Brief description of my work


During my Ph.D. I defined and implemented a new method for the translation of Esterel programs into sequential (C) code. The imperative semantics of Esterel suggests here the use of classical compilation and scheduling techniques. However, the constructive semantics of the language is not only based on classical control-flow. On-the-fly fact propagation mechanisms use the signal statuses that are already known to simplify the not-yet-executed code before execution can continue. Simulating this propagation at runtime can be very expensive. The propagation also induces a very complex causality that limits the scope and efficiency of the analysis and optimization algorithms.

I started by defining the micro-step operational semantics of Esterel (a conservative, deterministic extension of existing macro-step semantics). The new semantics is better adapted to sequential code generation. It also handles, for the first time in a native setting, the data-handling primitives of Esterel. A careful analysis of the semantic rules and of the existing compilers helped me develop a new compilation technique, based on a new intermediate representation (called GRC) for Esterel programs. A GRC specification is composed of a concurrent control-flow graph and a hierarchical state representation derived from the structure of the initial Esterel program. The GRC code has formal semantics, naturally derived from the constructive semantics of the digital circuits. Moreover, the Esterel-to-GRC and Esterel-to-circuits translation schemes are closely related, both semantically and syntactically, allowing us to prove a topological relation that is very important in practice (cyclic circuit iff cyclic GRC flowgraph). Thus, GRC is a model allowing the representation of all Esterel programs which also supports new, semantically correct optimization (control flow simplifications) and code generation (optimal state encoding, scheduling) techniques based on static analysis.

To evaluate the efficiency of the approach I wrote the grc2c optimizing Esterel compiler. The compiler fully handles the class of so-called acyclic Esterel programs, generating very efficient code (the fastest on most available examples, when compared with the output of existing compilers). A variant of the GRC format is currently used in the Esterel compiler system developed at Columbia University by the group of Stephen Edwards. It is also used at INRIA Sophia Antipolis as input for automatic code distribution techniques based on the SynDex tool.

The GRC code can also be considered as an intermediate step in the translation of Esterel into digital circuits, a step where basic Esterel structures (state, tests, and synchronizations) have not yet been encoded using logic gates. Then, the GRC-level optimizations result in efficient, low-cost optimizations of the circuit netlist, as shown by the good practical results of the prototype circuit optimizer scsimplify that I wrote. Simplifications are most obvious on the state encoding, where we are able to simultaneously reduce the number of latches, the circuit area, and its speed by using low-cost algorithms based on static analysis.

The grc2c and scsimplify prototype tools (almost 1Mo of C/C++/Lex/Yacc source code) have been transferred to the company Esterel Technologies, and our optimization techniques have been adapted for use in the EsterelStudio development environment.

These results have been presented at the Memocode2003 conference [7] and in a refereed book chapter [2]. The circuit optimization results have been presented in a technical report [10]. I am currently writing with Stephen Edwards and Gérard Berry a book on the Esterel language [4] (publishing contract signed, manuscript due in september 2005).

2. Post-doc:

Efficient GALS implementation of synchronous specifications

I am now a post-doc at INRIA, Rennes, France, financed by the European projects ARTIST (« Embedded Real-Time Systems ») et COLUMBUS (« Embedded Software Design for Mission-Critical Systems »). My main research direction is here the synthesis of GALS implementations from synchronous specifications. On this subject I mainly work with A. Benveniste and B. Caillaud. New collaborations have been recently started with the group of professor Yakovlev (Newcastle University), and with the INRIA projects ESPRESSO and AOSTE. I also continued the collaborations with R. de Simone (INRIA) and S. Edwards (Columbia University), focused on the compilation of Esterel.
2.1. Description of the problem

If the synchronous approach facilitates the specification and the verification of a large class of reactive systems, the problem of deriving correct physical implementations does remain. In particular, difficulties arise when the target implementation architecture has a distributed nature that does not match the synchronous assumption because of large variance in computation and communication speeds and because of the difficulty of maintaining a global notion of time. This is increasingly the case in complex microprocessors and Systems-on-a-Chip (SoC), and for many important classes of embedded applications in avionics, industrial plants, and the automotive industry.

For instance, many industrial embedded applications consist of multiple processing elements, operating at different rates, distributed over an extended area, and connected via communication buses (e.g. CAN for automotive applications, ARINC for avionics, Ethernet for industrial automation). To use a synchronous approach in the development of such applications, one solution is to replace the asynchronous buses with communication infrastructures (like the Timed-Triggered Architectures of H. Kopetz) that comply with a notion of global synchronization. However, such a fully synchronous implementation must be conservative, forcing the global clock to run as slow as the slowest computation/communication process. The overhead implied by the global synchronization is often large enough to convince designers to use the asynchronous buses mentioned above.

