SpecC System-Level Design Methodology Applied to the Design of a GSM Vocoder

A. Gerstlauer, S. Zhao, D. Gajski

Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA
{gerstl,szhao,gajski}@ics.uci.edu

A. Horak

Architecture and System Platforms
Motorola Semiconductor Products Sector
Austin, TX 78729, USA
Arkady.Horak@motorola.com

Abstract— In this paper we describe the steps and transformations of the SpecC system-level design methodology applied to the example of designing and implementing a voice encoding/decoding system with the purpose of demonstrating this design methodology on an industrial size example. Starting with the system specification, the design is gradually refined down to an optimal hardware/software implementation. The results show that the well-defined models, steps and transformations of the SpecC methodology lead to a significant productivity gain.

I. Introduction

It is a well-known fact that we are currently facing an increasing gap between the productivity of design teams and the possibilities offered by technological advances. Together with growing time-to-market pressures, this drives the need for innovative measures to increase design productivity by orders of magnitude.

The two commonly accepted solutions for this problem are raising the design abstraction to the system level and reusing *intellectual property* (IP) components. With this background in mind, we developed our IP-centric system-level design methodology which is based on the *SpecC* system-level design language. The methodology was specifically geared towards support by a set of system-level design automation tools.

This paper describes the SpecC system-level design methodology. We will show the different steps and transformations taking a specification down to an implementation using the example of a voice encoding/decoding system. The example was chosen to demonstrate the SpecC methodology on an industrial size design. For detailed information about this project the reader is referred to [1].

The voice encoder/decoder (*vocoder*) application used is part of the European GSM standard for cellular telephone networks. The lossy codec scheme was originally developed by Nokia and the University of Sherbrooke [2] and is based on widely used algorithms for speech encoding [3]. The so called *Enhanced Full Rate (EFR) speech transcoding* is standardized by the European Telecommunication Standards Institute (ETSI) as GSM 06.60 [4].

The rest of the paper is organized as follows: Section II briefly lists related work in the area of system-level design. Section III then gives an overview of the models and steps that comprise the SpecC methodology. In Section IV the specification of the vocoder in the SpecC language is described. Architectural exploration is shown in Section VI. Section VII then deals with the refinement during communication synthesis. The results after final simulation are presented in Section VIII. Finally, Section IX draws some conclusions and summarizes the paper.

II. Related Work

In the past, research in the area of hardware/software codesign has produced a number of codesign environments with focus on the hardware/software partitioning problem given a specification in an input language. Cosyma [5] and Vulcan [6] are early examples which use extensions of C (called C^x and HardwareC, respectively) for system specification. Other examples of codesign environments are Cosmos [7], Polis [8] and SpecSyn [9] (based on input languages SDL, Esterel and an extension of VHDL called SpecCharts, respectively), to name just a few. In general, those environments are limited in their design complexity to a target architecture template of a processor and an ASIC.

More recent work has extended the scope to distributed, heterogeneous multiprocessors with general target architectures consisting of an arbitrary network of processing elements. Prakash and Parker [10], Yen and Wolf [11], and Kalavade and Lee [12] developed algorithms for automatic partitioning into such general target architectures.

Interface and communication synthesis is targeted by the Chinook [13] and CoWare [14] systems. These environments focus on the automatic generation of the interface between hardware and software from a specification.

Recently, a couple of efforts have emerged that propose new languages for system-level specification and design. Both SystemC, which evolved from the Scenic [15] project into an open initiative, and CynLib, which was developed by CynApps, Inc., are C++ class libraries that add features to model hardware in C++. While both libraries turn C/C++ into a hardware description language and therefore, similar to SpecC, enable a shift from RTL to programming language based system-level design, the SpecC approach is different. SpecC extends ANSI-C by a set of carefully devised constructs required for system-level design. For simulation, these constructs are automatically translated into a C++ model that is very similar to a SystemC or CynLib description. Instead of writing the C++ model manually, the user is presented with a more abstract and powerful interface. In addition, it is much easier for synthesis tools to "understand" the explicit SpecC specification compared to the simulation-oriented C++ models.

III. SpecC Methodology

A methodology is a set of models and transformations that refine an initial, functional system specification into a detailed implementation. The SpecC system-level design methodology is depicted in Fig. 1 [16, 17]. It is based on four well-defined models: a specification model, an architecture model, a communication model, and an implementation model. The methodology is complemented by the SpecC system-level design language [16, 18] which is used to describe the models at each step. The SpecC language is a superset of ANSI C with explicit support for required features like concurrency, hierarchy, timing, communication, synchronization, etc.

