI for MPEG-2 AAC Decoder: a Case Study

Khaled Jamaleddine Eko Fajar Nurprasetyo
Hajime Yamashita Hiroto Yasuura
Department of Computer Science and Communication Engineering,
Kyushu University
6–1 Kasuga-koen, Kasuga, Fukuoka 816-8580 Japan
{*khaled, eko, hajime, yasuura*}*@c.csce.kyushu-u.ac.jp*

Abstract

The progress of both digital signal processing technology and LSI process technology are key factors in spreading digital media products in the market. System LSIs which support the function of video and audio data compression are utilized in these systems. This paper presents a case study on the development of an embedded system LSI for MPEG-2 AAC (MPEG-2 Advanced Audio Coding) decoder, based on a soft-core processor and a programming language, called Valen-C. Design results show that we can achieve cost reduction without losing the performance by optimizing the datapath width of the soft-core processor.

1 Introduction

In recent years, digital media products which utilize video and audio compression technology became widely used in the daily life. DVD (Digital Versatile Disc) video players, digital VCRs (Video Cassette Recorder), IRDs (Integrated Receiver Decoders) for digital broadcasting are examples of these products. The progress of both digital signal processing technology and LSI process technology are key factors in spreading these kind of products in the market. The MPEG-2 AAC (Advanced Audio Coding) coding/compression standard, being capable to compress audio data at a high rate is used in these products.

In this paper , we describe a design experience of an embedded system LSI for MPEG-2 AAC decoder. Our design method is based on a soft-core processor and Valen-C language[1].

In the initial design phase, the designer designs a system with a soft-core processor, RAM, ROM and logic circuits. The application program is written in Valen-C, in which the designer specifies the word length of each variable required for accurate computation. After verifying the functionality of the initial design, the system designer can modify several parameters of the soft-core processor, including the number of registers, the datapath width and the instruction set. Thus, the system designer can tune up the soft-core processor until the system performance satisfies the design constraints.

This paper is organized as follows: Section 2 presents the software we used for our target system. Section 3 presents our proposed design flow. In Section 4, we explain the variable size analysis of the source program. Section 5 summarizes the compilation of the program and the design of the soft-core processor. In section 6, we evaluate the performance of the developped system. Section 7 concludes our paper.

2 MPEG-2 AAC Decoder

MPEG-2 AAC is a very efficient technology of audio data compression. It was declared international standard by MPEG (Moving Picture Experts Group) at the end of April 1997 as ISO 13818-7. MPEG-2 AAC basically makes use of the signal masking properties of the human ear in order to reduce the amount of data. Doing so, the quantization noise is distributed to frequency bands in such a way that it remains inaudible.

The source program of the MPEG-2 AAC Decoder we used in our development is based on ARIB standard for BS digital broadcasting. Tables 1 and 2 show

respectively the specifications and the size of the program.

Table 1. Program specifications

coding method	MPEG-2 AAC (ISO/IEC 13818-7)
sampling frequency	32KHz, 44.1KHz, 48KHz
Maximum number of channel signals	5ch+LFE
Bit stream format	AAC Audio Data Transport Stream
Profile	Low Complexity

Table 2. Program list and size

name of the program	number of lines
config.c	381
copro.c	155
data tab.c	516
decdata.c	71
decoder.c	281
huffinit.c	71
huffdec1.c	102
huffdec2.c	675
huffdec3.c	100
hufftables.c	2733
intensity.c	89
portio.c	360
stereo.c	76
tns.c	228
trans.c	1055
window tab.c	1171
all.h	242
interface.h	182
port.h	55
tns.h	32
total	8575

3 Design Method

In order to design the MPEG-2 AAC decoder, we proposed a new design flow for embedded systems. Figure 1 shows the design flow of our system design method. And figure 2 shows the tools used for each design phase and its corresponding working flow. In the initial phase of the system design, the target application program of the embedded system is written in Valen-C language, and compiled with the retargetable Valen-C compiler [2]. In the other hand, we adopted Bung-DLX as the soft-core processor [3].

