to appear in the IEEE *Design & Test of Computers Magazine*
for the special issue on "Design, Test, & ECAD in Europe", 1997

Co-simulation and Software Compilation Methodologies for the System-on-a-Chip in Multimedia

Clifford Liem[1,2], François Naçabal[1,2], Carlos Valderrama[1], Pierre Paulin[2], Ahmed Jerraya[1]

1. Laboratoire TIMA, INPG (Institut National Polytechnique de Grenoble), Grenoble, France
2. Central R & D, SGS-Thomson Microelectronics, Crolles, France

Today's fabrication technologies are allowing increasing functionality on a single chip. Consequently, for complex consumer products like the SGS-Thomson single chip videophone [1], new design and verification methods are needed to address the interfunctioning on-chip hardware and software components. Complementing standard hardware practices, this article presents contributing techniques in hardware/software co-simulation and embedded software compilation.

Introduction

With the advances of submicron technologies, designers are able to put impressive numbers of components on a single microchip. Along with this capability, a vast choice of hardware and software components make the design and validation of the system an increasingly complex process. The SGS-Thomson series of videophone systems (e.g. STi1100 [1]) is an example of a system-on-a-chip which is indicative of this design challenge. This system has required a redefinition of design and functional validation methodologies from the standard ASIC (Application Specific Integrated Circuit) design flow. Figure 1 shows the main operators of the videophone system, which communicate through a set of busses.

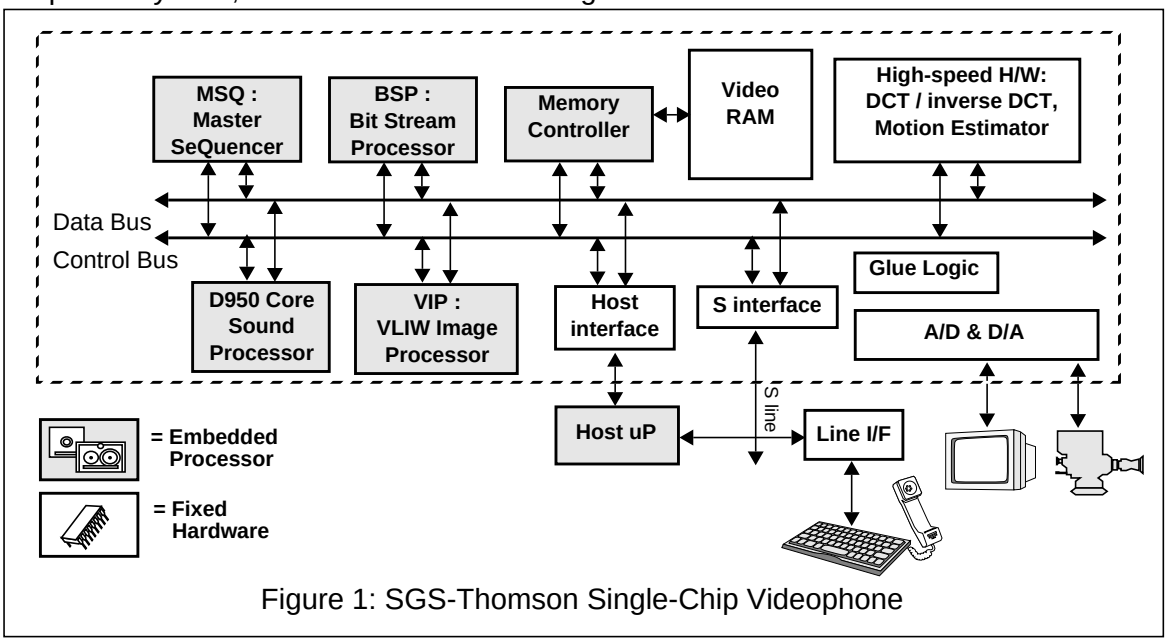

Figure 1: SGS-Thomson Single-Chip Videophone

Some of these operators are designed as fully hardwired blocks to meet the performance requirements (e.g. the Motion Estimator [2]); however, to keep pace with the evolving standards, many are designed as custom instruction-set processors or ASIPs (Application Specific In-

struction-Set Processors). These are dedicated, low-cost embedded processor cores which run software for their specialized task. With the constantly changing standards (e.g. H.261, H.263), block functions in software allow for late design changes and modifications, which are important in meeting the current market requirements.

In this system environment, two technologies are key to its success: hardware-software co-simulation and embedded software compilation. For functional validation of the system, a hardware-software co-simulation methodology is mandatory. The hardware of the system is written in behavioural-level and RTL-level VHDL [3], while the software for the ASIPs is written in C. Co-simulation is then needed for functional validation of this C-VHDL specification. Most VHDL simulators provide a means to integrate C routines in a simulation; however, this type of simulation is highly restrictive. It follows a master-slave co-simulation model allowing for a simple C program to be called from a master VHDL simulation. It does not allow for co-simulation within a distributed model, where several C programs are running in parallel. Moreover, it does not allow the C program to be the master.

New communication techniques are beginning to appear to resolve these restrictions [4][5]. For the videophone system, a C-VHDL co-simulation environment called *CoSim* [6] was developed which imposes no restrictions on the type of C being simulated. It is based on the Unix IPC (Inter Process Communication) utility and can model many intercommunicating processes at one time. In addition, the methodology allows standard C debugging tools like dbx and gdb to be used transparently. Thus, the user is free to debug VHDL and C all in the same simulation.

With a large proportion of the videophone function in software, an effective compilation methodology is critical. The requirements for the compiler system is that it be easily retargetable to a wide variety of processor architectures, from control-flow dominated architectures to data-flow dominated architectures. It must adhere to strict hardware requirements like bus interface protocols as well as handle architecture specialization. As well, since the system is real-time reactive, performance overheads with respect to hand code cannot be tolerated.

The challenges of code generation for custom embedded processors has just recently sparked the attention of the compiler community [8]. Solutions to the requirements of the videophone system have been met through the development of software compilers for three of the processor blocks using the retargetable rule-driven approach in the *FlexWare* system [9]. This C compiler environment is a flexible and efficient means of providing a compiler for a variety of architectures. The targeting time for each compiler was on the order of one person-month including validation; and results have shown that there was no loss in code quality when compared to hand code.

Design Methodologies for the Videophone Architecture

Prior Co-Design Methodology The design methodology for the current production chip is outlined in Figure 2 and described in detail in [3]. Briefly, the approach begins with an exploration and refinement of the C algorithms based on the models provided by the videotelephony standards committees (H.261, H.263). The architecture is then defined by a coarse partitioning onto software and hardware operators. VHDL models serve as the backbone simulation plane on this behavioural level. These models are then manually rewritten on a register-transfer level (RTL) for the hardware and as assembly code for the software. Commercial (RTL synthesis) and public domain tools (assemblers/linkers) then offer a path to physical implementation in the forms of netlists (hardware) and machine code (software).

