Cycle Accurate Simulation Model Generation for SoC Prototyping

Antoine Fraboulet, Tanguy Risset, and Antoine Scherrer
(Lip, ENS-Lyon, Citi, Insa-Lyon, Inria, UCBL, CNRS, EPML Soclib*)

Citi, Insa-Lyon, 21 av. Jean Capelle
69621 Villeurbanne Cedex
antoine.fraboulet@insa-lyon.fr

LIP, ENS Lyon, 46 allée d'Italie
69364 Lyon Cedex 07
{tanguy.risset,antoine.scherrer}@ens-lyon.fr

Abstract. We present new results concerning the integration of high level designed IPs into a complete System on Chip. We first introduce a new computation model that can be used for cycle accurate simulation of register transfer level synthesized hardware. Then we provide simulation of a SoC integrating a data-flow IP synthesized with MMAlpha and the SocLib cycle accurate simulation environment. This integration also validates an efficient generic interface mechanism for data-flow IPs.

1 Introduction

With the advent of multi-processors system on chip (MPSOC), fast cycle accurate hardware simulation is a major issue in the design process. Because most hardware design projects now include a significant software part, the simulation time increases a lot when it becomes precise enough to model highly unpredictable mechanisms such as cache misses or bus/network contentions. In addition the ideal design scheme where one could easily re-use IPs designed elsewhere, as it is naturally done in software development, is far from today's habits mainly because of the lack of standardization of communication protocols between IPs.

The SocLib* initiative proposes some advances in the resolution of these problems. First it proposes to write simulation models of IPs using SystemC with precise rules at various level of refinement: *Transaction Level Modeling* (TLM) for software-like prototyping and *Cycle Accurate, Bit Accurate Modeling* (CABA) for a cycle accurate simulation of the IP. Then, it proposes to use VCI (Virtual Component Interface [1]) communication low level protocol to interface easily various IP. Finally it suggests a certain programming style that enables the possibility of writing very efficient simulation engines (closer to step by step simulation than to event driven simulation [2]).

During the design of a MPSOC, some IPs are re-used (processors, memories, DMA, etc.) but some (usually few of them) must be designed specifically for the

* The authors would like to thank members of the Asim group in LIP6 Lab. for many fruitful discussion as well as for the production of many simulation models and tools used in this paper.

* http://soclib.lip6.fr/

application targeted. The design of these new IPs and the integration in the global system must be fast to prototype rapidly the whole application. For many years now, people have been working on *high level synthesis* to reduce the time, effort and potential errors due to manual design of hardware. These research have lead to prototypes and industrial tools that perform this *semi-automatic* design of hardware from high level (functional or behavioral) specifications. One can check the following academic tools: MMAlpha [3], Gaut [4], Paro [5], Compaan [6], Hugh [7] and the industrial initiatives: Pico [8] (Synfora, spin off from HP), CriticalBlue [9] and probably many more to come. Most of these tools address specific target circuits to reduce the search space during refinement, many of them target data-flow computations (sometimes referred to as *stream processing*) widely used in signal processing applications. This is the case for MMAlpha that is used in this work to generate highly parallel hardware accelerators for signal processing filters. It is important to notice that the *de facto* failure of general high level synthesis tools such as *behavioral compiler* from Synopsys highlighted the fact that high level synthesis was intrinsically difficult to achieve and that the long research effort provided to produce the tools mentioned above was justified.

However, most of these tools are now faced to the problem of integrating their IPs in MPSOC simulation and synthesis environments. The two most important problems encountered as soon as integration is envisaged are:

1. The choice of a communication protocol that provides data to the IP. This choice is driven by many factor: the simplicity of the protocol, the size and power of the hardware necessary for it and its impact on the performance of the global system.
2. The generation of a simulation model for the hardware synthesized which is compatible with the simulation platform used.

This paper explains which solutions were chosen for the MMAlpha tool and it also extracts some underlying generic solutions that can be re-used for other tools. More precisely, we propose a new target computation model for the translation from classical structural RTL representation of hardware to finite state machine representation. We also present an efficient algorithm for performing the translation. This algorithm has been used for CABA simulation model generation from MMAlpha. The generated simulation model have been integrated into a simple SoC composed of a MIPS R3000 processor, a standard RAM, a generic interconnect and a hardware accelerator generated with MMAlpha. This platform runs a simple signal processing program filtering audio files. Apart from the validation of the above mentioned computation model, this experiment was used to validate the SocLib approach and the use of a generic interface scheme for connecting external IPs generated by high level design tools in such a framework.

