G.729E Algorithm Optimization for ARM926EJ-S Processor

Technical Report CECS-03-09
March 21, 2003

Anshuman Tripathi, Shireesh Verma, Daniel D. Gajski

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA
(949) 824-8059

{anshuman, shireesh, gajski}@ics.uci.edu

G.729E Algorithm Optimization for ARM926EJ-S Processor

Technical Report CECS-03-09
March 21, 2003

Anshuman Tripathi, Shireesh Verma, Daniel D. Gajski

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA
(949) 824-8059

{anshuman, shireesh, gajski}@ics.uci.edu

Abstract

This report presents a methodology towards a complete software implementation of G.729 Annexe E on ARM926EJ-S processor running at 200 MHz clock speed. The methodology spans both, target independent as well as target dependent optimizations. The main focus is towards code speed up to reduce the per frame execution time as much below the 10 ms constraint as possible. Although no constraint is given for code size, the outcome of the optimization process has so far resulted in an overall reduction of the code size. The code was also run on an Aptix FPGA board to calibrate the ARMulator (ARM instruction set simulator). Although the optimizations presented here, were performed and analyzed on the G.729E algorithm, they can be used on any similar DSP code for speed up.

Contents

1 Introduction **1**
1.1 Project goal . . . 1
1.2 G.729 standard . . . 1
1.2.1 G.729 recommendation . . . 1
1.2.2 Speech quality . . . 1
1.2.3 Model of G.729 . . . 1
1.2.4 Speech encoding and decoding . . . 2
1.2.5 Encoder . . . 3
1.2.6 Annexe E . . . 4

2 Original C code **4**

3 Optimization methodology **4**
3.1 Test vectors and development tools . . . 4
3.2 Porting G.729 code to ARM926EJ-S . . . 6
3.3 Global-level optimizations . . . 6
3.3.1 Function inlining . . . 6
3.3.2 Post index addressing . . . 6
3.3.3 Loop invariant code motion . . . 6
3.3.4 32-Bit DPF format and operations . . . 6
3.3.5 Two Q15 operations in a single call . . . 7
3.3.6 Redundant optimization . . . 7
3.3.7 Processor specific adaptations . . . 7
3.3.8 Performance . . . 8
3.4 Function-level optimization . . . 8
3.4.1 Criterion for selecting functions . . . 9
3.4.2 Loop unrolling . . . 9
3.4.3 Loop merging . . . 9
3.4.4 Loop count reduction . . . 9
3.4.5 Declaring local variables . . . 10
3.4.6 Replacing functions by arithmetic operators . . . 10
3.4.7 Simpler test conditions . . . 10
3.4.8 Removal of Pointers . . . 10
3.4.9 Sign-extending/Zero-extending . . . 11
3.4.10 Selected functions for function level optimizations . . . 12
3.4.11 Performance . . . 12
3.5 Algorithmic changes . . . 12
3.5.1 Identifying algorithms to change . . . 13
3.5.2 Platform-independent changes . . . 13
3.5.3 Platform-dependent changes . . . 13
3.5.4 Function selection for algorithmic modifications . . . 13
3.5.5 Performance . . . 13
3.6 Assembly implementation of functions . . . 13
3.6.1 Selecting functions . . . 14
3.6.2 Implementation approaches . . . 14
3.6.3 Performance . . . 14

4 Measurement techniques **14**
4.1 Execution time . . . 14
4.2 ARMulator Benchmarking . . . 14

5 Results **15**
5.1 Execution time . . . 15

6 Optimization examples **16**
6.1 Optimizations in $L_shl()$ . . . 16
6.1.1 Function-Level Optimizations . . . 16
6.1.2 Algorithmic Changes in assembly . . . 16
6.1.3 Performance . . . 16
6.2 Optimizations in $Norm_Corr()$ . . . 16
6.2.1 Function-Level Optimizations . . . 16
6.2.2 Algorithmic Changes . . . 17
6.2.3 Assembly implementation . . . 17
6.2.4 Performance . . . 17

7 Conclusion **17**

8 Future research **18**

9 Acknowledgements **18**

A Appendix: Initial profile **19**

List of Figures

1	Speech reconstruction by filtering codebook excitation.	2
2	G.729 Speech synthesis model.	2
3	Encoding principle.	3
4	SoC design flow organization with code size in KB and data traffic in bytes.	5
5	Post index addressing example.	7
6	Single $L_macD()$ call example.	8
7	Instruction reduction with CLZ instruction usage.	8
8	Loop unrolling example.	9
9	Loop merging example.	9
10	Use of local variable example.	10
11	Left shift example.	10
12	Addition and Subtraction example.	10
13	if condition example.	10
14	C and Assembly version for code with pointers.	11
15	C and Assembly version for code without pointers.	11
16	Array access for implementation (a)with pointers (b) without pointers.	11
17	C and Assembly code for operation on a 32-bit quantity.	12
18	C and Assembly code for operation on a 16-bit quantity.	12
19	Percentage improvement in execution speed	15
20	Man months taken per optimization level	15

List of Tables

1 G.729 MOS Results . . . 2
2 Stages of software development process . . . 5
4 Performance after global optimizations . . . 8
3 Reduction in cycle count for a 30 frame testvector on G729E code observed after re-writing each of the functions in assembly. . . . 9
5 Performance after function level optimizations . . . 12
6 Performance after algorithmic optimizations . . . 13
7 Performance after mixed C assembly optimizations . . . 14
8 Execution statistics for a 30 frame test-vector set running on a Non-optimized G729 code at 6Mhz . . . 14
9 Execution statistics for a 30 frame test-vector set running on an optimized G729 code at 6 Mhz . . . 15
10 Scaling factor for the ARMULATOR. . . . 15
11 Performance after modifying $L_Shl()$ . . . 16
12 Performance after modifying $Norm_Corr()$ . . . 17

G.729E Algorithm Optimization for ARM926EJ-S Processor

A. Tripathi, S. Verma, D. Gajski

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA

Abstract

This report presents a methodology towards a complete software implementation of G.729 Annexe E on ARM926EJ-S processor running at 200 MHz clock speed. The methodology spans both, target independent as well as target dependent optimizations. The main focus is towards code speed up to reduce the per frame execution time as much below the 10 ms constraint as possible. Although no constraint is given for code size, the outcome of the optimization process has so far resulted in an overall reduction of the code size. The code was also run on an Aptix FPGA board to calibrate the ARMulator (ARM instruction set simulator). Although the optimizations presented here, were performed and analyzed on the G.729E algorithm, they can be used on any similar DSP code for speed up.

1 Introduction

1.1 Project goal

This project focuses on developing a software implementation of the Coding of Speech at 8 kbit/s and 11.8 kbit/s using Congugate-Structure Algebraic-Code-Excited Linear-Prediction (CS-ACELP) algorithm running on the ARM926EJ-S processor which meets the constraints specified in the algorithm. The emphasis is on developing a software implementation without any custom hardware. The algorithm standard is available as ITU-T recommendations G.729 and G.729E.

1.2 G.729 standard

1.2.1 G.729 recommendation

Speech compression technology is widely used in digital communication systems such as wireless systems, VoIP, and video conference technology. Speech compression reduces data redundancy and hence eases bandwidth requirements. The compression technique described in the ITU-T G.729 Recommendation is commonly employed in speech transmission systems because of the sophisticated quality of the reconstructed speech signal.

The G.729 algorithm standard models the functionality of the human vocal tract exploiting DSP techniques in order to synthesize speech or recreate it at the receiving end. Human speech is produced when air from the lungs is forced through an opening between two vocal folds called the glottis. Tension in the vocal chords caused by muscle contractions and forces created by the turbulence of the moving air force the glottis to open and close at a periodic rate. Depending on the physical construction of the vocal tract, oscillations occur between 50 to 500 times per second. The oscillatory sound waves are then modified as they travel through the throat, over the tongue, through the mouth and over the teeth and lips.

1.2.2 Speech quality

The Mean Opinion Score (MOS) is a commonly used test to assess speech quality. In this test, listeners rate a coded phrase based on a fixed scale. The MOS rating ranges from 0 to 5 and a MOS of four or higher is considered toll quality, which means that the reconstructed speech is almost as clear as the original speech. Tests have shown that encoding systems based on G.729 at 8 kbits/s provide toll-quality speech for most operating conditions. However, the encoding systems based on G.729E may not provide toll-quality reconstructed speech. quality may not be as good in some cases due to the included background noise. The details are shown in Table 1 [ITU-T P.810 96]. Here MNRU stands for Modulated Noise Reference Unit.

1.2.3 Model of G.729

The International Telecommunications UnionTelecommunications Standardization Sector (ITU-T) G.729 Recom-

Test Conditions		MOS
Clean		4.125
Encoder Background Noise	10 dB MNRU	3.65
	30 dB MNRU	3.975
Channel Errors	1.0% bit error	3.3
	0.1% bit error	3.95

Table 1. G.729 MOS Results

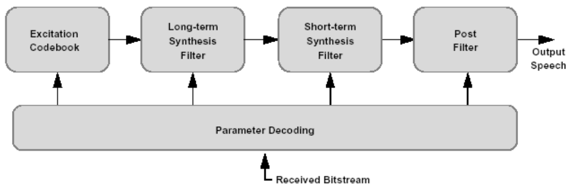

Figure 1. Speech reconstruction by filtering codebook excitation.

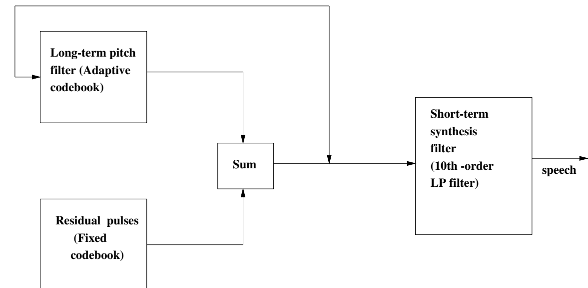

Figure 2. G.729 Speech synthesis model.

mendation defines an algorithm for encoding speech signals at 8 kbit/s using Conjugate-Structure Algebraic-Code-Excited Linear-Prediction (CS-ACELP). In this system, an analog voice signal is passed through a 300 Hz 3400 Hz bandpass filter and sampled at 8 kHz to yield digital data that is converted to a 16-bit linear PCM speech signal. An encoder analyzes the speech signal to extract the parameters of the CELP model. These parameters are encoded and transmitted in a bitstream. The decoder for this system uses the received parameters to retrieve the synthesis filter coefficients. The speech is then reconstructed by filtering the excitation codebook as shown in Figure 1. The vocoder operates on 10 ms frames with 5 ms look-ahead for linear-prediction (LP) analysis. Hence, the overall algorithmic delay is 15 ms.

Figure 2 shows the G.729 speech synthesis model. A sequence of pulses is combined with the output of a long-term pitch filter. Together, they model the buzz produced by the glottis and build the excitation for the final speech synthesis filter, which in turn models the throat and mouth as a system of loss less tubes.

The initial sequence of so-called residual pulses is constructed by assembling predefined pulse waveforms taken out of a given, fixed codebook. The codebook contains a selection of so-called fixed code vectors, which are basically fixed pulse sequences with varying frequency. In addition, applying a variable gain factor scales the pulse intensities.

The output of the long-term pitch filter is simply a previous excitation sequence, modified by scaling it with a variable gain factor. The amount by which excitations are delayed in the pitch filter is a parameter of the speech synthesis model and can vary over time. The long-term pitch filter is also referred to as adaptive codebook since the history of all past excitations basically forms a codebook with varying contents out of which one past excitation sequence the so-called adaptive code vector, is chosen.

Finally, the excitation, which is constructed by adding fixed and adaptive codebook vectors, is passed through the short-term speech synthesis filter, which simulates a system of connected loss less tubes. Technically, the short-term filter is a tenth order linear prediction filter meaning that its output is a linear combination (linear weighted sum) of ten previous inputs. The ten linear prediction coefficients are intended to model the reflections and resonance of the human vocal tract.

1.2.4 Speech encoding and decoding

Instead of transmitting compressed speech samples directly, the input speech samples are analyzed in order to extract the parameters of the speech synthesis model. These parameters are then transmitted to the receiving side where they are used to synthesize and reconstruct speech.

On the encoding side, the input speech is analyzed to estimate the coefficients of the linear prediction filter, removing their effects and estimating the intensity and frequency. The inverse filtering of the incoming speech removes the linear prediction effects. The remaining signal called the residual, is then used to estimate the pitch filter parameters. Finally, the pitch filter contribution is removed in order to find the closest matching residual pulse sequence in the fixed codebook.

At the receiver, the transmitted parameters are decoded,

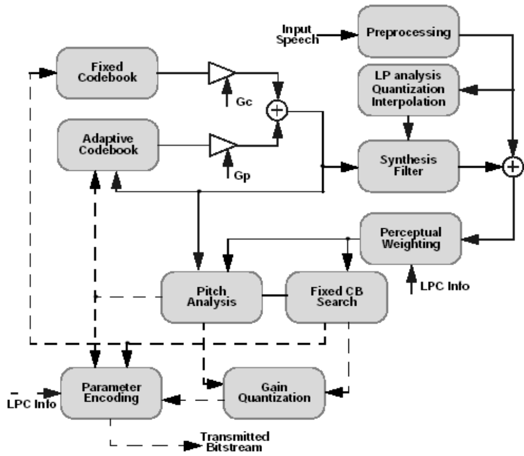

Figure 3. Encoding principle.

combining the selected fixed and adaptive code vectors to build the short-term excitation. The linear prediction coefficients are decoded and the speech is synthesized by passing the excitation through the parameterized short-term filter.

All together, this speech synthesis method has the advantages of achieving a high compression ratio since it tries to transmit only the actual information inherent in the speech signal. The filters of the speech synthesis model eliminate all the redundant relationships, which are due to the way the human vocal tract is organized. The vocal tract model provides an accurate simulation of the real world and is quite effective in synthesizing high quality speech. The speech model proves to simulate the real world quite effectively and synthesized speech is of high quality. In addition, encoding and decoding are relatively efficient to compute.