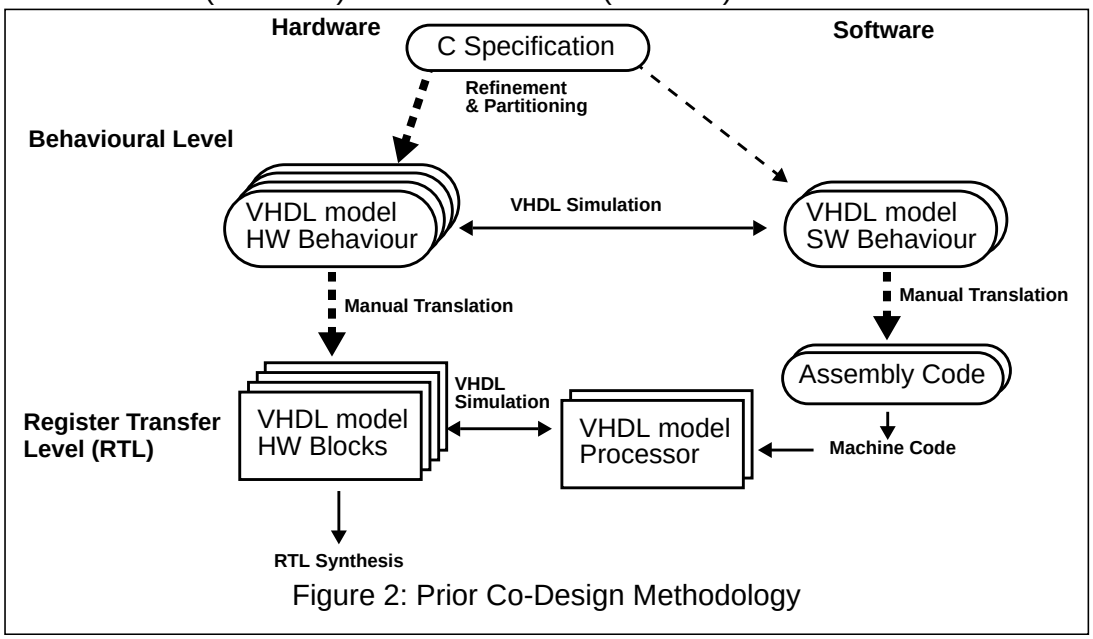

Figure 2: Prior Co-Design Methodology

Current Co-Design Methodology The current design methodology for the videophone is shown in Figure 3. Compared to the prior design methodology, the first notable aspect is that more function has moved from hardware to software. The reasons for this are many; however, the foremost remains the added flexibility to track the evolving algorithmic standards. At late design stages, software is much easier to modify than hardware, as long as each embedded processor has the foreseen capabilities.

Rather than the manual translations to and from VHDL, the software specification language is C, which requires fewer changes from the models provided by the standards groups. As a direct consequence, C-VHDL co-simulation for functional validation with the rest of the hardware system becomes an important component. In addition, software compilation to form a path to implementation is equally important.

Although there are fewer hardwired components, these hardware operators are usually critical components in other regards, for example, high speed, low power and low area. This drives the advancements to higher level synthesis (Behavioural in addition to RTL) to improve not only design time, but to make better high-level trade-offs (e.g. speed vs. area vs. power consumption). Behavioural synthesis for the videophone is discussed in detail in [2].

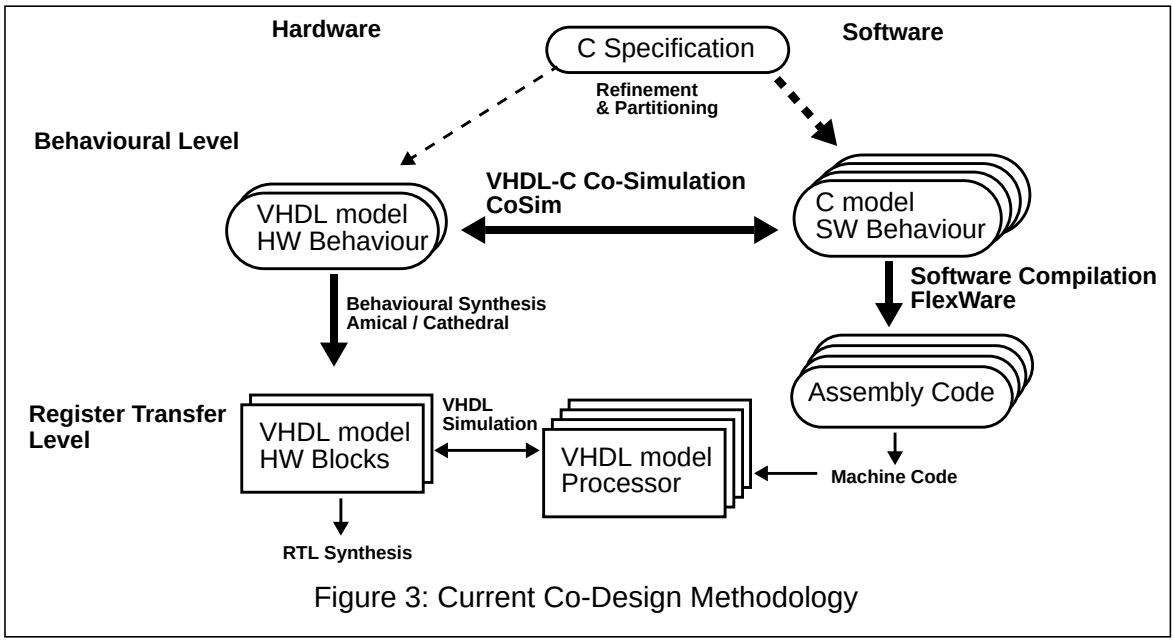

Figure 3: Current Co-Design Methodology

Hardware-Software Co-simulation

Most commercial VHDL simulators provide a basic means to invoke C functions during VHDL simulation; however, this approach imposes a fundamental constraint: the invocation of the C function must always start from the same entry point, as shown in Figure 4a. This fits in well in a master-slave communication model (Figure 4b), where the VHDL is the master that simply calls slave C routines, awaiting a response from each call. However, a C program could be the master of other hardware blocks (e.g. the MSQ), and therefore require multiple entry and exit points (Figure 4c). A distributed communication model (Figure 4d) with no notion of master or slave would alleviate this problem. In addition this model allows multiple C programs to run in parallel.