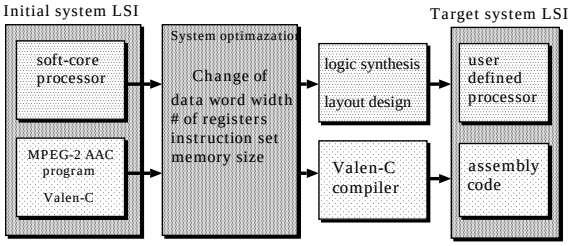

Figure 1. The design flow of MPEG-2 AAC decoder

3.1 Valen-C and Retargetable Compiler

Valen-C is an extension of the C language, and it is used to describe the source program of the embedded system. The control structures in Valen-C, such as *if* and *while* statements, are the same as C. However, the main feature of Valen-C is the specification of the required bit length of each variable in the program. In Valen-C, a programmer can appoint to each variable the minimum bit length required for accurate computation. Thus it becomes possible to reduce the cost of the datapath and the size of the data memory. This reduction also affects the power consumption in the datapath and data memory. For instance, if variables x, y, and z require 12, 20 and 24 bits respectively, the programmer can write $int12x; int20y; int24z;$ in the variable declaration.

3.2 Soft-Core processor

In processor design, the designer must determine the number, the size, and sorts of hardware components, called design parameters. The datapath width, the number of registers, the kind of functional units, the size of memories and the precision of ALUs are examples of design parameters.

A soft-core processor is a prototype of embedded processor, which has some design parameters. The designer can change the parameters for each application, and then obtains a customized processor optimized for the application. By using soft-core processors, development of embedded systems becomes easier and requires less time and effort.

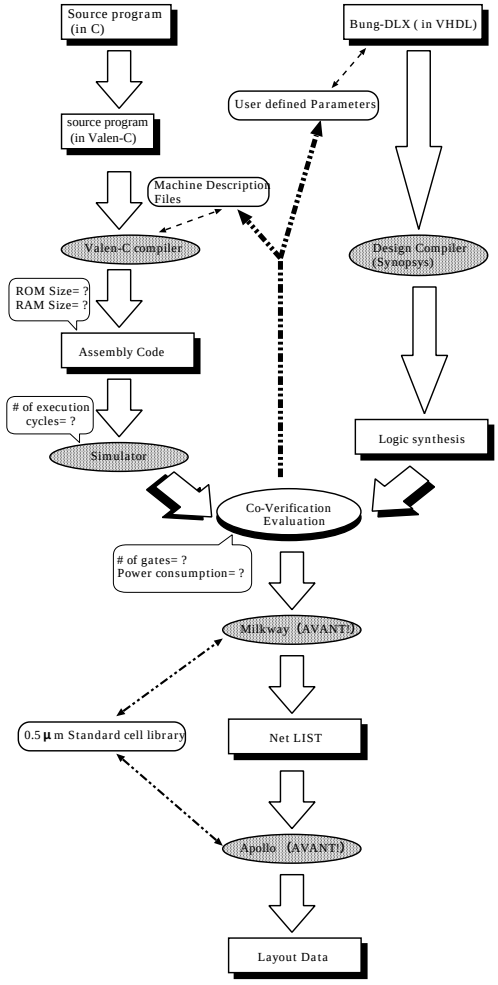

Figure 2. The tools used in the design flow

In the design of our embedded system, we adopted Bung-DLX as the soft-core processor. The main features of Bung-DLX are as follows:

- Targeted for controller and low-to-middle class embedded application.
- Serial nonpipeline architecture.
- Modifiability of an instruction set, datapath/ address width and the number of registers.
- Load/store architecture.
- Implemented purely in VHDL of about 7000 lines RTL code.

The designer can change the design parameters by only modifying the parameters in the following VHDL files: *bungtypes.vhd*, *bunginstructions.vhd* and *controltypes.vhd*.