The methodology starts with an executable specification written in SpecC. This specification model describes the desired functionality along with the requirements and constraints in an abstract and implementation-independent way.

The system-level synthesis flow consists of two major tasks: architecture exploration and communication synthesis. During architecture exploration the system architecture—a set of components and their connectivity—is allocated and the specification is partitioned onto the components and busses of the architecture. Scheduling determines the order of execution of the parts assigned to the inherently sequential components. Communication synthesis refines the abstract communication between components into a detailed implementation over the selected bus protocols, synthesizes interfaces and protocol transducers as necessary, and inlines protocols into synthesizable components.

The resulting communication model is then handed off to the backend tools for compilation of the software parts and behavioral synthesis of the custom hardware components.

At each step of the methodology, the current design is represented by a model in SpecC. After each refinement step a corresponding SpecC model of the design is automatically generated which reflects the decisions made by

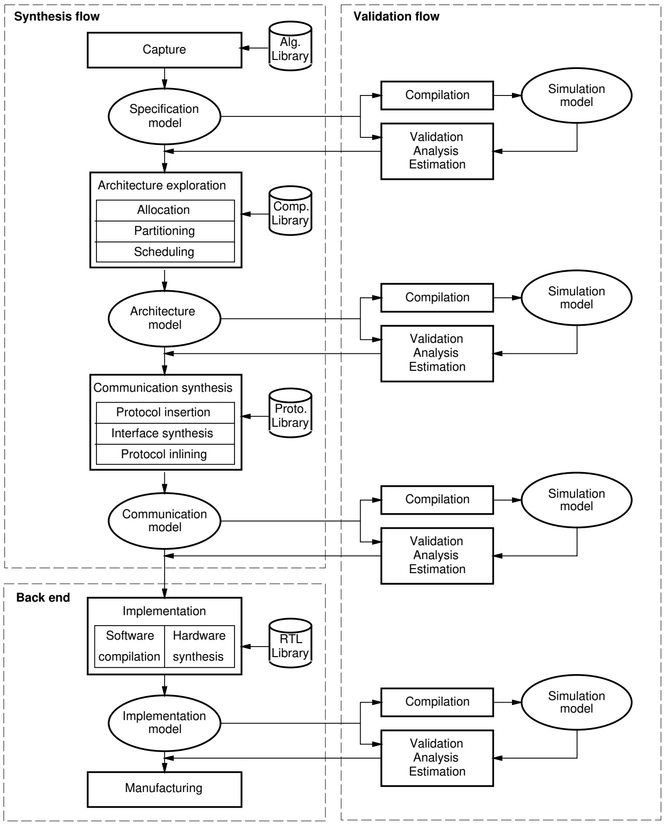

Fig. 1. SpecC methodology.

the designer through an interactive interface or with the help of automation algorithms.

In the validation flow that is orthogonal to the synthesis flow, simulation, analysis, estimation and verification of the SpecC models generated after each task are performed. Models are analyzed to estimate design metrics like performance, cost or power consumption. Each model is executable and can be simulated to validate design correctness using one testbench. Finally, the well-defined nature of the models and transformations enables formal methods for verification of design properties, for example.

IV. Specification

The first step in any design process is the specification of the system requirements, including both functionality and design constraints like performance, power consumption, etc.

As mentioned in the introduction, in the SpecC methodology the specification is formally captured and written in the SpecC system-level design language. In contrast to informal specifications (e.g. plain English), a formal specification unambiguously describes the system requirements, and can be executed for simulation or formally verified in order to validate the requirements. In

