GSM EFR Vocoder on a Configurable DSP Core
A quantitative analysis

Yann Bajot, Habib Mehrez

Laboratoire LIP6/ASIM UPMC Paris VI, 4 Place Jussieu, 75252 Paris Cedex 05, France
e-mail:yann.bajot@lip6.fr

Abstract

This paper deals with the implementation of the GSM EFR vocoder on a configurable DSP processor. We first present the highly configurable and modular architecture used in the model of our DSP core. Next, a quantitative analysis of the EFR algorithm regarding Instruction-Level Parallelism and Algorithm/Architecture Adequation is performed. This leads to define a cost-effective and powerful architecture with special features enabling an optimized implementation of the vocoder. The results obtained are shown and compared to those of state-of-art DSP Processors.

Introduction

Signal processing tasks performed in wireless systems require high performance. Third generation of mobile phones with internet capabilities will have to manage both data and voice, still increasing demand for complex DSP applications. Processors used in such systems have to deal with conflicting constraints: high performance for real-time functioning, low-power for energy savings, and low-cost.

One solution consists of using Application-Specific Instruction-Set Processors (ASIPs) [1]. These processors include a datapath and instruction-set optimized for the targeted applications, achieving the same or upper performances than a general DSP at a lower cost (silicon, power). ASIPs are also a good alternative to hard-wired blocks, since they provide more flexibility for the algorithm implementation. Within the context of System-On-Chip, ASIP cores provide efficient, flexible and low-cost computation engines. Since the design cycle of a new processor is very long, we propose in [2] a methodology for ASIP Core design. This method relies on a architectural model of the core, build on several configurable macro-blocks. This model is presented in the next section.

To illustrate interest of ASIPs, we examine the implementation of a complex DSP algorithm: the Enhanced Full-Rate (EFR) vocoder used in the Global System for Mobile Communications (GSM). A precise quantitative analysis of the targeted algorithm allows to extract relevant architectural features to be included in our core model. Then we can define an optimized cost-effective, powerful ASIP core which deliver high computation rate while minimizing hardware resources. Comparing results with high-end general DSPs shows the interest of the method.

Configurable DSP Core Architecture

We present here the architectural model that we use to design ASIP cores. There are two main reasons to use a configurable model rather than a completely redesigned architecture. First, the software tools can be more easily produced for a particular description of the core. If they are retargetable, the tools can even be automatically generated, taking into account the parameters of the core model. Secondly, the macro-block concept of the model enables a fast and efficient hardware implementation on any technological library [2]. However, the model must provide a high degree of configurability to ensure a good adequation with the target algorithm.

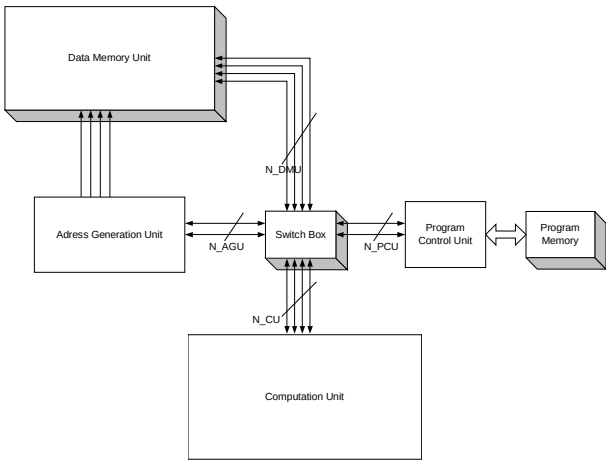

Fig 1: DSP Core model

Fig. 1 illustrates the global architecture of the core. It is centered around the 'Switch Box" unit, which routes data between the other units. Each unit is configurable, as well as the number of data busses between them. For instance, we configure the Data Memory Unit by specifying the bandwidth, and for each memory block its type (word width, single or double port) and addressing space. It permits to define heterogeneous memory structure including

memories of different types, to best match the data structures found in targeted algorithms. The address generation unit (AGU) performs data address computation for the memory unit, including special modes like modulo or bit-reverse. The number of address pointers is configurable.

