ATOMS - A Tool for Automatic Optimization of Gate Level VHDL Models for Simulation

Oliver Tschäche and Volkmar Sieh

Department of Computer Science III
University of Erlangen-Nürnberg
Martensstr.3
91058 Erlangen, Germany
Email: {ortschae,vrsieh}@immd3.informatik.uni-erlangen.de

KEYWORDS
VHDL, Optimization, Simulation

ABSTRACT

This paper proposes a new method to speedup simulation of VHDL models. Therefore, a tool named ATOMS (Automatic Optimization of VHDL Models for Simulation) reads in a VHDL source model and generates automatically a new optimized VHDL model which is simulated faster by the same simulator. ATOMS arises out of an approach to speedup simulation of a RISC computer system running its operating system. The CPU is modeled near gate level to calculate the rate of gate level faults covered by pin level faults. This is performed by software based fault injection experiments.

ATOMS achieves the speedup by reducing the number of processes and signals of the source model. The lower the abstraction level of the VHDL source model the higher is the speedup of the simulation. Though, optimizing gate level models promises the most efficient speedup. The average speedup of the simulation is measured to 5-8, the best measured speedup was 20.

INTRODUCTION

VHDL models of contemporary systems are as complex as the system itself. Synthesis of gate level models ends in a very complex network of logic gates, flipflops and latches. Simulating these gate level models requires a lot of CPU time. E.g. for a simulation interval of 1 ms the VHDL simulator needs 2.5 hours CPU time to simulate a VHDL model of a MVME188 32 bit RISC system (Motorola INC. 1992), although only the CPU, an M88000, is modeled near gate level and the periphery is modeled at an abstract behavioural level. The target machine simulating the MVME188 is a SPARCstation LX running the Modeltech VHDL-Simulator.

Efforts analyzing performance of simulation of VHDL models started with Hueber 1991. Levia O. 1991 and Bonomo et al. 1992 show, how to write VHDL models which can be simulated efficiently. This paper introduces ATOMS, a tool which optimizes VHDL models automatically.

The paper is divided into three further sections. The following section describes two tasks a VHDL simulator has to deal with and how the CPU time is split to these tasks. Based on this result optimization methods are presented.

The next section presents the implementation of ATOMS. The stages starting from reading in the VHDL source model and specifying the entity and architecture to optimize till the output of the automatically optimized model are explained.

Finally, the conclusion summarizes the most important results and the section future research explains what else can be done to speedup simulation.

OPTIMIZATION METHODS

CPU time used for simulation of VHDL models can be split into two parts. The time needed to

schedule processes and to switch between processes and the rest of CPU time which is used to execute the statements of the processes, doing the actual behaviour. Considering the VHDL model of an logic NAND gate with two inputs in figure 1, gate level models consist of many signals and processes and very few statements in the processes, one bold printed statement in figure 1.

```
ENTITY nand2 IS
  PORT(  i0,i1 : IN bit;
         o : OUT bit
  );
END nand2;

ARCHITECTURE behaviour OF nand2 IS
BEGIN
  PROCESS( i0,i1) BEGIN
    o <= i0 NAND i1;
  END PROCESS;
END behaviour;
```

Figure 1: VHDL model of a NAND gate

To calculate the CPU time used for execution of the statements, the VHDL model in figure 2 is simulated with *num*=1 and *num*=2 . The difference of the measured CPU times of the simulations is the time needed to process the inner for-loop (1000 additions). Running the Modeltech VHDL simulator on an SPARCstation LX, this time is 6.8s-3.9s=2.9s. The times are measured by using the UNIX command time which runs the simulator in batch mode.

```
ENTITY perf_calc IS
  GENERIC( num : integer );
END perf_calc;

ARCHITECTURE behaviour OF perf_calc IS
  SIGNAL o : integer;
BEGIN
  PROCESS
    VARIABLE c,d,e : integer;
  BEGIN
    FOR c IN 1 TO num LOOP
      e:=0;
      FOR d IN 1 TO 1000 LOOP
        e := e + d;
      END LOOP;
    END LOOP;
    o <= e;
    WAIT FOR 10 ns;
  END PROCESS;
END behaviour;
```