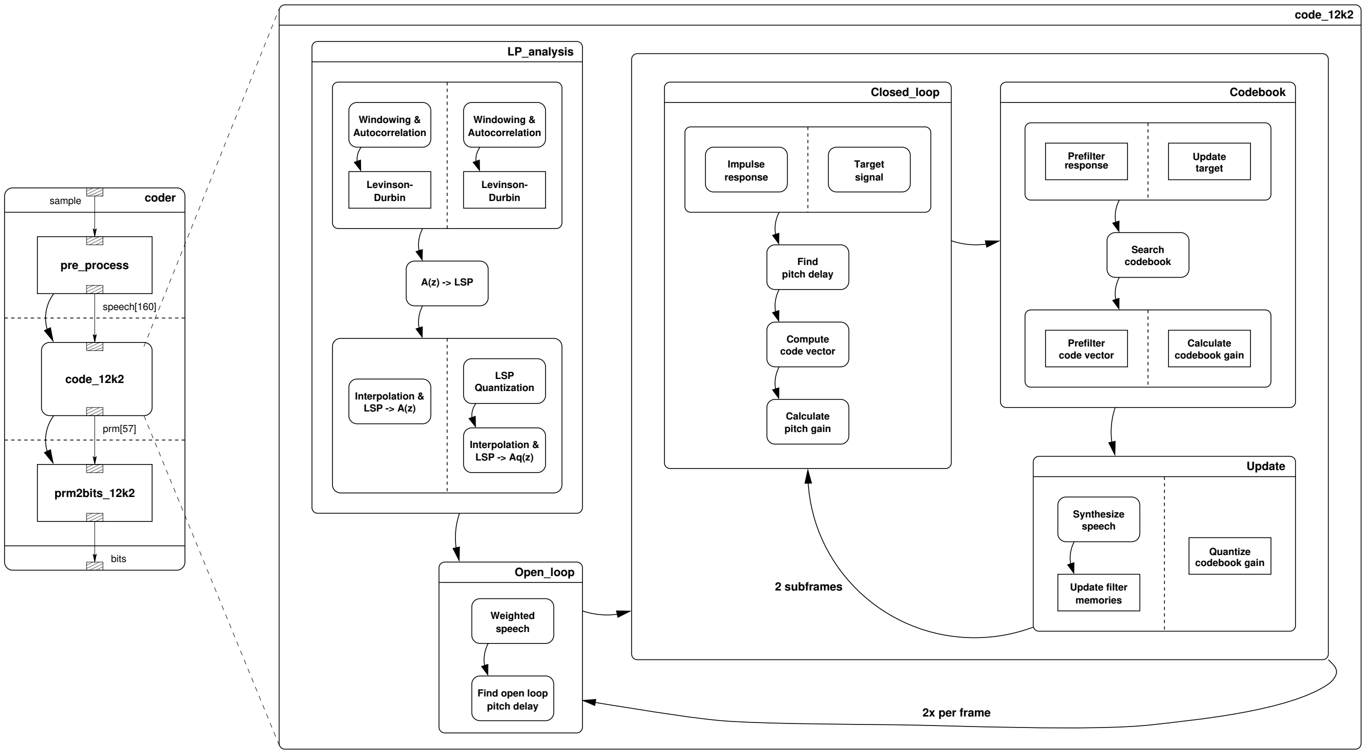

Fig. 2. Encoding.

addition, it serves as the input to the synthesis and exploration stages without the need for tedious rewrites.

The initial system specification written in SpecC should be as abstract as possible and free of any implementation details. Capturing the specification in a natural, clear and concise manner significantly eases understanding for both the human designer and the automation tools. Supported by the SpecC language, essential features of the specification, like parallelism, structural and behavioral hierarchy, or encapsulation and separation of communication and computation, should be expressed explicitly.

A. Vocoder Functionality

The GSM 06.60 standard for the EFR vocoder is accompanied by a bit-exact reference implementation of the vocoder functionality consisting of 13,000 lines of C code. Together with the standard document, this code defines the required functionality and was therefore used as the basis for the SpecC specification.

At the top level, the vocoder consists of independent coding and decoding behaviors running in parallel. Encoding and decoding transform a stream of speech samples at a rate of 104 kbit/s into an encoded bit stream with a rate of 12.2 kbit/s, and vice versa. Coding is based on a segmentation of the incoming speech into frames of 160 samples corresponding to 20 ms of speech. For each speech frame the coder produces 244 encoded bits.

The SpecC block diagram of the encoding part is shown in Fig. 2. Note that for simplicity only the first levels of the behavior hierarchy of the encoding part are shown. All together, the SpecC description of the vocoder contains 43 leaf behaviors. See [1] for a detailed description and discussion of the SpecC specification model.

At the top level, pre-filtering and framing, speech coding, and bit serialization run in a pipelined fashion. At the next level, the first step in the coding process is an extraction of linear-prediction filter parameters. Each frame is then further subdivided into subframes of 40 samples (5 ms). In two nested loops, open- and closed-loop analyses of pitch filter parameters and an exhaustive search of a predefined codebook are performed, followed by a filter memory update step.

In contrast to the reference implementation, the SpecC specification describes the vocoder functionality in a clear and precise manner. For example, available parallelism or behavior dependencies are explicitly shown. This greatly eases understanding and therefore supports quick exploration of different design alternatives at the system level in the first place. At each level, the SpecC specification hides unnecessary details but explicitly depicts the major aspects, focusing the view of the user and the tools onto the important decisions. On the other hand, SpecC being build on top of ANSI-C made it possible to resuse the C code of the reference implementation for each basic algorithm at the leaves of the SpecC behavior hierarchy.