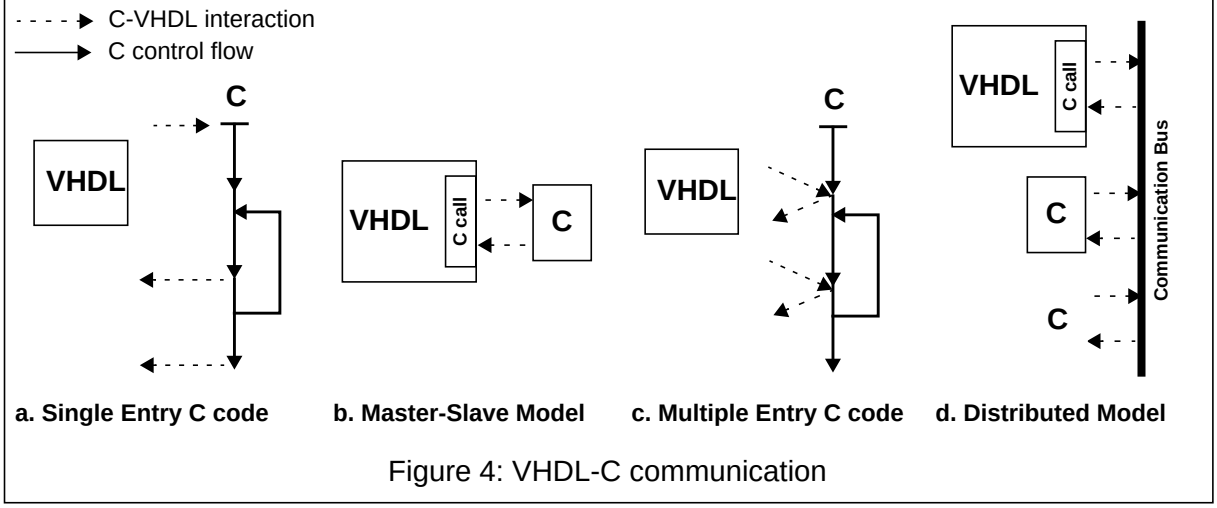

Figure 4: VHDL-C communication

The distributed model of Figure 4d can be realized using the Unix Inter-Process Communication (IPC) mechanism. In practice, the VHDL simulator capability to call a C function is still

used; however, that C function contains only the protocol to address IPC messages. This solution allows the designer to keep the C application code in its original form.

The key point in this methodology is the encapsulation of the IPC communication layers and the VHDL-C interface function call. This setup can be cumbersome, especially when repeated for several processors. Experiments conducted in this project showed that about 500 lines of C code were a minimum to implement an IPC-based mechanism. This motivated the development of *CoSim*: a tool which automatically generates these layers. Through a set of configuration options, this utility has been made flexible enough to suit a wide range of processor interfaces.

For a given C program, CoSim generates all the application-specific code, and links it with IPC and the VHDL- C interface libraries (Figure 5). The communication can then be performed through a single function named *io_transaction().*

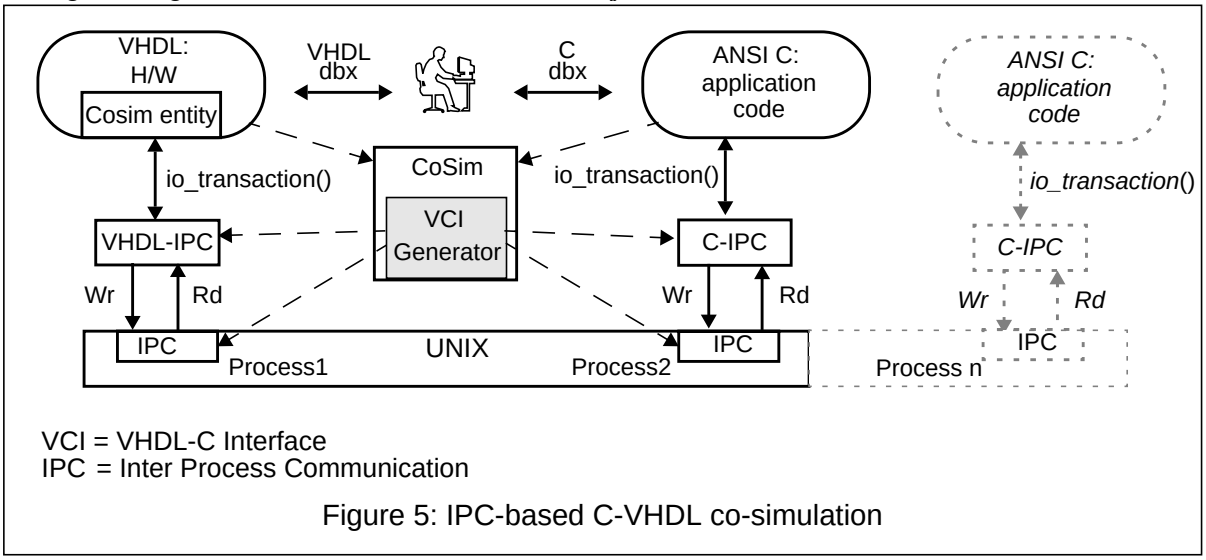

Figure 5: IPC-based C-VHDL co-simulation

When the generation of the communication layer is complete, the tool runs the VHDL simulator and the C program, automatically managing the IPC communication. An additional feature is that standard C debuggers can be used for co-simulation of C code with VHDL. This is not supported by commercial tools.

In systems with multiple processors like the videophone, CoSim generates one communication layer for each C application. To date, we have run a maximum of three C processes with VHDL without hitting any performance problems. The only obstacle we can envision for this methodology is a practical limit to the number of IPC communication channels which may run concurrently on UNIX platforms; however, we have not yet run experiments to determine this limit.

Implementation The generation of the C-VHDL interface is done through a process known as VCI (VHDL-C Interface)[7], shown in Figure 6. VCI is driven by a parameterized interface specification. This specification is created by a parsing of the VHDL entity to determine the communicating functions to C. The output of VCI is a set of VHDL and C files needed to interconnect the C-program to the VHDL structure during co-simulation. (It is worth noting that the tool can alternatively be used to generate the interconnection between C modules only.)

The C functions communicate with the VHDL simulator by means of the provision for externally callable languages (e.g. Synopsys *VSS* CLI, Cadence *Leapfrog* FMI). This information is con-

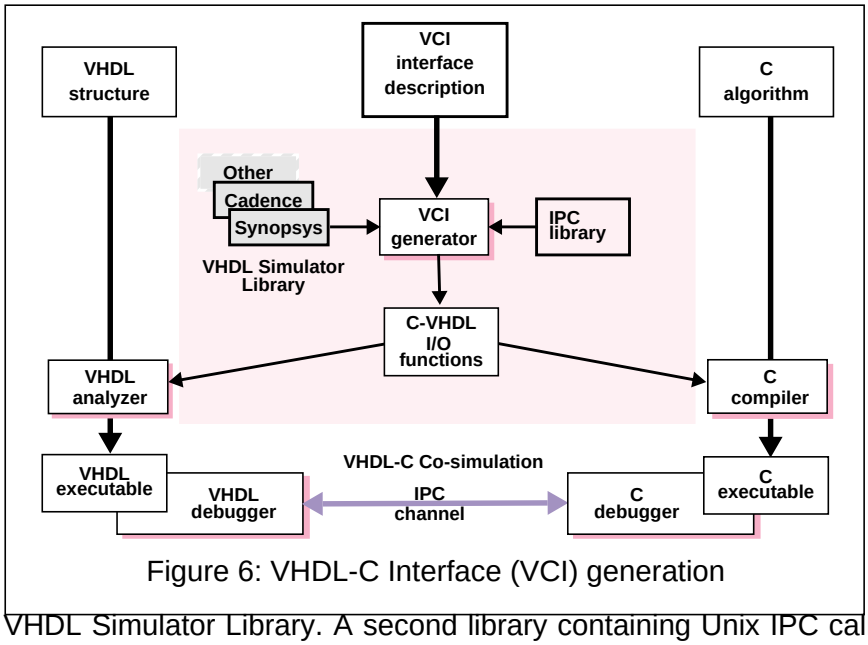

Figure 6: VHDL-C Interface (VCI) generation

tained in the VHDL Simulator Library. A second library containing Unix IPC calling functions completes the information necessary to produce the VHDL-C link.

