Reusable objects for optimized DSP design

A. Barabanov(*), M. Bombana(**), N. Fominykh(*), G. Gorla(**), A. Terekhov(*)
(**) *ITALTEL- DRSI-RSC, Milano (Italy)*
(*) *TEPKOM- St. Petersburg (Russia)*

Abstract. The design of embedded, customized and programmable digital signal processors (DSPs) imposes new challenges to system designers. In this paper we define a semi-automatic approach to the problem of an optimal DSP architecture selection and subsequent algorithmic mapping. Two phases are envisaged: an algorithmic development phase and an architectural development phase. The former is based on the identification of general algorithmic objects and their formalization in a library. The main criteria for the isolation of such objects is reusability. A 'what-if' analysis utilizes the elements of the library to generate and evaluate alternative algorithmic partitionings for the chosen architecture. The latter phase applies different optimization criteria including code optimization and heuristics. Two test benches in the area of mobile phone systems are presented to highlight the potential benefits of the proposed approach to the design of industrial devices.

1. Introduction

The history of DSP goes all the way from boards and ASIC based *hardware* to algorithm-specific *software* running on general purpose microprocessors. The former group includes the first technological approaches to perform embedded digital signal processing operations, usually known for the specific function performed, like modem, compression, etc., and not for any special architecture adopted, processor-like or algorithm specific. The latter group performs signal treatment, in most cases as a batch process on saved waveforms, sometimes with real time rendering, seldom, only in favorable cases, with full real time performance.

At the intermediate stage, dealing with real time processing and exploiting today's state of the art, are located the programmable DSPs, that are used in different moods and flavors by different designers as *"function and algorithm specific ICs"*, coupling specific software with specific hardware. They implement the evolution of those older versions of the ASICs performing signal-processing operations at the earlier stages of the technology that were complemented with pre-programmed, non-volatile memories implementing algorithmic-specific routines.

The adoption of this technology forces the designers of DSPs and of systems adopting DSPs to make consistently similar choices for categories of similar algorithms, those that usually refer to applications falling into the same discipline or even leading to the same category of products. Some examples of this are different designs adopting the same processor for slightly different algorithms dedicated to the same application or similar processors, competing about performances, used for the same kind of algorithms in one specific applicative field.

In this paper we describe a semi-automatic approach to the identification of re-usable algorithm-based semantical objects, a set of primitive entities meaningful for the processing of algorithms, not (or not just) for structural descriptions. They will be organized in a user-friendly library to support the system designer in the task of the optimal DSP architecture identification and subsequent algorithmic mapping. In section 2 the state of the art of design methodology is sketched in order to identify the basic requirements and constraints coming from the users. These elements provide the rationale for the following activity, and set the priorities which lead to the definition of a design flow for mixed systems. In section 3 we describe a design phase based on the identification of reusable algorithmic objects and their formalization into a library. First a taxonomy of objects is defined, with a granularity of six levels of complexity. From the analysis of different classes of algorithms in the telecom domain the candidates for hw and sw trade off evaluation are identified. This analysis has been done both manually and in a semi-automated way. The identified elements, modeled in C++ as design primitives and provided with profiling information, are grouped in an algorithmic library on which any other algorithmic solution can be decomposed and re-assembled, as a result of a ”what if” analysis.

In section 4 different optimization procedures minimizing costs are applied, including abstract representations of time properties, code optimization, transformation of the assembler code, heuristics. In section 5 the LPC algorithm, applied for source coding/decoding in the GSM specification [1], and the MBE algorithm [2] are shortly described, and used as a test bench to highlight the benefits of the proposed algorithmic approach. Exploitation of reusability was the key driving factor of this analysis. Finally section 6 contains some conclusions and the outline of future work.

2. Users requirements for the design of mixed hw and sw components

The present situation, mid-way between high performance multipurpose processors and ROM-based ASICs, could be labeled as that of *multi-function DSPs.* Different programmable DSPs, dedicated to the same application, often adopt similar architectures, only adapting size and structure of memories and buses to the specific algorithms, and implementing a dedicated instruction set. Larger differences exist between DSPs dedicated to different applications; however, in many cases the architectures of the programmable DSPs are very similar, just some co-processing units or the operation set or the i/o communication can be different. The design of a specific DSP to achieve challenging performance in a definite market niche is time and manpower expensive, and thus it is usually tackled only for the larger markets. ASIC-like techniques involving libraries, reuse and customizability are not pursued, thus under-exploiting smaller market niches. Basically the overall picture is very similar to the ’good old days’ of custom ICs before the advent of semicustom: specialized DSPs are available only for big undertakings. And system designers have to adapt the same techniques to make products for different applications. No library for modular approach or easy tuning and redesign is available. As a consequence, according to our perception, the efforts for the progress of today’s semiconductor processing technology must be concentrated on hw and sw co-design techniques.