Figure 2: VHDL model to estimate time of statement execution

Starting the simulator and loading the VHDL model needs 0.2s. Thus, the CPU time spent for scheduling is 3.9s-2.9s-0.2s=0.8s. Assuming that a logic expression like AND, OR, XOR, and so on is calculated as fast as an addition by the simulating machine the CPU time for processing simple statements like that of the NAND2 example is about 2.9s/1000=2.9ms. Thus, the CPU time used to schedule processes amounts to less than 0.4% (2.9ms/0.8s) of the CPU time needed for simulation. Because of the higher number of processes and signals in a gate level model the CPU time used by the scheduler constitutes **more** than 99.6%. Because of this portion of CPU time used for scheduling reducing the number of processes and signals promises the best reduction of CPU time.

A very simple method to reduce the number of processes is to search for flipflops which are triggered by the same event. For example, a 32 bit register consists of 32 single bit flipflops and so 32 processes in a gate level model. The statements of these processes can be collected in one single process. For three flipflops collected in one process the CPU time is 86% of the original model.

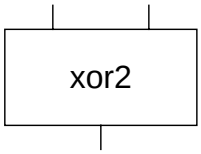
Another way to reduce the number of processes and signals of the VHDL source model is to search for connected combinational processes. A combinational process describes a combinational circuit like an AND gate, multiplexer or decoder. These components have no internal state unlike a flipflop or latch. Combinational processes can be collected in one single processes, if they are connected with signals and if they are feed-forward. To generate the correct logic function it is necessary to sort the statements of the combinational processes in the way the signals flow through them. This optimization method eliminates hazards and spikes, since the sorting of processes excludes delta cycles. An example for this method is shown in figure 3.

The speedup of this method is presented by an example of adders of different bit length. The adders are implemented as ripple carry adders built up from full adders which are modeled in one process. Thus, a 16 bit adder consists of 16 processes. Optimization of all adder models results in a model with one single process. For lengths of 1 to 32 bits (processes) the CPU time used for

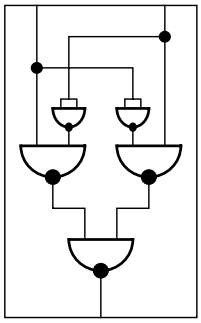
simulating the source and optimized VHDL models is measured. To this end, input vectors of the adders are stimulated with different wave tables changing their value in cycles of 100 ns:

- counter stimulus: the bits of the input vectors of the adders are concatenated to one vector. The number represented by this vector is incremented by 1 each cycle.
- random stimulus: the bits of the input vector are changed randomly each cycle.

```
ENTITY xor2 IS
  PORT(  i0,i1 : IN bit;
         o : OUT bit
  );
END xor2;
```




```
ARCHITECUTRE gate
       OF xor2 IS
  SIGNAL ii0,ii1 : bit;
  SIGNAL m1,m2 : bit;
BEGIN
  inv_i0 : nand2
    PORT MAP(i0,i0,o=>ii0);
  inv_i1 : nand2
    PORT MAP(i1,i1,o=>ii1);
  gen_m1 : nand2
    PORT MAP(i0,ii1,m1);
  gen_m2 : nand2
    PORT MAP(ii0,i1,m2);
  gen_result : nand2
    PORT MAP(m1,m2,o);
END gate;
```



```
ARCHITECTURE optimized OF xor2 IS
BEGIN
  PROCESS( i0,i1 )
      VARIABLE ii0,ii1,m1,m2 : bit;
  BEGIN
    ii0 := i0 NAND i0;
    ii1 := i1 NAND i1;
    m1 := i0 NAND ii1;
    m2 := ii0 NAND i1;
    o <= m1 NAND m2;
  END PROCESS;
END optimized
```

Figure 3: combinational optimization

The diagram in figure 4 shows the speedup for the different number of processes of the VHDL source model. For the counter stimulus the optimized model for less processes is simulated faster than the original. Then, starting at 7 processes it is simulated slower. The optimized model is slower because the stimulus activates less than two processes per cycle independent to the length of an adder. Thus, the statements of two processes are to be processed. The process of the optimized model processes the statements of all bits. Starting at seven processes it is faster to schedule the processes and process the statements of the source model than processing the statements for all bits in one process of the optimized model. If the stimuli activate a fraction of the processes of the source model like the random stimulus does the optimized model is simulated faster.