B. Vocoder Constraints

In addition to the constraint of 20 ms for encoding and decoding a complete frame, the GSM vocoder standard specifies a constraint of 30 ms for the total transcoder delay when operating coder and decoder in back-to-back mode. According to the standard, back-to-back mode is defined as passing the parameters produced by the encoder directly into the decoder as soon as they are produced. The transcoder delay is defined as the time from receiving a complete speech frame to synthesizing the last speech sample of the reconstructed frame. Hence, the transcoder delay constraint translates into a maximum delay of 10 ms for encoding and decoding the first subframe.

V. Analysis and Estimation

Accurate estimates of a wealth of design metrics like performance, power consumption, area/cost, etc. are crucial for effective architecture exploration, i.e. for allocation of and mapping on a system architecture. Exploration decisions made by the user or automated tools are based on an accurate assessment of the estimated design quality. On the other hand, in order to be able to explore a large part of the design space in a short amount of time, quick feedback is required. The evaluation of design metrics is performed at different stages of the design process with different trade offs of accuracy versus runtime.

A. Profiling

During functional simulation of the specification, the code is profiled to extract information about the dynamic and static behavior of the application. The target-independent specification characteristics are used for analysis of target architecture requirements. In addition, profiling data is back-annotated onto the design for user feedback and as input to other tools.

Initially, the vocoder specification was analyzed to obtain profiles about the relative complexity of the different parts. In terms of performance, for example, a weighted sum of the worst-case operation counts per behavior combined with a timing constraint gives the relative computational complexity in terms of weighted million operations per second (WMOPS).

Fig. 3 shows the WMOPS data for the different parts of the speech encoding based on a 20 ms constraint per frame. For each of the major parts the total WMOPS per frame and the WMOPS for the contributing subparts are shown. In all cases, the diagram includes the data for a single execution of the behavior and the sum over one complete frame. The decoding task which is not shown here has a total complexity of appr. 1.3 WMOPS. Therefore, it's complexity can be neglected compared to the encoding task.

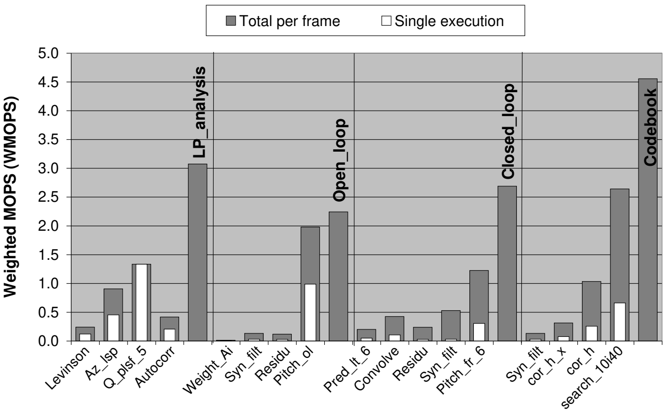

Fig. 3. Computational complexity of coder parts.

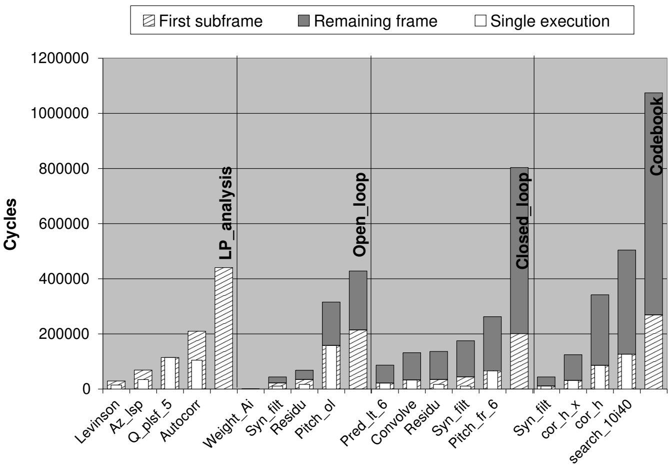

Fig. 4. Estimates of performance of coder parts.

B. Estimation

Using the profiling data, estimation of design metric values on a given target is performed. Retargetable estimators compute design metric values for a wide range of hardware and software implementations based on models of the target components in the IP library. Values are estimated statically without the need for time-consuming simulations.