Due to such considerations it is clear that severe strategies must be adopted to minimize the impact on design time and quality. Key points of this strategy is the use of standards and reuse. In general, adopting standards makes integration easier, exploits available know-how in the field producing a higher return on investment. Designers’ acceptance is also guaranteed by a gradual modification of the design flow to introduce new tools to cope with new and complex applications.

Reuse can be applied at different levels, allowing a quicker design time, more flexible implementations, and basically a return of investments. In the design world, reuse is associated to the generation of libraries, where parametric hw or sw cells/routines are available to users. Anyway this concept can be applied also in other ways: using cores out of the shelf, or existing DSP cores, applying logic synthesis tools to generate hw/sw interfaces or specific execution units.

Summarizing all the requirements coming both from the designers and from the market, the definition of a co-design environment should guarantee:

1. the possibility for the designer to conduct an exploratory activity in the first and most abstract design phases, applying a 'what-if' analysis in the hw/sw partitioning;
2. a quick and automatic generation of correct by construction hw and sw solutions (through VHDL logic synthesis and compilation);
3. the application of standard languages as C, C++ and VHDL for specifying systems and interfaces;
4. the application at all level of the concept of reuse as basic element to provide return of investment, shorter design cycles and shorter time to market;
5. a nice integration in the existing design flows, exploiting as much as possible, what is already available to users.

3. Object identification at algorithmic level

The goal of this activity was that of defining algorithm bound primitive objects, and of designing architectures able to be tuned to subsets of the classes of objects. Software reusability is large but at a rather low level; hardware reusability is very restricted, and the high level of the blocks forces suboptimal adoptions. The use of algorithm specific blocks, like dedicated hw implementations of some instructions or co-processors, can provide the proper granularity without compromising performance (as for microinstructions) or architecture and power consumption (as adapting standard blocks). basically, to make profitable use of libraries, these have to be reusable at the highest level that does not include product specific convention (like frame format or protocol conventions) and is not so generic to be trivial (like bit addition, byte logic etc.).
Multiple steps were involved in this task:

1. identification of the criteria to drive the partitioning phase;
2. formalization of general rules to be applied in the partitioning phase;
3. implementation of a semi-automatic application of the rules of point 2;
4. definition of a taxonomy of modules according to complexity;
5. definition of an algorithmic library specialized for sets of similar algorithms.

Point 1 provides the rationale for this activity, points 2 and 3 implement a semi-automatic 'what-if' analysis, point 4 and 5 define the data-base on which such analysis is conducted.

An object oriented approach was selected to define the modules and to encapsulate the information in their structures. C is widely used for system level specification and C-like languages ([3],[4]) and C++ have been used extensively ([5],[6]) for the specification of mixed systems. So C++ has been selected for the implementation of the environment based on the preceding five points. The link between this design phase and the following steps in the design flow is shown in the next section.

Criteria were identified from an analysis of users' needs and market requirements:

reusability, i.e. frequent appearance of the procedure in classes of algorithms;

encapsulation, i.e. independence from the rest of the algorithm;

semantic consistence, i.e. the completeness of behavior.

Moreover to be useful to the following design phases, the objects share the following features:

1. several implementation methods are associated to one behavior, i.e. to one object;
2. hierarchical descriptions are allowed;
3. hardware resources are taken into account as parameters of the object;
4. behaviors are independent from subsequent hw implementations.

Parametrization is the key for heavy reuse. Each object has its own mathematical meaning (behavior) which is not connected to any hardware implementation. Anyway the computation of the I/O mapping can be conducted in different ways. For example, the representation of variables with fixed point numbers can have different number of digits. Numerical methods for the same mathematical function can be very different and therefore give different accuracy of the results (for example, the implementation of the autocorrelation procedure in time-domain and in frequency-domain; various forms of filter implementations, and so on). In this way, parametrization can cover very different aspects of the object definitions.

Moreover performance parameters, such as computation time, accuracy, required memory, etc., are associated to every object. All these are used in the transformational phase (dynamical tuning) and they will guide the choice of one implementation method in the final object oriented program. Alternatives are selected evaluating cost functions of the performance parameters. To this end the input algorithm is transformed into a special object scheme containing all possible alternative implementations.