1.2.5 Encoder

Encoding is based on finding the parameters for the speech synthesis model at the receiving side, which will then be transmitted, to the decoder over the transmission medium. The speech synthesis model is a code-excited linear predictive (CELP) model. In this model, the locally decoded signal is compared with the original signal. The filter parameters are then selected to minimize the mean-square weighted error between the original and reconstructed signal. In order to synthesize speech in the decoder a 10th order linear predictive (LP) synthesis filter (Equation 1) is excited with a signal constructed by adding two vectors from the two codebooks as discussed later. The encoding principle is shown in Figure 3.

$$H(z) = 1/A(z) \tag{1}$$

The input samples are passed through a 140 Hz high-pass filter and a tenth-order LP analysis is performed on them. The resulting LP parameters are then quantized in the line spectral pair (LSP) domain with 18 bits. The input frame is divided into two 5 ms sub-frames to optimize tracking of the pitch and gain parameters and to reduce the complexity of the codebook searches. The interpolated LP coefficients are applied to the first sub-frame. The quantized and un-quantized LP filter coefficients are applied to the second sub-frame. The excitation in each sub-frame is represented by both an adaptive codebook contribution and a fixed codebook contribution. The adaptive and fixed codebook parameters are transmitted every sub-frame.

Adaptive codebook The adaptive codebook is based on a pitch synthesis filter, which is responsible for covering long-term effect. The output of the pitch filter is simply a previous excitation signal delayed by a certain amount (lag) and scaled with a certain gain. Since the delay/lag of the pitch filter can be fractional, the delayed excitation has to be interpolated (using a FIR filter) between the two adjacent (delayed by an integer lag) excitation values. The adaptive codebook component represents the periodicity in the excitation signal using a fractional pitch lag with 1/3 sample resolution. It is searched using a two-step procedure. An open-loop pitch lag is estimated per frame based on a perceptually weighted speech signal. The adaptive codebook index and gain are found by a closed-loop search around the open-loop pitch lag. The signal to be matched, referred to as the target signal, is computed by filtering the LP residual through the weighted synthesis filter. The adaptive codebook index is encoded with 8 bits in the first sub-frame and differentially encoded with 5 bits in the second sub-frame. The target signal is updated by removing the adaptive codebook contribution, and this new target is used in the fixed codebook search.

Fixed codebook The fixed codebook is a an algebraic codebook with 17 bits. The fixed or algebraic codebook covers any remaining pulse excitation sequence left after removing the short-term and long-term contributions. The fixed codebook contains 5 tracks (6 tracks for G.729E) with 8 possible positions each. For each track two positions are chosen (10 pulses all together/12 pulses for G.729E) and transmitted.

The gains of the adaptive and fixed codebooks are vector-quantized with 7 bits using a conjugate structure codebook. In general, the parameters for the two codebooks are chosen such that the error between the synthesized speech (at the output of the LP synthesis filter) and the original speech is minimized. However, for the codebook searches, the original speech is weighted by a weighting filter W(z) in order to account for the special properties of human acoustic perception.

1.2.6 Annexe E

The annexe E of G.729 standard provides the high level description of the higher bit-rate extensions ($11.8kbps$) of Recommendation G.729. It accommodates wide range of input signals, such as speech, with background noise and even music. Differences of this annexe from the initial standard are as follows:

1. A backward LP analysis is added for music signals and stationary background noises. The backward/forward decision selects speech (forward mode) or music (backward mode). The backward/forward procedure also reduces the number of switches needed to perform smooth switching between the appropriate filters. The LP mode and the related information better adapts post-filtering and perceptual weighting to either music or speech. This mode is also used for the error concealment.

2. Two algebraic excitation codebooks are added to extend the bit rate upto $11.8kbps$.

2 Original C code

The original C code provided was found to contain 14,000 line of code. The code had extensive use of pointers and a sizeable number of functions. This forced a change of strategy from attaining a profound understanding of the algorithm and proceeding with optimizations. The revised strategy entailed going into details of the specification document, several other resources from the web and the C code at the same time in order to be able to understand and break the design into algorithmic blocks such that the flow of data and control could be visualized easily.

The SoC (System-on-chip) environment (SCE) was chosen for further analysis for the following reasons:

Implementation choice At this stage the choice of implementation was open i.e. SW, HW or mixed SW/HW. So consideration was attributed to co-design issues and SoC design flow was chosen which is suited for the purpose on account of the features discussed below.

Separation of communication and computation Algorithmic functionality can be detached from the communication functionality. In addition, I/O of a computation can be explicitly specified to show data dependencies.

Parallelism Inherent parallelism in the system functionality can be exposed instead of having to artificially serialize functions assuming a serial implementation. In essence, all possible parallelism can be made available to the exploration tools in order to enhance optimization scope.

Hierarchy Hierarchy is used to group related functionality and abstract away localized effects at higher levels. For example, local communication and local data dependencies are grouped and hidden by the hierarchical structure. So the design is easier to handle.

Granularity The granularity of the basic parts for exploration can be chosen such that optimization possibilities and design complexity are balanced when searching the design space. Basically, the bottom level functions, which build the smallest indivisible units for exploration, reflect the division into basic algorithmic blocks.

In order to achieve the above, the design was ported to SoC environment and profiled using the SCE profiler. The provided the total number of computations for each algorithmic block and their memory requirements. It also gave the data traffic and control flow between them and their hierarchical organization as shown in Figure 4 . The data obtained from the SCE profiler enabled making decisions towards changing the structure of the design, re-arranging the computations among different algorithmic blocks to keep uniformity in the amount of, computations in each algorithmic block, data traffic between pairs of blocks and memory required for each block.

At this stage the implementation was constrained to a complete software implementation.

3 Optimization methodology

This section describes the optimization methodology used in implementing ITU G.729E reference code on ARM926EJ-S processor. The main steps performed during the implementation process are illustrated in Table 2 in the ideal order. Additionally, this section provides the typical problems encountered and their possible solutions. Finally, programming techniques are provided to optimize the code for speed.

3.1 Test vectors and development tools

The test vectors provided by the ITU-T were used to verify algorithmic equivalence between the ported and the original G.729E code. The test vectors used are listed below.

1. algthm.in

2. speech.in

3. algthm118.in

The development tools used included the debuggers *AXD* and *armsd*, *armprof* profiler and *armcc* compiler from the

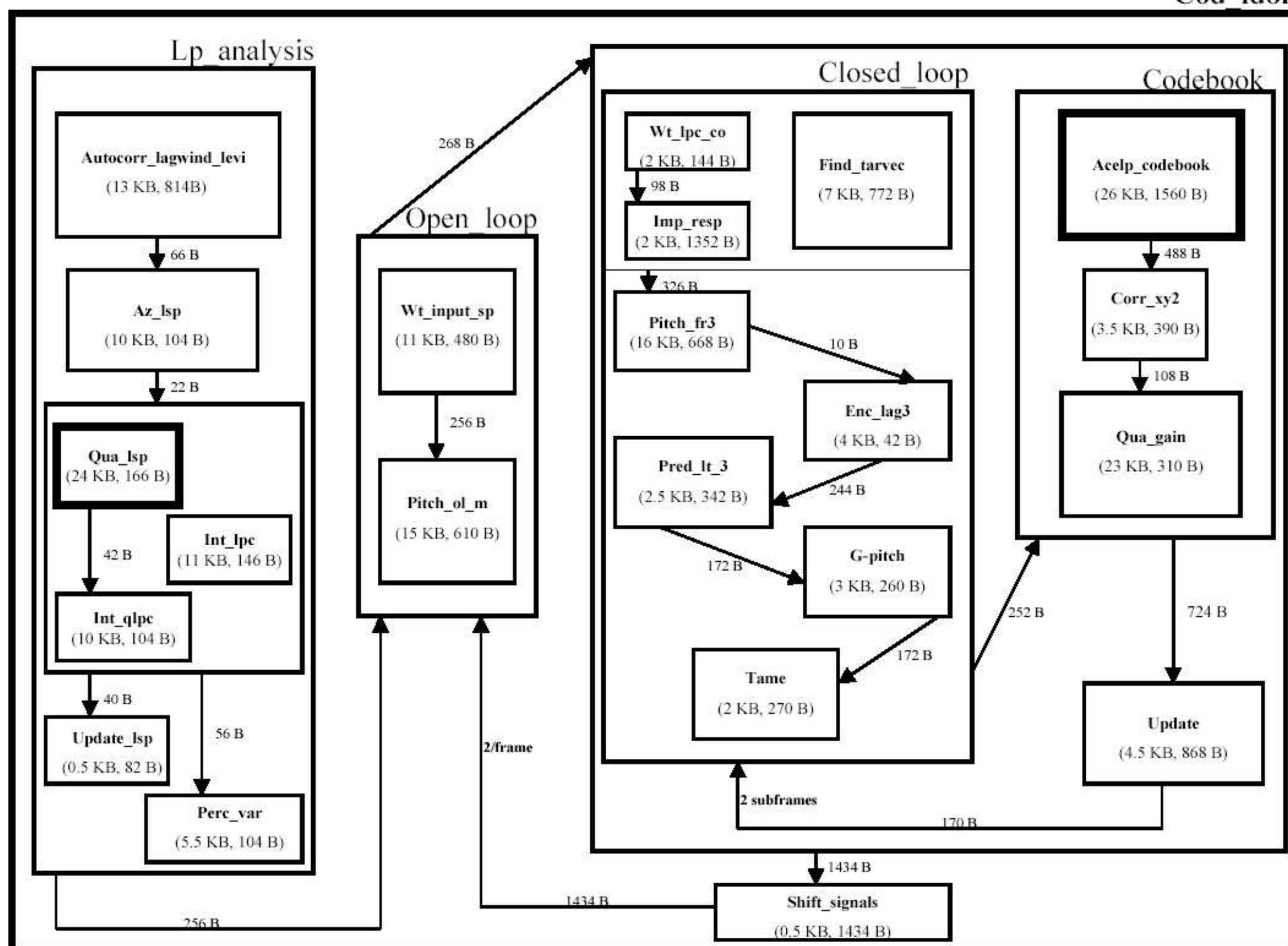

Figure 4. SoC design flow organization with code size in KB and data traffic in bytes.

Development Stage	*Description*
Porting to ARM926EJ-S	Modifying makefiles, choosing cpu parameters
Global-Level Optimizations	Inlining, Processor specific adaptations
Algorithmic Changes	Reducing computational overhead
Function-Level C Optimization	C optimization techniques
Writing functions in assembly	Exploiting assembly features for optimizations

Table 2. Stages of software development process

ARM Developer Suite 1.2. The compiler used to check the C code modifications and generate the test vectors for unit testing was gcc 3.01. The code was developed on the Solaris 8.0 platform. Hardware tests were run on the Aptix FPGA implementaton of the processor.

3.2 Porting G.729 code to ARM926EJ-S

There were not many considerations involved in porting the fixed-point C source code distributed by the ITU-T for the G.729E recommendation to ARM926EJ-S. The makefile was modified and the original C reference code was originally compiled with *armcc* compiler with only minor changes, and the resulting code passed the above listed test vectors on the software simulator. This verified that the compiler is ANSI C compliant.

3.3 Global-level optimizations

3.3.1 Function inlining

Function inlining is the process of replacing a function call with the called function within the callee function. This technique improves execution time by eliminating function-call overhead at the expense of larger code size, as the called function code is inserted in place of the function call. Small, frequently-called functions are the best candidates for inlining in the C code. Those functions which facilitated other optimizations illustrated in this report were short listed first. The short listing criterion also included the number and type (input or output) of parameters the function passed, the data type of the returned value and register allocation in the resulting assembly code.

Similarly, in assembly, called functions were inlined were it was found productive, i.e. did not cause register spilling or the called function was made redundant by the use of specific instructions in the callee function. In C, at the basic operations level, inlining was found to be counter productive. Upon inlining $L_add()$ in $L_mac()$ the cycles increased by 562 cycles per input frame (test performed on G.729). While there was 22.92% (Table 3) improvement seen in $L_mac()$ in assembly as $L_mult()$ and $L_add()$ were inlined in assembly using saturating arithmetic instructions.

The inlined functions are:

1. $sature()$
2. $L_mac()$
3. $L_msu()$
4. $add()$
5. $sub()$
6. $Mpy_32_16()$
7. $Mpy_32()$
8. $Div_32()$
9. $L_Extract()$
10. $L_Comp()$

3.3.2 Post index addressing

In vector operations, at many places the index is recalculated inside the loop which could easily be replaced by post index addressing load instructions removing the arithmetic instructions calculating the displacement from the index base by exploiting the order inherent in the vector elements. As seen in the Figure 5, the initial address calculation is moved out of the loop and subsequent addresses generated with post index addressing (2 bytes). This technique reduced at least one ADD or SUB instruction per load instruction. As shown in the figure the loop size was reduced by more than 50%.

3.3.3 Loop invariant code motion

The introduction of post index addressing enabled sizable loop invariant code motion. e.g. the initial address calculation for the array/vector is loop invariant and can now be safely moved out of the loop as shown in the Figure 5, with the insertion of post indexed load instructions given above. The address calculation overhead for each loop count was removed.