Communication between a VHDL process and a C executable is established by means of a single bidirectional message queue [4]. Information exchanged between the two parts contains the values for all the signals in the interface. For message-based communication, a synchronous half-duplex protocol is used. It is based on the continuous exchange of updates. A token represents each update, which allows the receiver to respond at a step following the send of the token. After sending a token, the sender is not permitted to emit another token before receiving one. This does not restrict the communication in any way, but rather, guarantees the synchronization of events.

Synchronization between C and VHDL is performed with an evaluation clock derived from the master clock. The transfer is done alternatively on each edge as shown in Figure 7. The mechanism also insures that one event and only one can be delivered to the C or to the VHDL at a time. No message overflow is allowed.

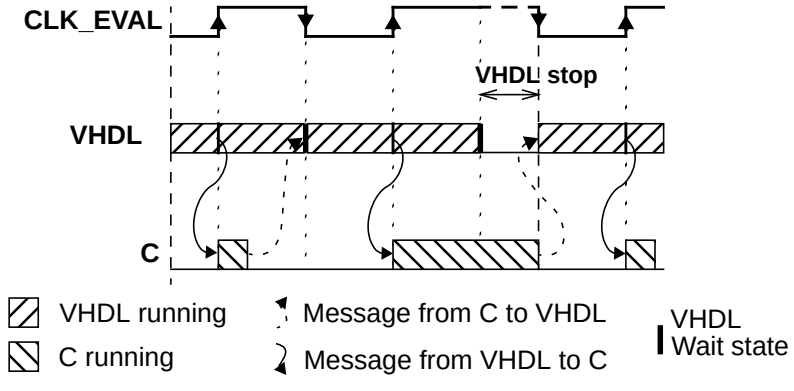

Figure 7: C - VHDL synchronization

Co-simulation Results In Figure 8, we compare C-VHDL co-simulation with simulation entirely in VHDL (RTL model of the target processor with the program loaded in ROM). We show results for the VIP (shown in Figure 1) with a number of different examples chosen from both the H.261 and H.263 recommendations. The simulation runtime is defined as follows:

- for the C-VHDL co-simulation: the time spent to run the compiled C algorithm (including the IPC communication) plus the time spent to simulate the rest of the system in VHDL;
- for the RTL simulation: the runtime of the VHDL simulator (model of the target processor and its environment).

This experiment has been done for six different algorithms. Figure 8 shows the comparison between the runtime (Y-axis) of co-simulation (dark gray bars) and the runtime of VHDL-only simulation (light gray bars), obtained for the different algorithms (on the X-axis, ordered by growing complexity).

The co-simulation is always faster than the RTL simulation, and the speed-up ratio increases with the complexity of the algorithms. In the most complex example, there is nearly a three-fold improvement in simulation time. Two reasons can explain these results:

- the VHDL simulator spends a lot of time simulating the behaviour of the processor for each machine assembly instruction; whereas, in C-VHDL co-simulation, the C is compiled to the workstation; and therefore, a small set of assembly instructions are simply executed on the workstation;
- the additional time required by the IPC communication is more than offset by the simulation speedup of the C versus the assembly running on the VHDL-RTL model.

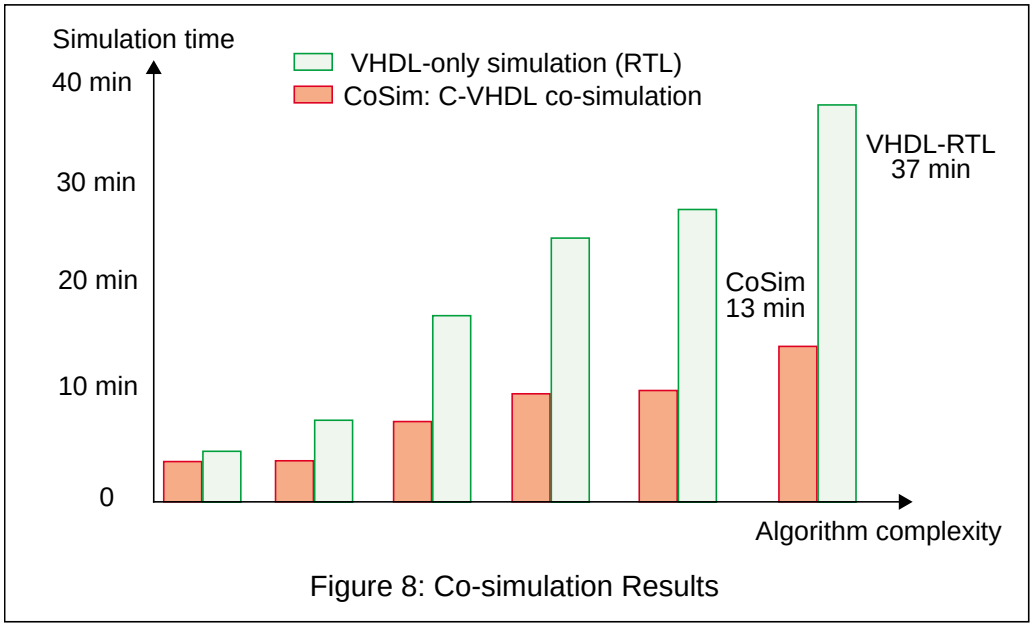

Figure 8: Co-simulation Results

In summary, the CoSim tool has provided the ability to functionally validate the communication between the components of the videophone system. In addition to the capability to write at a higher algorithmic level (C), simulation time has decreased when compared to pure VHDL simulation.

Embedded Software Compilation

The SGS-Thomson videophone contains a collection of very different embedded processor architectures ranging from control-flow oriented (the MSQ) to data-flow oriented (the VIP). Typically, compilers built for general computing environments are highly optimized for one sole target processor. In being built in this fashion, they are often rigid in functionality and suffer from low retargetability. A new style of architecture cannot be easily accommodated, as there is little flexibility in the steps of the compilation. A method supporting retargetability needs to have flexibility at each step in the compilation process.