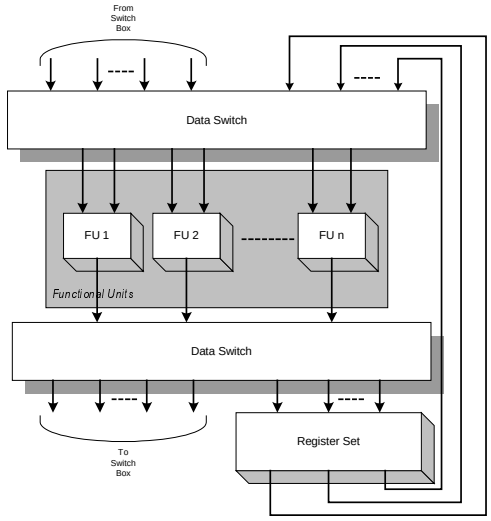

Fig 2: Computation Unit

The computation unit (CU) is shown on fig. 2. It is based on a highly parallel, VLIW-like architecture. A good way to reduce power consumption is provided by down-scaling the clock frequency of the processor. Using parallel architecture which exploits Instruction-Level Parallelism found in algorithm allows to work at lower frequencies. Unlike most DSP processors which address data in a load/store manner, our model allows direct memory access to operands. Data are provided by the switch-box unit and are routed towards functional units (FUs) by a data switch. FU outputs are driven either to switch box or to the register-set via a second data switch. We will see later that this feature significantly increases performance.

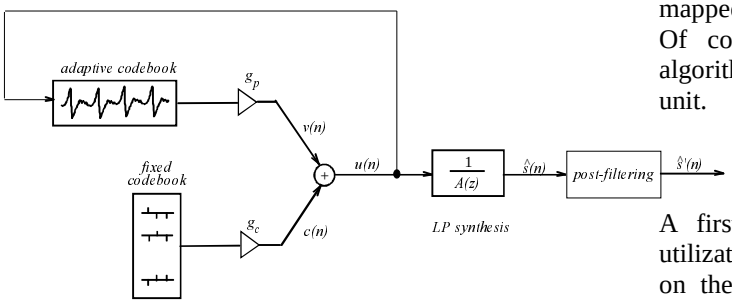

Fig 3: EFR Decoder

The configurability of the CU includes data width, FU types (ALU, Multiplier, BMU, or specialized units), data-switches connectivity, register-set size and granularity [2].

Presentation of the EFR vocoder

The GSM system uses three algorithms to encode and decode the voice: the EFR, Full Rate and Half Rate vocoders. The EFR is the most complex and provides the best quality, for a bit rate of 12.2Kb/s. It consumes about 20 MIPS on a single MAC architecture. EFR is a Code Excited Linear Prediction (CELP) vocoder [3]. For each frame of 20ms (160 samples), it tries to model the voice as a linear system (vocal tract) excited by a periodic signal (pitch) for voiced speech and by noise for unvoiced speech. The encoder computes coefficients of the linear filter (short-term filter) using the Linear Predictive Coding (LPC) analysis, twice by frame. Then the pitch parameters and the excitation model are computed by searching in two codebooks (adaptive for pitch and fixed for excitation). This work is done four times by frame. Finally the encoder transmits the LPC coefficients, and for each codebook the chosen vector and its associated gain. The decoder synthesizes the speech by filtering the sum of the two codebooks outputs with the short-term filter (fig. 3).

Analysis of the algorithm and impact on architecture

This analysis is based on the standard C code and test sequences of the EFR vocoder distributed by ETSI[4] [5]. This bit-true implementation uses 16-32 bit data types and relies on a set of basic operators which are used to perform all arithmetic computations. It includes elementary operators like addition (*add),* subtraction *(sub)*, multiply *(mul*), shifts *(shl, shr)*, as well as more complex operators such as "Multiply-Accumulate with saturation" *(l_mac)*, "Normalize" *(norm)*, or double precision operators. This source code is particularly well suited for a hardware implementation, knowing that almost all arithmetic operators used in the code can be mapped on functional units found in DSP processors. Of course, the more operators we use in the algorithm, the most complex will be the computation unit.