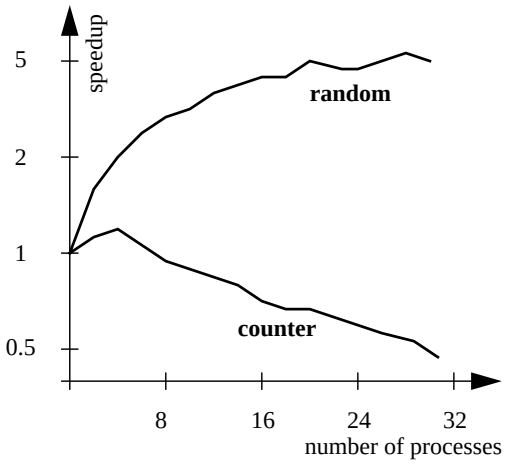


Figure 4: Speedup for ripple carry adders

Using the methods of combining flipflops and building combinational blocks concurrently VHDL models can be optimized to one single process if all triggered components like flipflops are sensitive to the same trigger (clock) signal. The combination of the two methods is not implemented yet.

IMPLEMENTATION

Contemporary models are so complex that they cannot be optimized manually because of the time needed to generate and to verify the optimized model. Therefore, we searched for a way to optimize VHDL models automatically. This is achieved by ATOMS.

ATOMS is developed in an UNIX environment. It is started with two parameters. The first parameter is the name of the entity and the second the

name of the appendant architecture which is to be optimize. In addition to the command line parameters ATOMS reads in a configuration file in which the user has to write the names of the VHDL source files. Then, ATOMS processes the VHDL source model in three stages:

1. Reading the VHDL source
2. Analysation of VHDL source
3. Opimization and output

The last stage puts out the optimized VHDL model, an architecture named 'optimized'. This architecture consists of signal declarations and the processes generated by the optimization stage. There are no component instantiations in the optimized architecture.

ATOMS supports different levels of VHDL subsets in different stages. In the stage reading VHDL source and transforming it into a syntax tree ATOMS supports the '93 VHDL standard. The next stages only support VHDL elements used by the VHDL model of the MVME188 system. The VHDL feature of resolved signals is not implemented yet.

The first stage of ATOMS processes the VHDL source in two moduls. The load module reads all files pointed to by the configuration file and converts them into a syntax tree. The load module consists of the lexical and grammatical description. To achieve best portability the GNU tools Flex and Bison are used to produce C modules of these descriptions which are derived from the ALLIANCE CAD toolset. The resolve module checks references to names introduced by USE-clauses and resolves them if they are valid.

The next stage of ATOMS analyses the semantics of the VHDL source. It is divided in two modules: the behavioural and structural analysis. Before the behavioural analysis all processes are explored for signals the process uses for input, output or sensitivity. Signals a process is sensitiv to are described in the process' sensitivity list. Input signals are the signals used in the process' statements to calculate new values at the right of variable/signal assign statements or forming conditions in IF- or CASE-statements. The output signals of a process are the signals referenced at the left of a signal assignment expression.

Behaviour analysis inspects all processes of the VHDL source in the syntax tree whether they serve conditions for a combinatorical process or not. These are:

- All input signals must be sensitivity signals and must be on the sensitivity list of a process. Wait statements are not allowed.
- To a value variables must be assigned before they are used in expressions.
- IF and CASE statements must have a default alternative. An IF statement must have an ELSE alternative and a CASE statement must have an WHEN OTHERS alternative.
- All alternatives of an IF or CASE statement must assign values to exactly the same variables and signals.

If a process meets all of these conditions it is combinatorical. If only one of these conditions do not meet the process is assumed to be non combinatorical.

