Co-design of Emulators for Power electric Processes Using SpecC Methodology

Slim Ben Saoud
L.E.C.A.P.-E.P.T./ I.N.S.A.T.
B.P. 676, 1080 Tunis Cedex, TUNISIA
SlimBenSaoud@fulbrightweb.org

Daniel D. Gajski and Andreas Gerstlauer
Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA

***Abstract*–Emulation of CMS [1] systems is an interesting approach to complete the validation of new digital control unit and to perform the diagnosis tasks. However to be efficient, the emulator have to run in real time in order to reproduce exactly the physical process functioning.**

This paper describes the design of an *Autonomous Emulator* employing the system-level design methodology developed at CECS-UC Irvine (SpecC methodology). Starting from the abstract executable specification written in SpecC language, the emulator is gradually refined and mapped to a final communication model. This model can then be used with backend tools for implementation and manufacturing.

***Index Terms*– Co-design, Embedded Systems, Autonomous Emulators, Electromechanical systems**

I. INTRODUCTION

In this work we propose to design an autonomous real time emulator for electromechanical systems using SpecC methodology [1,2]. This emulator will be used with the control device either for complete validation of this one at the development and validation stage or for diagnosis at normal functioning stage. In both cases, the emulator should behave like the physical system in real time.

Today, realization of this emulator is not possible using standard electronic components. Therefore, we oriented our work to the development of new embedded systems specific to these applications of emulation.

Realization of this emulator is essentially faced to the execution time constraints. Indeed the emulator has to replace high dynamic systems in real time. So we distinguish three approaches in which real time emulator can be implemented: Digital, Analog and Hybrid. These approaches were discussed in previous publications [3,4,5].

The main difficulty in this realization is to satisfy both specifications of real time functioning and of flexibility. Association of these characteristics imposed uses of digital approaches, which guaranty user-friendliness.

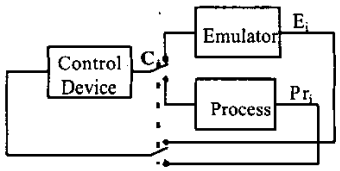
On the other side, improvements in VLSI technology have to led the wide spread use of specific processor, which may also be used to realize complete system.

According to these considerations, our work is oriented to the design of embedded systems specific to the emulation application. Therefore, we study the synthesis process of the emulation systems directly from their specification. This approach requires the ability to synthesize specified functions into software or hardware to meet the given constraints. It has the most flexibility since software, architecture and each component are custom made. However, it requires a well-defined methodology with clear steps, easy transformations and efficient tools to help the designer in the synthesis process.

In this paper, we describe a new approach based on the SpecC methodology for the design of an autonomous real-time emulator. We present at the beginning an introduction to the emulation principles. From this description and according to the SpecC methodology, we deduce the specification model of the emulator. Then, we describe the different steps and transformations used to convert this model to a communication model, which can be then transformed to an implementation model ready for manufacturing.

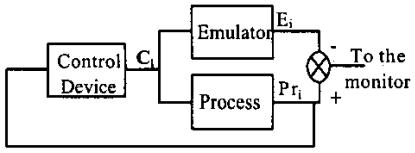
II. EMULATOR PRINCIPLES

The objective of the emulation approach in the electric drive applications is to design an electronic system, which can reproduce the physical system functioning in real time and with high precision. This system, called emulator, will be used for both of the new control device validation (Fig. 1) and of diagnosis (Fig. 2).



C_i Control signals / E_i Emulator outputs / Pr_i Process outputs

Fig. 1 Structure of the emulation application



C_i Control signals / E_i Emulator outputs / Pr_i Process outputs

Fig. 2 Structure of the diagnosis application

In both cases, the emulator has to reproduce accurately the process functioning. Therefore, it has to compute the System State and to convert the obtained results in forms identical to those obtained by sensors.

For this and in order to obtain precise results, we should use precise models of different components used on this

[1] Static Converters / electric Motors / Sensors

0-7803-7474-6/02/$17.00 ©2002 IEEE

process and a high performance computing system with the lowest computing step as possible (1µs or less).

On the other hand, this emulator must be flexible, easily configurable by the user according to her/his application, and it must allow the storage of results and if necessary monitoring.