In section 2 we introduce the new computation model that we would like to promote for high level design tools. In Section 3 we present the SoC simulation performed using SocLib and MMAlpha. we have improve the interface mechanism between that hardware and software that was presented in [10] so that it can be reused for any data-flow IP in the SoC platform, simulation results concerning the whole SoC platform are presented there too.

2 FSM representation for RTL simulation models

Now that high level synthesis tools are gaining importance, the question is raised of how to generate efficient simulations models from internal representation of the tools which usually corresponds to *classical* RTL representation. In this generation process, a crucial issue is the efficiency of the CABA simulation. The community now widely agree that finite state machine (FSM) is a good computation model for simulation because it can be easily understood by designers and it can be efficiently simulated. Following initial ideas of Jennings [11], many works have focussed on the problem of improving MPSOC simulation performances by reducing the inefficiency introduced by event driven simulation. In particular, the results of [2] state that if the FSM is represented in the form of Moore-Mealy machine, the simulation kernel can produce much more efficient simulations.

A moore-mealy machine is an automaton with an internal state and three functions: the `transition` function, the `moore` function and the `mealy` function. The standard definition of these functions is the following: the `transition` function uses the inputs and the current internal state to update the internal state. The `moore` function uses the internal state to provide outputs and the `mealy` function uses the inputs and the internal state to produce output (see the left of figure 2). In hardware component modeling, the internal state is composed of the internal registers of the component and the state is computed at each clock cycle.

This section present a new computation model that improves SystemC code quality and performances when it is used for RTL hardware modeling. A more precise description of this new model can be find in the research report [12].

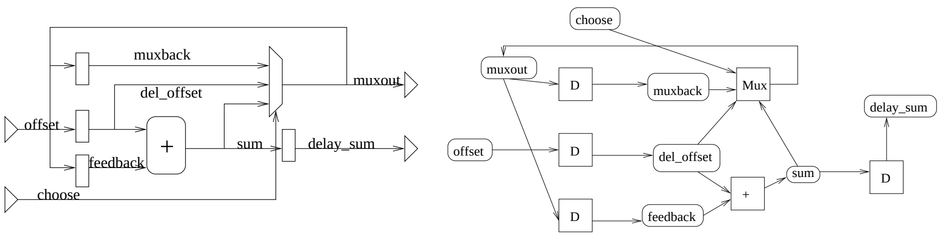


Fig. 1. Original Jennings' RTL example and its representation as a RTL graph

2.1 A new FSM model for hardware modeling

In [11], Jennings introduced the *directed cyclic bipartite graph*, referred here as the RTL GRAPH, for abstracting RTL descriptions. We use the original Jennings' example in figure 1 to illustrate it (in the following we denote by the *module* the whole hardware component to be abstracted).

Applied on the RTL graph of figure 1, a straightforward algorithm for generating a Moore-Mealy machine will lead to the graph represented on the left

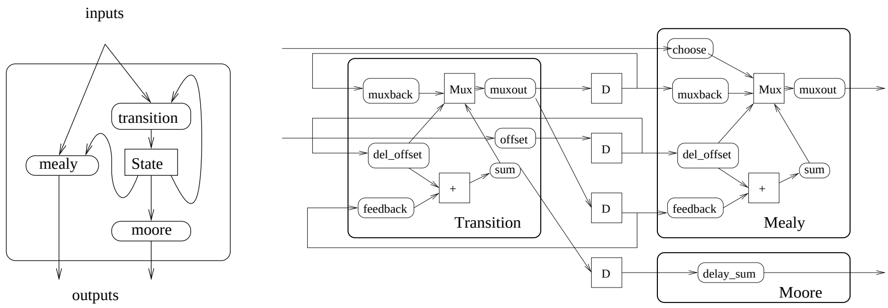


Fig. 2. The standard Moore-Mealy machine used for writing CABA simulation models of IPs (on the left) and its application for the hardware of figure 1.

of figure 2. The computations that are used in more than one function (here `transition` and `mealy`) are duplicated. Code duplication can be error prone (if the code is modified during the debugging process for instance) and it also implies an increased complexity during simulation because computation will be done twice.