Structural analysis resolves component instantiations and generates a planar, non hierarchical view of the VHDL source model. Starting with the architecture which should be optimized structural analysis searches in the architecture body for processes and component instantiation statements. If a component instantiation statement is found the same procedure searches recursively in the architecture body of that component for further processes and component instantiation statements. At the end of structural analysis for each process a list with the signals connected to this process exists and for each signal a list of connected processes exists.

The last stage of ATOMS does the optimization and the output of the optimized VHDL model. One optimization module searches for non combinatorical processes which are sensitive to one single signal. Processes sensitive to the same signal are combined to one process sensitive to this signal and the processes optimized in this way are removed from the list of processes.

A further module of the optimizing stage searches for blocks of connected combinational processes to build a single optimized process for each block. To achieve this, the signals of the first combinational process in the list of processes are searched for other connected combinational processes. Then, the signals of this process are examined to which other combinational processes they are connected. Interrogation of all signals of all interconnected processes results in a combinational block. Starting from the input signals of this block it is searched for processes having only input signals which are input signals of the block. The output signals of these processes are added to the list of input signals and the search for processes depending only on these signals starts again. The sequence in which the processes are found is the same as the sequence in which the statements of the processes must be copied to the optimized process. Finally all processes used to build the new block are removed from the list of processes.

The processes left in the list cannot be optimized by the current implementation of ATOMS. To complete the optimized model they are copied unchanged.

In the current implementation the optimization stage of ATOMS does not handle resolved signals. Thus, only parts of MVME188 model can be optimized by ATOMS.

CONCLUSION

ATOMS is a tool which speeds up simulation of VHDL models of low abstraction level. For gate level models the average speedup is about 5-8.

The speedup depends on the stimuli. Thus, it is possible that the optimized model is simulated slower for very special stimuli.

ATOMS supports only a subset of VHDL '93. The most important restriction is that ATOMS cannot handle resolved signals.

FUTURE RESEARCH

ATOMS optimizes VHDL models so that the time spent for scheduling and switching processes is decreased and the time spent to process the statements is increased. No efforts are made to decrease the time spent to process the statements of a process. This could be a potential place to speedup the simulation of the optimized model (Aho et al. 1996).

Since ATOMS rearranges the VHDL source model additional information should be generated which makes it possible to map the states of the signals from the source model to the optimized model and vice versa.

REFERENCES

Aho, A. V.; R. Sethi and J. D. Ullman 1996. *Compilers.* Addison-Wesley, ISBN 0-201-10088-6.

Balboni, A.; P. Cavalloro; M. Mastretti; A. Bonomo; E. Paschetta; G. Buonanno; and D. Sciuto. "A Set of Tools for VHDL-Code Quality Evaluation", Proceedings of the 1994 VHDL-FORUM for CAD in EUROPE.

Bonomo A.; P. Garino; G. Ghigo; A. Balboni; and M. Mastretti. 1992."VHDL optimization techniques for coding and simulation", Technical Report, CSELT Torino.

Hueber M. 1991. "VHDL experiments on Performance", Proceedings of the 1991 EURO-VHDL Conference.

Jenn, E.; J. Arlat; M. Rimén; J. Ohlsson; and J. Karisson. 1994. "Fault Injection into VHDL Models: The MEFISTO Tool." In digest of papers of the 24. International Symposium on Fault-Tolerant Computing (Pasadena, California, June 27-30, 1994), IEEE Computer Society, 66-75

Levia, O. 1991. "Writing High Performance VHDL Models", Proceedings of the 1991 EURO-VHDL Conference.

Motorola INC. 1992. *MVME188A VMEmodule RISC Microcomputer User's Manual*.

Sieh V.; O. Tschäche; and F. Balbach. 1996. "VHDL based Fault Injection with VERIFY", internal report University of Erlangen-Nürnberg, IMMD III

Tschäche O. 1996. "Automatische Optimierung von VHDL-Modellen" (German), Diplomarbeit am Lehrstuhl IMMD III der Friedrich-Alexander Universität Erlangen-Nürnberg, postscript version available from http://faui30t.informatik.uni-erlangen. de:1200/Staff/ortschae/papers/dipl.ps