Optimizing a Vocoder for Operation on a StarCore™-Based DSP

By Kim-Chyan Gan

Like most high-performance DSPs, the StarCore™ DSP core uses a very-long-instruction-word (VLIW)[1] architecture running at a high frequency. This architecture fetches very long program instructions and decodes them into multiple simple instructions that are processed in parallel using the atomic functional units in the DSP core. The functional units include multiple Arithmetic Logic Unit (ALUs) and a data-fetching unit. Programming such a DSP is difficult because the programmer must schedule parallel instructions explicitly. This application note describes how to optimize an existing vocoder algorithm using function categorization, which allows the programmer to predict the achievable target performance at the beginning of the project cycle. The approach also permits the programmer to dissect the complexity of the functions, which is useful in project planning. Dissection identifies which functions can yield the greatest gain after optimization. The methodology was applied to a G.723.1A vocoder to optimize it for StarCore SC140 core operation. The resulting vocoder application consumed the lowest number of MIPS ever seen for such a vocoder application.

CONTENTS

1. High-performance superscalar DSP is not considered in this application note since the programmer does not have to schedule the instructions explicitly.

© Freescale Semiconductor, Inc., 2004. All rights reserved.

1 Function Profiling

First, we ported the vocoder to the SC140 core architecture and profiled the performance. Based on the profiling results, the functions were categorized into one of two groups. Group I functions, typically arithmetic computations, accounted for about 20 percent of the code size but consumed about 80 percent of the execution time. Group II functions, typically control code, accounted for about 80 percent of the code size but consumed only about 20 percent of the execution time. This 80/20 rule is consistent with most typical DSP applications. Most of the optimization effort was focused on the Group I functions.

We further categorized the Group I functions to determine which were the most time consuming and therefore the most likely to yield improved performance through optimization. Such an assumption is not always valid, but this approach offers a starting-point that is most likely to minimize the work effort required. An additional consideration was to decouple instructions within a function so that all ALUs were used effectively. Profiling does not yield the information required for instruction decoupling. However, there is a recommended metric of potential optimization represented by the following equation:

Equation 1

Metric = Total cycle consumed within a function * improvement factor.(1)

The improvement factor depends on the number of ALUs and the DSP architecture. For example, the SC140 core is a four-ALU DSP, so the instructions within a function can be fully decoupled into four computations, with a potential improvement factor of 4.

2 Cycle Improvement Factor Based on Categorization

We predicted the improvement factor after optimization for the Freescale MSC8101 device, which has a single four-ALU VLIW-based core. Using a brute force approach equates the improvement factor with the number of ALUs, in this case a 3–4 fold improvement. The efficiency is not exactly equal to the total number of ALUs because sometimes you cannot decouple all functions into four independent calculations.

Because the brute force approach is general and not highly inaccurate, we further categorized the Group I functions to obtain a more accurate prediction improvement factor prediction. When an algorithm such as a vocoder is first developed, it is almost a direct mapping between the ITU-C specification mathematical formulas and a high-level programming language, such as C. The implementation maximizes the number of computations from a mathematical view, but it is not optimized for a specific DSP architecture. Categorization maps the specific mathematical calculations to the DSP architecture, which then allows you to estimate how many cycles can be gained by optimization. We categorized Group I functions as codebook, filtering, or energy functions. Evaluation of these vocoder function categories yields the estimated improvements shown in **Table 1**.

Table 1. Improvement Factor Using Function Categorization

Function	Two-ALU DSP	Four-ALU DSP
Codebook search on excitation pulse	1.5–4X	3–8X
Filtering (IIR, FIR, ARMA)	2.5–3.5X	5–7X
Energy, Correlation	1.5–2X	2.5–3.5X

.

2.1 Codebook Search

Because a codebook search typically consumes the greatest number of cycles during vocoder execution, it offers the best opportunity for optimization improvement. The ITU implementation optimizes the codebook search function by minimizing the number of computations required. This offers opportunities for optimizing this code for the DSP architecture by using such techniques as algorithm transform and data packing, holding values in registers to avoid multiple memory reads, and so on. Register allocation is extremely important in codebook searches because the searches include a number of excited pulses and accumulative metrics. All of these can be stored in registers to reduce the number of memory reads and writes. Compiled code has a very low efficiency because it stores most data in memory. This code must be written in optimized assembly code.