According to these previous considerations, the emulator structure will be composed of three main modules [5]:

- Computing module: it computes, according to the digitized models, the system state variables.
- Emulation sensors module: it converts numeric results on different other forms identical to those obtained by the used sensors.
- Monitoring module: it performs the acquisition of new parameters and the storage of results. It can also perform some monitoring tasks.

III. SPECC METHODOLOGY

Managing the complexity at higher levels of abstraction, in the design of systems-on-chip (SOCs) or embedded systems in general, is not possible without having a very well defined system-level design flow [6,7,8].

Therefore, in this project we propose to use the SpecC methodology [1,2], which is a set of models and transformations on these models:

- The models written in programming language (SpecC language) are executables descriptions of the same system at different levels of abstraction in the design process.
- The transformations are a series of well-defined steps through which the initial specification is gradually mapped onto a detailed implementation description ready for manufacturing.

This methodology is based on 4 well-defined models, namely a specification model, an architecture model, a communication model, and finally, an implementation model.

In this paper we focus on the synthesis flow which contains the steps of specification, architecture exploration and communication synthesis. Implementation can then be done easily using standard tools.

IV. SPECIFICATION MODEL

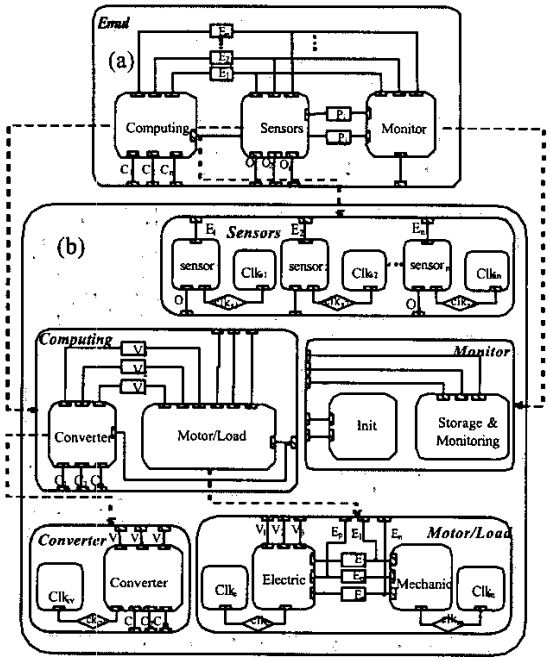
According to the SpecC language syntax [1], the emulator can be described by a main behavior (*Emul*) including different sub-behaviors: one for the computing module, one for sensors and one for monitoring (Fig. 3a).

These behaviors are usually composed of other sub-behavior according to the modular structure of the physical system (Fig. 3b). So, we use a sub-behavior for each of the converter, motor/load and sensor component and we add separate sub-behaviors for initialization process and for the monitoring module (including storage). These behaviors can also be decomposed of child-behaviors as we usually do with the motor/load behavior. Indeed, this behavior is usually decomposed on mechanic-child-behavior and electric-child-behavior.

Computing steps used with different modules are not the same since they don't have the same temporal characteristics. So we propose, for more flexibility, to add to each of the obtained child-behaviors a clock-generator behavior that controls its execution. The occurrence of the clock event is defined according to the module temporal characteristics.

However, we usually use the same computing step for the CM system (1µs) and different computing steps for each different type of sensor...

The obtained specification model shown on Fig. 3 was validated for the case of DC and AC systems.



C_i Control signals / E_i Emulator outputs / P_i Process parameters
V_i Converter voltages / O_i Sensors outputs

Fig. 3 Detailed Specification model of the emulator

Constraints: One of the most important constraints with this application is the computing time of the CM System State. Indeed, this computing must be done, according to the given models, in a periodic manner with a computing step of 1µs or less. While this seems to be realizable using High performance standard processors for the case of simple system using simple models, it is not possible for high performance application that needs sophisticated models and smaller computing step. Temporal problems of this computing, implemented into a processor, are related to the computing load and to the use of interruption for the periodic functioning management.

Other constraints such as the resolution of the used DAC devices and the resolution and precision of the used sensors (encoders, resolvers,...) will certainly influence the design procedure.

