A technique for generating efficient simulators

Przemyslaw Bakowski
Université de Nantes
IRESTE, la Chantrerie
44087 Nantes
France

Jean-Luc Dubois
Université de Lille I
I.E.E.A., Bât. M3
59650 Villeneuve d'Ascq
France

Adam Pawlak
G.M.D.
Project E.I.S.
5205 St. Augustin
F.R. Germany

Abstract
We present in this paper a methodology aiming at reducing the size of generated simulators and at increasing their simulation speed. This technique consists in abstracting the functionality of an architecture's control part into a data table and mapping it onto the main memory of a host computer. The Lille University simulator called LIDO was used as a testbed for experiments. Some decision making capabilities of our generator based on static analysis of characteristics of an architecture and empirically refined are presented then. Finally, we discuss results of comparative tests with a VHDL simulator.

1 Introduction
Our interest in this paper will be focussed on the generation of efficient simulators from HDLs to the C language. As a working example we have used the LIDO [1] language developed at the University of Lille. In the LIDO system, the functional verification of initial specifications is realized by two complementary tools: the interactive global interpretor, and the generator of models in the C language (C-models) for fast compiled simulators (Fig. 1). The interactive interpretation used in the system offers a rich user interface including graphic, symbolic and numeric tracing, and profiling. This permits to analyse profoundly all aspects of modelled architectures, but heavily burdens (5 to 10 simulated instructions per second). Therefore, we have developed a translator to the C language in order to obtain portable, flexible, compact and fast simulators for target machines performing operations on up to 128 bit-words, described in the LIDO language [1], basically at the RT level.

The first version of our LIDO to C generator has been based on a direct translation, i.e. without any modification of the input description. The compactness of a C-code and a simulator's speed in this first implementation of the tool were very promising, i.e. comparable (if not better) with other published results [4, 6, 7]. But, in the case of large systems such as MC680xx or network controlling ICs, the generated simulators size proved to be too large and not compilable (e.g., the simulator of a MC68000 description is over two megabytes of source code) at all. The problem thus was to reduce the size of the simulator's source code in order to guarantee the feasibility of compiling models of large scale architectures. Some process oriented simulation tools, like VHDL simulators, solved this problem by splitting up the model into design units whose activities are coordinated by a kernel process. The main drawback of this solution is that it penalizes the simulation speed and restricts host computers to several Mips rate machines. Another well-known (hardware) solution is a construction of a specialized highly parallel hardware but an extremely high cost of this approach limits tremendously its class of users. Thus, there is a great need for an efficient solution on a typical computer reducing a simulator's source code without reducing its execution speed.

2 Simulation-oriented pre-synthesis
2.1 The idea of the solution
In this paper, a software solution is proposed, which aims at reducing both the simulator's size and its simulation time. The size and speed limits are mainly due to the traditional translation scheme algorithm to
implies the identification and separation of:

2.2 Our solution

It constitutes a synthesis of programmed structures based on a mapping of all control path functions onto the main memory of a host computer (data representation), whereas all data path functions are mapped on the processing unit of a host computer (code). This technique is known as "ROM approach" and used during a model design phase to implement logic functions. This solution, however simple in principle implies several transformational and optimization difficulties. For example, in some cases an enormous size of a RAM memory may be experienced, e.g. a control path with 32 logic inputs and 128 logic outputs needs the space of 84 Giga bytes. Thus, this design step requires a good functional and/or structural decomposition of models fortunately provided by the use of structured HDLs.

This partitioning must be realized by a hardware designer. On one hand, the functional decomposition implies the identification and separation of:

- sequencing functions (the highest level of control path),
- decoding operations (the lowest level), and
- synchronization functions.

To do so, and in order to obtain compact control path descriptions, a designer may use among others various techniques such as:

- the property extraction, i.e selecting the smallest subset of conditions sufficient for a decision, and
- the parameterization mechanism, i.e static generation of commands depending on the specific execution condition.

On the other hand, the structural decomposition has to be done using control structures (predefined in LIDO) such as PLAs, decoders and memory units. When not interested in a structural decomposition a hardware designer may use CDFGs.