After allocation of the vocoder target architecture, estimates of the vocoder performance running on the chosen target processor (DSP56600, see Section A) were computed by compiling and statically simulating the code on the target processor. The resulting number of cycles for each behavior of the encoding task are shown in Fig. 4 with a maximal processor clock frequency of 60 MHz.

VI. Architecture Exploration

A. Allocation

The first task of the architectural exploration process is the allocation of a system target architecture consisting of a set of components and their connectivity. Allocation selects the number and types of processing elements (PEs), memories and busses in the architecture, and it defines the way PEs and memories are connected over the system busses. Components and protocols are taken out of a library and can range from full-custom designs to fixed IPs.

For the vocoder, allocation initially started out with a least-cost solution based on a pure software implementation using a single, low-cost standard processor. The specification profiles showed that the standard is based on a 16-bit fixed-point implementation with an operation mix typical of signal-processing applications (e.g. multiply-accumulate, array accesses, tight loops with fixed bounds). For the vocoder, we selected the Motorola DSP56600 processor [19] out of the DSPs available from Motorola as the one satisfying those criteria.

However, execution time estimation determined that a pure software solution would not satisfy the timing constraints (20 ms per frame, 10 ms per subframe) even with the DSP running at theoretical peak performance. Hence, reallocation became necessary. Since the specification is inherently sequential in nature, a speedup through parallel execution of tasks on multiple components could not be expected. Therefore, it was decided to add a custom hardware component in order to reduce execution time by implementing parts of the specification in hardware.

B. Behavior Partitioning

After an architecture has been allocated, the first step in implementing the specification on the given architecture is to map the SpecC behaviors onto the architecture's processing elements. In the refined model after behavior partitioning an additional level of hierarchy is inserted with top-level behaviors representing the components of the architecture. Formerly local variables used for communication between behaviors mapped to different components now become global, system-level variables. Behaviors and channels are added for synchronization between behaviors mapped to different components in order to preserve the execution semantics of the original specification.

In case of the vocoder, a least-cost partition was obtained by gradually moving tightly coupled groups (minimizing communication overhead) into hardware until the constraints were satisfied. Mapping the codebook search, the most critical part of the specification, into hardware was sufficient to obtain a valid solution. The codebook search is sped up by an estimated factor of 10 in a hardware vs. software implementation.

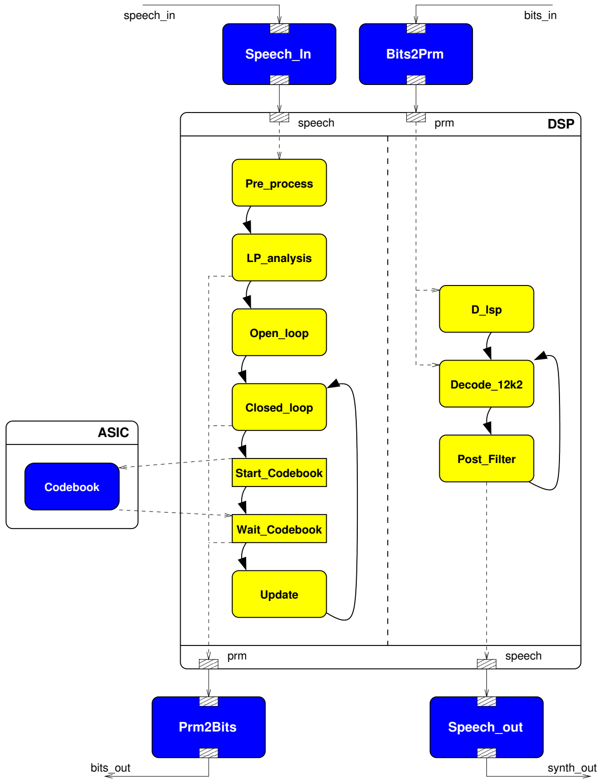

Fig. 5. Vocoder model after behavior partitioning.

The SpecC model after behavior partitioning for the vocoder is shown in Fig. 5. The DSP core running encoding and decoding tasks is supported by a custom coprocessor for the codebook search and four custom hardware I/O components. Synchronization behaviors (`Start_Codebook` and `Wait_Codebook`) have been added as part of the encoding task to coordinate execution between the DSP and the custom hardware.

C. Variable Partitioning

After behavior partitiong, communication between behaviors mapped to different PEs is performed via global, shared variables. Global variables have to be assigned to local memory in the PEs or to a dedicated shared memory component. In the refined model after variable partitioning, global variables are replaced with abstract channels and code is inserted into the behaviors to communicate variable values over those channels.