3.3.4 32-Bit DPF format and operations

The ITU-T G.729E uses a representation of 32-bit double-precision numbers known as double precision format (DPF). The 32-bit DPF format was designed for 16-bit processors which do not support 32-bit operations. Thus, firstly the 32-bit word is split into higher and lower 16-bit half words as shown in Equation 2 3 and 16-bit operations are applied on them. Although ARM926EJ-S provides 32-bit operations, these operations had to be implemented in DPF format to maintain bit-exactness of the original ITU-T implementation.

$$hi = L_32 >> 16 \tag{2}$$

$$lo = (L_32 - (hi << 16)) >> 1 \tag{3}$$

$$L_32 = hi << 16 + lo << 1 \tag{4}$$

Here L_32 is a 32-bit signed integer, and hi and lo are 16-bit signed integers. The range of values for L_32 is shown in Equation 5.

```
1  /* Original C code */
2
3  for (i = 0; i <= n; i++)
4  s = L_mac(s, x[i], h[n-i]);
5
6  ;Compiler generated assembly
7  ;code
8  ---------------------------
9  MOV   r0,#0
10 MOV   r4,#0   ; value of i
11 CMP   r5,#0   ; value of n
12 BLT   |L1.92|
13 |L1.48|
14     SUB  r1,r5,r4 ; n-i
15     ADD  r1,r7,r1,LSL #1
16       ; address of h[n-i]
17     LDRSH r2,[r1,#0]
18               ; load
19     ADD   r1,r6,r4,LSL #1
20     LDRSH r1,[r1,#0]
21     BL    L_mac
22     ADD   r1,r4,#1
23     MOV   r4,r1,LSL #16
24     MOV   r4,r4,ASR #16
25     CMP   r4,r5
26     BLE   |L1.48|
27
28 ;Modified assembly code
29 -----------------------
30 ADD r11,r6,r5,LSL #1
31       ;last address h[n-i]
32 MOV r0,#0
33 MOV r4,r6
34 ADD r10,r7,r5,LSL #1
35       ;first address h[n-i]
36 |L1.48|
37     LDRSH r2,[r10],#-2
38       ;each element 2 bytes
39     LDRSH r1,[r4],#2
40     BL    L_mac
41     CMP   r4,r11
42     BLE   |L1.48|
```

Figure 5. Post index addressing example.

$$0x8000000 <= L_32 <= 0x7ffffffe \quad (5)$$

The DPF format and the operations based on it are originally defined in detail in the G.729E oper_32b.c file. The principal operations defined for DPF include:

1. $Mpy_32()$: multiplication of two 32-bit DPF values
2. $Mpy_32_16()$: multiplication of a 32-bit DPF value with a signed 16-bit value
3. $Div_32()$: division of two 32-bit DPF values

These functions originally took their 32-bit parameters in 16-bit half words and returned a 32-bit word which was re-split into half words by the calling function.

Therefore, the optimization took advantage of the processors 32-bit architecture without violating the DPF format. The two 16-bit half words, which were originally processed in two separate variables, were combined into a single 32-bit value using only one variable as shown in Equation 4. Thus, functions that originally received two pointers to two 16-bit arrays could now operate with pointer to a single 32-bit array. This optimization step reduced the number of parameters passed, to the called function, by half.

3.3.5 Two Q15 operations in a single call

As the basic data element (parameter code) was two bytes long, two contiguous data elements could be passed to a function residing in the upper and lower halves of the register. This reduces the number of function calls by half. This was utilized by adding an extra function ($L_macD()$) performing two $L_mac()$ operations, one on the upper half word and the other on the lower half word of the argument registers as shown in the Figure 6. This technique also utilized the two saturating arithmetic units in parallel and also reduced the function call overhead by 50%.

3.3.6 Redundant optimization

The *armcc* is found to generate compare instructions to compare a loop counter with zero, for a speedy loop termination, if the terminating value was passed to it as an argument. While this is a legitimate way for speed optimization, its sometimes applied on global variables which cannot hold zero or negative values and thus the compare and branch pair becomes redundant and can be removed. This removes two instructions per for loop.

3.3.7 Processor specific adaptations

Modifications made to the code which exploit some specific features of the ARM926EJ-S processor are discussed below.

```
;Compiler generated assembly
;code loop
;loop block running n times
LDRSH    r1,[r5,#0]
MOV      r2,r1
BL       L_mac

;Modified assembly code loop
;loop block running n/2 times
LDRH    r1,[r5,#0]
LDRH    r2,[r5,#2]!
ADD     r2,r1,r2,LSL #16
MOV     r1,r2
BL      L_macD
```

Figure 6. Single *L_macD*() call example.

```
;Modified code:
CLZ var_out, L_var1
SUB var_out,var_out,#1

;Compiler generated assembly code:
MOV   var_out, #0
|L1.1952|
CMP   L_var1, #0x40000000
ADDLT var_out, var_out,#1
MOVLT var_out, var_out,LSL #16
MOVLT var_out, var_out,ASR #16
MOVLT L_var1, L_var1, LSL #1
BLT   |L1.1952|
```

Figure 7. Instruction reduction with CLZ instruction usage.

Saturating arithmetic G729 annexe E like other DSP algorithms utilizes saturating, Q15 and Q31 arithmetic. The basic arithmetic operations are provided in the basic_op.c and oper_32b.c files.

In saturating arithmetic when on addition the result overflows its corrected to the maximum possible positive signed number, while on subtraction the value is corrected to the most negative signed number. While in Q15 and Q31 format the N bit signed value is treated to have a binary point following the sign bit, where N is 16 and 32 bits respectively.

These saturating arithmetic instructions are not part of the ANSI C and thus their implementation in C assembles into large assembly code. But due to their frequent use many DSP processors implement them in hardware. The DSP extensions of ARM926EJ-S also provide the same. Thus the C code was modified and the operations written in assembly incorporating these instructions. The improvements provided by this are listed in the Table 3.

Exploiting processor specific instructions Along with the above DSP instructions the ARM926EJ-S processor provides a single cycle instruction, to return the leading number of zeros in a specified register, named CLZ. This instruction was found to be beneficial for normalization operations, and was found to reduce the code size, and improve speed drastically for left shift operations(*L_shl*), see Table 3. For *norm_l*() function the use of CLZ instead of the for loop as shown in Figure 7 gave rise to a reduction of 16 cycles per call leading to a reduction of 1625 cycles per frame.

Multiple load operation The ARM926EJ-S can load a register with four contiguous bytes from memory. This would reduce the load instructions by half by loading two consecutive two byte data elements from the memory in one load operation. This was not exploited as the data bytes are not correctly aligned if the address is not a multiple of four. This leads to instruction overhead for loops iterating an odd number of times and for data byte alignment and defeats the purpose.

3.3.8 Performance

The Table 4 lists the cycle count and MCPS (see Measurement Techniques)before and after performing optimization in this category.

Version	*Cycle count*	*MCPS*	*% Impr.*
Initial version	42419339	4242	0
Optimized version	39025791	3898	8.1

Table 4. Performance after global optimizations

3.4 Function-level optimization

The optimization techniques presented here can be performed without a global knowledge of the algorithms and without a detailed analysis of data and control flow. The general approach for optimizing each function includes:

1. Establishing the function interface and separating the function from the rest of the code to facilitate analysis.
2. Adding test code that saves the function input and output values before and after each function call.
3. Optimizing the function and verifying the output to ensure that it remains the same as the corresponding reference output.

Function name	*Total cycles*	*Reduction*	*% Reduction*
Original	42422412	0	0.00
L_add	36409140	6013272	14.17
L_sub	42040012	382400	0.90
L_mac	32697692	9724720	22.92
L_shl	41068443	1353969	3.19
sature	42224987	197425	0.47
Total	30774723	11647689	27.45

Table 3. Reduction in cycle count for a 30 frame testvector on G729E code observed after re-writing each of the functions in assembly.

4. Integrating the optimized function with the rest of the code and verify that it passes the ITU-T test vectors.

3.4.1 Criterion for selecting functions

The selection of functions for optimization was based primarily on profiling information (Appendix), focusing on the most time-critical functions. All the functions which consumed more the 5% of the processor cycles while executing were short listed. This set of functions was called the Z3 set. The main criteria used to select the functions to be optimized was the following information.

Number of calls per frame. Based on this information, the decision is made whether a small, frequently-called function should be inlined. This information is most useful during the global-level optimizations.

Total number of cycles per frame. This is the most useful information for selecting the functions to optimize.

3.4.2 Loop unrolling

This technique repeatedly instantiates the loop body with corresponding indices. Loop unrolling proves quite rewarding in some cases where it gives up to 2% improvement in terms of total number of cycles for each of its applications. An example is illustrated in Figure 8.

```
1  /* original code */
2  for(i=MP1; i <M_BWDP1; i++)
3   {
4    prev_filter[i] = 0;
5   }
6
7  /* modified code */
8  prev_filter[MP1] = 0;
9         |
10        |
11 prev_filter[M_BWDP1-1] = 0;
```

Figure 8. Loop unrolling example.

3.4.3 Loop merging

Combining two or more loops into a single loop loads the ALU more efficiently, as illustrated in Figure 9. This technique yields up to 1% improvement in terms of total number of cycles for each of its applications.

```
1  /* original loops */
2  for(i=0; i<L_WINDOW; i++)
3  {
4  y[i] = mult_r(x[i], hamwindow[i]);
5  }
6  for(i=0; i<L_WINDOW; i++)
7  {
8  sum = L_mac(y[i], y[i]);
9  }
10
11 /* merged loops */
12 for(i=0; i<L_WINDOW; i++)
13 {
14 y[i] = mult_r(x[i], hamwindow[i]);
15 sum = L_mac(y[i], y[i]);
16 }
```

Figure 9. Loop merging example.

3.4.4 Loop count reduction

Loop count reduction is reducing the number of times a loop is run. The ARM926EJ-S has two saturating arithmetic functional units, thus in vector operations in each loop two

```
/* using local variables
inside for loop */
for(j=1; j<i; j++)
 {
  Word32 t0;
  t0 = Mpy_32_new(K, A[i-j]);
  An[j] = L_add(t0, A[j]);
 }
```

Figure 10. Use of local variable example.

```
/* original shift */
alp = L_shl(t0, 1);

/* modified shift */
alp = t0 << 1;
```

Figure 11. Left shift example.

saturating arithmetic operations can be executed in parallel for optimum processor utilization, thereby reducing the number of loop counts by half. This was optimally used with the function *L_macD*() performing two long mac operations in one call (Figure 6).

3.4.5 Declaring local variables

Variables are declared as close as possible, in the code hierarchy, to their area of use (in C) to help the compiler identify their life cycles. This improves register allocation but requires more stack memory. An example is shown in the Figure 10. This was not probed further.

3.4.6 Replacing functions by arithmetic operators

The right shift operator $>>$ is used in a variable shift displacement to save a function call *L_shr*(). Similarly, the left shift operator $<<$ is used instead of the *L_shl*() function to save a function call if overflow or underflow does not occur after a shift operation. An example is shown in the Figure 11. Also some additions or subtractions where add() or sub() functions were used were replaced by addition operator + or subtraction operator - after analyzing the code . An example is shown in the Figure 12.

However, the C operators $*$, $+$, $-$ were not applied to operations on fractional values that only employ intrinsic functions (*L_mult*(), *L_add*()) because it is difficult to ascertain whether overflow, underflow or saturation would occur or not.

```
/* Original addition and subtraction */
alp_exp = add(alp_exp, temp);
i = sub(i,1);

/* modified addition and subtraction*/
alp_exp = alp_exp + temp;
i--;
```

Figure 12. Addition and Subtraction example.

```
/* original if construct */
if (sub(abs_s(K>>16), 32750)){

/* modified if construct */
if (abs_s(K>>16) > 32750){
```

Figure 13. if condition example.

3.4.7 Simpler test conditions

The test conditions were simplified to reduce the amount of computation. e.g. the conditions which involved subtraction were replaced with direct comparisons. This saves a subtraction operation. An example is shown in the Figure 13.

3.4.8 Removal of Pointers

Impact of pointers on code speedup was probed. Almost all occurrences of pointers were replaced with 'non-pointer' implementations. Upon simulation on the *ARMulator*, the modified code took greater number of execution cycles compared to the original code.

Subsequently, the C and assembly codes were analysed and the following example illustrates the reason. Let $h[8]$ be an eight element array of short data type which is 2 bytes in size and p be a pointer to this array. To perform the following multiplication operation inside a *for* loop,

$$Result = h[i] * h[i]$$

all variables and array addresses in a function are pushed on the stack pointer (SP). Figure 14 shows the code with the pointer implementation and Figure 15 shows without.

So, the implementation without pointers adds extra overhead of assembly instructions for moving the content of SP to a register, a left shift operation to double the array index and an addition operation. Although we need a *Mov* operation for the original implementation also, it would be outside the *for* loop so it gets executed only once while in case of the modified code it has to be performed in each iteration.

```
C code                   Assembly

p = h;                   Mov r1, sp
                         ;which means
                         the array
                         address from
                         the SP is
                         being moved
                         to reg. r1.


Res = (*p)*(*p);         LDRH r2,[r1,0]
/* this construct
is inside a for          ;loads reg.
loop */                  r2 with the
                         data at the
                         address stored
                         at r1.

                         MUL r3, r2, r2
                         ;performs the
                         multiplication.
```

Figure 14. C and Assembly version for code with pointers.

This is diagrammatically explained in Figure 16. The left hand side of the figure shows the data access through a pointer which involves reading the address of the memory location and then accessing it. While the right hand side of the figure shows the computations required to be performed on the address of the memory location. The double of the array index has to be added to the address of the memory location. This modification was abandoned as it was found to be counter productive.

3.4.9 Sign-extending/Zero-extending

Using the most appropriate data type for variables is important, as it can reduce code and/or data size and increase performance considerably.

ARM926EJ-S processor has a 32-bit wordsize. So, wherever the design permits, it is best to avoid using short (Word16) as local variables. For these, the compiler needs to reduce the size of the local variable to 16-bits after each assignment. This is called sign-extending for signed variables and zero-extending for unsigned variables. It is implemented by shifting the register left by 16-bits, followed by a signed or unsigned shift right by the same amount, taking two instructions (zero-extension of an unsigned char takes one instruction). These shifts can be avoided by using int

```

C code              Assembly

Res = h[i]*h[i];    ADD r2, sp, r1, LSL #1
                    ;moves the address
                    stored at SP to
                    reg. r2, adds it
                    to twice the array
                    index corresponding
                    to the element being
                    accessed and finally
                    stores the result in
                    reg. r2. The index
                    is doubled since
                    its data type
                    occupies 2 bytes,
                    so each element
                    corresponds to 2 bytes.

                    LDRH r2,[r2, 0]
                    ;loads reg. r2 with
                    the data at the address
                    stored at r2.

                    MUL r3, r2, r2
                    ;performs the
                    multiplication.
```

Figure 15. C and Assembly version for code without pointers.