V. ARCHITECTURE EXPLORATION

Architecture exploration is the first part of the system synthesis process that develops system architecture from the specification model. The purpose of architecture exploration is

to map the computational parts of the specification onto the components of system architecture. The steps involved in this process are allocation, partitioning and scheduling. Through this process, the specification model is gradually refined into the architecture model.

At this stage, we propose to study the case of an *Autonomous Emulator*: the emulator is associated to an external memory on which will be stored parameters and emulation results. The emulator functioning is then independent from other computing systems. At the starting step, it reads new parameters from the external memory block and then, at each new storage period, it will write new results into successive registers of this memory. Another processor used for initialization and monitoring will manage this memory when the emulation is off-line in order to initialize the process parameters and to restore emulation results. This solution is very useful when used with the micro-controller of the control device since it will not disturb the control tasks. However it introduces some delay in the monitoring operation.

A. Allocation and Behaviors Partitioning

The first step in architecture exploration is to allocate a set of processing elements (PEs) and to map the behaviors of specification onto the allocated PEs. In this structure and according to the previous considerations, the emulation system will be composed of three PEs: the EmulCore hardware component, the software component (processor) and the memory block. This memory, shared by the two active components, is used for storage of parameters and emulation results.

Fig. 4 represents the obtained refined model after behavior partitioning. In this model, two behaviors (*init & storage*) are added to *PE1 (EmulCore)* in order to synchronize and establish communication with the software and memory components. This communication becomes system-global and it is moved to the top-level connecting the PE behaviors. Synchronization is done at the beginning and the end of the emulation procedure by using two events *Start* and *End*.

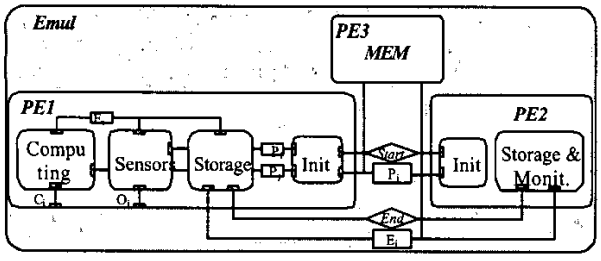


Fig. 4 Architecture model after allocation and behavior partitioning

B. Variable Partitioning and Scheduling

The number of exchanged variables between HW and SW components is very limited (usually less than 10), so we chose to use local copies for these variables in each PEs. Therefore in the refined model obtained after variable partitioning, global variables for results (E_i) and parameters (P_i) are replaced with their respective abstract channels C_{Pi} and C_{Ei}. Code is inserted into behaviors to communicate variable values over these channels. Note that data are exchanged in a vector type: one for parameters and one for results.

Scheduling The next step in the architecture exploration process is to schedule behavior executions on the inherently sequential processing elements. We consider that processing elements have a single thread of control only. Therefore, behaviors mapped to the same PE can only execute sequentially and have to be serialized.

In this application and according to the previous specifications, synchronization between HW and SW components are performed by using of two events *Start* and *End*.

The *Start* event can be associated to the parameters variables Pi and encapsulated into a Message-Passing channel that models the abstract communication semantics of blocking, unbuffered message-passing between any two client-behaviors.

On the other hand, the *End* event will be connected to a processor interrupt input. The interruption program is used to set a flag *F_end* when this interruption is activated. When the software program needs the emulation results for monitoring, it tests this flag. If it is set, it performs the data read sequences from memory, otherwise it waits for the interruption activation.

C. Channel Partitioning

In this application, the obtained architecture target is composed of two active PEs (PE1 and PE2) that share a memory block (PE3). Therefore, only one bus is used into which all the global channels and their implementation are encapsulated.

The obtained refined model after variable partitioning, scheduling and channel partitioning is shown on Fig. 5.