3.3 Design Flow

Our proposed design flow of MPEG-2 AAC decoder consists of the following phases:

- Phase 1: The source program, which was originally written in C, is rewritten in Valen-C language. In this phase, the bit width of each variable, after been analysed, is used. for instance, if the variable a requires at most 14 bits, the programmer can write $int14a$; in the variable declaration.
- Phase 2: The soft-core processor is customized to different soft-core processors by choosing different design parameters such as the datapath width and the address size of the data memory. All those modifications are made in HDL level.
- Phase 3: The source program of the MPEG-2 AAC decoder is compiled for the customized soft-core processors. Here we use the Valen-C compiler which generates the assembly code from the source program and the design parameters of the customized processor. As a result we get different embedded systems from different customized processors and assembly codes. At this point the designer can estimate the size of both the data memory and the instructions memory of each system.
- Phase 4: We perform an evaluation of the systems generated in phase 3. That is we check the number of execution cycles, the size of the memory, the number of gates and power consumption. And

we evaluate the impact of the design parameters on the system performance. Thus we can choose the best embedded system that satisfies the design constraints among the systems we have made in the previous phase.

4 Variable Size Analysis

We define the effective size of a variable as the smallest size which can hold both maximum and minimum values of that variable. In order to make good use of Valen-C, the effective size of each variable in the source program needs to be analyzed. The number and types of the variables in MPEG-2 AAC decoder are as follows:

- Int: 352
- Unsigned: 21
- Pointers: 186
- Long: 241
- Long long: 29
- Short: 73
- Char: 16

In this paper we used two approaches to analyze the effective size of variables.

4.1 Static Analysis

When the maximum value of an unsigned integer variable x is n_{max}, the effective size of x, $e(x)$, is given as follows[4]:

$$e(x) = \lceil \log_2(n_{max} + 1) \rceil \quad (1)$$

For a signed integer x with a maximum value n_{max} and a minimum value n_{min}, $e(x)$ is defined as follows:

$$e(x) = \lceil \log_2 \mathcal{N} \rceil + 1 \quad (2)$$

where

$$\mathcal{N} = \max(|n_{max}| + 1, |n_{min}|)$$

Tables 3 and 4 show the static analisys results of effective size. In table 4, by analyzing the effective size of arrays, which relatively require more memory, an important decrease of the data memory in the system can be expected.

Table 3. Analisys results (1)

effective size	number of variables
1 bit	55
2 bits	18
3 bits	23
4 bits	18
5 bits	2
6 bits	3
7 bits	5
8 bits	4
9 bits	2
10 bits	3
11 bits	1
12 bits	1

Table 4. Analisys results of the arrays

variable	type	effective size
window long sin[1024]	long	23 bits
window long ksb[1024]	long	23 bits
window short sin[128]	long	23 bits
window short ksb[128]	long	23 bits
brx table[512]	long	32 bits
tns table[2][16]	long	32 bits
new bitrev2048[480]	short	10 bits
new bitrev256[56]	short	7 bits
l zcos256[64]	long	32 bits
l zsin256[64]	long	32 bits
l zcos2048[512]	long	32 bits
l zsin2048[512]	long	32 bits
sfb 48 1024[49]	short	11 bits
sfb 48 128[14]	short	8 bits
sfb 32 1024[51]	short	11 bits
scale g[4]	long	23 bits
q table[8192]	long	23 bits

4.2 Dynamic Analysis

Static analysis is an efficient method to analyse the effective size of variables. However, in many cases when we can not know the assigned value of a variable unless we run the program, such as the case of loops, static analysis becomes insufficient. As a solution to this problem, we adopted also dynamic analysis in our case study.

In dynamic analysis, we run the program and monitor the values assigned to each variable. For this reason, we use the function which monitors the variable in the assignment statement. The factors of the monitoring function are the variable and its assigned value. Figure 3 shows a part of the program used for dynamic analysis.

```
short subgroup_inc;
subgroup_inc = 0;
for (j = 0; j < ncells[i]; j++)
{
 subgroup_inc += cellsize[j] ;
}
```

⇨

```
short subgroup_inc;
subgroup_inc = 0;
for (j = 0; j < ncells[i]; j++)
{
 subgroup_inc += cellsize[j] ;
checkbits_short("subgroup_inc", subgroup_inc);
}
```

Figure 3. An example of a program used for dynamic analysis

The monitoring function checks the value assigned to the variable and verifies the bit width required to memorize it. After that, it keeps the bit width temporarily in a table. Next, when the monitoring function checks the same variable with a different assigned value, it compares the new bit width with the bit width already memorized in the table, and keeps the bigger bit width in the table, and so on. Thus, the required bit width of the variable is analized while running the program.