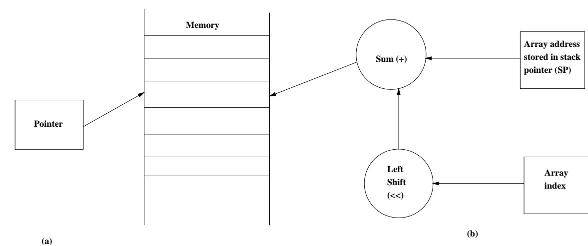

Figure 16. Array access for implementation (a)with pointers (b) without pointers.

```
/* C code for adding
1 to a 32-bit quantity */
int word32 (int a)
 {
  return a + 1;
 }

;Corresponding Assembly
;code (compiler) generated
ADD r1,r1,#1
MOV pc,lr
```

Figure 17. C and Assembly code for operation on a 32-bit quantity.

```
/* C code for adding 1
to a 16-bit quantity */
short word16 (short a)
{
return a + 1;
}

;Corresponding assembly
;code (compiler generated)
ADD r1,r1,#1
MOV r1,r1,LSL #16
MOV r1,r1,ASR #16
MOV pc,lr
```

Figure 18. C and Assembly code for operation on a 16-bit quantity.

and unsigned int for local variables. This is essentially important for calculations which first load data into local variables and then process the data inside them. Even if data is input and output as 16-bit quantities, it leads to better performance processing them as 32-bit quantities as can be seen from the code fragments shown. Figure 17 shows the C and assembly code for adding 1 to a 32-bit number (int type). Figure 18 shows the C and assembly codes for adding 1 to a 16-bit number (short int type). The assembly code is shortest for the 32-bit operation.

So far this is applied to some functions and the improvement in terms of cycles is considerable (by 2.2% , in codebook) in terms of number of execution cycles.

Version	*Cycle count*	*MCPS*	*% Impr.*
Initial version	42419339	4242	0
Optimized version	37753211	3766	11.2

Table 5. Performance after function level optimizations

3.4.10 Selected functions for function level optimizations

The Z3 group of selected functions was extended to include $Syn_filt()$, $Convolve()$, $Residue()$, $Cor_h_X()$, $Autocorr()$, and $Norm_Corr()$. As they are smaller and easier to manipulate. However, the optimizations were further extended to functions $Cor_h()$, $Lsp_pre_select()$, $Levinsone()$, $D4i40_17()$, $Pre_Process()$, $Az_lsp()$, $Chebps_10()$, $Chebps_11()$, $Lag_max()$, $Pred_lt_3()$, $Get_lsp_pol()$, $Qua_gain()$. There are still functions left to which these optimizations are to be applied.

3.4.11 Performance

The Table 5 lists the cycle count and MCPS (see Measurement techniques) before and after performing optimization. Comparing with Table 4, it is observed that optimizations under this category provided 3.1% improvement.

3.5 Algorithmic changes

It is worthwhile to determine if frequently-accessed data structures could be changed so that the functions access data sequentially and lengths of data arrays are multiples of four(thereby filling the complete 32-bits of a register). It is also useful to examine the relationship between the results of an algorithmic module and its internally computed values. Often the number of variables a module computes and stores is significantly larger than the number of variables returned and need to be probed for reduction.

For example, the output of a function might be the offset of a single value, while at the same time numerous intermediate internal variables might have to be computed and stored internally to obtain it. Examining the relationships of the output and the internal variables allows to determine if the output could be obtained with lesser computations and/or with fewer internal variables. However while performing this analysis special emphasis was given on maintaining bit-exactness with the standard specification. This is particularly important for cases where the sequence of operations is changed from that of the original reference code.

3.5.1 Identifying algorithms to change

Precise determination of which algorithms to optimize is difficult. It involves a further search from the Z3 set. The following two guidelines reduce the scope of this search to a manageable range:

1. Using profiling data to select only the most time-critical functions from the Z3 set.
2. Including the functions that provide inputs to and receive outputs from the functions in Z3 set.

Although the functions in step 2 are usually small and not included in the Z3 set, they are usually easily modified so that the data they provide or receive from the functions selected in step 1 is accessed and manipulated more efficiently. The three most time-consuming modules, which together accounted for more than 50% of the total execution time.

Ideally, algorithmic modifications should have been taken up before performing function-level optimizations. This could have saved significant time in this iterative optimization process. However, this is possible only if the improvement in execution time after function-level optimization or assembly implementation could be predicted fairly accurately, which cannot be done without an in depth understanding of algorithms and optimization techniques. Hence, after the algorithmic modifications are done and the final C implementation passed the test vectors, another round of function-level optimizations have to be taken up. The result yields a faster C implementation and also serves as an excellent reference code for the assembly implementation.

3.5.2 Platform-independent changes

Platform-independent changes to algorithms are improvements in code which apply to all processor types and C compilers. The following are general guidelines which were followed:

1. Replacing the time-critical operations div() and log() with multiplications.
2. Removing repeated computations of the same value.
3. Reordering computations to avoid repeated fetches of the same value.
4. Reducing the number of test conditions (e.g. *if* statements)

3.5.3 Platform-dependent changes

Platform-dependent changes to algorithms essentially reorder and restructure data. They also reorder and regroup computation blocks to take advantage of the parallel architecture of a particular processor. H. They are as follows:

Version	*Cycle count*	*MCPS*	*% Improvement*
Initial version	42419339	4242	0
Optimized version	35208051	3510	17.26

Table 6. Performance after algorithmic optimizations

1. Data structure addressing is preferably done sequentially (linearly), using indices rather than pointers.
2. DPF formats are translated to native 32-bit representation wherever possible.
3. Sequential, identical and related computations are grouped together.

3.5.4 Function selection for algorithmic modifications

Algorithmic modifications were made after performing the function-level optimizations. The initial selected function set for the algorithmic changes focusing on modules that grouped several functions, is as follows:

1. $D4i40_17() + Cor_h() + Cor_h_X()$
2. $Lag_max() + Pitch_ol()$
3. $Norm_Corr() + Pitch_fr3()$
4. $Autocorr() + Lag_window() + Levinsone()$
5. $Az_lsp() + Chebps_10() + Chebps_11()$
6. $cor_h_e() + cor_h_vec() + search_ixiy()$

Changes have been made to above 1 through 4. The results are quite rewarding considering the improvements we can obtain for an essentially non-parallel architecture. The above list of functions can be used to demonstrate the given optimizations and can be further applied on other function groups.

3.5.5 Performance

The Table 6 lists the cycle count and MCPS (Measurement techniques before and after performing algorithmic modifications on some functions. Comparing with Table 5, it is observed that optimizations under this category provided 6.06% improvement.

3.6 Assembly implementation of functions

In general, rewriting C functions in assembly increased speed and reduced code size by implementing optimizations overlooked by the compiler. However, the *armcc* compiler is quite efficient, hence most code was kept in C with specific functions re-written in assembly. While the optimization techniques enshrined in this report can be used on the

Version	*Cycle count*	*MCPS*	*% Impr.*
Initial version	42419339	4242	0
Optimized version	24603216	2452	42.19

Table 7. Performance after mixed C assembly optimizations

remaining C code thereby giving further improvement. This was left for future work due to time limitations. The functions to be implemented in assembly were selected on the basis of profiling information, as discussed as follows.

3.6.1 Selecting functions

The entire code is profiled, and the most time-critical functions are identified for the Z3 set (as discussed earlier). The estimates of ideal execution times are compared with the C optimization results. Functions which are near-optimal are retained in the optimized C version and the remainder are candidates to be implemented in assembly.

3.6.2 Implementation approaches

There are two basic approaches to implementing assembly.

Modifying compiler output This approach is most useful for relatively simple functions for which the compiled code is close to the optimal version. For instance, registers can be more optimally allocated,or the number of pointers needed to fetch data can be reduced.

Coding directly in assembly The functions that are complex or that could use better assembly instructions were written using this approach. The optimized C code is used as a reference for testing to ensure the bit-exactness after performing platform-dependent optimizations. In such cases, assembly implementation is based on, another C version or standard recommended algorithm. However, it is best to use the C code as a reference, regardless of other optimizations employed.

3.6.3 Performance

The Table 7 lists the cycle count and MCPS before and after writing some functions in assembly. Comparing with Table 6, it is observed that optimizations under this category provided 24.93% improvement.

4 Measurement techniques

Various tools and techniques are used to measure the execution time, code size and data size of the code.

Cache options	*Cyls*	*Cyls/fr*	*mS/fr*
I and D caches off	17143184	571439.47	95.24
I cache On, D cache Off	12380106	412670.20	68.78
I cache Off, D cache On	13431503	447716.77	74.62
I and D caches On	9171550	305718.33	50.95

Table 8. Execution statistics for a 30 frame test-vector set running on a Non-optimized G729 code at 6Mhz

4.1 Execution time

The execution time of functions were evaluated using the number of simulated cycles spent in the functions. This measurement is a good approximation of actual processing time. Four tools were used to gather execution time information: the ARM compiler (*armcc*), the ARM ISS (*ARMulator*), and the ADS Debuggers (*AXD* and *armsd*) and the profiler *armprof*.

The profiler provides information on the average time spent in each function as a percentage of the total number of cycles. The profiler provides the overall number of cycles spent in each function with and without descendants (in absolute and percentage values) as well as the number of times the function was called. The profiler was used to determine which functions are the most time-critical and where to direct further optimization efforts. It reports the number of cycles used to execute each function.

The metric used for optimization measurements was million cycles per second (MCPS). The number of MCPS required to encode a frame is obtained by multiplying the measured number of cycles by the number of frames to be processed per second (for G.729E, 100 frames of 10 ms each have to be processed per second), and dividing the result by 1,000,000. For instance, if it takes 42,420,000 cycles to encode a frame, the processing power required is (42,420,000 100) / 1,000,000 = 4242 MCPS.

4.2 ARMulator Benchmarking

As the algorithm needs to be executed with real time constraints it was imperative to calibrate the *ARMulator* to real time. Thus, the ARMULATOR was calibrated against an Aptix FPGA board with Instruction and Data caches. The Tables 8, 9, provide the statistics for the same.

The cycle count provided by the *ARMULATOR* needs to be scaled by a factor of 0.222 to provide the correct cycle count including instruction and data caches on an Aptix FPGA implementaton of the processor, as shown in Table 10.

Cache options	*Cyls*	*Cyls/fr*	*mS/fr)*
I and D caches off	12929209	430973.63	71.83
I cache On, D cache Off	9756321	325210.70	54.20
I Off cache, D cache On	9700514	323350.46	53.89
I and D caches On	6961778	232059.27	38.68

Table 9. Execution statistics for a 30 frame test-vector set running on an optimized G729 code at 6 Mhz

G.729E version	*Armulator cyls*	*Emulator cyls*	*Ratio*
Original	42419339	9171550	0.216
Optimized	30590320	6961778	0.22
Average	36504829.5	8066664	0.222

Table 10. Scaling factor for the ARMULATOR.

5 Results

The following section presents the results of porting the G.729E code to the ARM926EJ-S processor. The primary performance data (MCPS) is presented in comparison to the effort applied to achieve that performance. The major milestones shown on the graph include code versions after each of the following steps:

1. Initial porting to the ARM926EJ-S.
2. Global-level optimization, including inlining of DPF functions.
3. Initial function-level optimization, before algorithmic changes.
4. The final C version, after algorithmic changes and re-optimization at function level.
5. The final mixed implementation, including selected functions in assembly.

5.1 Execution time

The percentage of improvement in execution speed through the different versions of the project are summarized in the Figure 19. While the distribution of man-months taken over the various optimization levels is illustrated in Figure 20.

The global-level optimizations, especially inlining small DPF functions, proved to be beneficial for both execution time and code size. They reduce the time by 8.2% (344 MCPS). The code size also decreased slightly, primarily due to the removal of operations performed before and after

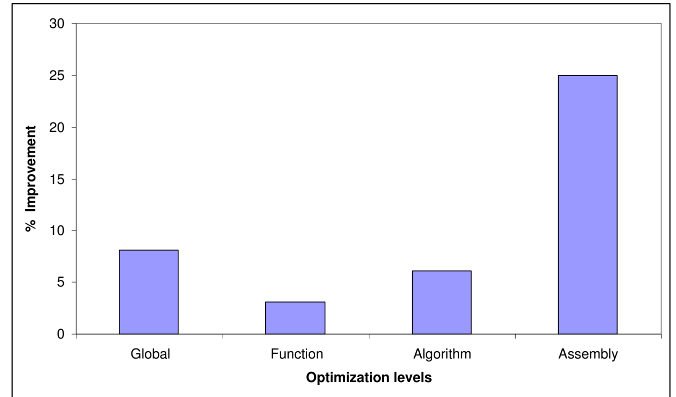

Figure 19. Percentage improvement in execution speed

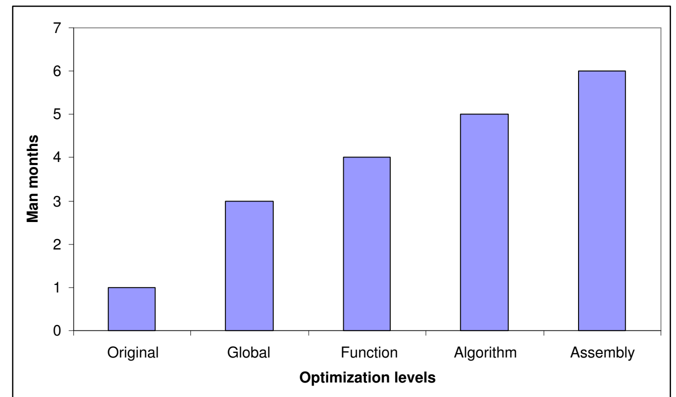

Figure 20. Man months taken per optimization level

function calls which were now redundant. The function-level C optimizations reduce execution time by an additional 3.1% (132 MCPS). Algorithmic changes reduce execution time by another 6.06% (256 MCPS). The phases after project-level optimization included some redundant work because some time-critical functions were optimized both before and after algorithmic changes. After the mixed C and assembly implementation the time reduces by further 25% (1058 MCPS). The final execution time of 2452 MCPS is substantially better than our original target of 4242 MCPS which is an improvement of approximately 42%. This suggests that with one more iteration of the optimization process an improvement of up to 50% can be achieved.