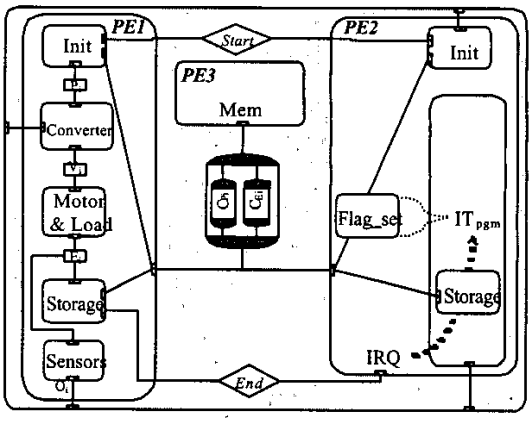


Fig. 5 Architecture refined model after channel partitioning

VI. 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.

For the illustration of our application we use the DSP56600 (Motorola) as the chosen IP component for implementing the software behaviors (PE2) [9].

A. Protocol Insertion

During the protocol insertion, a description of the used protocol is inserted into the corresponding virtual system bus channel.

The abstract communication primitives provided of the bus channel are rewritten into an implementation using the primitives provided by the protocol layer. The outer application layer of the bus channel implements the required semantics over the actual bus protocol. This includes tasks like synchronization, arbitration, bus addressing, data slicing, and so on.

All the abstract bus channels in the model are replaced with their equivalent hierarchical combinations of protocol and application layers that implements the abstract communication of each bus over the actual protocol for that bus.

To illustrate our study, the protocol channel in the system bus and the wrapped processor model describe and implement the DSP56600 bus protocol according to its timing diagram shown in Fig. 6 [9].

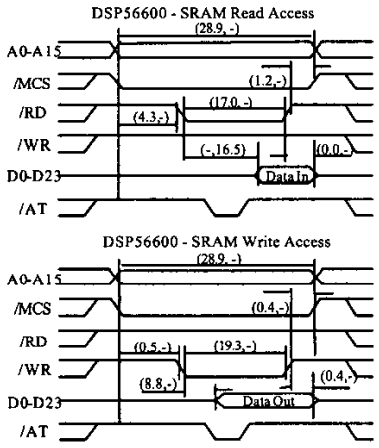


Fig. 6 Protocols of the DSP56600 external bus

Fig. 7 shows the emulator model after the insertion of the DSP bus protocol as the system bus protocol, and after the processor behavior has been replaced with a model of the real processor with a wrapper. Transducer will be added, if needed, between hardware protocol and the system bus protocol. On the other hand, memory must be able to respond to the read and write requests from DSP and ASIC. This again requires design of a bus interface for memory to respond to bus read/write requests. The behaviors and complexity of such interfaces depends on the time-constrained behavior of these components at their ports. So, we must discuss the timing behavior of these components and then analyze these interfaces [10].

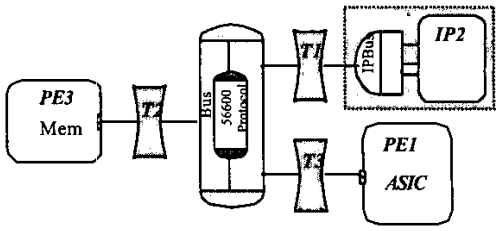


Fig. 7 Communication model after protocol insertion

Note that in order to simplify this illustration, we assume that the hardware component has the same protocol as the system bus (DSP56600 protocol).

On the other hand, we experimented with Samsung memory KM68257C [11], which is a CMOS static RAM and has 8 common input and output lines. The different pins and the memory specification for read and write cycles are shown in Fig. 8.

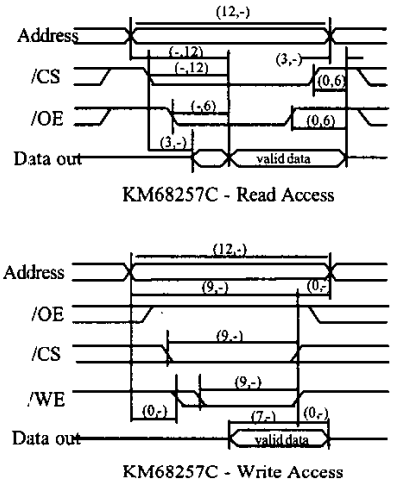


Fig. 8 Memory bus protocol (KM68257C)

According to these specifications, this memory is fast enough and will be used directly without any additional interface according to the schema of Fig. 9 Therefore, no interfaces are required with this architecture model.