Table 5. Types of test stream files

test file	encoded channel	samp. freq. [KHz]
L4 48k	1	48
L4 44k	1	44.1
L4 32k	1	32
L5 48k	2	48
L5 44k	2	44.1
L5 32k	2	32
L6 48k	3	48
L6 44k	3	44.1
L6 32k	3	32

Table 6 shows the analysis results, and figure 4

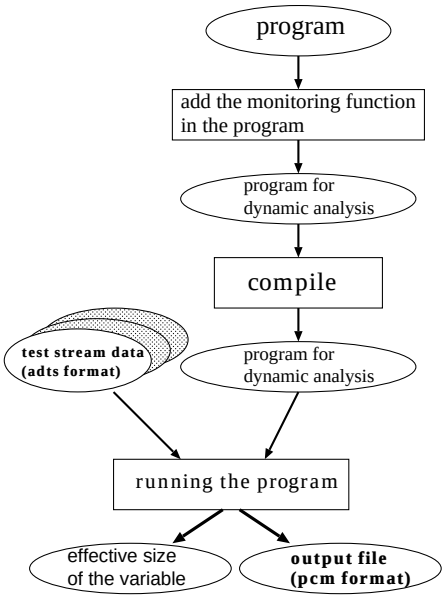

Figure 4. The dynamic analysis flow

shows the dynamic analysis flow. We used nine different kinds of test stream data files (adts format) to analyze the variables dynamically (see table 5).

Table 6. Dynamic analisys results

variable	type	effective size
tot sfb	int	20 bits
base	int	18 bits
buff ch	int	6 bits
subgroup inc	short	12 bits
cell inc	short	7 bits
offset	int	10 bits
cb	int	9 bits
top	int	11 bits
bot	int	11 bits
fac trans	int	16 bits
l right q	long	32 bits
l left q	long	32 bits

4.3 Analisys Result Verification

In order to verify the correctness of the effective size analisys explained in sections 4.1 and 4.2, we set a maximum value for the bit width of each variable. In other words, we used the operation AND in the program so that the proper lower bits of the variables can be taken

out. As a result, the program was compiled and running normally. Further more, the resulting output files (pcm format) before and after analyzing the effective sizes of the variables were identical.

5 Compilation of the Program and Design of the Soft-Core Processor

5.1 Compilation of the Program

After rewriting the MPEG-2 AAC decoder program in Valen-C, we compiled it with the Valen-C compiler. Customized Bung-DLX with different even datapath widths are used as soft-core processors.

When the datapath width changes from 32 bits to 30 bits, the total memory size increases. While the datapath width decreases from 30 bits to 24 bits, the total memory size also decrease. However, when the datapath width changes from 24 bits to 22 bits, the total memory size increases. This is because 24-bits variables are assigned to *long* instead of *int*. The compilation results show that Bung-DLX with a datapath width of 24 bits is the best soft-core processor for our embedded system. The next section explains the design of the 24-bits Bung-DLX.

5.2 Design of the Soft-Core Processor

Bung-DLX with a datapath width of 24 bits was adopted as the soft-core processor for the embedded system. Table 6 summarizes the design results of the customized Bung-DLX. Figure 5 shows the layout result of the designed chip.

Table 7. Design results of Bung-DLX

Design process	CMOS $0.5\mu m$ three metal layers and one poly silicon layer
Chip size	$4.76mm \times 4.76mm = 22.66mm^2$
Gates	5,837
Transistors	129,133
I/O pins	67

6 System Performance Evaluation

In this section, we evaluate the performance of the developped embedded system by comparing the performance of a 32-bits system with the system developped with a datapath width of 24 bits. For this evaluation, we utilized Bung-DLX Simulator, developped with a cycle-accurate simulator toolkit for soft-core processors [5].