The scheme looks like a graph in which nodes are objects and links correspond to implementation methods. An object scheme represents the source algorithm where no commitment to instances has been made for any object. Therefore the scheme has not the structure of a DSP network. The scheme may be treated as a general type graph because of recursive calls.

According to these requirements all objects were divided for into 6 levels;

Level 0. Logical operations with floating-point and fixed-point numbers. These operations are very close to hardware and they have a simple implementation. The main examples are shift, truncation and assignment operators.

Level 1. Arithmetic operations with floating-point and fixed-point numbers. These operations are more complicated then logical operations. Some of them may have different implementations in hardware. Consider as an example operations from the class FIXED used in the object oriented design of GSM algorithm (Section 5). The implementation of the arithmetic operations in this class is a mixture between integer and floating-point. The operands are considered as floating- point but no shift is made in arithmetic operation as it is for integer operands. This helps to control overflow and underflow errors at the stage of dynamic tuning.

Level 2. Complex arithmetic and additional functions of floating-point and fixed- point numbers. The fixed point arithmetic chosen in the final algorithm version reflects the hardware structure of the DSP processor. Although fixed complex arithmetic is usually not implemented in hardware (for example, complex MAC), these operations are very used in DSP algorithms.

Level 3. Matrix and vector arithmetic; sequential elementary signal functions and the basic procedures of signal processing like FFT and filtering. Sequential operations are inherent to signal processing and therefore special hardware tools should be taken into account on the stage of hw/sw partitioning. A special research has been conducted to select a set of widely used filters. As a result, the first order FIR and IIR filters, lattice filters, the convolution filters with both time and spectral implementations were described as objects.

Level 4. Object-oriented operations which are either standard in the signal processing theory or mathematically meaningful. They do not belong to Level 3 because of their complexity. Most of these operations correspond to well known macros collected in libraries in MATLAB, FORTRAN, C and other standard languages, for example butterfies, direct/inverse FFTs, etc.
Level 5. Specific signal processing functions. If a part of the input algorithm possesses all object features then it can be isolated and used in the design flow. Such non standard objects can be useful in specific application domains, for example Viterbi decoder, Shur recursive algorithm, etc.

Objects hierarchy, linked to structural definitions, is allowed. Each object can be composed of elements belonging to the same or lower levels. Objects of the first four levels are not related to a specific application domain, so maximal reuse is expected. These objects are rather basic (standard arithmetic and logical operations as well as their vector and matrix generalizations) and no ad-hoc rules are required for their identification. For the objects included in the two highest levels (4 and 5) the criteria for object identification can be extrapolated from a similar field, i.e. criteria of a search restriction when the best method for some algorithm implementation is being found.

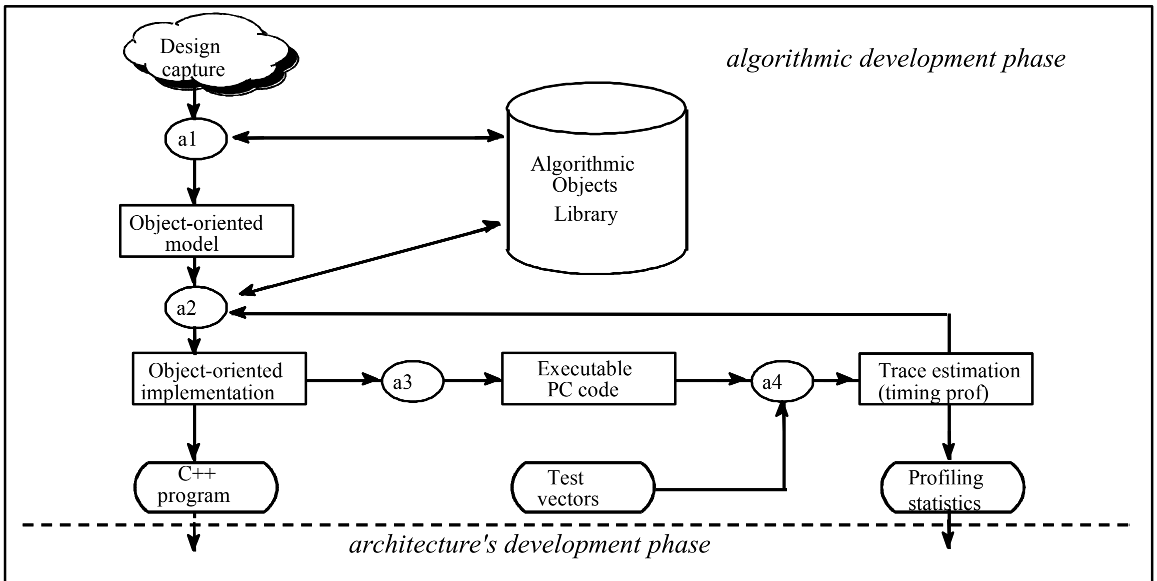
Design capture
algorithmic development phase
a1
Algorithmic Objects Library
Object-oriented model
a2
Object-oriented implementation
a3
Executable PC code
a4
Trace estimation (timing prof)
C++ program
Test vectors
Profiling statistics
architecture's development phase

Figure 1. Algorithmic design flow.