Compilation Techniques Traditionally, a compiler is built as a succession of refinement steps as shown in Figure 9a. While this is successful for many general-computing architectures, retargetability is often restricted to the back-end code generation phase. This can mean that optimizations are not driven by the architecture characteristics. Furthermore, in the case of embedded processors, architecture idiosyncracies like heterogeneous register files and special-purpose hardware functions cannot be handled.

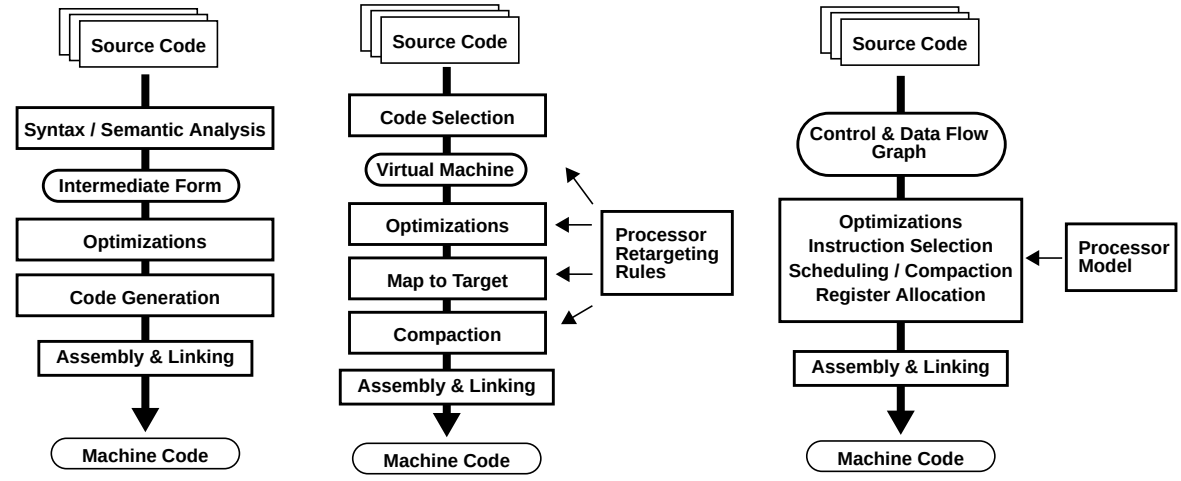

Figure 9: Compilation Techniques

Figure 9b and c depict recent approaches to compilation for embedded processors [8]. In these approaches, information of the processor architecture drives the succession of the compilation steps. This feature is essential for embedded processors, where the architecture characteristics are unlike standard microprocessors. Processors for real-time reactive systems often contain a limited number of registers which are specialized for certain functions, as well as encoded instruction words, and special functions to communicate with the rest of the system. The compiler task of mapping onto these functions requires special attention.

Rule-driven Compilation This approach to compilation was first presented by Gurd in [10]. It is based on step-wise progressive refinement and provides a programming environment for compiler development. The compilation process is divided into four main phases, which are shown by example in Figure 10.

1. **Virtual code selection.** The developer defines a virtual machine which resembles in functionality the instruction-set of the real machine, but is sequential in operation. Those proces-

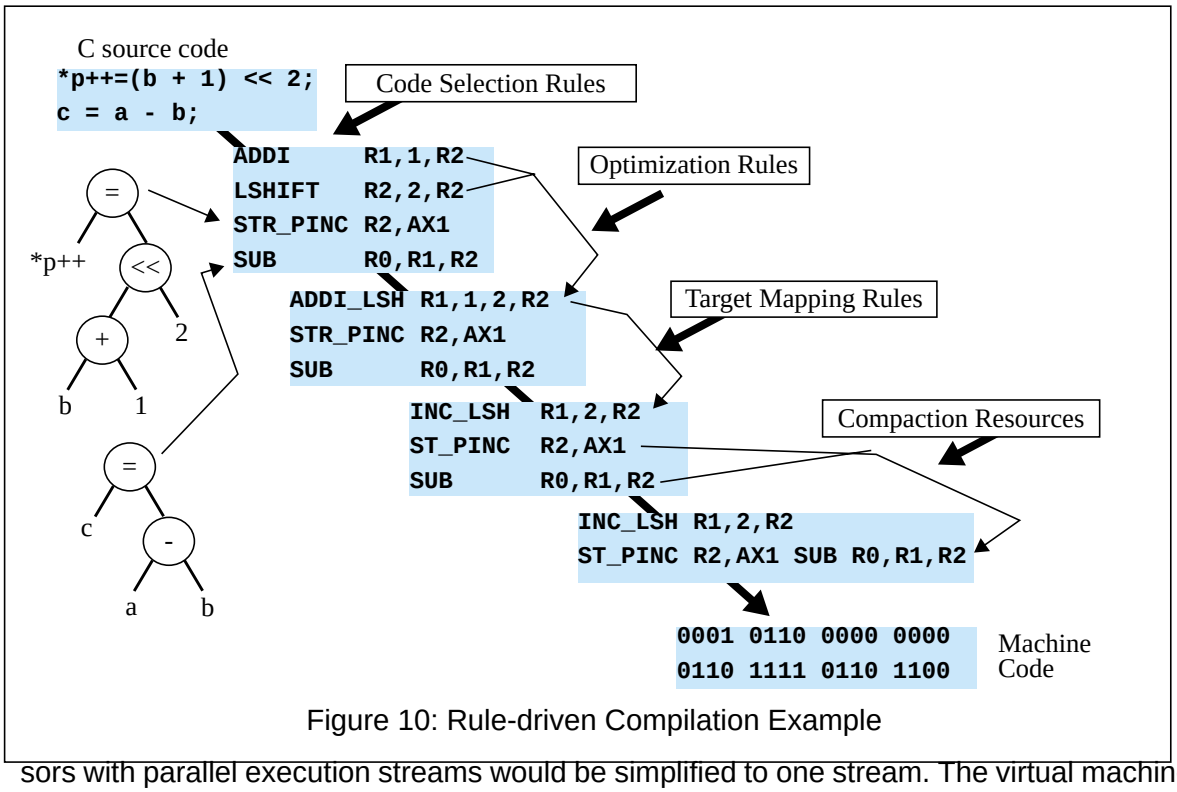

Figure 10: Rule-driven Compilation Example

sors with parallel execution streams would be simplified to one stream. The virtual machine description contains two main parts: 1) a description of resources including register sets and addressing modes, and 2) a set of code selection rules.

The definition of the available register sets classifies these resources into functional categories. For example, it indicates which C data-types each register may hold. The definition of the addressing modes indicates the manner in which variables are to be retrieved from memory.

For the code selection rules, the developer defines the mapping between the C code onto the virtual machine instruction set. For each operation which may occur, the developer provides a rule which is defined in a programming language. This rule will be triggered upon matches to the source code and executed at compile time. This approach allows the developer to provide simple rules for the most common cases and more complex mappings for special features of the architecture. For example, the developer may restrict the use of certain registers whose function are constrained by the architecture. This is important to support special-purpose registers which are often found in embedded processors. Register assignment within register sets is performed after code selection using a coloring approach. This is done in a manner which satisfies the constraints imposed by selection rules.