6 Optimization examples

This section presents details of the optimization process for the functions, $L_shl()$, and $Norm_Corr()$. The function-level, algorithmic, and assembly changes for each function are discussed, as well as the effect of these changes on the efficiency of code generated by the compiler.

6.1 Optimizations in $L_shl()$

The $L_shl()$ function arithmetically shifts the 32-bit input argument left by positions provided by the second 16-bit argument. If second argument is negative, it calls the right shift function and right shifts the 32bit argument. Finally, it saturates the result in case of underflows or overflows.

The main steps in the reference C code are:

1. Checking if the 16-bit argument is negative.
2. Calling the right shift function if above test returns true.
3. Multiplying the 32-bit argument by 2.
4. Checking for overflows each time in the above iteration.

6.1.1 Function-Level Optimizations

The function calls function L_shr() if the number of shift positions specified are negative. This check operation takes eight instructions. While in the static code the L_shl instruction is called 74 times where 41 instances have hard-coded positive shift arguments. Thus by defining a second shift function L_shlf() which does not perform sign check for calling L_shr() function reduces eight redundant instructions per L_shl() call.

Using L_shlf: Therefore all L_shl calls with hard-coded positive shift arguments were replaced by the new function L_shlf.

Version	*Cycle count*	*MCPS*	*% Impr.*
Initial version	42419339	4242	0
Optimized version	40934662	4094	3.4

Table 11. Performance after modifying $L_Shl()$

6.1.2 Algorithmic Changes in assembly

1. The ARM926EJ-S provides an instruction, CLZ to return the number of leading zeros in a given register data. This instruction was utilized to return the number of zeros or ones for a positive and negative number respectively in the given register.
2. The above returned value was compared with the number of shifts required, and if found greater, the overflow flag was set.

6.1.3 Performance

The Table 11 lists the $L_Shl()$ cycle count and MCPS before and after optimization.

6.2 Optimizations in $Norm_Corr()$

The $Norm_Corr()$ function finds the normalized correlation (correlation divided by the square root of the energy of filtered excitation) between the target vector and the filtered past excitation. The main steps of this function in the reference C code are as follows:

1. Computing the filtered excitation for the minimum delay.
2. Scaling the excitation vector to avoid overflow.
3. Computing the energy of the filtered excitation to check overflow.
4. Scaling the filtered excitation to avoid overflow and computing the energy of the scaled filtered excitation.
5. Computing the normalized correlation vector For every possible delay between minimum and maximum and modifying the excitation for the next iteration.

6.2.1 Function-Level Optimizations

The C optimizations applied to $Norm_Corr()$ to speed up the function include the following:

1. Replacing the tests that use subtraction with direct comparisons. for example, replace if $(sub(a,b) > 0)$ with if $(a > b)$.

2. Replacing functions calls with operators when integer values are used. for example, replacing $i = sub(i, 1)$ with $i--$.

3. Replacing the $L_shl()$ function call with the $<<$ operator.

4. Replacing unmodified variables with constants defined in the G.729 reference code.

5. Inlining of DPF functions

6.2.2 Algorithmic Changes

The major modifications to the algorithms in $Norm_Corr()$ included the following:

1. Computation of the scaled filtered excitation vector only when overflow occurs. In the ITU-T speech.in test vector, overflow occurs in only about 200 out of 3750 frames.

2. Exploiting the 32-bit capabilities of the processor by using 32-bit variables instead of the DPF format defined in the G.729 standard, and replacing the $Mpy_32()$ function with a new function $Mpy_32_new()$ which works with native 32-bit data but preserves bit-exactness. The code listing is shown in the appendix.

3. Eliminating the else branch of the if()else statement to reduce the number of branch-like instructions.

4. Computing the energy in the same loop which computes the new scaled filtered excitation values to avoid extra memory moves. However this modification did not work as intended due to suboptimal register utilization.

These modifications were first applied to the un-optimized C code to verify bit-exactness. The function was then re-optimized in C.

6.2.3 Assembly implementation

Assembly implementation was not used in this case except for the arithmetic functions which were implemented using saturating arithmetic instructions.

6.2.4 Performance

The Table 12 lists the $Norm_Corr()$ cycle count and MCPS before and after optimization.

Version	*Cycle count*	*MCPS*	*% Impr.*
Initial version	42419339	4242	0
Optimized version	41146758	4115	3

Table 12. Performance after modifying $Norm_Corr()$

7 Conclusion

This report describes porting G.729E C code to the ARM926EJ-S platform and optimizing the code while maintaining the bit-exactness of the original code. The approach used and recommended has the following key components:

1. C source optimizations for selected functions, with and without algorithmic changes.

2. Assembly implementation/optimization of selected time-critical functions.

The C optimization of selected functions was done at both the global and function level. These optimizations were algorithm independent. Execution time profile, provided by the profiler, was used for selecting functions for optimization. The global-level optimizations involve profiling the initially ported code, inlining frequently-called functions, and optimizing the C code so that the compiler produces code that is better adapted to the ARM926EJ-S architecture. In the function-level optimization, several techniques including loop unrolling and loop merging were employed. These two optimizations lead to an improvement by 11.2 % (476 MCPS). The total development time for these two phases of the project was 4 man-months.

Further run-time reduction is achieved by applying algorithmic changes to critical functions grouped in modules. This involves a comprehensive understanding of the algorithms. Four such modules were chosen for optimizations. The profiler generated information assisted in identifying functions for implementing algorithmic changes. The algorithm-modified functions were then re-optimized at the function level. The functions initially selected were termed Z3 set and collectively took more than 50% of total execution time of the optimized C version. This Z3 set of functions was expanded to include functions that provide inputs to and receive outputs from the functions in the it. The algorithmic changes employed included both architecture-independent modifications and ARM926EJ-S specific adaptations. Basic algorithmic changes included modifying vector sizes and internal pointer offsets to multiples of four, sequential array addressing, searches with interval division by four, and use of native 32-bit data format. Algorithmic changes reduced run time from 3766 MCPS to 3510 MCPS and improved by 6.06%. The development time for this phase was 4 man-months. These improvements were made

after the initial C optimization, due to which some functions had to be optimized again at function level.

Assembly implementation was the final phase of development because it is very difficult to change the structure of an implementation in assembly, especially after algorithmic changes. Functions targeted for assembly implementation are primarily those for which the *armcc* compiler could not produce efficient code, and those which execute for large number of cycles. Particular emphasis was given to loops, where each cycle saved results in a significant reduction in execution time. This final phase involved the implementation of selected critical functions in assembly. The optimized C implementation served as a reference for the assembly implementation. It was used to verify that the intended modifications comply with bit-exactness requirements and it also provide a pseudo-code for the algorithm. The criteria for selecting functions for this phase was primarily based on the difference between the estimated performance and actual performance of the compiler generated code for each function. An improvement by 25% (1058 MCPS) was obtained, with a development time for this phase of 6 man-months.

The *ARMulator* was bench-marked with Aptix FPGA implementation of ARM926EJ-S by executing the code on it and a correction factor of 0.222 was obtained for determining accurate number of cycles from the *ARMulator* results.

With the present effort the code has speeded up by approximately 42% and the steps for the next optimization iteration are defined. It is predicted, with these optimizations applied on the complete code can speed up by 10% more.

8 Future research

Many optimizations on similar algorithms were surveyed and it was found that further optimizations could be explored for the given system. Till now the emphasis of optimization has been on the algorithm and the implementing C code.

Algorithmic optimizations: The present algorithmic optimizations provided about 7% improvement and further optimizations upto 10% more seem feasible.

Code size and Memory usage: The code size and memory utilization needs to be be explored. Using the correct sizes of caches and TCM can improve or nullify the code speed-up gained above. In the course of the project, functions which could be reduced, collapse d or contained redundant code, and data look up tables which could be modified, were marked. The authors believe these reductions can be made without significant penalty on the execution speed.

Harvard architecture: Optimizations utilizing the internal Harvard architecture of ARM926EJ-S could also be explored.

Multiple channels: Code modification / optimization to support multiple channels is not yet implemented.

The complete code speedup can be ascertained after implementing the optimizations elaborated here on the complete code. While the code could be further speeded up if the constraint of not using customized hardware is removed or by using a full DSP processor.

9 Acknowledgements

The authors are thankful to Mohammed Ben Romadhane and Sveirrir Olafsson of Conexant Systems Inc. for having funded this project. A special thanks to Ramanand Mandayam for the overall coordination and support with the ARM tools. They strongly acknowledge the help and support extended by Steve Barrett in setting up the code on the FPGA boards. And most importantly they thank David Braun for sharing his valuable time, expertise and ARM libraries. And, lastly they acknowledge the help and support extended by Robert Larsen of CECS in reviewing this report.

References

[ITU-T P.810 96] ITU-T P.810 Recommendation. *Modulated noise reference unit (MNRU) (1996).*

[Steve] Steve Furber, *ARM System Architecture, ARM Co., 1996*

[Target Guide] Debug Target Guide, ARM Developer Suite version 1.2

[ITU-T G.729 96] ITU-T G.729 Recommendation. *Coding of speech at 8 kbit/s using Conjugate-Structure Algebraic-Code-Excited Linear-Prediction(CS-ACELP) (1996).*

[ITU-T G.729E 98] ITU-T G.729E Recommendation. *Coding of speech at 8 kbit/s using Conjugate-Structure Algebraic-Code-Excited Linear-Prediction(CS-ACELP): Annexe E 11.8 kbit/s CS-ACELP speech coding algorithm (1998).*

[Mot. Vocoder] A. Gerstlauer, S.Zhao, D.Gajski, A. Horak, *Design of a GSM Vocoder using SpecC Methodology", UCI, Technical Report, 1999.*

[SpecC Book] D. Gajski, F. Vahid, S. Narayan, J. Gong, *Specification and Design of Embedded Systems, Prentice Hall, Englewood Cliffs, New Jersey, 1994.*

[ARM DSP] Hedley Francis, ARM DSP-Enhanced Extensions, 2001

[APP NOTE] Writing Efficient C for ARM, Application notes 34, 1998.

[Tech Ref] ARM926EJ-S (Rev 0), Technical Reference Manual, 2002.