Due to the high variable access frequencies in the vocoder, local copies of the data arrays communicated between hardware and software components were kept in each PE. Code was inserted to exchange modified array contents over a corresponding global message-passing channel at synchronization points between hardware and software.

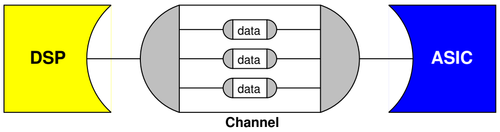

Fig. 6. Vocoder model after channel partitioning.

D. Channel Partitioning

Channel partitioning is the process of mapping and grouping the abstract, global communication channels between components onto the busses of the target architecture. In the refined model, system busses are represented by additional top-level channels and channel partitioning is reflected by hierarchically grouping and encapsulating the abstract, global channels under the top-level bus channels.

In case of the vocoder, there exists only one bus in the system architecture which connects the DSP to the custom hardware components. Therefore, all communication is mapped to that bus. In the resulting SpecC description of the refined vocoder, a single channel representing the system bus is inserted at the top level. All components are connected to the bus channel and all abstract channels for communication between behaviors are grouped under the top-level channel (Fig. 6).

E. Scheduling

Scheduling determines the order of execution of behaviors that execute on inherently sequential PEs. Scheduling may be done statically or dynamically. In static scheduling, each behavior is executed according to a fixed schedule. In the refined model after scheduling, behaviors inside each component are exectued sequentially according to the computed schedule. Redundant synchronization between the behaviors is removed during optimization. In dynamic scheduling, the execution of behaviors on each component is determined at run-time. An application-specific run-time scheduler is created during refinement.

Fig. 7 shows the scheduling of the parallel coding and decoding tasks running on the DSP core. Due to the dynamic timing relation between encoding and decoding tasks, a dynamic scheduling scheme is implemented. The coding task represents the main program which executes in synchronization with the arrival of new speech input frames. Whenever a new packet of encoded bits arrives for decoding, the encoding task is interrupted in order to execute the corresponding part in the decoding sequence, and encoding resumes after the decoding step is finished.

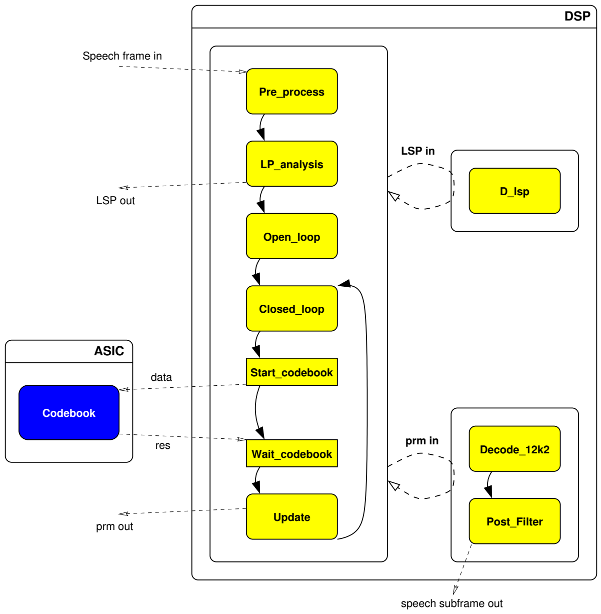

Fig. 7. Dynamic scheduling of vocoder tasks.

VII. Communication Synthesis

The purpose of communication synthesis is to refine the abstract communication in the architecture model into an actual implementation over the wires of the system busses. This requires insertion of communication protocols for the busses, synthesis of protocol transducers to translate between incompatible protocols, and inlining of protocols into hardware and software.

A. Protocol Insertion

Fig. 8 shows the vocoder model after insertion of the DSP56600 bus protocol for the system bus. The bus protocol is modeled as a SpecC channel in the protocol library. The protocol channel is inserted into the top-level bus channel and all communication over the system bus is implemented using the primitives provided by the protocol.

The processor IP is replaced with a bus-functional model of the processor taken out of the IP library. The IP model in the library is encapsulated in a SpecC wrapper which encloses the fixed IP protocol and provides an abstract, canonical interface to the IP. A protocol transducer component is inserted which bridges the gap between the wrapped IP component and the bus. Note that in this case the transducer will be optimized away during protocol inlining since IP and bus protocol are equivalent.