2.2 Filtering

A direct implementation of the filtering function uses multiply and accumulation (MAC) and shifting as its mathematical formula. The MAC or shifting operation consumes the same number of cycles as the number of filter taps. Using the modulo addressing feature in MSC8101 DSP architecture, shifting can be omitted, resulting in 2X performance gain. Using a multisampling technique further increases performance by 3X in the MSC8101 device. Thus, the total gain is around 6X. The inefficiency implementation is due to IIR since part of the computation cannot be fully decoupled because part of the current computation depends on the result of the previous sample. The modulo addressing, together with the multisampling technique, reduces the efficiency of the compiler. Thus, the filtering function is usually written in optimized assembly.

2.3 Energy

Usually, energy or correlation code is written in optimized C, which has more than 90 percent of the efficiency of optimized assembly. Not all the energy functions can be computed using split computation because of the saturation.[2] In this case, there is no improvement in optimization. However, this problem can be solved by mixing the energy calculations with other computations, which highly complicates the register allocation and lowers compiler efficiency. In this case, the entire function must be converted into assembly code.

In some functions, SIMD instructions within the DSP can be used. For example, the SC140 **max2** instruction can be used to search through the maximum value within an array. Combined with split computation, this technique gives an improvement factor ranging from 5 to 7.

Special instructions available in the DSP should be fully utilized. For example, the SC140 core has a **maxm** instruction that computes the maximum of the absolute value. This special instruction can improve performance by a factor of 2. In addition, the SC140 core has the **subl** instruction, which can improve the FFT[3] algorithm by 10–20 percent.

By categorizing vocoder functions, the developer can give a more accurate target performance after profiling the ported code. Accurate performance prediction can be invaluable in planning a project. Using the profiling information, the programmer knows which functions yield the best cycle gains after optimization using Equation (1).

2. When all the elements in an array are positive, split-computation has the same result as the original computation even if saturation happens. However, the results may be different if an array contains negative values.
3. FFT is used in recent Vocoders, such as AMR.

3 Complexity and Code Size of Vocoder Functions

The complexity of a function is proportional to the development time. The more complex the function, the more time it takes to write. Code size may play a critical role in determining the DSP system performance, which highly depends on the memory hierarchy of the DSP architecture.

3.1 Complexity

The vocoder functions are further arranged according to their complexity, as shown in **Table 2**. Complexity can be used to estimate the project development schedule. If a person takes one day to write an energy function, it will take that person five days to write a correlation function.

Table 2. Complexity of Functions

Function	Complexity	Optimization Approach
Codebook search	20X	Algorithm transform
Filtering	8X	Multisampling, modulo addressing
Auto/Cross-Correlation	5X	Multisampling
Energy	1X	Split-computation

Filtering is optimized through multisampling and modulo addressing. Multisampling allows several independent operations to execute simultaneously. The number of independent operations should be the same as the number of ALUs in the DSP.[4] Modulo addressing should be set up at the beginning of the filtering function. The function should be restored to linear addressing before termination. Modulo addressing makes the debugging slightly harder. The complexity of filtering is 8X. Auto or cross correlation uses multisampling for optimization. Its complexity is 4X. Split-computation is applied to energy functions, which have a 1X complexity.

3.2 Code Size

The code size increases before and after optimization depending on instruction encoding schemes and the number of ALUs, which determines the loop-unrolling factor. Multisampling and split-computation increase the code size since these optimization techniques unroll a loop into several independent computations. Some optimization techniques—such as compact storage, efficient calculation, avoiding memory storage by using registers, and so on—may decrease the code size.

In the SC140 core, a codebook search yields a 0–50 percent code size increase after optimization. Filtering, correlation, and energy increase code size by 50–200 percent due to the loop unrolling. The total code size of the optimized vocoder is around 5–20 percent more than for its ported counterpart. Code size is harder to estimate than performance. Usually, the cycle-intensive code executes a large number of times. However, it is not the case for code size. Any variance (such as additional control code) may affect the code size significantly.

4. If an ALU can execute a dual MAC (SIMD instruction), the number of independent operations should be twice the number of ALUs in the DSP.

3.3 Reducing Complexity by Compiler and Architecture

In the competitive business environment of today, time-to-market is critical for product development. Both the DSP architecture and the compiler play a very important role in reducing the product development cycle. For example, the SC140 core has a five-stage pipeline and orthogonal registers. A short pipeline can prevent complicated scheduling. Programming assembly in the SC140 can be relatively easy and can reduce the development cycle. The SC140 compiler very efficiently compiles Group II functions. For some Group I functions, it can achieve more than 90 percent of the assembly efficiency.