We propose to distribute the computations performed by the hardware in the three FSM functions without duplicating any one. For this we use a slightly modified definition of the three functions: the `moore` function inputs are unchanged, the `mealy` function will take as input some inputs of the module and some outputs of the `moore` function. Both functions `moore` and `mealy` will produced outputs that will be used by the transition function (while in the original definition, the `transition` function used only module inputs and register outputs). This is illustrated on the left of figure 3.

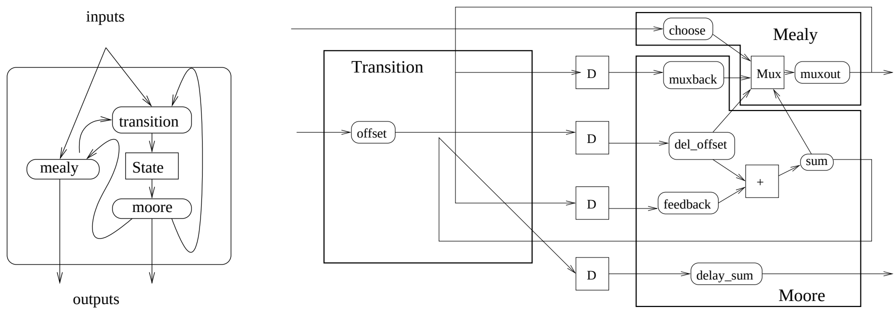


Fig. 3. The new machine used for writing CABA simulation models of IPs (on the left) and its application for the hardware of figure 1

With this target model in mind, a possible FSM for the module of figure 1 would be the one of figure 3 where no code duplication occurs. The construction of this FSM can simply consist of a rewriting of the RTL graph. Indeed, each hardware operator will be present in one and only one of the three functions: `transition`, `moore` or `mealy`. A clever traversal of the RTL graph will label each hardware operator of the graph with the type of function it belongs to (`transition`, `moore` or `mealy`). The nets will be classified too, but they may be duplicated if they are connected to operators of different type (we want the operands and results of an operator to be of the same type, this will ease the translation to a programming language).

The algorithm is *efficient* in the sense that its complexity is linear, assuming that one can have access in a constant time to registers. Indeed, each node of the graph is traversed only once (except for the duplications at the end of the algorithm but we can safely assume that the number of registers is small compared to the total number of nodes of the graph). It is *clever* in the sense that it does not duplicate any computation, hence the resulting simulation model of the hardware will be faster. The complexity of the graph on the right of figure 2 and figure 3 illustrates it.

A more detailed description of the algorithm can be found in [12].

2.2 RTL simulation experiments with SystemC

In a SystemC implementation of this new FSM machine, we must ensure that the three methods corresponding to `mealy`, `moore` and `transition` functions are evaluated in a valid order. In [12], we explain precisely how it can be ensure in the SystemC framework.

For our experiments we have derived a specific hardware module: a non-pipelined version of the DLMS filter presented in [13, 3]. The RTL version of the architecture has been derived with MMAlpha from the same specification as the one used in [3]. The designer chose not to pipeline the architecture and hence obtained moore and mealy computations in each cell. Moreover, each mealy computations are connected from cell to cell, hence a complex dynamic scheduling occurs at simulation run time. This architecture is composed of `N` cell with `N-2` identical cells.

We generated SystemC code compliant with the model introduced above for this architecture with `N`=30 and `N`=60 cells. This gave rise to three type of cell: the first cell of the array, the last cell of the array and the middle cell repeated `N-2` times. Each of these SystemC cell where implemented with this new FSM machine convention. Then we took the middle cell and modify it by hand so that it can stick to the classical Moore-Mealy machine. This consisted in duplicating some of the code to use only cell inputs or register outputs in each method. It is worth noting that this represented an increase of 60% of the computations lines and 120% in the number of variables declared in the methods.

These simulation where done with systemC-2.0.1 on linux operating system (processor pentium 4M, 1.4 GHz with 512 Mo ram). C++ Compilation was done with GCC 3.3 without any optimization options. We also did the simulation with the SystemCass simulation kernel [2]. The results are presented in table 1.