A Appendix: Initial profile

```
Name                            cum%     self%    desc%     calls
__heap_extend                  96.06%    0.00%   96.06%         1
  __rt_heap_extend                       0.00%    0.00%         1
  __rt_stackheap_init                    0.00%    0.00%         1
  __rt_lib_init                          0.00%    0.00%         1
  exit                                   0.00%    0.00%         1
  main                                   0.00%   96.06%         1
----------------------------------------------------------------------
main                           96.06%    0.00%   96.06%         1
  fwrite                                 0.00%    0.00%        30
  fread                                  0.00%    0.00%        31
  fopen                                  0.00%    0.00%         3
  atoi                                   0.00%    0.00%         1
  _printf                                0.00%    0.04%        31
  _printf                                0.00%    0.01%        15
  Pre_Process                            0.12%    0.85%        30
  Init_Pre_Process                       0.00%    0.00%         1
  prm2bits_ld8e                          0.04%    0.00%        30
  Coder_ld8e                             0.50%   94.44%        30
  Init_Coder_ld8e                        0.00%    0.01%         1
----------------------------------------------------------------------
__argv_alloc                   96.06%    0.00%   96.06%         0
  __heap_extend                          0.00%   96.06%         1
----------------------------------------------------------------------
Coder_ld8e                     94.95%    0.50%   94.44%        30
  Copy                                   0.02%    0.00%       120
  Copy                                   0.04%    0.00%       180
  Enc_lag3                               0.00%    0.00%        60
  G_pitch                                0.07%    0.12%        60
  Pitch_fr3                              0.02%    7.92%        60
  Pitch_ol                               0.12%    9.77%        30
  perc_var                               0.02%    0.06%        30
  Qua_gain                               0.31%    1.14%        60
  Parity_Pitch                           0.01%    0.00%        30
  Pred_lt_3                              0.34%    1.28%        60
  Int_lpc                                0.00%    0.44%        29
  Int_qlpc                               0.01%    0.23%        29
  Weight_Az                              0.04%    0.67%       270
  L_shl                                  0.23%    0.00%      6000
  L_shl                                  0.18%    0.00%      4800
  L_mac                                  0.01%    0.03%      2400
  round                                  0.22%    0.01%      2400
  extract_h                              0.00%    0.00%      3600
  extract_h                              0.00%    0.00%      4800
  negate                                 0.13%    0.00%       180
  L_mult                                 0.03%    0.00%      6000
  L_mult                                 0.03%    0.00%      4800
  mult                                   0.00%    0.00%       747
  mult                                   0.00%    0.00%       787
  shl                                    0.00%    0.00%        60
  shr                                    0.00%    0.00%        30
  sub                                    0.04%    0.03%      3688
  sub                                    0.05%    0.04%      4892
```

```
  add                                       0.00%    0.01%     2603
  add                                       0.00%    0.00%      817
  Az_lsp                                    0.08%    4.07%       30
  Levinsone                                 0.84%    6.67%       60
  Lag_window                                0.00%    0.00%       30
  Autocorr                                  1.00%    1.74%       30
  Residue                                   1.58%    3.81%      180
  Syn_filte                                 0.32%    1.16%       60
  Syn_filte                                 1.64%    5.84%      300
  Convolve                                  0.52%    1.08%       60
  ACELP_10i40_35bits                        0.46%   23.43%       60
  Lag_window_bwd                            0.02%    0.19%       30
  autocorr_hyb_window                       1.31%    2.34%       30
  set_lpc_mode                              0.00%    5.33%       30
  Lsp_prev_update                           0.00%    0.02%       30
  Qua_lspe                                  0.00%    6.74%       30
  test_err                                  0.00%    0.00%       60
  Corr_xy2                                  0.12%    0.22%       60
  update_exc_err                            0.00%    0.00%       60
----------------------------------------------------------------------
L_mac                             31.81%   11.73%   20.07%   1559356
  L_add                                     9.96%    0.00%   1469754
  L_mult                                   10.11%    0.00%   1534020
----------------------------------------------------------------------
ACELP_10i40_35bits                23.90%    0.46%   23.43%       60
  L_msu                                     0.00%    0.00%      180
  L_sub                                     0.00%    0.00%      720
  L_add                                     0.00%    0.00%      300
  L_mult                                    0.00%    0.00%      180
  mult                                      0.00%    0.00%      540
  shl                                       0.00%    0.00%      180
  add                                       0.11%    0.53%    58320
  cor_h_x_e                                 0.68%    1.35%       60
  set_sign                                  0.18%    0.89%       60
  cor_h_e                                   0.73%    2.78%       60
  cor_h_vec                                 3.05%    6.19%     1440
  search_ixiy                               2.04%    4.23%      720
  build_code                                0.35%    0.25%       60
----------------------------------------------------------------------
L_mult                            13.03%   13.03%    0.00%   1977254
----------------------------------------------------------------------
L_add                             10.65%   10.65%    0.00%   1571129
----------------------------------------------------------------------
Residue                            9.90%    2.90%    6.99%      330
  L_shl                                     0.56%    0.00%    14400
  L_mac                                     1.80%    3.08%   240000
  round                                     1.35%    0.09%    14400
  L_mult                                    0.09%    0.00%    14400
----------------------------------------------------------------------
Pitch_ol                           9.89%    0.12%    9.77%       30
  Lag_max                                   3.29%    6.26%       90
  L_sub                                     0.00%    0.00%       30
  L_mac                                     0.04%    0.08%     6690
```

```
  mult                                0.00%   0.00%       60
  shl                                 0.07%   0.00%     2736
  sub                                 0.00%   0.00%      120
----------------------------------------------------------------------
Lag_max                       9.56%   3.29%   6.26%       90
  Mpy_32                              0.00%   0.00%       90
  L_Extract                           0.00%   0.00%      180
  Inv_sqrt                            0.00%   0.01%       90
  L_sub                               0.05%   0.00%     3720
  L_mac                               2.28%   3.91%   304800
  extract_l                           0.00%   0.00%       90
----------------------------------------------------------------------
cor_h_vec                     9.24%   3.05%   6.19%     1440
  L_mac                               1.77%   3.03%   236152
  round                               1.03%   0.07%    10944
  mult                                0.06%   0.09%    10944
  add                                 0.01%   0.10%    11520
----------------------------------------------------------------------
Syn_filte                     8.99%   1.98%   7.01%      360
  L_shl                               0.56%   0.00%    14400
  L_msu                               1.70%   3.21%   144000
  round                               1.35%   0.09%    14400
  L_mult                              0.09%   0.00%    14400
----------------------------------------------------------------------
L_msu                         8.55%   2.96%   5.59%   250363
  L_sub                               4.04%   0.00%   250363
  L_mult                              1.55%   0.00%   235918
----------------------------------------------------------------------
Pitch_fr3                     7.94%   0.02%   7.92%       60
  Norm_Corr                           1.93%   5.89%       60
  Interpol_3                          0.02%   0.06%      280
  sub                                 0.00%   0.00%      901
  add                                 0.00%   0.00%       63
----------------------------------------------------------------------
Norm_Corr                     7.82%   1.93%   5.89%       60
  Mpy_32                              0.02%   0.07%      990
  L_Extract                           0.03%   0.08%     1980
  Inv_sqrt                            0.06%   0.19%      990
  L_shl                               1.46%   0.00%    37260
  L_sub                               0.00%   0.00%       60
  L_mac                               0.60%   1.04%    81600
  extract_h                           0.00%   0.00%    37260
  negate                              0.04%   0.00%       60
  L_mult                              0.23%   0.00%    36270
  shr                                 0.03%   0.00%     3330
  sub                                 0.01%   0.01%     1920
  add                                 0.06%   0.33%    36270
  Convolve                            0.52%   1.08%       60
----------------------------------------------------------------------
Levinsone                     7.52%   0.84%   6.67%       60
  Div_32                              0.13%   0.91%     1069
  Mpy_32                              0.79%   2.06%    26618
  L_Comp                              0.38%   0.30%    15426
```

```
  L_Extract                          0.26%    0.75%    16499
  norm_l                             0.04%    0.00%     1064
  L_abs                              0.00%    0.00%     2133
  L_shl                              0.15%    0.00%     4132
  L_shr                              0.00%    0.00%     1064
  L_negate                           0.00%    0.00%      471
  L_sub                              0.01%    0.00%     1064
  round                              0.09%    0.00%     1050
  L_add                              0.17%    0.00%    25499
  abs_s                              0.59%    0.00%     1009
  sub                                0.00%    0.00%     1009
  add                                0.00%    0.00%     1004
----------------------------------------------------------------------
Qua_lspe                    6.74%    0.00%    6.74%       30
  Lsp_lsf2                           0.04%    0.08%       30
  Lsf_lsp2                           0.02%    0.00%       30
  Lsp_qua_cse                        0.00%    6.58%       30
----------------------------------------------------------------------
Lsp_qua_cse                 6.58%    0.00%    6.58%       30
  Get_wegt                           0.00%    0.03%       30
  Relspwede                          0.02%    6.52%       30
----------------------------------------------------------------------
Relspwede                   6.55%    0.02%    6.52%       30
  shl                                0.00%    0.00%       60
  add                                0.00%    0.00%      600
  Lsp_prev_extract                   0.06%    0.09%       60
  Lsp_expand_2                       0.01%    0.00%       60
  Lsp_expand_1                       0.00%    0.00%       60
  Lsp_get_quante                     0.01%    0.11%       30
  Lsp_expand_1_2                     0.01%    0.01%       60
  Lsp_pre_select                     1.34%    3.27%       60
  Lsp_select_1                       0.18%    0.55%       60
  Lsp_select_2                       0.27%    0.55%       60
  Lsp_get_tdist                      0.00%    0.02%       60
  Lsp_last_select                    0.00%    0.00%       30
----------------------------------------------------------------------
search_ixiy                 6.27%    2.04%    4.23%      720
  L_msu                              0.54%    1.02%    46080
  L_mult                             0.30%    0.00%    46080
  mult                               0.28%    0.41%    46080
  add                                0.28%    1.38%   151200
----------------------------------------------------------------------
sature                      6.06%    6.06%    0.00%   660726
----------------------------------------------------------------------
round                       5.86%    5.47%    0.39%    57896
  L_add                              0.39%    0.00%    58480
----------------------------------------------------------------------
set_lpc_mode                5.33%    0.00%    5.33%       30
  L_shr                              0.00%    0.00%       30
  L_mac                              0.00%    0.00%      300
  L_mult                             0.00%    0.00%       30
  shr                                0.00%    0.00%       30
  sub                                0.00%    0.00%      300
```

```
  add                                      0.00%    0.00%       30
  extract_l                                0.00%    0.00%       30
  Residue                                  1.31%    3.17%      150
  ener_dB                                  0.34%    0.39%      210
  Int_bwd                                  0.06%    0.04%       30
  tst_bwd_dominant                         0.00%    0.00%       28
  calc_stat                                0.00%    0.00%       28
--------------------------------------------------------------------
L_sub                             4.67%    4.67%    0.00%   289532
--------------------------------------------------------------------
Lsp_pre_select                    4.61%    1.34%    3.27%       60
  L_sub                                    0.12%    0.00%     7680
  L_mac                                    0.57%    0.98%    76800
  sub                                      0.89%    0.70%    76800
--------------------------------------------------------------------
L_shl                             4.45%    4.45%    0.00%   113258
  L_shr                                    0.00%    0.00%     1172
--------------------------------------------------------------------
Az_lsp                            4.16%    0.08%    4.07%       30
  div_s                                    0.06%    0.05%      299
  norm_s                                   0.00%    0.00%      299
  L_shr                                    0.00%    0.00%      598
  L_msu                                    0.00%    0.00%      150
  L_mac                                    0.00%    0.00%      150
  extract_h                                0.00%    0.00%      300
  negate                                   0.08%    0.00%      120
  L_mult                                   0.00%    0.00%     1798
  L_mult                                   0.00%    0.00%     1636
  L_mult                                   0.00%    0.00%      300
  shl                                      0.00%    0.00%      299
  shr                                      0.02%    0.00%     2400
  abs_s                                    0.17%    0.00%      299
  sub                                      0.00%    0.00%       30
  sub                                      0.01%    0.00%     1198
  sub                                      0.00%    0.00%      150
  add                                      0.00%    0.00%      290
  add                                      0.00%    0.00%     1200
  add                                      0.00%    0.01%     1636
  add                                      0.00%    0.00%      150
  extract_l                                0.00%    0.00%      598
  Chebps_11                                0.61%    2.99%     3166
--------------------------------------------------------------------
autocorr_hyb_window               3.65%    1.31%    2.34%       30
  Mpy_32_16                                0.01%    0.03%      930
  L_Extract                                0.01%    0.03%      930
  L_shl                                    0.03%    0.00%      930
  L_shr                                    0.01%    0.00%     1860
  L_mac                                    0.79%    1.37%   106950
  L_add                                    0.00%    0.00%      930
  mult                                     0.02%    0.03%     4350
--------------------------------------------------------------------
Chebps_11                         3.61%    0.61%    2.99%     3166
  Mpy_32_16                                0.20%    0.52%    12664
```

```
  L_Extract                                0.20%    0.57%    12664
  L_shl                                    0.49%    0.00%    12664
  L_msu                                    0.14%    0.27%    12664
  L_mac                                    0.20%    0.36%    28494
  extract_h                                0.00%    0.00%     3166
  L_mult                                   0.01%    0.00%     3166
----------------------------------------------------------------------
cor_h_e                           3.51%    0.73%    2.78%       60
  __rt_sdiv                                0.04%    0.00%     2400
  Inv_sqrt                                 0.00%    0.00%       60
  norm_l                                   0.00%    0.00%       60
  L_shl                                    0.00%    0.00%      120
  L_shr                                    0.00%    0.00%       60
  L_mac                                    0.17%    0.30%    24000
  extract_h                                0.00%    0.00%    21660
  negate                                   1.82%    0.00%     2400
  mult                                     0.12%    0.19%    21600
  shl                                      0.07%    0.00%     2400
  shr                                      0.02%    0.00%     2460
  add                                      0.00%    0.00%       60
----------------------------------------------------------------------
Convolve                          3.22%    1.04%    2.18%      120
  L_shl                                    0.18%    0.00%     4800
  L_mac                                    0.73%    1.26%    98400
  extract_h                                0.00%    0.00%     4800
----------------------------------------------------------------------
add                               3.22%    0.55%    2.66%   290811
  sature                                   2.66%    0.00%   291011
----------------------------------------------------------------------
Mpy_32                            3.22%    0.89%    2.32%    29967
  L_mac                                    0.44%    0.76%    59934
  L_mult                                   0.19%    0.00%    29967
  mult                                     0.36%    0.54%    59934
----------------------------------------------------------------------
mult                              3.16%    1.26%    1.89%   205988
  sature                                   1.89%    0.00%   206654
----------------------------------------------------------------------
negate                            3.03%    3.03%    0.00%     3984
----------------------------------------------------------------------
sub                               3.00%    1.68%    1.31%   143786
  sature                                   1.31%    0.00%   143814
----------------------------------------------------------------------
Autocorr                          2.75%    1.00%    1.74%       30
  L_Extract                                0.00%    0.01%      330
  norm_l                                   0.00%    0.00%       30
  L_shl                                    0.01%    0.00%      330
  mult_r                                   0.04%    0.06%     7200
  L_mac                                    0.59%    1.01%    79230
  shr                                      0.01%    0.00%     1680
----------------------------------------------------------------------
L_Extract                         2.63%    0.68%    1.94%    42594
  L_shr                                    0.41%    0.00%    42598
  L_msu                                    0.50%    0.94%    42598
```

```
  extract_h                          0.00%    0.00%    42598