2. **Optimizations.** Instructions for the virtual machine may be passed to a series of optimization routines, such as a peephole optimizer, which transforms sequential occurrences of operations into more efficient operations through simple replacements. The user indicates a source and target sequence of code using keywords and wildcards. As the transformations are activated recursively, the code may significantly be improved. At this point in the compilation, it is also easy to add custom optimization sequences, since the input is very well defined. This was done in the definition of the virtual machine. For example, a data-routing optimizer may be inserted to determine the best movement of data through the machine, given the structural connectivity of the hardware.

3. **Mapping to the target machine.** The optimized sequence of virtual instructions are transformed into operations for the real machine. Each transformation again follows a rule provided by the developer. Each rule indicates a source piece of code and a target implementation in the form of micro-operations representing bit fields of the instruction-set.
4. **Code compaction.** Micro-operations are compacted into real instructions. The compaction procedure executes based on constraints of both the bit-field formats and read/write/occupy resources which are indicated by the developer. The compactor attempts to push the maximum number of micro-operations to the earliest possible positions. The straight-forward tasks of assembly and linking immediately follow compaction.

The open programming concept The rule-driven compilation approach is built upon the concept of an open programming environment. All the rules are defined in a well-structured programming language. For example, in the first step of sequential code selection, the compiler developer has at his/her disposal a language which contains a set of high-level primitives corresponding to information which is generated as syntax trees of the source program. In providing functions using these primitives, the developer defines the mapping from syntax trees to assembly code, as shown in the example in Figure 10. This allows the developer to provide simple mapping rules for the majority of cases, and more complex rules for instructions which activate areas of architecture specialization. It is possible to provide sophisticated mapping functions to handle architecture features which do not fall into the well-known categories of standard processors.

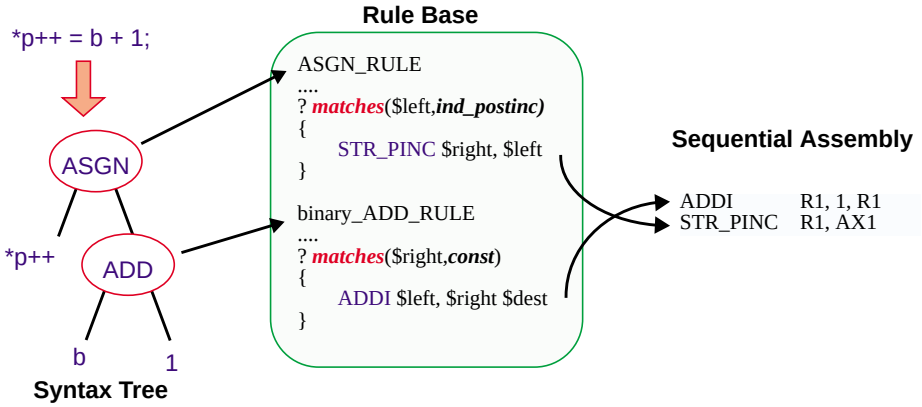

Figure 11: Sequential code selection

A high level of open programming is provided at all the steps of the rule-driven compiler, allowing a very flexible development system. The user is able to retarget the system upon the application of suitable mapping functions. The quality of the compiler is directly proportional to the amount of development time spent on optimization strategies. Moreover, previous compiler development experience may be leveraged for processors with similar features.

Assessment of the approach Although the rules for this type of compiler must be written by an experienced developer, the retargeting time is relatively short. Each of the compiler systems built for processors of the videophone were done within one person-month, and each produce acceptable code quality. Code size overhead varied between 0 and 30% in comparison with hand-code. While execution time overhead was not measured, the algorithm time constraints were met in all applications.

The main strength of rule-driven compilers is in the inherent flexibility of the approach. The compiler developer has the means for describing specific rules and strategies for efficiently mapping higher-level constructs onto the processor, based on his knowledge of the architecture idiosyncracies. Standard rules are put in place based on previous compiler experience, and primitives are available to manipulate the compiler for new architectures with unforeseen specialization.

When compared to a traditional compiler approach (Figure 9a), the rule-driven approach allows for faster development time through the ability to retarget at each phase of compilation. The environment provides high level mechanisms for quickly capturing compilation strategies. The quality of the results depend on the compiler development effort. In the few cases when the code quality is inadequate, a custom optimization module is incorporated with little effort.

When compared to model-based retargetable compilation approaches (Figure 9c) [8], the rule-driven approach requires retargeting development time; whereas, in principle, a model-based compiler requires only a small change to the model to arrive at a new compiler. However, in our experience [9], the retargeting time is compensated by the applicability of the rule-driven approach to a very broad set of processor architectures, from low-end microcontrollers to VLIW DSPs (Very Large Instruction Word Digital Signal Processors). In addition, architecture specific idiosyncracies may be handled by case-by-case development strategies.

Compiler Retargeting Compilers were developed for three embedded processors of the videophone system: the MSQ, the BSP, and the VIP (Figure 1). For each of the architectures, a functional rule base was typically developed in two person weeks, or roughly half of the total targeting time including validation. This allows early feedback to the architecture design team before the final refinements are made. Each compiler supports a subset of C; however, support of the entire functionality of the architecture is always available.

As an example, we show in Figure 12 the MSQ (Micro-SeQuencer), which is the top-level control unit of the videophone system. The architecture is a single execution stream controller providing standard ALU operations (ADD, SUB, AND, OR, CMP, SHIFT); as well as standard control operations (BR conditional/unconditional, BR indirect). Reserved instructions perform the function of the bus interface protocol. A unique property of this block is a unit known as the scheduler (SCH) which can affect the position of the program counter independent of the nat-

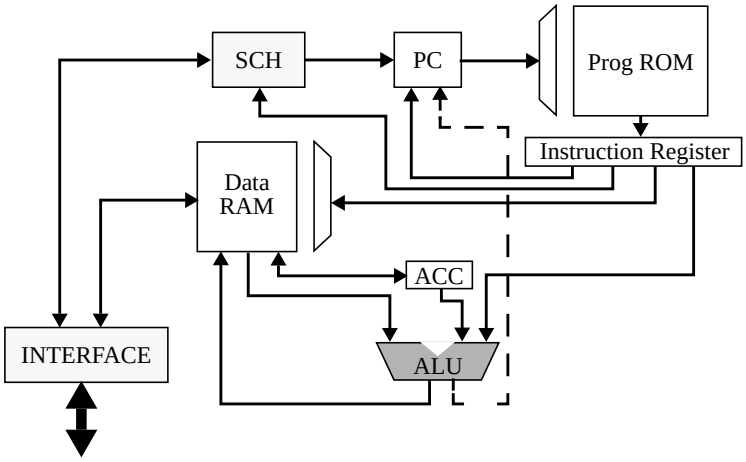

Figure 12: MSQ Top Controller of the SGS-Thomson Videophone

ural order of the program. The scheduler can access the interface directly and make decisions depending on values from the exterior.