These results show an improvement, between 30% and 50%, in the simulation performances obtained by SystemC simulation with the new model compared to

	SystemC		SystemCass	
	new model	Moore-Mealy	new model	Moore-Mealy
Cell	0.58	0.76	0.13	0.2
module N=30	8.82	15.07	3.09	5.1
module N=60	17.56	30.63	6.14	10.34

Table 1. Simulation performances, in seconds, for a simulation of 100 000 clock cycles

equivalent hardware described as a classical Moore-Mealy machine. Simulation with SystemCass show that our improvement can be added to the simulation acceleration provided by static scheduling techniques. Note that simulation performance of SystemC and SystemCass should not be compared here because SystemCass currently does not implement real bit-true mechanism for handling the `sc_int` type (`int` type is used). The simulation time decrease is not huge, but we point out that this is not only the main advantage of our new coding style, it has other advantages: (i) it suppresses code duplication which is error prone, and (ii) it generates simulation models that are closer to final RTL implementation. This property could be used, for instance to derive automatically VHDL RTL models from the corresponding SystemC cycle accurate simulation models.

3 Soc platform simulation

Choosing the good simulation platform is a crucial issue in MPSOC design because it implies a non-negligible investment for using the environment and developing the simulation models. Commercial tools may offer powerful environment but the lifetime of a tool will frequently be much shorter than the lifetime of an IP. We believe that the open source nature of SystemC** is adapted to this kind of constraint. We have used SocLib for setting up the SoC simulation environment and the MMAlpha tool for high level synthesis of dedicated hardware.

The main component of SocLib is currently a set of open source SystemC CABA simulation models for common IPs (MIPS processor, RAM, NoC, busses, DMA,. . .). The SocLib simulation models use the VCI (Virtual Component Interface [1]) communication low level protocol to interface with other IPs.

In this framework, a MPSOC platform consists of a set of hardware IPs together with some software code running on the programmable IPs (here the MIPS processor), which imply the use of a compiler. We used GCC targeted to MIPS core. As we used a generic network on chip interconnect (and not a standard bus), it is mandatory to provide a simple operating system that ensure correct behavior of the whole system (memory coherency, exception handling, etc.). Hence to be usable, SocLib must be associated to such an operating system compatible with

** http://www.systemc.org/

the compiler used. We are currently using Mutek [14] which provides a very light implementation of the Posix threads API.

MMAlpha [3] is a toolbox for designing regular parallel architectures (systolic like) from recurrence equation specifications expressed in the Alpha language. It is one of the only existing tools that really automate the refinement of a software specification downto RTL description within the same language: Alpha. MMAlpha's methodology for refinement successively introduces time (global synchronous clock), space (Mapping to 1 or 2-D array of processors), control generation and finally RTL level generation.

We have developed a translator from AlpHard (hardware description language, subset of Alpha) to SystemC, based on the principles explained in section 2. The translator also provides testbench that allow hierarchical debugging and which uses a stimuli file format which is common to the C code generator and the VHDL code generator of MMAlpha.

3.1 The SOC platform

We have chosen a classical linux audio signal processing application called *Gramofile*. Gramofile processes audio files (.wav format) and proposes various simple filters like LP's tick removal. We have extracted a simple filter (referred as `filter.c`), and translated it in Alpha by hand. The translation was validated by replacing the original `filter.c` file by the C code generated from the `filter.alpha` by MMAlpha. Through all the refinement stages of MMAlpha, this C translation can be used to test the validity of the current description.