The traditional approach to the search restriction is a decomposition of the source algorithm: the algorithm description in detail is replaced by a block description which is essentially shorter and hence it admits an exhaustive search for optimization. As a result, the source algorithm is written by a flow chart that is customary for specialists who often use the method indicated for better understanding and analysis of the algorithm. Individual blocks along which a search is made are called objects. Thus the following rule is justified:

R1. Similar parts of programs form a single object.

The encapsulation condition requires the independence of the object from the rest of the program. Therefore R1 is constrained by the following condition:

C1. The length of input and output parameters which must be set whenever the object is called must be less than the length of the body of the object.

Thresholds for the comparison of such lengths are selected taking into account the following conditions: 1) frequency of the object in the set of algorithms; 2) memory restriction and restriction on the total number of objects in the system; 3) organization of a search in the object system; 4) complexity increase of the search procedure with increasing number of objects. R1 with condition C1 operates at an abstract level without taking into account the implementation in hardware. A further condition on thresholds is the following:

C2. Thresholds for accepting a new object according to C1 must be decreased if the possible object has relatively low complexity but represents a bottle neck for memory or speed.

C2 formalizes the feedback in the process of algorithm partitioning, dynamic tuning and code optimization.

The objects so identified are grouped in libraries. These libraries are of generic use, but some elements may be application domain dependent. The huge background in signal processing theory provides a wide set of pre-defined modules. Many functions have been extensively studied, and are the first candidates for the building of the object library. Reusability in these cases is guaranteed by the underlying mathematical theory. An algorithm having a standard name becomes a block which can be considered as a single operator in different DSP algorithms.

The automation of object selection involves two steps: the development of some methods of program re-engineering to apply rules and identify objects, and the design of a language for manipulating them, for optimizing their selection and for generating a final program in the object oriented style.

The entry point for this design phase is a description of the algorithm. The solution is independent from the adopted style: flow charts, mathematical formulas or a computable program in an algorithmic language are allowed. The final result of this design phase will an object- oriented program in C++.

The designer writes the specification of the algorithm using as much as possible elements of the library (figure 1). The ideal solution of constructing a flow chart made only of blocks taken from the library is seldom reached. When this is not possible, the library must be customized and new objects defined by the user.

Program re-engineering is applied in cases when the algorithm is described in a programming language. In some cases, for clearness sake or for efficiency reasons, a pre-compiler based on some heuristic rules is used prior to the application of the re-engineering program. The results of the automatic application of these rules to the test bench will be shown in section 5.

4. Architectural mappings and optimizations

Timing constraints are a basic part of the specification of any mixed hw/sw components. This information is lacking in the algorithmic analysis of previous section. Therefore we would like to improve our technology by adding special features to formulate time constraints specifications, taking into account the requirements dynamic statistics (profiling). Software is developed together with a statistically representative test set; the profiling characteristics received as a result of final program execution on this test set are necessary to create quantitative estimations of the alternative decisions.

This design phase has the goal of optimizing the cost of the final device varying the distribution between software and hardware, selecting a program structure (in a wide class of equivalent program transformations) and a hardware architecture (taking into account that the target device is intended just for this specified task). The proposed design flow is shown in figure 3.

Our methodology is based on the assumption that the input for the development process contains:

the precise definition of all external and internal events;

algorithms for all processes, including internal signals interaction;

clearly specified real-time restrictions for external events.