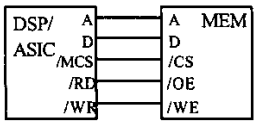


Fig. 9 DSP-ASIC / Memory interfacing model

Parameters and results are stored in the memory at precise corresponding locations with precise addresses that correspond to free locations in the DSP and ASIC memory spaces.

According to this architecture model, we distinguish two masters sharing the same bus and the same memory component:

- DSP: it writes at the beginning parameters values to their corresponding locations in the memory block, and reads results tables;
- ASIC: it reads at the beginning parameters from their corresponding locations, and writes results at each execution of the storage behavior (Ts period).

So, cautions must be taken in order to avoid communication conflicts. In this project, we implement the case of the solution without management of the bus, which represents a simple bus management protocol.

In this case the required synchronous passing semantics are realized as follows: The DSP determines new parameters and it stores them in their locations inside the memory block. Then, it signals by an asynchronous event to the emulator that data are ready and it continue execution of other tasks without using its external bus. The emulator waiting for the start signal from the DSP, receives this event and then execute the init behavior before beginning emulator computing. At each Ts period, the emulator increments the memo ry address and store new results in a table. At the end of its computing, the ASIC signals by an asynchronous event to the DSP that results are ready in the memory. This event will interrupt the DSP program and set a flag. The DSP, tests this flag when results are required in order to perform new acquisition.

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 obtained after protocol inlining is shown in Fig. 10.

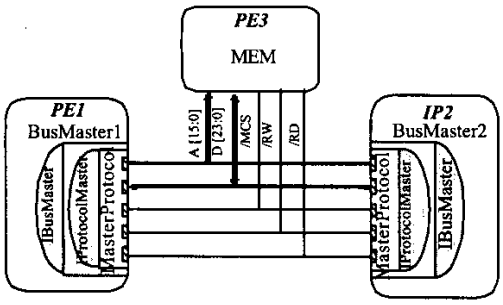
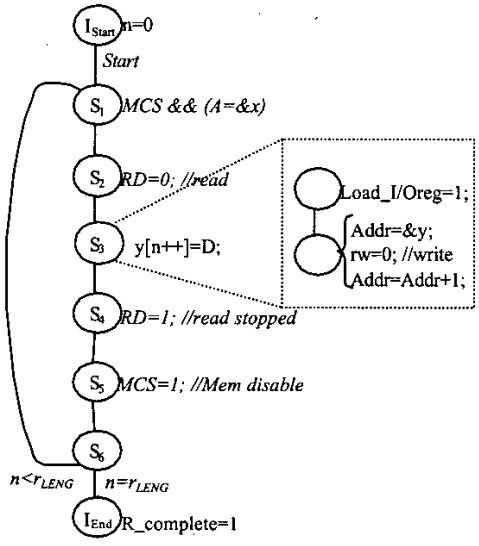


Fig. 10 Communication model after protocol inlining

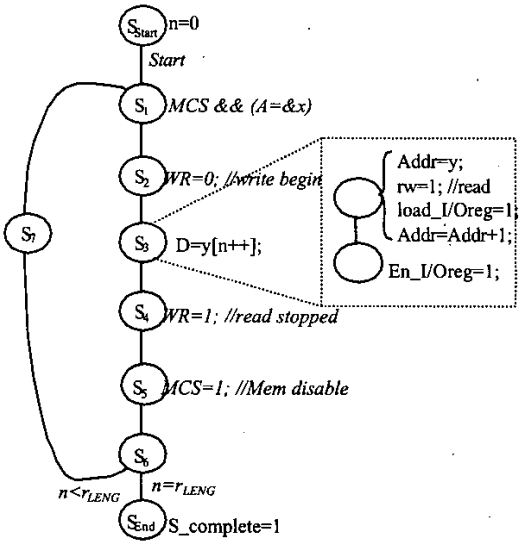
The *Init* SFSMD synchronizes with the software on the DSP, it receives the start signal from it. Then, it reads the process parameters over the processor bus from the memory, and starts the emulator computing. At each new storage period (Ts=x_s hor), the storage SFSMD writes emulation results into the memory table. When finished, the ASIC send an event to the DSP signaling that data are ready in the memory and that bus is "free".

For the ASIC, communication primitives are inlined into the exchanges sub-behavior (*Init* and *storage*).