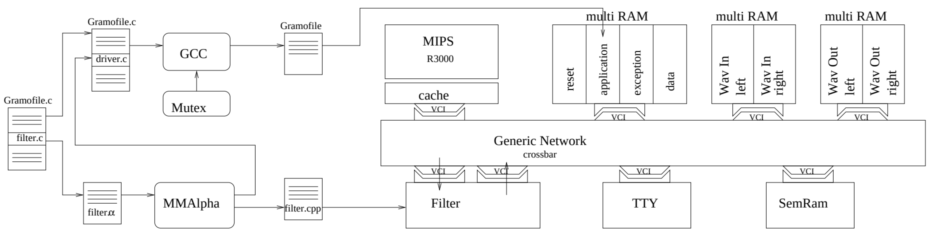
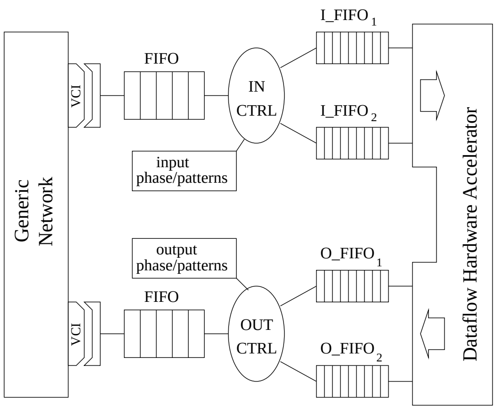


Fig. 4. The SoC simulated and the global design methodology. Blank square box represent C++ simulation models of IPs.

The target platform chosen and the software design method are represented on figure 4. A single MIPS R3000 processor with its associated data and instruction cache is used. Standard memories are used for storing the program and data, a special memory is used for semaphores. A dedicated component is used for displaying output (referred as TTY). The hardware accelerator performing a filter is generated with MMAlpha. All these components are connected via VCI ports to a simple network. The software running on the MIPS, in addition to bootstrapping information, is composed of the gramofile program cross-compiled with GCC to a MIPS target. In this gramofile program, a specific function (referred as `filter.c`) has been translated in Alpha (MMAlpha input language) in order to generate the hardware accelerator IP. During this high level synthesis process, MMAlpha generates a `driver.c` which replaces the `filter.c` function in gramofile and ensures that data are correctly exchanged between the memory and hardware accelerator.




```
// input phase 0
for(i=0;i<26144;i++){
  SEND(psl_mirr1,30);
  SEND(psr_mirr1,30);
}
// input phase 1
SEND(psl_mirr1,20);
SEND(psr_mirr1,20);
// output phase 0
RECV(pres_left,8);
RECV(pres_right,8);
// output phase 1
for(i=0;i<26144;i++){
  RECV(pres_left,30);
  RECV(pres_right,30);
}
// output phase 2
RECV(pres_left,18);
RECV(pres_right,18);
```

Fig. 5. Hardware and Software interfaces for MMAlpha data-flow IP. Controllers must be configured according the software communication pattern used in the software driver on the right hand side. Both the hardware and software can be parameterized to allow a maximum throughput using burst communications and/or stream interleaving

In this paper we do not describe precisely the design of the accelerator with MMAlpha (see [3] for details), the RTL representation obtained in Alphard for a simple mean filter is composed of `F`=8 cell containing two additions and some register each, plus a shift for the final division. This accelerator is associated with an interface (mentioned hereafter as the Alpha *controller*) which connects on the bus and carefully dispatches the data coming from/to the memory on the right input/output ports, this is represented on figure 5. The controller is based on the principles explained in [10] extended to handle burst mode (see [12] for details). Note also that this hardware accelerator comes with a *driver* represented on the right of figure 5, i.e. a piece of software code that will replace the original `filter.c` function in the C code. This driver is also generated by MMAlpha together with a set of parameters that tunes the controller accordingly.

3.2 System and simulation performances

The SoC simulation results are presented in the table 2. In this implementation we used a simple memory transfer mechanism driven by the processor. This dual read/write generates a significant overhead due to memory access and system interconnect latency taken into account for each word transfered by the processor to the accelerator or back to memories. This overhead can be overcome by using a DMA engine to speedup memory copies using burst transfers between the memories and the hardware accelerator.

complete simulation time	29.21 s
complete simulation cycles	600000 cycles
simulation speed	20540.9 cycles/seconds
hardware pipeline throughput	20450 cycles

Table 2. Performances obtained from the complete SoC simulation

It is interesting to compare these results with approach of Quinton et al. [15] where a very similar application is prototyped by using emulation on a real platform (Lyrtech SignalMaster, with a DSP and a FPGA) rather than simulation. Their implementation uses an API provided by Lyrtech for hardware-software interface (roughly the equivalent of our interface controller and driver), unfortunately we have no precise estimation of the performance of the interface. In [15] the simulation time of the complete application per filter sample is approximately $6\mu s$ while our simulation reaches $29.21/20450 \simeq 1.4ms$ per filter sample (250 times slower for a simpler application). Hence, the prototyping approach is much better as soon as simulation time is targeted. However, if cycle accurate simulation is sought, our interface mechanism is much closer to what will be on the final chip than the one used in [15] and the behavior of our simulation is exactly what will occur on the real system.