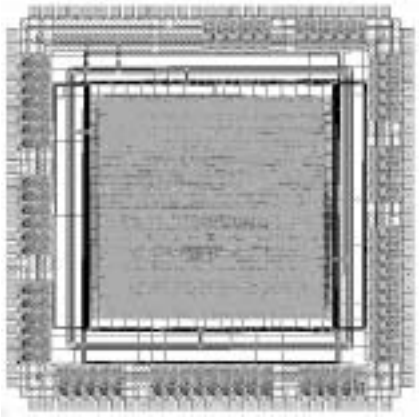

Figure 5. The layout result of the designed chip

6.1 Bung-DLX Simulator

6.1.1 Overview

We developed an instruction level simulator using the proposed toolkit in [5].

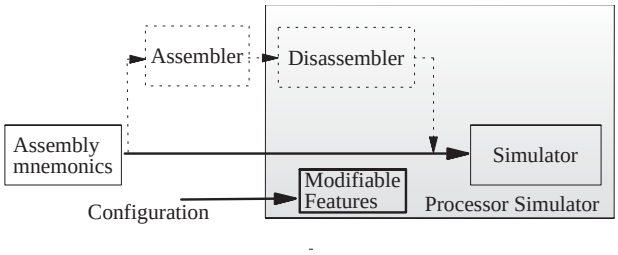

Figure 6. Traditional vs. our approach

The innovation we have taken is decoupling processor internal disassembler (i.e. instruction decoder) from the simulation description and taking a symbolic instruction format as the input of simulator as is shown in figure 6. On consideration, that during architecture/instruction set exploration phases it is hardly needed to encode instructions into bitmap. And instruction bitmap encoding has no direct effect to simulator accuracy[1].

[1] Indeed a good instruction format encoding is crucial for realizing an efficient instruction decoder circuit in hardware implementations.

The aforementioned approach has yet several advantages. Firstly a slight simulation speed gain is obtained, since no controller to encode instruction bitmaps into internal instruction tokens is needed. Secondly, any kind of instructions is easily formed including those of difficult to express in assemble-disassemble approach like a variable length instruction, a multi operand instruction and an instruction with complex encoding. So the system designer does not need to bother with instruction encoding which has no direct relation to architecture explorations.

6.1.2 Instruction Description

Being aware that instruction removal, addition and modification might be frequently carried out during architecture exploration, we keep the description format of instruction set as simple as possible but yet easy to extend. The supplied information includes a mnemonic format, an instruction behavior, an instruction pipeline cycle count and optionally an instruction bitmap for RTL generation.

The mnemonic is formed as a list of symbol. The instruction behavior is expressed using a list of behaviors which is then converted to a lambda[2] construction. Within behavior list, any valid scheme expression can be included as far it can be converted into lambda expression.

The $Inst : reg$ procedure inserts into the controller lookup table the opcode as key. The table entry is a lambda expression acquired by converting the behavior list.

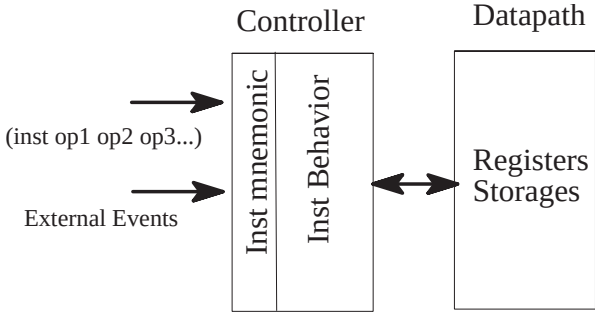

Figure 7. Bung-DLX simulator architecture

6.1.3 Processor Description and Runtime

In abstract view, Bung-DLX simulator consists of a controller and a datapath. The controller is actually a lookup hash table keyed with instruction mnemonic or processor external events (interrupt etc) and related instruction/event behavior at the entry side. The processor controller is generated based on registered instructions and events. Execution procedure of single instruction is performed as below.

```
(let ((lambda (lookup opcode)))
  (apply lambda (op1 op2 op3 ...)))
```

Bung-DLX Simulator datapath is populated with storages only. Signals coming from controller and interconnections among storages in original Bung-DLX VHDL RTL are precisely emulated by behavior entries in the table of Bung-DLX simulator controller. Better simulation speed is one advantage of this approach.