2.3 Generation steps

Given a description of a complex architecture in the LIDO language, decomposed as suggested above, the generator creates an internal representation for each storage element, and binds it to the associated identifier. Then the transformation of control path functions into a PLA form takes place providing that the corresponding option was used. The last step is the generation of the C-model and the possible update of the internal library. The analysis is performed during all steps. Before proceeding to the simulation of the architecture, the user has to compile the C-model, the required libraries if needed, and to link all modules together. Simulators generated in this way do not offer a good user interface. One shall use among others a symbolic debugger to display the values of identifiers during a simulation. This restriction may be an advantage when mixing different C-models. As data types used in different C-models have not to be standardized (a symbolic debugger accepts the whole C language as input), a generator can utilize target data types well adapted to modelled storage elements. Thus, the user interface is no more a bottleneck for the simulation efficiency.

The pre-synthesis is realized in two stages. At the first one, all complex modules like sequencers, PLAs and decoders are transformed into optimized and complete PLA structures dedicated to our method with the following transformation steps:

- separation of control and data flow:
  - identification of a data-flow in PLA and Graph LIDO units,
  - a signal (furtherly used as a guard) is generated in place of the data flow,
  - the data-flow (guarded) is copied into a LIDO Functional Unit in the data path,

- transformation of control structures into PLAs,

- PLA expansion:
  - pterms are produced for all combination of input values so that the OR operation will not be performed at run-time,
  - the (abstracted) PLA functionality is mapped into a data table

- data table compaction.

The last step of this stage (compaction) may be skipped for a given PLA due to a decision of the generator. This decision is based on the predicted ratio between:

- the size saved by our compaction algorithm (calculated), and
- the simulation time lost due to a more complex access to a data table (predicted using previous experiments and current PLA characteristics).
The second stage is the first part of the back-end. At this stage, data tables are transformed into C-tables. The resulting C-models contain several tables (RAM blocks) with data implying control path functions. In this way the evaluation of all AND-lines of a given PLA appearing when direct translation is used is replaced by the calculation of the RAM address (determined by the PLA inputs) plus the access to the value. The OR operation has been realized during the expansion step and does not appear any more. The closely related form of a synthesized control path with an underlying memory system facilitates the generation of control path prototypes based on RAM/PROM chips. Thus, such a tool could easily produce an EDIF output defining the memory components and their netlist. It provides a simple, rapid and inexpensive way to develop experimental hardware architectures.

3 Results

3.1 Performance criteria and evaluation technique

The criteria considered in our measurements concern the resources needed to perform a simulation, i.e., memory and time requirements. The first one considers the HDL model size and especially the source code size of the simulator, whereas the second reflects the need for a CPU time. The last one encompasses: the description compilation time, the C-model compilation time, and especially the simulation time of a model.

The number of events performed by second is often used as a characteristic of a simulator. But neither the VHDL simulator nor our generated simulators provide this number. The reason is that, such a calculation would increase the simulation time and the object code size. Another parameter being the number of transactions (NT for short) performed by second is calculated by the simulators. We often noticed when simulating with our C-models that NT may reduce and even invert the result of the comparison between two models. This is also the case with two models of the 16-bit up counter. If we now mention the corresponding NT (196609 for the functional description and 747103 for the gate level description), one can make sure of it when looking at the Table 2, 513 and 604 are the respective NT/s.

The key parameters for the simulation speed-up method being the simulator's size and a CPU simulation time proposed in this paper and obtained in a numerous experiments were carefully analyzed using the regression technique. Equations providing for a prediction of the simulator's size and the CPU simulation time have been presented in [3]. We made our experiments first on a HP 300 workstation, but the resulting figures were according to the results obtained on SUN workstations.

3.2 Comparison with a VHDL simulator

We have compared the performance of our generator and its C-models with the Intermetrics VHDL simulator as VHDL is now a standard in the field of HDLs and it uses the C language as "an intermediate format". Another common feature is the ability to communicate designs: through VHDL models for the one hand, via portable C-models for the other hand. To make them compatible (different VHDL models or LIDO-C-models working together), conventions must...
be defined. A discussion of those being mostly still an open problem is not our purpose in the context of this paper.

The comparisons we made, were based on the simulation of a 16-bit up counter using a functional description and the corresponding four 4-bit up-counter (a structural description at a gate level) and for a more complex design - a RISC architecture control path. Counter models were simulated for a count from 0 to \( 2^{16} \) whereas the RISC control path was simulated for 105 clock periods. Table 2 and Table 3 show these results. Time figures are not normalized since they were obtained on the same host computer. In all cases, our C-models are faster than the VHDL simulator. The higher the level of abstraction, the greater is the gap: 17.6 and 49.1 times respectively for the counter, 139 times for the RISC architecture (when comparing pre-synthesis and the ROM approach). The ROM approach improve the simulation speed by almost 37 percent. Moreover, the time lost in the whole object code production process, i.e. generation and compilation for the one hand, VHDL compilation, model generation and linking of processes on the other hand, points to another advantage of our generator.