For the MSQ, the mapping of the C source to standard arithmetic and control operations was relatively straight-forward. The only issues that arise are in the routing of data to the special ACC register and appropriate locations in the RAM. This is easily handled in virtual code selection. Architecture specific features required special attention, such as the mapping of case statements onto the indirect branching instruction. This instruction requires alignment upon specific bits. This is handled in the mapping to the target machine. The rule simply emits an alignment directive along with the assembly code.

For the interface to the rest of the chip, volatile register variables were defined for the access port. In this manner, compiler rules map reads and writes of these variables onto the appropriate interface instructions. It is important that the variables are defined as volatile, otherwise, compiler optimizations could remove the read and write accesses.

The C compiler for the BSP (Bit Stream Processor) (Figure 1) had similar targeting issues to that of the MSQ. The main differing issue was the treatment and optimization of bit manipulation operations used heavily in one part of the videotelephony process. The handling of the register interface was reused from the MSQ.

For the VIP (VLIW Image Processor) (Figure 1), the declaration of compaction resources was a fundamental development issue, due to the very large instruction word (VLIW) and multiple execution streams. In addition, several built-in functions which correspond directly to hardware functions needed to be designed. Often, hardware functionality which does not correspond to a simple C operation requires the provision of a *reserved* built-in function in C. These are supported by the compiler and also allow the developer to provide equivalent workstation functions for co-simulation purposes. Again, the register interface was reused from the MSQ. In this case, compaction is disallowed with interface functions. This can be guaranteed through a careful definition of the compaction resources.

C Compiler Results For the MSQ architecture, a subset of the H.261 code was taken from the prior co-design methodology. This code had previously been written in VHDL and was hand-translated to assembly (see Figure 2 VHDL model SW Behaviour). The examples contain a cross section of the different types of tasks the MSQ performs.

This code, originally written in VHDL, was rewritten in C nearly line-for-line and compiled using the MSQ C compiler. We then compared the compiled code with the hand-translated code written in assembler. The results are shown in Figure 13.

On average, the compiled code size is roughly equal to the hand code size. This indicates that for this processor, the compiler performs as well as an assembly-level programmer. This was possible because of the natural mapping from C to the instruction-set of the controller architecture. Only special cases needed to be addressed using the flexibility of rules.

In some cases the compiler produces more compact code than the hand code. One may argue that the hand-coded assembly could have been improved in some cases; however, this reflects the industrial reality of tight schedules that preclude exploring all possible optimizations of assembly code. In addition, for real-time systems the assembly programmer is concerned only with meeting the timing constraints. If the constraints are met, there is no need to further optimize, especially if it means costly development time on the hand assembly code level.

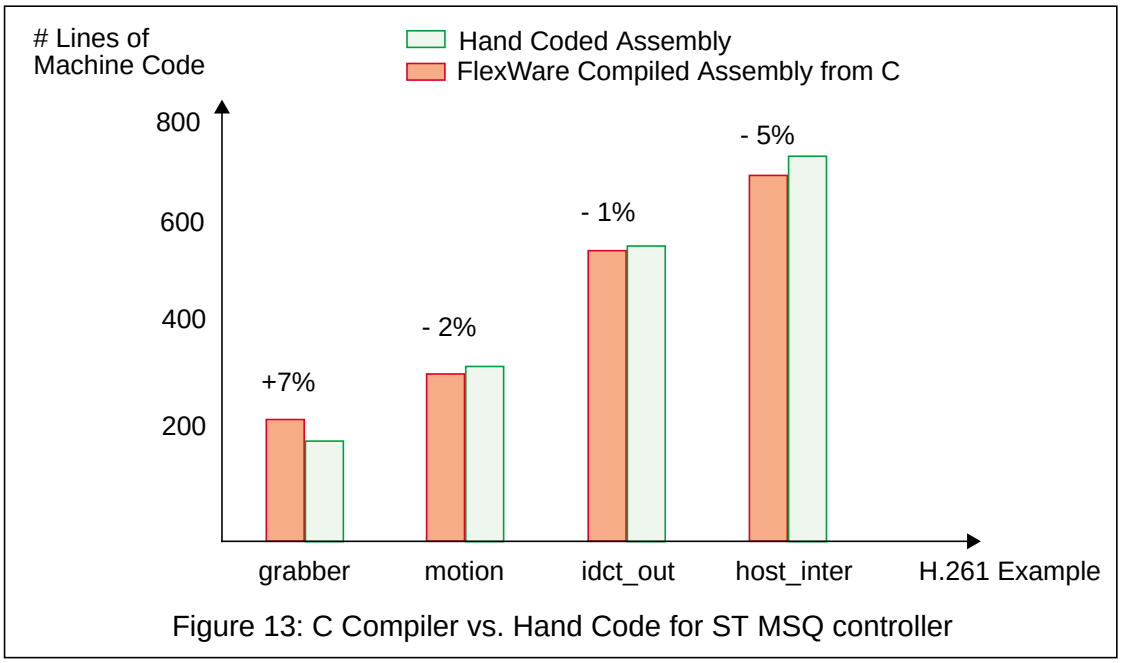

Figure 13: C Compiler vs. Hand Code for ST MSQ controller

For the BSP and VIP video processors, C compilers were also developed. The compiled code met all code size and performance constraints. Although we have no comparisons with hand code, this is a fortunate outcome of the previous benchmark which led to a decision by the design team to write all the code in C.

The results shown in this section indicate good quality results using a retargetable compiler methodology for several blocks of the videophone architecture. However, we cannot yet claim to have all the optimization capabilities in place for more complex architectures. For highly specialized arcitectures, the programmer must, in some cases, manipulate the source code based on his knowledge of the architecture to arrive at better results. This is not the most ideal fashion to program, as it discourages code portability, but it does provide the user with the important first step to accepting a compiler methodology over assembly-level coding. The support of the widest retargeting range independent of the source coding style producing quality compiler results is the goal of all research in the area of compilers for embedded processors [8].

Conclusion

Embedded processors and embedded software are rapidly becoming an essential part of the emerging system-on-a-chip. This is mostly in response to the desire of incorporating flexibility in order to track evolving standards and changing markets. Functional validation of C application code with VHDL and embedded software compilation play critical roles in this design process.

For functional validation, we have presented an effective approach to VHDL-C co-simulation, which has advantages over commercially approaches. In addition to improved flexibility on the style of C which may be simulated, it provides a distributed model of communication where multiple C programs can run in parallel. As well, the model allows the transparent incorporation of standard C debugging facilities. Results have shown that a simulation speedup of up to a factor of three is achievable over pure RTL-based VHDL simulation.