The clear separation between communication and computation enables replacement of a general component with an IP model plus wrapper and transducer at any stage of the design process. The wrapper specifies how to interface the IP model with the rest of the design. For simulation purposes, any model of the IP component that provides a suitable programming interface can be hooked into the SpecC simulator through the wrapper.

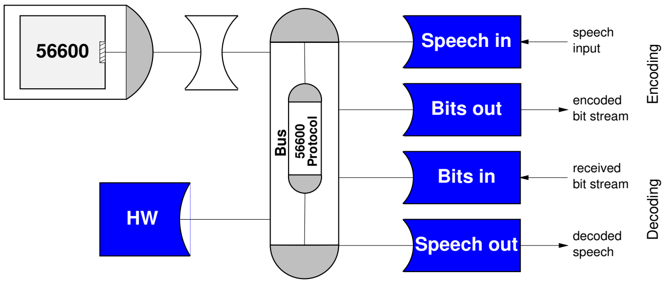

Fig. 8. Vocoder model after protocol insertion.

B. Protocol Inlining

Protocol inlining is the process of inlining the channel functionality into the connected components and exposing the actual wires of the busses. The communication code is moved into the components where it is implemented in software or hardware. On the hardware side, FSMDs that implement the communication and bus protocol functionality are synthesized. On the software side, bus drivers and interrupt handlers that perform the communication using the processor's I/O instructions are generated or customized.

The communication model of the vocoder as the result of protocol inlining is shown in Fig. 9. In case of the vocoder, all data transfers on the processor bus are initiated by the DSP. High-level handshaking and synchronization between hardware and software is realized using interrupt-based handshaking. For details about the implementation of the hardware/software interface and communication see [16, 1].

VIII. Results

The communication model is a bus-functional model in which bus transactions are modeled bit-exactly and with accurate timing, but components internally are still modeled at a functional level and annotated with estimated delays only. The communication model is handed off to the traditional backend tools for further refinement of the components.

For the software parts of the vocoder, C code was generated which was then compiled into assembly code for the Motorola DSP. For the hardware part behavioral synthesis was performed to create an RTL description of the custom hardware. Details about the backend process, the vocoder design and the methodology in general can be found in [16, 1].

In the final implementation model, both communication and computation is timing-accurate. Components are replaced with their cycle-accurate models as the result of the backend process or as stored in the library in case of hard IPs. For the vocoder, the processor was replaced with a model that integrates an DSP56600 instruction set simulator (ISS) executing the compiled code into

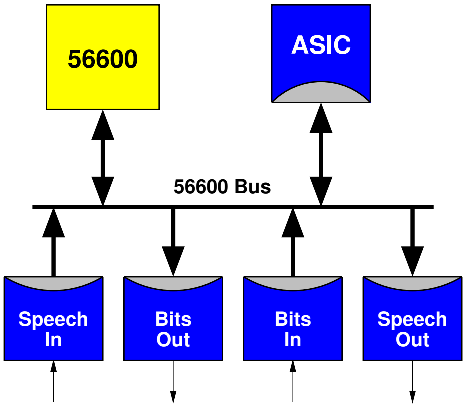

Fig. 9. Vocoder communication model.

the SpecC simulation. For custom hardware, the components were replaced with behavioral and structural RTL models, i.e. FSMD descriptions and RTL netlists.

In order to validate performance, a simulation of the final system model is needed. To avoid costly and time-consuming cosimulation of the whole implemenation model, a mixed communication/implementation model was used for simulations. For example, the cycle-accurate ISS model of the processor was simulated together with bus-functional models for the custom hardware. Similarily, hardware implementation models are validated using a bus-functional model of the processor for simulation.

Table I lists the simulation results in relation to the constraints as given by the vocoder standard, based on a maximal processor clock frequency of 60 MHz. All constraints are satisfied, leaving room for additional optimization by lowering the clock frequency in order to reduce power consumption, for example.

TABLE I
Worst-case delays for vocoder in back-to-back operation.

	Cycles	ms	Constraint
First subframe	366809	6.11	10 ms
Total frame	642351	10.71	20 ms

IX. Conclusions

In this paper we demonstrated the SpecC methodology on a vocoder example. Starting with the specification, the design was taken down to an actual implementation through a series of well-defined steps and transformations. The SpecC source code of the specification, architecture and communication models can be downloaded from the SpecC web page [20].

The well-defined nature of the models and transformations based on the SpecC language helps focusing design efforts on central issues, provides the basis for design automation tools, and enables application of formal methods, e.g. formal verification or equivalence checking.