Operation Count

A first interesting information concerns operator utilization rate. The following estimations are based on the dynamic simulation of the encoder source code with a test sequence of 163 frames (about 3 seconds of speech). Results for each operator are weighted with their estimated execution time on a single MAC architecture. Fig. 4 shows that 75 percent of the computation load are used by only 9 operations on a total of 41. Multiply-accumulate

operations alone takes almost half of the load, which still points out importance of this kind of operator in DSP algorithms.

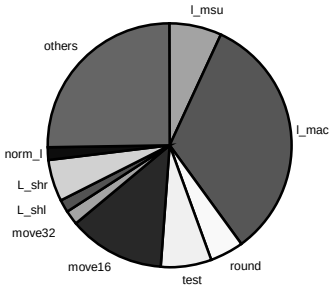

Fig 4: Weighted operation counts

10 operators are not used and can be removed from the functional units. For instance, carry management can be removed from ALU since L_add_c and L_sub_c are unused. It is the same for 32 bit logic operations. Saturation appears only in multiply-accumulate operations, so it should be implemented in a MAC unit. The low rate of the mult_round operation (0.23 %) does not justify integration of a costly rounding operator in MAC units. However, 32 bits shift and normalize operations (8 % of the load) requires use of a Shifter/Normalizer Unit (see fig. 5). At last, the use of a register-set with double precision capabilities (16/32 bit word addressing) can significantly increase performance at a reasonable cost by suppressing the need for *extract /deposit* operators.

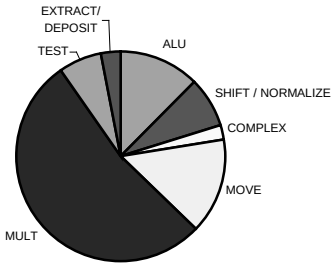

Fig 5: Operation counts by type

This first study of operation counts already give us some details about the final architecture. Now we can go further by examining the most time consuming functions used in the algorithm.

Critical functions

A common rule of thumb for DSP applications says that 20 percent of the application consumes 80 percent of the overall execution time. The remaining 80 percent represents sequential, data-dependant control operations that are difficult to speed up and take few advantages of parallel architectures. But DSP subroutines forming the critical 20 percent have a high Instruction-Level Parallelism (ILP) and can be significantly speed up by architectural optimizations. Techniques like VLIW architecture, specialized functional units and large instruction words with multiple operations encoded allow to maximize the use of the available ILP.

Fig. 6 presents the seven most consuming subroutines found in the EFR encoder. As seen before, results are based on the execution of the encoder on a single MAC, with a direct memory access architecture. The number of Millions of Operations per second (MOPS) required for the complete encoder is about 12 MOPS on this architecture.

Function Name	MOPS	%
Search_10i40	2,45	20,3
Norm_Corr	1,49	12,3
Lag_max	1,15	9.5
Cor_h	1,39	11.5
Syn_flt	0,74	6.1
Chebps	0,71	5.9
Convolve	0,42	3.5
	8,36	69,1
Encoder	12.1	100

Fig 6: Critical Functions

These subroutines take 70 percent of the overall required MOPS. Alone, the function used for search in fixed codebook requires one fifth of the computation load. By optimizing the execution of these subroutines on our DSP core we should drastically enhance the overall performance of the application. The model of the computation unit presented above supports multiple functional units, e.g. MAC units. Here, we limit the available parallelism by assuming an architecture based on two MAC and two ALU units, which is adopted by almost all today's high-end DSP processors. Fig. 5 shows that EFR makes a heavy use of multiply-based operations, so two MACs should improve overall performance by allowing two multiply operations per cycle.

Direct Memory Access architecture

Our ASIP model works in a memory to memory manner. It allows to load operands from memory, perform computation and store results back to memory in a single instruction. This feature is quiet unusual in DSPs, which rather use a load/store architecture. Each technique has its advantages and drawbacks. Direct Memory access often requires longer instruction pipeline, whereas load/store needs more registers. However, for algorithms whose inner loops frequently access memory, the latter often requires complex optimization techniques such as

software pipelining to produce efficient code. Fig. 7 shows that the load/store implementation of the EFR requires 5.3 MOPS more than for direct memory access. The EFR seems to be a memory-intensive algorithm and justify the choice for this architecture.