The compiler should be used extensively to write optimized C for the Group I functions. Optimized C serves a couple of purposes. First, optimized C can be used to emulate the assembly code and aid in the debugging in writing assembly. Second, it serves as a quick way to verify whether an algorithm transformation is correct. Writing assembly code takes a long time. For example, in energy calculation of two-ALU DSPs, split summing can yield different results than the original because of a different order of computation and saturation in ITU intrinsic function. Writing a Group I function in optimized C allows earlier and faster detection of any defects. Third, optimized C code can sometimes be used in Group I, avoiding the time-consuming process of writing assembly code and shortening the development cycle. The SC140 compiler can achieve more than 90% of the efficiency of assembly in compiling an energy/correlation function.

4 G.723.1A Results

The original ITU G.723.1A code was ported to the SC140 platform. Profiling was conducted to find the functions that consume the most cycles. These functions were categorized into three different types. Then the ported and optimized functions were compared to find the improvement factor, as shown in **Table 3**. The results are coincident with theory presented in**Section 3**, *Complexity and Code Size of Vocoder Functions,* on page 4, and the improvement factor is in the range predicted in **Table 2**.

Table 3. Cycle Counts of Functions in the G.723.1A Vocoder

Categorization	Function	ITU Cycles	Optimized Cycles	Improvement Factor
Cookbook search	Find_best (6.3)	51052	7283	7.0
	D4i64_lbc (5.3)	35466	11003	3.2
Filtering	comp_ir	6651	1019	6.5
	error_wght	17535	2626	6.7
	sub_ring	6668	1016	6.6
	upd_ring	6242	1023	6.1
	Spf	4603	868	5.3
	Synt	3006	586	5.1
Energy/Auto/ Cross-Correlation	Estim_pitch	20848	6763	3.1
	cor_h	1459	532	2.7
	cor_h_x	3021	1128	2.7

The code sizes of ported and optimized G.723.1A functions on the SC140 core are gathered, as shown in **Table 4**.

Table 4. Code Sizes of Functions in G.723.1A Vocoder

Categorization	Function	ITU Code Size	Optimized Code Size	Expansion Percentage
Cookbook search	Find_best (6.3)	2352	3154	34.1
	D4i64_lbc (5.3)	2496	2364	–5.3
Filtering	comp_ir	304	684	125.0
	error_wght	192	560	191.7
	sub_ring	336	768	128.6
	upd_ring	272	666	144.9
	Spf	480	888	85.0
	Synt	112	356	217.9
Energy/Auto/ Cross-Correlation	Estim_pitch	416	1296	211.5
	cor_h	1424	716	–49.7
	cor_h_x	224	704	214.3

The million cycles per second (MCPS) and the resource of G.723.1A on the SC140 core are displayed in **Table 5**. This G.723.1A has the lowest MCPS (5.32 MCPS) ever seen in the market, more than 40 percent lower than its closest competitor, Delphi, who stands at 7.5 MCPS on the TI C64X.

Table 5. MCPS and Resources of G.723.1A on the SC140 Core

Library	MCPS	Code	Initialized Data	Stack	Persistent Data	Scratch Data
Encode	4.73	25042	42	2056	1482	770
Decode	0.59	8670	8	2040	420	0
Common	—	5410	18709	—	—	—

5 References

The following documents are available at the website listed on the back cover of this application note.

[1] Bogdan Costinescu, Costel Ilas, *Developing High-Performance DSP Software Products Using C*, Embedded Systems Show, London, May 2001.

[2] STCR140MLAN/D, *StarCore Multisample Programming Technique*, 1999.

[3] MNSC140CORE/D, *SC140 DSP Core Reference Manual*.

[4] MNSC100CC/D, *SC100 C Compiler User's Manual*.

[5] MSC8101RM/D, *MSC8101 Reference Manual*.

NOTES:

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations not listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GMBH
Technical Information Center
Schatzbogen 7
81829 München, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T. Hong Kong
+800 2666 8080

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters which may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals” must be validated for each customer application by customer’s technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. StarCore is a trademark of StarCore LLC. All other product or service names are the property of their respective owners.

© Freescale Semiconductor, Inc. 2004.

AN2562
Rev. 2
12/2004