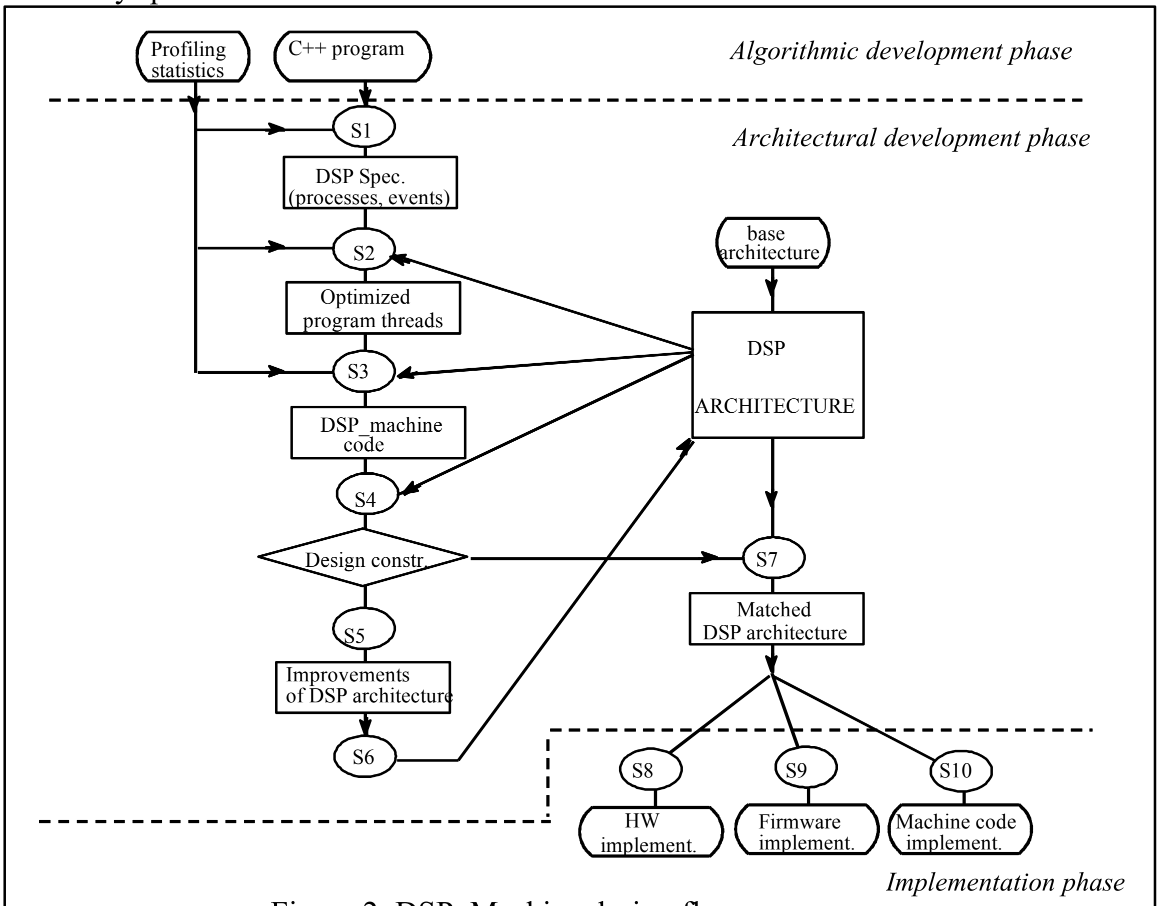

Figure 2. DSP_Machine design flow.

Algorithmic descriptions already contain a significant part of information necessary for the complete system specification of the device. For the needs of the next steps the specifications should be expanded with:

precise description of events handling (buffering, interruptions, scanning etc.);

indication of the temporal relations between signals;

The first point, though insignificant on an algorithm development phase, has a great influence on a target device complexity and must be strictly specified. For the second point, not only is required to specify these relations for the separate processes but for the whole system.

At this stage it is necessary to represent the algorithm in a maximum parallel form to give an opportunity of conveyer execution. Sometimes the initial algorithm must be redesigned to meet this requirements.

At the stage of global optimization (S2) traditional compilers optimization based on global analysis is executed. The result is a transformation of the initial program into an internal representation. Note that the stage may already use the architecture-dependent information about relative complexities of elementary operations. A problem connected with global optimization is the maximum static planning of processes: every external event generates a sequence of internal events (the transition in finite automata model); external events are asynchronous, but inherited sequence of internal events is mostly static. So all calculations inside this sequence can be planned statically, in particular, all the resources can be distributed statically.

At the stage of synthesis of the machine code (S3) the compiler uses a locally optimal translation maintaining the source program structure. The most crucial aspect is to load all processor equipment elements in accordance with the conveyer organization of the execution. In addition to methods used in compilers for RISC / VLIW architectures, we use the results of static planning of execution, trying to put different program chains onto different hardware elements.