  extract_l                          0.09%    0.00%    42598
----------------------------------------------------------------------
cor_h_x_e                   2.04%    0.68%    1.35%       60
  norm_l                             0.00%    0.00%       60
  L_abs                              0.00%    0.00%     2400
  L_shl                              0.09%    0.00%     2400
  L_shr                              0.00%    0.00%      300
  L_sub                              0.03%    0.00%     2400
  L_mac                              0.36%    0.62%    49200
  round                              0.22%    0.01%     2400
  L_add                              0.00%    0.00%      300
  sub                                0.00%    0.00%       60
----------------------------------------------------------------------
Mpy_32_16                   1.87%    0.52%    1.34%    32046
  L_mac                              0.23%    0.41%    32110
  L_mult                             0.20%    0.00%    32110
  mult                               0.19%    0.29%    32110
----------------------------------------------------------------------
Pred_lt_3                   1.62%    0.34%    1.28%       60
  L_mac                              0.35%    0.61%    48000
  round                              0.22%    0.01%     2400
  negate                             0.04%    0.00%       60
  sub                                0.02%    0.01%     2400
  add                                0.00%    0.00%       26
----------------------------------------------------------------------
Qua_gain                    1.46%    0.31%    1.14%       60
  Gbk_presel                         0.00%    0.01%       56
  Mpy_32_16                          0.15%    0.39%     9600
  L_Extract                          0.00%    0.01%      300
  Gain_update                        0.00%    0.01%       60
  Gain_predict                       0.06%    0.09%       60
  norm_l                             0.00%    0.00%      180
  div_s                              0.01%    0.00%       60
  L_deposit_l                        0.00%    0.00%     3960
  L_deposit_h                        0.00%    0.00%      300
  L_shl                              0.00%    0.00%      240
  L_shr                              0.02%    0.00%     2700
  L_sub                              0.03%    0.00%     2100
  L_add                              0.06%    0.00%     9660
  extract_h                          0.00%    0.00%      360
  negate                             0.08%    0.00%      120
  L_mult                             0.00%    0.00%      540
  mult                               0.04%    0.06%     7680
  shl                                0.00%    0.00%       56
  shr                                0.00%    0.00%       60
  sub                                0.01%    0.01%     1680
  add                                0.00%    0.03%     3938
  extract_l                          0.00%    0.00%     1980
----------------------------------------------------------------------
set_sign                    1.08%    0.18%    0.89%       60
  Inv_sqrt                           0.00%    0.01%      120
  L_abs                              0.00%    0.00%     2400
```

```
  L_shl                               0.00%    0.00%      120
  L_mac                               0.04%    0.08%     6720
  extract_h                           0.00%    0.00%      120
  negate                              0.75%    0.00%      984
  L_mult                              0.01%    0.00%     2400
  mult                                0.00%    0.00%       60
---------------------------------------------------------------------
Div_32                       1.04%    0.13%    0.91%     1069
  Mpy_32_16                           0.03%    0.08%     2138
  Mpy_32                              0.03%    0.07%     1069
  L_Extract                           0.04%    0.14%     3207
  div_s                               0.24%    0.20%     1069
  L_shl                               0.03%    0.00%     1069
  L_sub                               0.01%    0.00%     1069
---------------------------------------------------------------------
Pre_Process                  0.97%    0.12%    0.85%       30
  Mpy_32_16                           0.07%    0.19%     4800
  L_Extract                           0.03%    0.10%     2400
  L_shl                               0.09%    0.00%     2400
  L_mac                               0.04%    0.09%     7200
  round                               0.22%    0.01%     2400
  L_add                               0.01%    0.00%     2400
---------------------------------------------------------------------
Lsp_select_2                 0.82%    0.27%    0.55%       60
  L_sub                               0.03%    0.00%     1920
  L_mac                               0.06%    0.12%     9600
  mult                                0.05%    0.08%     9600
  sub                                 0.11%    0.08%     9900
---------------------------------------------------------------------
abs_s                        0.81%    0.81%    0.00%     1368
---------------------------------------------------------------------
shr_r                        0.79%    0.79%    0.00%        0
---------------------------------------------------------------------
ener_dB                      0.74%    0.34%    0.39%      210
  L_shr                               0.03%    0.00%     3638
  L_mac                               0.12%    0.21%    16800
  sub                                 0.00%    0.00%      210
  add                                 0.00%    0.03%     3428
  extract_l                           0.00%    0.00%      210
---------------------------------------------------------------------
Lsp_select_1                 0.73%    0.18%    0.55%       60
  L_sub                               0.03%    0.00%     1920
  L_mac                               0.06%    0.12%     9600
  mult                                0.05%    0.08%     9600
  sub                                 0.11%    0.08%     9900
---------------------------------------------------------------------
Weight_Az                    0.71%    0.04%    0.67%      270
  round                               0.59%    0.03%     6330
  L_mult                              0.03%    0.00%     6330
---------------------------------------------------------------------
L_Comp                       0.70%    0.39%    0.30%    15486
  L_deposit_h                         0.00%    0.00%    15486
  L_mac                               0.11%    0.19%    15486
```

```
-----------------------------------------------------------------------
div_s                          0.63%    0.34%    0.29%     1476
  L_sub                                 0.19%    0.00%    12030
  sature                                0.10%    0.00%    12047
-----------------------------------------------------------------------
L_shr                          0.61%    0.61%    0.00%    63274
-----------------------------------------------------------------------
build_code                     0.61%    0.35%    0.25%       60
  L_shr                                 0.00%    0.00%      600
  L_mult                                0.00%    0.00%      600
  mult                                  0.00%    0.00%      550
  shl                                   0.00%    0.00%      300
  shr                                   0.02%    0.00%     2200
  sub                                   0.07%    0.05%     6336
  add                                   0.01%    0.08%     9427
  extract_l                             0.00%    0.00%      600
-----------------------------------------------------------------------
L_add_c                        0.56%    0.56%    0.00%        0
-----------------------------------------------------------------------
Int_lpc                        0.44%    0.00%    0.44%       29
  Lsp_lsf                               0.08%    0.25%       60
  Lsp_Az                                0.01%    0.12%       30
  shr                                   0.00%    0.00%      600
  add                                   0.00%    0.00%      300
-----------------------------------------------------------------------
Lsp_Az                         0.43%    0.03%    0.39%       90
  Get_lsp_pol                           0.04%    0.27%      180
  L_shr_r                               0.05%    0.00%      900
  L_sub                                 0.01%    0.00%      900
  L_add                                 0.00%    0.00%      900
  extract_l                             0.00%    0.00%      900
-----------------------------------------------------------------------
Corr_xy2                       0.35%    0.12%    0.22%       60
  norm_l                                0.00%    0.00%      180
  L_shl                                 0.00%    0.00%      180
  L_mac                                 0.04%    0.09%     7200
  round                                 0.01%    0.00%      180
  negate                                0.04%    0.00%       60
  shr                                   0.02%    0.00%     2400
  sub                                   0.00%    0.00%      120
  add                                   0.00%    0.00%      180
-----------------------------------------------------------------------
Lsp_lsf                        0.34%    0.08%    0.25%       60
  L_shl                                 0.01%    0.00%      600
  round                                 0.05%    0.00%      600
  L_mult                                0.00%    0.00%      600
  shl                                   0.01%    0.00%      600
  sub                                   0.09%    0.07%     8338
  add                                   0.00%    0.00%      600
-----------------------------------------------------------------------
Inv_sqrt                       0.33%    0.08%    0.25%     1260
  norm_l                                0.04%    0.00%     1211
  L_deposit_h                           0.00%    0.00%     1169
```

```
  L_shl                              0.04%    0.00%     1211
  L_shr                              0.03%    0.00%     4221
  L_msu                              0.01%    0.02%     1211
  extract_h                          0.00%    0.00%     1127
  shr                                0.01%    0.00%     1211
  sub                                0.03%    0.03%     3633
  add                                0.00%    0.00%     1127
  extract_l                          0.00%    0.00%     1169
----------------------------------------------------------------------
Get_lsp_pol                 0.32%    0.04%    0.27%      180
  Mpy_32_16                          0.02%    0.07%     1800
  L_Extract                          0.02%    0.07%     1800
  L_shl                              0.06%    0.00%     1800
  L_msu                              0.00%    0.01%      900
  L_sub                              0.02%    0.00%     1800
  L_add                              0.00%    0.00%     1800
  L_mult                             0.00%    0.00%      180
----------------------------------------------------------------------
shr                         0.27%    0.27%    0.00%    23652
  shl                                0.00%    0.00%        3
----------------------------------------------------------------------
Int_qlpc                    0.24%    0.01%    0.23%       29
  Lsp_Az                             0.02%    0.25%       60
  shr                                0.00%    0.00%      600
  add                                0.00%    0.00%      300
----------------------------------------------------------------------
shl                         0.24%    0.24%    0.00%     7939
----------------------------------------------------------------------
Lag_window_bwd              0.21%    0.02%    0.19%       30
  Mpy_32                             0.02%    0.06%      900
  L_Extract                          0.02%    0.07%     1830
  norm_l                             0.00%    0.00%       30
  L_shl                              0.03%    0.00%      930
  L_add                              0.00%    0.00%      930
----------------------------------------------------------------------
G_pitch                     0.20%    0.07%    0.12%       60
  norm_l                             0.00%    0.00%      120
  div_s                              0.00%    0.00%       48
  L_shl                              0.00%    0.00%      112
  L_mac                              0.03%    0.06%     4800
  round                              0.00%    0.00%      112
  shr                                0.02%    0.00%     2496
  sub                                0.00%    0.00%      216
----------------------------------------------------------------------
Gain_predict                0.15%    0.06%    0.09%       60
  L_Extract                          0.00%    0.00%       60
  Log2                               0.01%    0.00%       60
  Pow2                               0.03%    0.00%       60
  L_shl                              0.00%    0.00%       60
  L_shr                              0.00%    0.00%       60
  L_mac                              0.01%    0.03%     2700
  extract_h                          0.00%    0.00%       60
  L_mult                             0.00%    0.00%       60
```

```
  sub                                     0.00%    0.00%        60
  extract_l                               0.00%    0.00%        60
----------------------------------------------------------------------
Lsp_prev_extract                 0.15%    0.06%    0.09%        60
  L_deposit_h                             0.00%    0.00%       600
  L_shl                                   0.01%    0.00%       600
  L_msu                                   0.02%    0.04%      2400
  extract_h                               0.00%    0.00%      1200
  L_mult                                  0.00%    0.00%       600
----------------------------------------------------------------------
Lsp_lsf2                         0.13%    0.04%    0.08%        30
  L_shr                                   0.00%    0.00%       300
  L_mult                                  0.00%    0.00%       300
  mult                                    0.00%    0.00%       300
  shl                                     0.00%    0.00%       300
  sub                                     0.04%    0.03%      4178
  add                                     0.00%    0.00%       300
  extract_l                               0.00%    0.00%       300
----------------------------------------------------------------------
norm_l                           0.13%    0.13%    0.00%      3055
----------------------------------------------------------------------
Lsp_get_quante                   0.12%    0.01%    0.11%        30
  Copy                                    0.00%    0.00%        30
  add                                     0.00%    0.00%       300
  Lsp_expand_1_2                          0.01%    0.01%        60
  Lsp_prev_compose                        0.02%    0.01%        30
  Lsp_stability                           0.02%    0.01%        30
----------------------------------------------------------------------
mult_r                           0.11%    0.04%    0.06%      7200
  sature                                  0.06%    0.00%      7200
----------------------------------------------------------------------
Copy                             0.11%    0.11%    0.00%       455
----------------------------------------------------------------------
extract_l                        0.11%    0.11%    0.00%     50928
----------------------------------------------------------------------
Int_bwd                          0.11%    0.06%    0.04%        30
  L_shr                                   0.01%    0.00%      1860
  L_mult                                  0.00%    0.00%      1860
  shr                                     0.01%    0.00%      1860
  sub                                     0.00%    0.00%        60
  add                                     0.00%    0.01%      1860
  extract_l                               0.00%    0.00%      1860
----------------------------------------------------------------------
Interpol_3                       0.08%    0.02%    0.06%       280
  L_mac                                   0.01%    0.02%      2240
  round                                   0.02%    0.00%       280
  sub                                     0.00%    0.00%       280
  add                                     0.00%    0.00%       112
----------------------------------------------------------------------
perc_var                         0.08%    0.02%    0.06%        30
  L_shr                                   0.00%    0.00%        55
  L_sub                                   0.00%    0.00%        55
  L_mult                                  0.00%    0.00%        49
```

```
  mult                                        0.00%    0.00%       26
  shl                                         0.01%    0.00%      626
  shr                                         0.00%    0.00%      175
  abs_s                                       0.03%    0.00%       60
  sub                                         0.00%    0.00%      799
  add                                         0.00%    0.00%       58
  extract_l                                   0.00%    0.00%       53
-----------------------------------------------------------------------
_printf                              0.08%    0.01%    0.07%       46
  __vfprintf                                  0.06%    0.01%       46
-----------------------------------------------------------------------
__vfprintf                           0.07%    0.06%    0.01%       46
  strlen                                      0.00%    0.00%        2
  _printf_display                             0.01%    0.00%       30
-----------------------------------------------------------------------
Lsp_expand_1_2                       0.07%    0.03%    0.03%      120
  shr                                         0.01%    0.00%     1080
  sub                                         0.01%    0.00%     1094
  add                                         0.00%    0.00%     1094
-----------------------------------------------------------------------
L_shr_r                              0.06%    0.06%    0.00%      960
  L_shr                                       0.00%    0.00%      960
-----------------------------------------------------------------------
fputc                                0.05%    0.02%    0.03%        0
  __flsbuf                                    0.03%    0.00%      978
-----------------------------------------------------------------------
Random                               0.04%    0.04%    0.00%        0
-----------------------------------------------------------------------
__rt_sdiv                            0.04%    0.04%    0.00%     2400
-----------------------------------------------------------------------
prm2bits_ld8e                        0.04%    0.04%    0.00%       30
-----------------------------------------------------------------------
Lsp_prev_compose                     0.04%    0.02%    0.01%       30
  L_mac                                       0.00%    0.01%     1200
  extract_h                                   0.00%    0.00%      300
  L_mult                                      0.00%    0.00%      300
-----------------------------------------------------------------------
Lsp_stability                        0.03%    0.02%    0.01%       30
  L_deposit_l                                 0.00%    0.00%     1080
  L_sub                                       0.01%    0.00%      810
  sub                                         0.00%    0.00%       60
  add                                         0.00%    0.00%        3
-----------------------------------------------------------------------
Pow2                                 0.03%    0.03%    0.00%       60
  L_shr_r                                     0.00%    0.00%       60
  L_deposit_h                                 0.00%    0.00%       60
  L_shr                                       0.00%    0.00%       60
  L_msu                                       0.00%    0.00%       60
  extract_h                                   0.00%    0.00%       60
  L_mult                                      0.00%    0.00%       60
  sub                                         0.00%    0.00%      120
  extract_l                                   0.00%    0.00%       60
-----------------------------------------------------------------------
```

```
Log2                            0.03%    0.03%    0.00%      120
  norm_l                                 0.00%    0.00%      120
  L_deposit_h                            0.00%    0.00%      120
  L_shl                                  0.00%    0.00%      120
  L_shr                                  0.00%    0.00%      240
