# Designing a Reactive Processor with Esterel v7

## A Case Study

Malte Tiedje  Claus Traulsen

*Dept. of Computer Science*
*Christian-Albrechts-Universität zu Kiel*
*Olshausenstr. 40, D-24098 Kiel, Germany*
{*mti,ctr*}*@informatik.uni-kiel.de*

## Introduction

Esterel [3] is a synchronous language, developed to model reactive systems with hard real-time constraints. It offers, beside a precise synchronous semantics, constructs to express various forms of preemption as well as concurrency. Esterel programs can be either compiled to VHDL, and then synthesized to hardware, or to C, in order to be executed on a COTS processor. They can also be executed on a *reactive processor*, like the *Kiel Esterel Processor* (KEP) [2], which directly supports concurrency and preemption. The KEP was originally designed directly in VHDL. This implementation turned out to be very efficient. For comparison, we decided to reimplement the KEP, using Esterel as the description language. We expected the following advantages from this implementation:

- The implementation in Esterel gives a formal reference for the behavior of the KEP.
- Via the C code generation, we also get a software simulation of the KEP, which can be used for testing the implementation and in classes on reactive processing.
- Having a non-trivial project to evaluate the efficacy and efficiency of designing hardware with the current Esterel v7, as well as the usability of Esterel-Studio [1]. In particular, we wanted to test whether efficient hardware could be developed in Esterel, without prior knowledge in hardware design.

## Developing with Esterel v7 and Esterel-Studio

We used the software generation from Esterel for regressions tests. While the generated code is assured to behave exactly like the generated hardware, the software generation is much faster. The software can be extended to log its input in a simple trace format. When an error occurs, these trace file can be executed inside the Esterel-Studio simulator, to see the internal state of the model when the error occurs.

Esterel-Studio allows to formally verify properties of the program like the equivalence of different modules and assertions, which can be either written manually, or generated automatically, *e. g.*, to test for overflows. While a global proof of all assertions is not possible for medium size projects, the equivalence test turned out to be useful. A simple and probably inefficient description of a module can be written and tested. Thereafter, an optimized version of the same module is written, and its equivalence to the original implementation can be formally proven.

Esterel-Studio also allows early performance evaluation, by giving a rough approximation of the used registers without actually synthesizing the hardware. While these values can differ from the actual register usage, they indicate whether the implementation scales.

Current compilers reject all programs with cyclic signal dependencies. Unfortunately, it is not always clear what the compiler considers as cyclic. Esterel-Studio tries to mark the cycles in the program code, but actually finding and solving a cycle is by no means trivial.

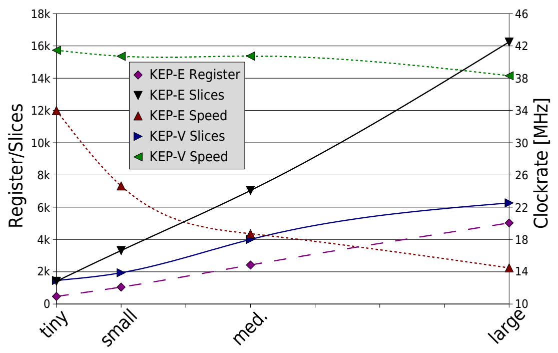

Fig. 1: Resource usage of the Esterel (KEP-E) and VHDL (KEP-V) implementation

The generated hardware is less efficient than the original implementation of the KEP, as can be seen in Fig. 1. Note, however, that the Esterel implementation is a prototype with less optimizations. While the register usage computed by Esterel-Studio scales well, the synthesized hardware is inferior to the VHDL implementation. Subtle changes of the Esterel description can have a significant influence on the efficiency of the generated hardware. Unfortunately, the most efficient implementation is often the least readable. In many cases, the compiler should be able to perform the optimizations internally, and not leave this burden to the developer. So far the compiler optimizations are performed on a circuit representation of the program, therefore, the optimization does not scale for larger projects. This can be avoided, however, by using modular compilation, where the optimizations are performed on each module separately. Still, an earlier optimization directly on the Esterel level might be better.

Interfacing existing cores with generated hardware can be problematic. Care has to be taken, that the inputs are stable at the start of an Esterel tick when they are sampled [4]. These problems can be avoided by using request-acknowledgment mechanisms.

## Lessons learned

With Esterel v7, hardware can be described easily and rapidly. While in our experiment the generated hardware is less efficient than the manually written VHDL code, its performance is acceptable for our purposes. We expect that the performance could be increased significantly by further tuning the code, or—preferably—by making the synthesis tool smarter about high-level optimizations.

The possibility to formally verify the equivalence of different modules, and in particular the generation of hard- and software from the same Esterel description proved to be very useful in practice, since it allowed fast functional testing, without a complete hardware synthesis.

Currently, we are developing a more dataflow oriented reactive processor. This implementation makes use of additional design features of Esterel v7, like multi-clocks.

## References

[1] Esterel Technologies, *Esterel studio* (2008), http://www.esterel-eda.com.

[2] Li, X., M. Boldt and R. von Hanxleden, *Mapping Esterel onto a multi-threaded embedded processor*, in: *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'06)*, San Jose, CA, 2006.

[3] Potop-Butucaru, D., S. A. Edwards and G. Berry, "Compiling Esterel," Springer, 2007.

[4] Tiedje, M., "Beschreibung des Kiel Esterel Prozessors in Esterel," Diploma thesis, Christian-Albrechts-Universität zu Kiel, Department of Computer Science (2008), http://rtsys.informatik.uni-kiel.de/~biblio/downloads/theses/mti-dt.pdf.