An interpreter automatically customized to the current hardware architecture is used to check the conformity to the initial temporal restrictions (S4). This interpreter does not execute all the machine instructions. Only execution time is important here. At this stage we can rely upon static decomposition of processes with statically known separate statements frequencies and their relative costs in the current architecture. When the estimation results are negative (violation of temporal restrictions and excessive productivity) the flow proceeds to S5, where a set of possible updating of the architecture are proposed (special instructions on the base of the same equipment, additional functions for already presented processing blocks, additional hardware accelerators). In case of positive result the flow finalizes the processor architecture (S6).

In S6 the designer evaluates which proposals of stage S5 will be taken into account. According to that, the architecture is modified accordingly in all its aspects (e.g. compilers, interpreters etc. must be automatically tuned to new architecture). After the fulfillment of this stage, it is necessary to map the initial problem to the already updated architecture of the processor, i.e. pass through all stages of the considered technological cycle, from the stage of global optimization.

At the stage of final processor architecture consolidation (S7), the final refinements of processor architecture are executed in accordance with the requirements of the application, including the removal of unused instructions, the register size, the depth of procedure stacks, the optimization of coding, etc.

The design technology, based on the VHDL, is applied for hardware design (S8). In future, the possibility of automatic generation of the VHDL-programs corresponding to the given specifications of processor architecture can be considered. Although a great experience in microprogramming generation (for fixed structure processors) is available (S9), this task still is only partially automatic at the moment. Finally the stage of executable code generation (S10) is activated and all programs are compiled and linked together with a special dispatcher program (real time kernel).

5. A case study in digital communication systems: GSM and MBE

A. Description of the test benches. One of the most powerful speech analysis techniques is the method of linear predictive analysis (LPC) ([7],[8]). This method has become the predominant technique for estimating the basic speech parameters, such as pitch, formants, spectra, vocal tract area functions, and for representing speech for low bit rate transmission or storage. The importance of this method lies both in its ability to provide extremely accurate estimates of speech parameters, and in its relative speed of computation.

The basic idea behind linear predictive analysis is that a speech sample can be approximated as a linear combination of past speech samples. By minimizing the sum of the squared differences (over a finite interval) between the actual speech samples and the predicted ones, a unique set of predictor coefficients can be determined. Different computational techniques can be used for obtaining the values of LPC parameters, and are considered as variations in this algorithmic family ([9],[10],[11],[12],[13]).

This transcoding procedure has found application in the full rate traffic channel (TCH) of the pan European digital Mobile Radio (DMR) system (GSM - Full Rate Speech Transcoding) [Rec6.10]. In this case it performs the mapping between input blocks of 160 speech samples in 13 bit uniform PCM format to encoded blocks of 260 bits and from encoded blocks of 260 bits to output blocks of 160 reconstructed speech samples. The sampling rate is 8000 sample/s, leading to an average bit rate for the encoded bit stream of 13 kbit/s. The coding scheme is the so-called Regular Pulse Excitation - Long Term Prediction - Linear Predictive Coder (RPE-LTP).

The Multi-Band Excitation (MBE) Speech Model was recently developed ([14],[15]). This model uses a more flexible representation of the speech excitation than traditional speech models. As a consequence, it is able to produce more natural sounding speech, and it is more robust in the presence of acoustic background noise. This properties have made the MBE speech model a prime framework for the development of high quality low-rate speech coders. The name derives from the fact that a large number (up to twenty) of frequency bands are considered. Speech coders based on the MBE speech model estimate a set of model parameters for each segment of speech: a fundamental frequency, a set of voiced or unvoiced decisions (V/UV) which characterize the excitation signal, and a set of spectral amplitudes which characterize the spectral envelope. Once the model parameters have been estimated for each segment, they are quantized and transmitted to the decoder.