Simulator runtime features such as breakpoints, cycle execution and state monitoring are realized using statically or dynamically hooked functions to post-clock execution queue. A function hooking can be performed in the following manner.

```
(define (monitor-every-1000-cycle)
  (if (= 0 (rem cycle 1000))
      (disp:intrn reg-file mdr cycle)))
(post-clock-hook monitor-every-1000-cycle)
```

6.2 Evaluation Results

Table 7 shows a comparison between a 32-bits system and the 24-bit system we have developped. We calculated the number of gates and the total cell area with Design Compiler (Synopsys). The RAM size and number of words were calculated by Bung-DLX simulator and we used the resulting assembly code after compiling the program with Valen-C compiler. We also used the power model of the cell library to calculate the power consumption of the processor as the sum of the cell internal power and the net switching power.

Table 8. 32-bits system vs. 24-bits system

Evaluation items	32-bits	24-bits	difference
# of gates	7989	5837	-26.9 %
Cell area (mm^2)	3.438	2.492	-27.5 %
RAM size (kByte)	157.300	124.170	-21.0 %
# of words	39325	41390	+5.2 %
Power (mW)	68.115	61.281	-10.0 %

An important cost reduction of the total cell area (-26.9%) and data memory size (21%) was achieved. The power consumption in the datapath and the control path also reduced by 10%. However the number of words increased by 5.2%. In total, we can conclude that the design cost after narrowing the datapath width has improved without sacrificing the system performance.

[2] Anonymous functions, a concept that is well known in functional programming

7 Conclusion

We have presented a design experience of an embedded system LSI for MPEG-2 AAC decoder based on a soft-core processor. Firstly, We compiled the software program with Valen-C compiler and then tuned up the soft-core processor for lower design cost. Design results show that we can achieve cost reduction by changing the datapath width.

8 Acknowledgment

We would like to express our thanks to Semiconductor Technology Academic Research Center (STARC) for their support. We are very grateful to Sanyo Electric Co, Ltd. for providing us the source code of the MPEG-2 AAC decoder.

The VLSI chip in this study has been fabricated in the chip fabrication program of VLSI Design and Education Center (VDEC), the University of Tokyo with the collaboration of Hitachi Hokkai Semiconductor Ltd. and Dai Nippon Printing Corporation.

References

[1] H. Yasuura, H. Tomiyama, A. Inoue and F. N. Eko, "Embedded System Design Using Soft-Core Processor and Valen-C," Journal of Information Science and Engineering, , No. 14, pp.587-603, August 1998.

[2] A. Inoue, H. Tomiyama, T. Okuma, H. Kanbara, and H. Yasuura, "Language and Compiler for Optimizing Datapath Width of Embedded Systems," IEICE Trans. Fundamentals, Vol. E81-A, No. 12, pp. 2595-2604, Dec. 1998.

[3] F. N. Eko, A. Inoue, H. Tomiyama, and H. Yasuura, "Soft-Core Processor Architecture for Embedded System Design," IEICE Trans. on Electronics, Vol. E81-C No.9 pp1416-1423 Sep. 1998.

[4] H. Yamashita, H. Tomiyama, A. Inoue, F. N. Eko, T. Okuma, and H. Yasuura, "Variable Size Analysis for Datapath Width Optimization," Proc. of Asia Pacific Conference on Hardware Description Languages (APCHDL'98), pp. 69-74, July 1998.

[5] E. Nuprasetyo and H. Yasuura, "A Cycle-Accurate Simulator Toolkit for Soft-Core Processors," Proc. of Asia Pacific Conference on cHip Design Languages (APCHDL'99), pp. 11-16, October 1999.

[6] Reference Manual for Kyushu-u Std. Cell Lib. `http://www.vdec.u-tokyo.ac.jp/DesignLib/Kyushu/`