Therefore, exchanges SFSMD models are created and inserted into the ASIC SFSMD model (Fig. 11).



(a) *Init* SFSMD



(b) *Storage* SFSMD

Fig. 11 HW communication SFSMDs (Autonomous emulator)

Fig. 12 shows the implementation of the interconnection between the three used components after final inlining.

After validation by simulation, the obtained communication model will be ready for use directly to generate the implementation model. Indeed, The leaf behaviors of the design model will be fed into different tools in order to obtain their implementation [12].

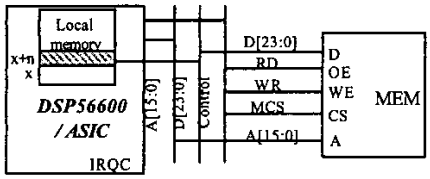


Fig. 12 DSP/Memory interfacing model

VII. CO-DESIGN APPROACH PERFORMANCES

An application to the case of DC system is done and demonstrates the efficiency of this approach. The development of this emulator is performed very easily (in few hours) using the defined approach and the previous study of the emulator structures.

The used co-design approach presents a simplified design process based on well-defined, clear and structured models at each exploration step. This enables quick exploration and synthesis.

In each of the tasks the designer can make design decisions and generate a corresponding SpecC model. Then, the design can be statically analyzed or simulated for validation of design correctness in terms of functionality, performance, and other constraints. A simulation model is compiled after each step, which can be run on the host computer to validate correctness for simulation.

The presented synthesis flow demonstrates the efficiency of a system-level design methodology, which allows moving the design to higher levels of abstraction, in order to increase productivity.

Using components and protocols libraries with specific tools for step automation will facilitate the design process and reduce further more the time-to-market. The user will be able in the near future to design her/his emulator and implement it (using for example FPGA circuits) in few hours without the need of any high qualification.

Development of these tools and libraries represent our main objectives for the future works.

VIII. CONCLUSIONS

In this paper, we present a design process of an autonomous emulator, based on the SpecC methodology. This methodology presents a simplified design process based on well-defined, clear and structured models at each exploration step. This enables quick exploration and synthesis.

According to this methodology and during the synthesis process, three models are constructed: the specification model, the architecture model and the communication model. For each of them we discussed the mechanism of refinement and we proceeded to a validation by simulation. The correct output demonstrates the correctness of our models.

IX. ACKNOWLEDGMENTS

The authors would like to thank the Fulbright Scholar Program for supporting this project.

X. REFERENCES

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

[2] A. Gerstlauer, R. Dömer, Junyu Peng, D. Gajski, "System Design: A Practical Guide with SpecC", Kluwer Academic Publishers, 2001

[3] S. Ben Saoud, J.C. Hapiot, "Parallel architectures applied to real time emulation", IECON2K IEEE International Conference on Industrial Electronics, Control and Instrumentation, October 22-28, 2000, pp1719-1724

[4] S. Ben Saoud, J.N. Contensou, J.C. Hapiot, "ASIC dedicated to real time emulation", SDEMPED'99 IEEE International Symposium on Diagnostics for Electrical Machines, Power Electronics and Drives, September 1-3,1999

[5] S. Ben Saoud, B. Dagues, J.C. Hapiot, "Universal emulator of static converter / electrical machines / sensors sets. Test of new control unit", CESA'98 Computational Engineering in Systems Applications, April 1-4,1998

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

[7] R.K. Gupta, "Co-Synthesis of Hardware and Software for Digital Embedded Systems", Kluwer Academic Publishers, 1995

[8] R. Niemann, "Hardware/Software Co-Design for Data Flow Dominated Embedded Systems", Kluwer Academic Publishers, 1998

[9] Motorola, Inc., Semiconductor Products Sector, DSP Division, DSP 56600 16-bit Digital Signal Processor Family Manual, DSP56600FM/AD, 1996

[10] J. Peng, L. Cai, A. Selka, D. Gajski, "Design of a JBIG Encoder using SpecC Methodology", University of California, Irvine, Technical Report ICS-TR-00-13, June 2000

[11] Samsung Semiconductor Inc., North America, SRAM Products, "KM68257C", 1998

[12] 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