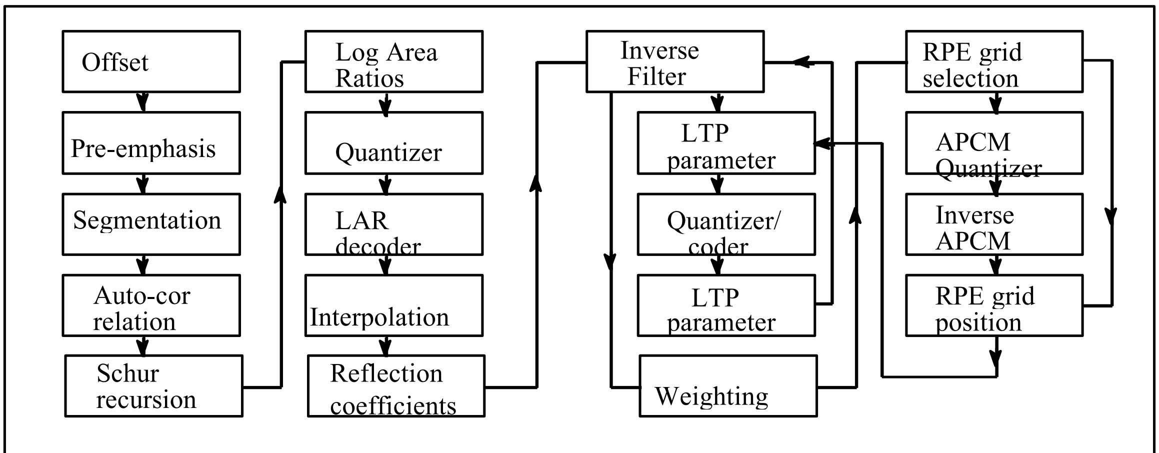

Figure 3. GSM Speech Encoder building blocks.

B. Description of the algorithms in objects. DSP layer in the GSM system consists of a transmitting and a receiving section. The former includes the Discontinuous Transmission Handler of the Transmitter (TX DTX), coding and modulation. The latter includes demodulation, decoding and the Discontinuous Transmission Handler of the Receiver (RX DTX). The modules which are subject to non trivial hw/sw partitioning are included in the following sub-blocks: equalization in demodulation-decoding; speech encoder and voice activity detector in TX DTX; speech encoder in RX DTX.

The objects identified in the specification of the GSM speech encoder/decoder are shown in Fig. 3. Each object has a well-defined mathematical meaning. Some of them include smaller objects as their sub-parts. The level 3 objects include the following elementary DSP functions: vector segmentation, quantizer, de-quantizer, LTP gain decoder, pushing the buffer, substitution of a vector segment. Level 4 objects consist of the standard DSP functions: autocorrelation, LAR update (piece-linear coding), sequential quantizer, interpolation, LTP gain coder, APCM quantizer. For the most part these objects consist of sequential operations of Levels 0 and 1. The following specific objects belong to Level 5: Schur recursion, LTP parameters update, signal compression by decimation.

The objects extracted satisfy the requirements of reusability, encapsulation and semantic consistence indicated in Section 1. The autocorrelation and Schur recursion are used twice in the voice activity detector block inside TX DTX. Moreover, the speech decoder system can be as a part of the speech encoder and includes the same objects. The classes FIXED, COMPLEX and MATRIX, defining the objects belonging to Levels 1 and 2, implement various kinds of arithmetic operations in order to be tuned to specific hardware or for program accuracy improving and debugging. This proves objects reusability. The encapsulation requirement is essential for the objects of Levels 4, 5. The number of internal variables in the extracted objects is greater than the number of input and output data. Moreover, the computations have some specific features and they are independent of the rest of the algorithm. Finally, semantic consistence is self-evident. These blocks are considered as natural units in the investigation of the algorithm properties.

The time profiling allows to quickly identify the units crucial for time constraints. The objects "Inverse filter A(z)" and "LTP parameters" take 65 per cent of the computing time, so they are good candidates for hardware implementation.

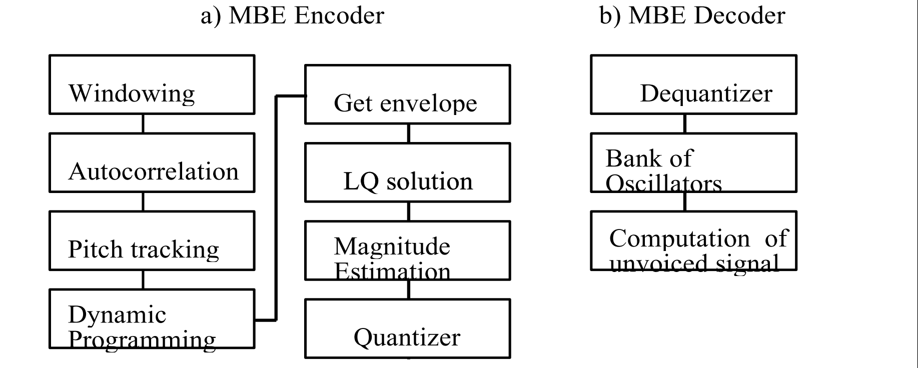

Figure 4. MBE building blocks.