	Direct Memory	Load / Store
MOPS	8.0	13.3

Fig 7: Required MOPS for a single Mac DSP

Memory Structure

At first we have to define the memory bandwidth. Maximizing the use of the 2 Macs architecture requires four 16 bit data per cycle. Some algorithms are able to exploit the data locality to reduce the number of memory accesses needed. Unfortunately, the study of critical functions has shown that an effective bandwidth of four words per instruction was needed to take full advantage of the two Macs. Now we have to define the memory structure: number of memory banks , single or multi-ported, and word length. This is an important factor that will determine the final cost of the chip. Compared to its associated memory, the silicon area of a DSP core becomes more and more negligible as the integration density increase. As an example, Fig. 8 illustrates the cost and performance differences between single and double-port memories for a 0.35 process.

	Single Port	Double Port
Surface	1	1.7
Cycle time	1	1.1
Dyn. Power	1	3

Fig 8: Double Port Memory Cost

The study of critical functions shows that 32 bits operations as well as SIMD operations are unusual and do not justify the use of 32 bit memories. We use double-port memories for simultaneous accesses with random addresses. For sequential accesses, two single-memory banks can provide two data per cycle without the cost penalty of the double-port solution. In this case, data have to be dispatch between banks according to their address parity. To ensure the four words bandwidth, we need two double-port and four single-port memory banks, as shown in fig. 9.

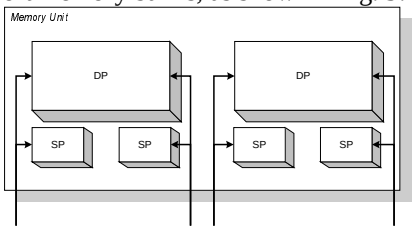

Fig 9: Memory Structure

Architecture Optimization

The first modification to our model consists to reduce hardware cost by keeping only one ALU. Indeed, the study of critical functions showed that it was enough to sustain computation throughput.

Fig. 10 compares the performance of the Bi-Mac architecture vs. Single-Mac. The overall MOPS required by Bi-Mac is **5.57**, vs. 8.36 for Single-Mac. It means a 35 percent gain. But we can see that certain costly functions as *search_10i40* and *cor_h* do not take great advantage of the bi-mac structure. They require other special architectural features to be speed up. Our study points out 3 main architectural enhancements that can significantly improve the execution of these critical functions:

1. Have an instruction able to perform two parallel computations (add or mac) with an immediate operand. Indeed, the most computations in the *search_10i40* function use an immediate as operand, that prohibits to use parallel instructions in classic DSP instruction-sets.
2. Add the array addressing capability to the Address Generation Unit. This feature is particularly useful in the two critical functions and allows to fetch data referenced as array elements: *tab[i][j].*
3. Allow to write a result to two memory locations simultaneously. This feature drastically improve the *cor_h* function execution.

We can verify on fig. 10 that these changes in architecture really speed up the *cor_h* and *search_10i40* functions. Fig. 11 shows the contribution of each of these optimizations on the overall MOPS count.

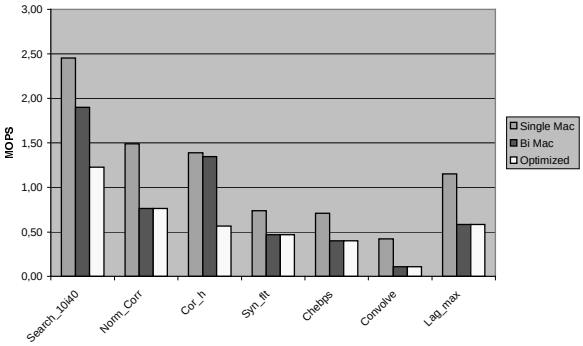

Fig 10: Critical functions speed up

Implementation of (1) and (3) is quiet cheap. For (1), we have to insert an heavy encoded instruction performing a 16 bit operand and two computation. It is feasible in a VLIW instruction-set. For (3), all we have to do is to configure the switch box to route a result from the CU to two memory busses , and add an associated managing instruction. Using (2) seems

to be more difficult. Array address computation needs a multiply-accumulate operation, which suppose to include a Mac in the AGU. Moreover, we have to encode not just one but two indexes for only one operand, which take a large place in the instruction word, even VLIW.