4 Conclusions

We have presented a pre-synthesis method based on an initial control path specification which overcomes the main drawback of automatically generated models. This method, the use of C as target language and our "generation philosophy", i.e. never postpone to run-time what can be done at generation time, leads to very efficient simulators. The equations obtained using the regression analysis, and several comparative results have proved this fact. Equations and results from static analysis are used during generation to adapt the pre-synthesis algorithm. The internal library offers flexibility during design, while the generation of C tables "underlying memory like" provides an additional flexibility to our C-models as they may be parameterized. Furthermore, our generator could easily produce an EDIF output and opens an inexpensive way to develop hardware prototypes. Our generator could work with other C-models providing some commonly used techniques for mixing models (e.g., defining interfaces or common data types) are being applied.

References


---

### Table 1: Comparison of results for control paths of several processors

<table>
<thead>
<tr>
<th></th>
<th>direct</th>
<th>presyn.</th>
<th>SRR</th>
<th>execution time</th>
<th>direct</th>
<th>presyn.</th>
<th>AR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>size (bytes)</td>
<td></td>
<td></td>
<td></td>
<td>size (bytes)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8085</td>
<td>106</td>
<td>17.3</td>
<td>6.1</td>
<td>12.15</td>
<td>4.8</td>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>Riscp</td>
<td>28</td>
<td>3.6</td>
<td>7.7</td>
<td>2.9</td>
<td>0.7</td>
<td>4.1</td>
<td></td>
</tr>
<tr>
<td>Riscp4</td>
<td>37</td>
<td>7</td>
<td>5.3</td>
<td>12.0</td>
<td>2.9</td>
<td>4.1</td>
<td></td>
</tr>
<tr>
<td>Riscp8</td>
<td>41</td>
<td>10.4</td>
<td>3.9</td>
<td>23.4</td>
<td>5.2</td>
<td>4.5</td>
<td></td>
</tr>
<tr>
<td>Riscm</td>
<td>14.6</td>
<td>6.3</td>
<td>2.3</td>
<td>2.0</td>
<td>1.4</td>
<td>1.4</td>
<td></td>
</tr>
<tr>
<td>Toypp</td>
<td>12.9</td>
<td>5.1</td>
<td>2.5</td>
<td>2.8</td>
<td>1.25</td>
<td>2.2</td>
<td></td>
</tr>
<tr>
<td>Toymp</td>
<td>6.7</td>
<td>3.4</td>
<td>2</td>
<td>1.4</td>
<td>1.1</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>Trip</td>
<td>4.6</td>
<td>2.2</td>
<td>2.1</td>
<td>1.0</td>
<td>0.48</td>
<td>2.1</td>
<td></td>
</tr>
</tbody>
</table>

### Table 2: 16-bit up-counter simulation time

<table>
<thead>
<tr>
<th></th>
<th>functional description</th>
<th>gate level description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIDO simulator</td>
<td>VAX 8810</td>
<td>0s 82c</td>
</tr>
<tr>
<td>VHDL simulator</td>
<td>VAX 8810</td>
<td>40s 35c</td>
</tr>
</tbody>
</table>

### Table 3: RISC architecture simulation time

<table>
<thead>
<tr>
<th></th>
<th>LIDO generator</th>
<th>VHDL simulator</th>
</tr>
</thead>
<tbody>
<tr>
<td>technique used</td>
<td>algo. translation</td>
<td>behavioral</td>
</tr>
<tr>
<td>HDL model size</td>
<td>4K</td>
<td>12K</td>
</tr>
<tr>
<td>C-model size</td>
<td>27.5K</td>
<td>123K</td>
</tr>
<tr>
<td>object code production time</td>
<td>18.93s</td>
<td>43.56s</td>
</tr>
<tr>
<td>simulation time</td>
<td>18.93s</td>
<td>22mn 45s</td>
</tr>
</tbody>
</table>

SRR - Size Reduction Ratio, AR - Acceleration Ratio, m - indicates microprogrammed control path.