4 Conclusion

In this paper we have presented an efficient computation model for performing cycle accurate hardware simulation of RTL description of hardware. This representation is particularly useful for high level synthesis tools that provides *generated* simulation models rather than manually written simulation models. We have validated the simulation time improvements provided by this new computation model but we believe that the quality of the code generated is a more important advantage of our model.

We have also presented a complete SoC simulation integrating IPs coming from different places: manually written IPs developed for standard SoC platform and a hardware accelerator IP generated with MMAlpha and associated with a generic hardware interface and software driver, both generated by MMAlpha. This integration is a case for the use of the SocLib environment and it also highlighted the good efficiency of our generic hardware/software mechanism.

References

[1] Alliance, V.: Virtual component interface standard (ocb specification 2, version 1.0) (2000)

[2] Pétrot, F., Hommais, D., Greiner, A.: A simulation environment for core based embedded systems. In: Annual Simulation Symposium, Atlanta, GA, U.S.A (1997) 86–91

[3] Guillou, A.C., Quinton, P., Risset, T., Wagner, C., Massicotte, D.: High level design of digital filters in mobile communications. Technical Report 1405, Irisa (2001)

[4] Sentieys, O., Diguet, J., Philippe, J.: Gaut: a high level synthesis tool dedicated to real time signal processing application. In: EURO-DAC. (2000) University booth stand.

[5] Bednara, M., Teich, J.: Interface synthesis for fpga based vlsi processor arrays. In: Proc. of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA 02), Las Vegas, Nevada, U.S.A. (2002)

[6] Kienhuis, B., Rijpkema, E., Deprettere, E.: Compaan: Deriving process networks from matlab for embedded signal processing architectures. In: 8th International Workshop on Hardware/Software Codesign (CODES'2000). (2000)

[7] Augé, I., Donnet, F., Gomez, P., Hommais, D., Pétrot, F.: Disydent: a pragmatic approach to the design of embedded systems. In: Design, Automation and Test in Europe Conference and Exhibition (DATE'03 Designers' Forum), Paris, France (2002)

[8] R. Schreiber et al.: High-Level Synthesis of Non Programmable Hardware Accelerators. In: IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP 2000), Boston (2000)

[9] Hounsell, B., Taylor, R.: Co-processor synthesis a new methodology for embedded software acceleration. In: Design, Automation and Test in Europe Conference and Exhibition (DATE'03 Designers' Forum), Paris, France (2004)

[10] Derrien, S., Guillou, A.C., Quinton, P., Risset, T., Wagner, C.: Automatic synthesis of efficient interfaces for compiled regular. In: Internationnal Samos Workshop on Systems, Architectures, Modeling and Simulation (Samos), Samos, Grece (2002)

[11] Jennings, G.: A case against event-driven simulation for digital system design. In: Proceedings of the 24th annual symposium on Simulation, IEEE Computer Society Press (1991) 170–176

[12] Fraboulet, A., Risset, T., Scherrer, A.: Cycle accurate simulation model generation for soc prototyping. Technical Report 2004-18, LIP (2004)

[13] Naresh, R.S., Keshab, K.P.: Pipelined Adaptative DFE Architecture Using Relaxed Look-Ahead. IEEE Transactions on Signal Processing **43** (1995) 1368–1385

[14] Pétrot, F., Gomez, P.: Lightweight implementation of the posix threads api for an on-chip mips multiprocessor with vci interconnect. In: Design, Automation and Test in Europe Conference and Exhibition (DATE'03 Designers' Forum), Munich, Germany (2003)

[15] Charot, F., Nyamsi, M., Quinton, P., Wagner, C.: Architecture Exploration for 3G Telephony Applications Using a Hardware–Software Prototyping Platform. In: Proc. of Samos IV, Samos, Greece (2004)