Architecture Enhancement	MOPS Reduction (%)
Bi Mac	33
Complex immediate instruction (1)	10
Array Addressing (2)	5
Simultaneous memory write (3)	2

Fig 11: MOPS Reduction

A reasonable solution consists to implement only the (1) and (3) optimizations. In this case, the final MOPS required for the seven critical functions of the EFR encoder is 4.56. Knowing that our evaluation do not take into account some powerful features found in DSPs like min/max, bit-manipulation instructions or conditional execution, we can reasonably expect that the proposed architecture will require around 6 MOPS for the complete encoder.

Comparison with other DSP processors

References [7] and [8] presents the implementation of the EFR vocoder on two powerful DSPs: the Siemens Carmel and the Motorola/Lucent Star Core. Carmel is a dual Mac, direct memory access DSP with a memory structure similar to the one shown above [9]. Its two Macs, two ALUs , and rich instruction set (bit manipulation and SIMD instructions) allow to execute the EFR encoder in 6.4 MOPS. The difference with our implementation is probably due to architectural enhancements we presented before. Even if this difference is not large, the ASIP solution is cheaper, with its single ALU and 32 bits (vs. 40 for Carmel) functional units. Moreover, further optimizations on the architecture can be done in order to reduce the cost and power consumption of the core, to exactly match the algorithm requirements: we can minimize the size and port number of the register-set, reduce the number of address pointers, remove unused instructions from the instruction-set, reduce the data switches complexity, etc.

Star Core is currently one of the most powerful DSPs with its four Macs based architecture. Results for the implementation of the EFR are similar to the Carmel ones. It seems to confirm what we notice during the study of the critical functions: the EFR algorithm does not offer a high amount of ILP for a multi-Mac architecture, and data dependencies often prohibits use of SIMD programming. This kind of architecture probably should take more advantage from 2D, SIMD intensive algorithms as found in image processing.

Conclusion

We have presented here an example of Algorithm/Architecture Adequation. Starting from the target application, the GSM EFR vocoder, we led a quantitative analysis which allowed us to include in the core design several architectural enhancements to speed up algorithm critical functions. In the same time, hardware model of the core can be reduced to exactly match the application requirements, saving power and silicon area. Thus we can define an optimized, low-cost ASIP architecture.

The use of a configurable model of ASIP core must permit a fast implementation of architectural ideas. Using configurable hardware macro-blocks allows to quickly get an optimized layout on any technological library [2]. Our future works rely on retargetable software tools (assembler and compiler) to manage the configurable architecture of our ASIP core model.

References

[1] Goossens G, Liem C., Cornero M., Nacabal F., “Embedded Software in Real-Time Signal Processing Systems: Application and Architecture Trends”, Proceedings of the IEEE, Vol. 85, no.3, March 1997

[2] Bajot Y., Mehrez H., “A Macro-Block Based Methodology for ASIP Core Design”, Proceedings of the ICSPAT, Orlando, 1999.

[3] GSM 06.60 (EN 300 726): “Enhanced Full Rate (EFR) speech transcoding”, ETSI, 1998

[4] GSM 06.53 (ETS 300 724): “ANSI-C Code for the GSM Enhanced Full Rate (EFR) speech codec”, ETSI, 1998

[5] GSM 06.54 (EN 300 725): “Test sequences for the GSM Enhanced Full Rate (EFR) speech codec”, ETSI 1998

[6] Eyre J., “Optimizing DSP Software for the Latest Processors” Lecture presented at ICSPAT/DSP World, September 15, 1998, www.bdti.com.

[7] Mingardon S., Niggebaum R., “Implementation of the Enhanced Full Rate vocoder on the CARMEL DSP Core”, Proceedings of the ICSPAT, Orlando, 1999.

[8] Halahmi D. & al., “GSM EFR Vocoder on Star Core 140”, ”, Proceedings of the ICSPAT, Orlando, 1999.

[9] Eyre J, Bier J., “Carmel Enables Customizable DSP”, Microprocessor Report, Volume 12, Number 17, 1998