  L_msu                                  0.00%    0.00%      120
  extract_h                              0.00%    0.00%      240
  sub                                    0.00%    0.00%      360
  extract_l                              0.00%    0.00%      120
-----------------------------------------------------------------------
Get_wegt                        0.03%    0.00%    0.03%       30
  norm_s                                 0.00%    0.00%       30
  L_shl                                  0.01%    0.00%      622
  extract_h                              0.00%    0.00%      622
  L_mult                                 0.00%    0.00%      622
  shl                                    0.00%    0.00%      300
  sub                                    0.00%    0.00%      812
  add                                    0.00%    0.00%      281
-----------------------------------------------------------------------
__flsbuf                        0.03%    0.03%    0.00%      993
  _sys_istty                             0.00%    0.00%        2
  malloc                                 0.00%    0.00%        2
  _writebuf                              0.00%    0.00%       40
-----------------------------------------------------------------------
Lsp_get_tdist                   0.02%    0.00%    0.02%       60
  L_shl                                  0.01%    0.00%      600
  L_mac                                  0.00%    0.00%      600
  extract_h                              0.00%    0.00%      600
  L_mult                                 0.00%    0.00%      600
  mult                                   0.00%    0.00%      600
  sub                                    0.00%    0.00%      600
-----------------------------------------------------------------------
Lsf_lsp2                        0.02%    0.02%    0.00%       30
  L_shr                                  0.00%    0.00%      300
  L_mult                                 0.00%    0.00%      300
  mult                                   0.00%    0.00%      300
  shr                                    0.00%    0.00%      290
  sub                                    0.00%    0.00%      300
  add                                    0.00%    0.00%      300
  extract_l                              0.00%    0.00%      300
-----------------------------------------------------------------------
Lsp_prev_update                 0.02%    0.00%    0.02%       30
  Copy                                   0.02%    0.00%      120
-----------------------------------------------------------------------
L_sat                           0.02%    0.02%    0.00%        0
-----------------------------------------------------------------------
Gain_update                     0.01%    0.00%    0.01%       60
  L_Comp                                 0.00%    0.00%       60
  Log2                                   0.01%    0.00%       60
  L_shl                                  0.00%    0.00%       60
  extract_h                              0.00%    0.00%       60
  mult                                   0.00%    0.00%       60
  sub                                    0.00%    0.00%       60
```

```
----------------------------------------------------------------------
Gbk_presel                        0.01%    0.00%    0.01%        56
  L_deposit_l                              0.00%    0.00%        60
  L_shl                                    0.00%    0.00%       180
  L_shr                                    0.00%    0.00%       598
  L_sub                                    0.00%    0.00%       598
  L_add                                    0.00%    0.00%        60
  extract_h                                0.00%    0.00%       240
  L_mult                                   0.00%    0.00%       778
  mult                                     0.00%    0.00%        60
  sub                                      0.00%    0.00%       330
  add                                      0.00%    0.00%       330
----------------------------------------------------------------------
Init_Coder_ld8e                   0.01%    0.00%    0.01%         1
  Copy                                     0.00%    0.00%         1
  Set_zero                                 0.01%    0.00%        15
  Lsp_encw_resete                          0.00%    0.00%         1
----------------------------------------------------------------------
L_abs                             0.01%    0.01%    0.00%      6933
----------------------------------------------------------------------
perc_vare                         0.01%    0.01%    0.00%         0
----------------------------------------------------------------------
Gain_update_erasure               0.01%    0.01%    0.00%         0
----------------------------------------------------------------------
norm_s                            0.01%    0.01%    0.00%       329
----------------------------------------------------------------------
extract_h                         0.01%    0.01%    0.00%    123173
----------------------------------------------------------------------
Parity_Pitch                      0.01%    0.01%    0.00%        30
  shr                                      0.00%    0.00%       210
  add                                      0.00%    0.00%       180
----------------------------------------------------------------------
Set_zero                          0.01%    0.01%    0.00%        15
----------------------------------------------------------------------
L_deposit_l                       0.01%    0.01%    0.00%      5100
----------------------------------------------------------------------
__rt_memcpy                       0.01%    0.00%    0.01%        82
  __rt_memcpy_w                            0.01%    0.00%        82
----------------------------------------------------------------------
_printf_display                   0.01%    0.01%    0.00%        30
  strlen                                   0.00%    0.00%        30
  __rt_udiv10                              0.00%    0.00%        51
----------------------------------------------------------------------
Lsp_expand_2                      0.01%    0.01%    0.00%        60
  shr                                      0.00%    0.00%       300
  sub                                      0.00%    0.00%       300
  add                                      0.00%    0.00%       300
----------------------------------------------------------------------
__rt_memcpy_w                     0.01%    0.01%    0.00%        82
----------------------------------------------------------------------
fwrite                            0.00%    0.00%    0.00%        30
  __rt_memcpy                              0.00%    0.00%        44
  __flsbuf                                 0.00%    0.00%        15
```

```
-----------------------------------------------------------------------
Lag_window                      0.00%    0.00%    0.00%        30
  Mpy_32                                 0.00%    0.01%       300
  L_Extract                              0.00%    0.01%       300
-----------------------------------------------------------------------
test_err                        0.00%    0.00%    0.00%        60
  L_sub                                  0.00%    0.00%       147
  sub                                    0.00%    0.00%        60
  add                                    0.00%    0.00%        86
-----------------------------------------------------------------------
Lsp_expand_1                    0.00%    0.00%    0.00%        60
  shr                                    0.00%    0.00%       240
  sub                                    0.00%    0.00%       258
  add                                    0.00%    0.00%       258
-----------------------------------------------------------------------
L_deposit_h                     0.00%    0.00%    0.00%     17735
-----------------------------------------------------------------------
Lsp_last_select                 0.00%    0.00%    0.00%        30
  L_sub                                  0.00%    0.00%        30
-----------------------------------------------------------------------
calc_stat                       0.00%    0.00%    0.00%        28
-----------------------------------------------------------------------
update_exc_err                  0.00%    0.00%    0.00%        60
  Mpy_32_16                              0.00%    0.00%       114
  L_Extract                              0.00%    0.00%       114
  L_shl                                  0.00%    0.00%       118
  L_sub                                  0.00%    0.00%       116
  L_add                                  0.00%    0.00%       116
  sub                                    0.00%    0.00%        76
-----------------------------------------------------------------------
Enc_lag3                        0.00%    0.00%    0.00%        60
  sub                                    0.00%    0.00%       178
  add                                    0.00%    0.00%       232
-----------------------------------------------------------------------
L_negate                        0.00%    0.00%    0.00%       471
-----------------------------------------------------------------------
__rt_get_argv                   0.00%    0.00%    0.00%         1
  _handle_redirection                    0.00%    0.00%         4
  _sys_command_string                    0.00%    0.00%         1
-----------------------------------------------------------------------
atoi                            0.00%    0.00%    0.00%         1
  strtol                                 0.00%    0.00%         1
  __rt_errno_addr                        0.00%    0.00%         1
-----------------------------------------------------------------------
Lsp_encw_resete                 0.00%    0.00%    0.00%         1
  Copy                                   0.00%    0.00%         4
-----------------------------------------------------------------------
_fflush                         0.00%    0.00%    0.00%         5
  _writebuf                              0.00%    0.00%         1
-----------------------------------------------------------------------
fclose                          0.00%    0.00%    0.00%        12
  _sys_close                             0.00%    0.00%         5
  free                                   0.00%    0.00%         3
```

```
  fflush                                   0.00%    0.00%        5
----------------------------------------------------------------------
fopen                             0.00%    0.00%    0.00%        3
  __rt_memclr_w                            0.00%    0.00%        3
  malloc                                   0.00%    0.00%        3
  freopen                                  0.00%    0.00%        3
----------------------------------------------------------------------
_initio                           0.00%    0.00%    0.00%        1
  setvbuf                                  0.00%    0.00%        3
  __rt_memclr_w                            0.00%    0.00%        3
  freopen                                  0.00%    0.00%        3
----------------------------------------------------------------------
_terminateio                      0.00%    0.00%    0.00%        1
  free                                     0.00%    0.00%        3
  fclose                                   0.00%    0.00%        6
----------------------------------------------------------------------
tst_bwd_dominant                  0.00%    0.00%    0.00%       28
  shl                                      0.00%    0.00%       19
  add                                      0.00%    0.00%       28
----------------------------------------------------------------------
Init_Pre_Process                  0.00%    0.00%    0.00%        1
----------------------------------------------------------------------
fflush                            0.00%    0.00%    0.00%        5
  fseek                                    0.00%    0.00%        5
  _fflush                                  0.00%    0.00%        5
----------------------------------------------------------------------
free                              0.00%    0.00%    0.00%        6
  __user_libspace                          0.00%    0.00%        6
  __Heap_Free                              0.00%    0.00%        6
----------------------------------------------------------------------
fseek                             0.00%    0.00%    0.00%        5
  _sys_istty                               0.00%    0.00%        5
----------------------------------------------------------------------
__rt_exit                         0.00%    0.00%    0.00%        1
  __rt_lib_shutdown                        0.00%    0.00%        1
----------------------------------------------------------------------
_terminate_user_alloc             0.00%    0.00%    0.00%        1
----------------------------------------------------------------------
_init_user_alloc                  0.00%    0.00%    0.00%        1
----------------------------------------------------------------------
_init_alloc                       0.00%    0.00%    0.00%        1
  __user_libspace                          0.00%    0.00%        1
  __Heap_ProvideMemory                     0.00%    0.00%        1
  __Heap_DescSize                          0.00%    0.00%        1
  __Heap_Initialize                        0.00%    0.00%        1
----------------------------------------------------------------------
malloc                            0.00%    0.00%    0.00%        6
  __user_libspace                          0.00%    0.00%        6
  __Heap_Alloc                             0.00%    0.00%        6
----------------------------------------------------------------------
__rt_errno_addr                   0.00%    0.00%    0.00%        2
  __user_libspace                          0.00%    0.00%        2
----------------------------------------------------------------------
```

```
__rt_memclr_w                     0.00%    0.00%    0.00%          6
-----------------------------------------------------------------------
__rt_udiv                         0.00%    0.00%    0.00%          1
-----------------------------------------------------------------------
strtol                            0.00%    0.00%    0.00%          1
  __rt_ctype_table                         0.00%    0.00%          1
  _strtoul                                 0.00%    0.00%          1
  __rt_errno_addr                          0.00%    0.00%          1
-----------------------------------------------------------------------
_sys_command_string               0.00%    0.00%    0.00%          1
-----------------------------------------------------------------------
_sys_open                         0.00%    0.00%    0.00%          6
  strlen                                   0.00%    0.00%          6
-----------------------------------------------------------------------
_sys_close                        0.00%    0.00%    0.00%          5
-----------------------------------------------------------------------
_sys_write                        0.00%    0.00%    0.00%         41
-----------------------------------------------------------------------
exit                              0.00%    0.00%    0.00%          1
  __rt_exit                                0.00%    0.00%          1
-----------------------------------------------------------------------
_sys_istty                        0.00%    0.00%    0.00%          7
-----------------------------------------------------------------------
_writebuf                         0.00%    0.00%    0.00%         41
  _sys_write                               0.00%    0.00%         41
-----------------------------------------------------------------------
_strtoul                          0.00%    0.00%    0.00%          1
  _chval                                   0.00%    0.00%          2
-----------------------------------------------------------------------
__filbuf                          0.00%    0.00%    0.00%         11
  _sys_read                                0.00%    0.00%         11
  malloc                                   0.00%    0.00%          1
-----------------------------------------------------------------------
freopen                           0.00%    0.00%    0.00%          6
  _sys_open                                0.00%    0.00%          6
  fclose                                   0.00%    0.00%          6
-----------------------------------------------------------------------
__Heap_Initialize                 0.00%    0.00%    0.00%          1
-----------------------------------------------------------------------
__Heap_DescSize                   0.00%    0.00%    0.00%          2
-----------------------------------------------------------------------
__Heap_Alloc                      0.00%    0.00%    0.00%          6
-----------------------------------------------------------------------
__Heap_ProvideMemory              0.00%    0.00%    0.00%          1
  __Heap_Free                              0.00%    0.00%          1
-----------------------------------------------------------------------
__Heap_Free                       0.00%    0.00%    0.00%          7
-----------------------------------------------------------------------
__rt_lib_init                     0.00%    0.00%    0.00%          1
  _fp_init                                 0.00%    0.00%          1
  _get_lc_ctype                            0.00%    0.00%          1
  __user_libspace                          0.00%    0.00%          1
  __Heap_DescSize                          0.00%    0.00%          1
```

```
  _init_alloc                           0.00%    0.00%        1
  _init_user_alloc                      0.00%    0.00%        1
  _initio                               0.00%    0.00%        1
  __rt_get_argv                         0.00%    0.00%        1
----------------------------------------------------------------------
__rt_lib_shutdown              0.00%    0.00%    0.00%        1
  _terminate_user_alloc                 0.00%    0.00%        1
  _terminateio                          0.00%    0.00%        1
----------------------------------------------------------------------
__user_libspace                0.00%    0.00%    0.00%       20
----------------------------------------------------------------------
__rt_ctype_table               0.00%    0.00%    0.00%        1
  __user_libspace                       0.00%    0.00%        1
----------------------------------------------------------------------
__rt_udiv10                    0.00%    0.00%    0.00%       51
----------------------------------------------------------------------
__rt_stackheap_init            0.00%    0.00%    0.00%        1
  __user_initial_stackheap              0.00%    0.00%        1
  __user_libspace                       0.00%    0.00%        1
----------------------------------------------------------------------
__rt_heap_extend               0.00%    0.00%    0.00%        1
  __user_libspace                       0.00%    0.00%        1
----------------------------------------------------------------------
strlen                         0.00%    0.00%    0.00%       38
----------------------------------------------------------------------
_sys_exit                      0.00%    0.00%    0.00%        1
----------------------------------------------------------------------
_chval                         0.00%    0.00%    0.00%        2
----------------------------------------------------------------------
_handle_redirection            0.00%    0.00%    0.00%        4
----------------------------------------------------------------------
fread                          0.00%    0.00%    0.00%       31
  __rt_memcpy                           0.00%    0.00%       38
  __filbuf                              0.00%    0.00%       11
  __rt_udiv                             0.00%    0.00%        1
----------------------------------------------------------------------
setvbuf                        0.00%    0.00%    0.00%        3
----------------------------------------------------------------------
_get_lc_ctype                  0.00%    0.00%    0.00%        1
----------------------------------------------------------------------
__user_initial_stackheap       0.00%    0.00%    0.00%        1
----------------------------------------------------------------------
__rt_fp_status_addr            0.00%    0.00%    0.00%        1
  __user_libspace                       0.00%    0.00%        1
----------------------------------------------------------------------
_sys_read                      0.00%    0.00%    0.00%       11
----------------------------------------------------------------------
_fp_init                       0.00%    0.00%    0.00%        1
  __rt_fp_status_addr                   0.00%    0.00%        1
----------------------------------------------------------------------
```