For software compilation, we have presented an effective environment for the retargeting of a C compiler to a variety of embedded processor architectures. From control-dominated to data-dominated architectures, the same development environment may be used to provide high quality compilers. The methodology has provisions to deal with embedded processor peculiarities. Furthermore, the approach handles architecture constraints such as interface protocols and it allows the mapping of reserved built-in functions onto specialized hardware operations. With a retargeting time on the order of one person-month, results have shown that the resulting compiler produces machine code comparable to hand-coded quality. Depending on the complexity of the processor architecture, optimization development time determines the total retargeting time.

As the amount of software content on the modern system-on-a-chip continues to grow, new co-design and co-validation methodologies are needed to support the design cycle. We have presented two critical technologies for today's design flow; however, there are also many other practical related issues including: compiler validation, source-level debugging, and instruction-set simulation. In addition, the embedded processor methodology opens new opportunities for areas such as instruction-level profiling, static and dynamic analysis of instruction utilization, and instruction-set design.

Acknowledgments

The authors would like to thank the members of the Integrated Video Telephone design team at SGS-Thomson Microelectronics, especially Olivier Deygas, José Sanchez, and Michel Harrand. Being early users of the tools, they were essential to the development of production quality software.

References

[1] M. Harrand et al., "A Single Chip Videophone Encoder/Decoder", *Proceedings of the IEEE International Solid-State Circuits Conference*, Feb. 1995, pp. 292-293

[2] E. Berrebi, P. Kission, S. Vernalde, S. DeTroch, J.C. Herluison, J. Fréhel, A. Jerraya, I. Bolsens, "Combined Control-flow Dominated and Data-flow Dominated High-level Synthesis", *Proc. of the Design Automation Conference*, June 1996, pp. 573-578.

[3] P. Paulin, J. Fréhel, M. Harrand, E. Berrebi, C. Liem, F. Naçabal, JC Herluison , "High-Level Synthesis and Codesign Methods: An Application to a Videophone Codec", *Proc. of EuroDAC/EuroVHDL*, Sept. 1995.

[4] C.A. Valderrama, A. Changuel, P.V. Raghaven, M. Abid, T. Ben Ismail, A.A. Jerraya, "A Unified Model for Co-simulation and Co-synthesis of Mixed Hardware/Software Systems", *Proc. of ED&TC*, March 1995.

[5] S. Vercauteren, B. Lin, H. DeMan, "Constructing Application-Specific Heterogeneous Embedded Architectures from Custom HW/SW Applications", *Proc. of the Design Automation Conference*, June 1996, pp. 521-526.

[6] F. Naçabal, O. Deygas, P. Paulin, M. Harrand, "C-VHDL Co-Simulation: Industrial Requirements for Embedded Control Processors", *Proc. of EuroDAC/EuroVHDL Designer Sessions*, Geneva, Sept. 1996.

[7] C.A. Valderrama, F. Naçabal, P. Paulin, A. Jerraya, “Automatic Generation of Interfaces for Distributed C-VHDL Co-simulation of Embedded Systems: an Industrial Experience”, *Proc. of the Int. Workshop on Rapid Systems Prototyping*, June 1996.

[8] Code Generation for Embedded Processors, ed. by P. Marwedel, G. Goossens, Kluwer Academic Publishers, 1995.

[9] C. Liem, P. Paulin, M. Cornero, A. Jerraya, “Industrial Experience using Rule-driven Retargetable Code Generation for Multimedia Applications”, *International Symposium on System-Level Synthesis,* Cannes, France, Sept. 1995, pp. 60-65.

[10] R.P. Gurd, “Experience Developing Microcode Using a High-Level Language”, Proc. of the 16th Annual Microprogramming Workshop, Oct 1983, pp. 179-184.

Biographical Information: Clifford Liem

Clifford Liem received the Bachelor of Science degree in Physics from St. Francis Xavier University, Antigonish, Nova Scotia, Canada in 1989 and the Master of Engineering degree in Electronics from Carleton University, Ottawa, Ontario, Canada in 1991. From 1991 to 1994, he held an engineering postion at Bell-Northern Research / Northern Telecom Ltd. in Ottawa where he worked on retargetable compilation, processor modelling, and RTL synthesis.

He is currently completing the Ph.D. degree at the TIMA laboratory, Institut National Polytechnique de Grenoble (INPG) in tight co-operation with SGS-Thomson Microelectronics on methods and tools for embedded instruction-set processors.

In addition to his research interests in retargetable compilation for embedded processors, he is involved in instruction-set analysis and design, as well as hardware-software co-design.

Biographical Information: François Naçabal

François Naçabal received the M.S. degree in microelectronics from the Pierre-et-Marie Curie University of Paris, France in 1993. He is currently working toward the Ph. D degree in the Institut National Polytechnique of Grenoble, France in cooperation with SGS-Thomson Microelectronics.

He has been working on hardware-software cosimulation and compilation for embedded software. His research interests include hardware-software codesign, architecture exploration and performance analysis for digital signal processors.

Biographical Information: Carlos Valderrama

Carlos Valderrama received an Electrical-Electronics degree from the National University of Cordoba, Argentine in 1989 and a M.S. degree in microelectronics from the Federal University of Rio de Janeiro, Brazil in 1993. He is currently working toward the Ph. D degree in the Institut National Polytechnique of Grenoble, France.

He has been working on a VHDL compiler-simulator and on hardware-software codesign. His research interests include hardware-software codesign in general, cosimulation and cosynthesis in particular.

Biographical Information: Pierre Paulin

Pierre Paulin received the B.Sc. degree in engineering physics in 1982 and the M. Sc. degree in electrical engineering in 1984 from the Universite Laval, Quebec city, Canada. He obtained the Ph.D. degree in electronics engineering in 1988 from Carleton University, Ottawa, Canada, as part of a cooperative research project with BNR/Northern Telecom, Ottawa. This led to the development of the force-directed scheduling approach that has since been used in many high-level synthesis systems.

From 1989 to 1994, Dr. Paulin was with BNR/Northern Telecom, Canada, where he was responsible for embedded software development tools. He is currently a manager in the Central R&D division of SGS-Thomson Microelectronics in Grenoble, France and is responsible for the Embedded Systems Technology group.

Dr. Paulin was awarded the Design Automation Conference (DAC) best presentation award in 1986, and his 1987 and 1989 DAC papers were nominated for best paper awards. His interests are in the fields of hardware/software co-design, embedded systems, compilation, high-level synthesis and system-level design tools.

Biographical Information: Ahmed Jerraya

Ahmed Amine Jerraya received the engineering degree from the university of Tunis at Tunisia in 1980, a Docteurs Ingenieur degree in 1983 and a Docteur d'Etat degree in 1989 from the University of Grenoble in France. He has held a full research position with the CNRS (Centre National de la Recherche Scientifique) in France since 1986.

He participated in several successful french CAD projects such as the LUCIE project in the early 80's and more recently the high-level synthesis system AMICAL. He served as a general chair for the 9th International Symposium on System Synthesis (1996).

He is currently the head of the system-level synthesis group within TIMA/CNRS-INPG-UJF where he is developing methods and tools for behavioral synthesis and hardware-software codesign.