Also the objects of the MBE vocoder/decoder (figure 4) have a hierarchical structure composed of objects of the lower levels. The standard procedures of FFT, Hamming windowing, vector normalization, norm evaluation and vector multiplication (level 3) are used several times. Level 2 objects include: pitch tracker, dynamic programming, quantizer (log-increment and uniform), LQ estimates (constant and piece-wise), magnitude estimation. The decoder has a bank of harmonic oscillators, a similar quantizer/de-quantizer object and also standard tuning harmonic signal and piece-linear interpolation. Some other objects are specific and belong to Level 5: errors vs pitch period, get residuals, direct pitch estimate, get envelope, unvoiced envelope update, - from the encoder, and unvoiced signal computation from the decoder.

Shifts, cycle shifts and other operations of Level 0 are used in both RPE-LTP and MBE vocoder algorithms. The sequential arithmetic operations like norm evaluation, vector multiplication and quantizer appeared several times as subsystems. Despite the relatively small size of the programs some basic signal processing operations appear several times. They are FFT, autocorrelation, Schur algorithm, interpolation.

Each extracted object has a clear semantic. It was noteworthy that a purely syntactic analysis of the program, based on the rules of section 3, resulted in a partitioning very similar to that done manually by the programmer. The MBE encoder program contains 19 objects. 16 of them were exactly extracted by the re-engineering program which identified totally 28 candidates for separate procedures. For the MBE decoder the re-engineering program found 6 objects over 8 in a range of 20 candidates. The superfluous candidates are rejected in an interactive session with the user. These numerical results confirm that the partitioning the program into objects satisfying reusability, encapsulation and completeness conditions closely matches the design steps of an experienced designer

7. Conclusions and future work

The results obtained up to now show that the presented approach is promising for solving some of the open problems in the codesign area. We focused on the specification phase, due to the relative weight that this phase covers in the telecom domain. Quick identification of data-path modules already described and profiled in a library, allows quick definition of a partitioning strategy and an early indications for binding. Future research will be devoted to expand the second phase of the design flow here proposed (architectural mapping and optimizations), and to link it with the algorithmic approach to exploit synergies in view of reducing design time and enhancing quality for the manufacturing of mixed hw/sw components.

References

[1] GSM Recommendation, '13 kbit/s Regular Pulse Excitation - Long Term Prediction - Linear Predictive Coder for use in the Pan-European Digital Mobile Radio System', CEPT/CCH/GSM 06.10, Sept. 1988

[2] John C. Hardwick, Jae S. Lim, 'The application of the IMBE speech coder to mobile communications', Proc. ICASSP '91, 1991

[3] OLYMPUS

[6] COSYMA

[5] C. Weiler, U. Kebschull and W. Rosenstiel, 'C++ base classes for specification, simulation and partitioning of hw/sw systems', Proc. ASP-DAC '95, 1995

[6] G. Kock, U. Kebschull and W. Rosenstiel, 'A prototype environment for hw/sw codesign in the COBRA Project', Proc. 3rd Workshop on hw/sw Codesign, 1994

[7] L.R. Rabiner and R.W. Schafer, 'Digital Processing of Speech Signals', New York, Prentice-Hall, 1978

[8] J.D. Markel, A.H. Gray, 'Linear Prediction of Speech', New York, Springer-Verlag, 1976

[9] Thomas Parsons, 'Voice and Speech processing', New York: McGraw-Hill, 1986

[10] Kevin T. Malone, Thomas R. Fischer,'Trellis-searched adaptive predictive coding of speech', IEEE Trans. on Speech and Audio Proc., Vol.1, n.2, Apr 1993

[11] Roy C. Snell, Fausto Milinazzo, 'Formant location from LPC analysis data', IEEE Trans. on Speech and Audio Proc., Vol.1, n.2, Apr 1993

[12] W.P. LeBlanc, B. Bhattacharya, S. A. Mahmoud, 'Efficient search and design procedures for robust multi-stage VQ of LPC parameters for 4 kb/s speech coding', IEEE Trans. on Speech and Audio Proc., Vol.1, n.4, Oct 1993

[13] Alv I. Aarskog, Hans C. Guren, 'Predictive coding of speech using microphone/speaker adaptation and vector quantization', IEEE Trans. on Speech and Audio Proc., Vol.2, n.2, Apr 1994

[14] Daniel W. Griffin, Jae S. Lim, 'Multiband Excitation Vocoder', IEEE Trans. on ASSP, Vol. 36, n. 8, Aug. 1988

[15] Daniel W. Griffin, Jae S. Lim, 'A new model-based speech analysis/synthesis system', Proc. ICASSP '85, Tampa, Mar. 1985