The final implementation of the vocoder consists of 70,500 lines of compiled assembly code and 45,000 lines of synthesized RTL code. The design of the vocoder was accomplished by two people working on the project part-time over the course of six months. Simply following the well-defined steps of the SpecC methodology helped to reduce the design effort significantly.

Since the corresponding tools are not available yet, the implementation was done mostly manually at this point. Our current research is aimed at providing the envisioned tool support. With the availability of automated tools that will cover a large part of the tedious and error-prone synthesis tasks the time-to-silicon will be reduced even further. The time spent on the actual design tasks of the vocoder project was about 12 weeks only.

Acknowledgments

The authors would like to thank Motorola for supporting this project. Also we would like to thank Lukai Cai, Hongxing Li from UCI and Justin Denison, Mike Olivarez from Motorola for help in synthesis of the codebook search.

References

[1] A. Gerstlauer, S. Zhao, D. Gajski, A. Horak, *Design of a GSM Vocoder using SpecC Methodology*, University of California, Irvine, Technical Report ICS-TR-99-11, February 1999.

[2] K. Järvinen *et. al.*, "GSM enhanced full rate speech codec," *Proceedings ICASSP*, pp. 771-774, 1997.

[3] R. Salambi *et. al.*, "Design and description of CS-ACELP: a toll quality 8 kb/s speech coder," *IEEE Transactions on Speech and Audio Processing*, Vol. 6, No. 2, pp. 116-130, March 1998.

[4] European Telecommunication Standards Institute (ETSI), *Digital cellular telecommunications system; Enhanced Full Rate (EFR) speech transcoding (GSM 06.60)*, Final Draft, November 1996.

[5] A. Osterling, T. Benner, R. Ernst, D. Herrmann, T. Scholz, W. Ye, "The Cosyma system," in J. Staunstrup, W. Wolf (ed.), *Hardware/Software Co-Design: Principles and Practice.* Kluwer Academic Publishers, 1997.

[6] R. Gupta, G. De Michelli, "Hardware-software cosynthesis for digital systems," *IEEE Design and Test of Computers*, September 1993.

[7] C. Valderrama, M. Romdhani, J. Daveau, G. Marchioro, A. Changuel, A. Jerraya, "Cosmos: a transformational co-design tool for multiprocessor architectures," in J. Staunstrup, W. Wolf (ed.), *Hardware/Software Co-Design: Principles and Practice,* Kluwer Academic Publishers, 1997.

[8] F. Balarin *et al.* , *Hardware-Software Co-Design of Embedded Systems, The POLIS Approach,* Kluwer Academic Publishers, 1997.

[9] D. Gajski, F. Vahid, S. Narayan, J. Gong, *Specification and Design of Embedded Systems,* Prentice Hall, 1994.

[10] S. Prakash, A. Parker, "SOS: synthesis of application-specific heterogeneous multiprocessor systems," *Journal of Parallel and Distributed Computing*, vol. 16, pp. 338-351, 1992.

[11] Y. Li, W. Wolf, "Hardware/software co-synthesis with memory hierarchies," *Proceedings ICCAD*, pp. 430-436, 1998.

[12] A. Kalavade, E. Lee, "A global criticality/local phase driven algorithm for the constrained hardware/software partitioning problem," *Proceedings of the International Workshop on Hardware-Software Codesign*, 1994.

[13] P. Chou, R. Ortega, G. Borriello, "The Chinook hardware/software co-synthesis system," *Proceedings ISSS*, 1995.

[14] K. Rompaey, D. Verkest, I. Bolsens, H. De Man, "CoWare—a design environment for heterogeneous hardware/software systems," *Proceedings EuroDAC*, 1996.

[15] R. Gupta, S. Liao, "Using a programming language for digital system design," *IEEE Design & Test of Computers*, April 1997.

[16] D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, S. Zhao, *SpecC: Specification Language and Methodology,* Kluwer Academic Publishers, 2000.

[17] D. Gajski *et al.*, *Methodology for Design of Embedded Systems*, University of California, Irvine, Technical Report ICS-TR-98-07, March 1998.

[18] R. Dömer, J. Zhu, D. Gajski, *The SpecC Language Reference Manual*, University of California, Irvine, Technical Report ICS-TR-98-13, March 1998.

[19] Motorola, Inc., Semiconductor Products Sector, DSP Division, *DSP56600 16-bit Digital Signal Processor Family Manual*, DSP56600FM/AD, 1996.

[20] SpecC home page, `http://www.cecs.uci.edu/~specc/`.