Gathering advantages of both the synchronous and the asynchronous approaches, the Globally Asynchronous Locally Synchronous (GALS) architectures are emerging as the architecture of choice for implementing complex specifications in both hardware and software. In a GALS system, locally-clocked synchronous components are connected through asynchronous communication lines. Thus, unlike for a purely asynchronous design, the existing synchronous tools can be used for most of the development process, while the implementation can exploit the more efficient/unconstrained/required asynchronous communication schemes.

The problem we considered is that of synthesizing GALS implementations from modular synchronous specifications. Each component of the synchronous specification must be fitted with a wrapper (a real-time executive) that reads the asynchronous inputs and schedules them into synchronous input events before giving them to the component and triggering reactions (clock ticks) of the component. As the synchronous paradigm is often used in the development of safety-critical systems, input reading and the system itself must be deterministic, or at least predictable. It is therefore essential to consider classes of synchronous specifications that facilitate the development of efficient wrappers which make input reading deterministic while not restricting the behavior of the system.

The difficulty is to synthesize wrappers that preserve the functionality of the specification, its correctness, and which do not add redundant signalling (otherwise we fall onto inefficient, globally-synchronized solutions). Previous work in this direction include that of Benveniste et al. (endo/isochronous systems) and, in a circuit-oriented framework, that of Carloni and Sangiovanni-Vincentelli (latency-insensitive systems).

Like in previous approaches, the correctness criterion we considered is the preservation of semantics by the GALS implementation. Formally, the GALS implementation preserves the semantics of the synchronous specification if its set of possible executions (asynchronous traces) coincides with the set of asynchronous observations of executions of the synchronous specification. This criterion ensures that (1) the GALS implementation is safe (because all its executions are covered by the verification of the specification) and (2) the functionality of the specification is preserved.

The problem we solved is that of characterizing large classes of synchronous components for which small, simple wrappers (e.g. wrappers that trigger a transition as soon as enough input is available) produce deterministic, efficient, and semantics-preserving GALS implementations. These classes of systems can then be considered as the implementation space, and the remaining problem is that of making given synchronous systems belong to these classes (by adding supplementary signalling).

2.2. Contribution

Our first contribution has been the definition of the notions of weak endochrony and weak isochrony, which characterize a large class of synchronous systems for which small, simple wrappers produce deterministic, semantics-preserving GALS implementations. Weak endochrony is a delay insensitivity property ensuring that the order in which asynchronous inputs are read by a synchronous component does not influence the outcome of its computation (so that very simple wrappers produce deterministic
implementations). Weak isochrony is a compatibility property insuring that a GALS system formed of weakly endochronous components is semantics-preserving. The two properties are decidable and it is easy to transform a general synchronous component into a weakly endochronous one by adding new signals (the problem is how to do it efficiently). Weak endochrony generalizes the notions of endochrony and latency-insensitivity by allowing computations of a synchronous component to run concurrently. Thus, it potentially supports GALS wrappers and communication protocols that are less constraining and less expensive in terms of communication than existing ones. From a theoretical point of view, weak endochrony is a non-trivial extension of the classical (Mazurkiewicz) trace theory to a synchronous setting. To our knowledge, it is also the first attempt to integrating concurrency in a synchronous model.

Weak endochrony and weak isochrony are formulated in a classical synchronous model – the Mealy machines – which is macro-step and non-causal. However, an efficient solution to our implementation problem must take into account essential aspects of any implementation, such as the causality and the communication through read/write primitives. Our second contribution has been the definition of a model for the representation of asynchronous implementations of synchronous specifications. The model covers classical implementations, where global synchronization is preserved by means of signalling, and GALS implementations, where the global clock is removed. Our model offers a unified framework for reasoning about two essential correctness properties of an implementation: the preservation of semantics and the absence of deadlocks due to inter-component communication. Our model offers a good abstraction level for reasoning about desynchronization, by revealing the intricate relations between (1) causal dependencies, concurrency and conflicts in the micro-step semantics of a synchronous specification and (2) the correctness (semantics preservation) of its GALS implementation. In particular, the model allows us to prove that the absence of deadlocks in a micro-step synchronous specification whose components are weakly endochronous implies the correction of the GALS implementation. As the correctness of the synchronous specification implies the absence of deadlocks, checking the correctness of the GALS implementation is checking the compositional weak endochrony property (which can be done efficiently).


2.3. Work in progress

With our new model, we are exploring the development of GALS circuits made of synchronous IPs (collaboration with the group of professor Yakovlev, of Newcastle University). Our work aims at using asynchronous logic wrappers to encapsulate the components of a modular synchronous circuit into delay insensitive components. Our model seems well-suited to analyze such designs, involving both synchronous and asynchronous parts.

We are also considering the application of our results to the efficient GALS implementation of high-level synchronous specifications (in collaboration with the ESPRESSO and AOSTE projects of INRIA). The goal is here to adapt our results to specification languages like Signal, Esterel, and Lustre, and to develop symbolic analysis and synthesis techniques for the transformation of general high-level specifications into weakly endochronous ones.

A third research direction concerns the (still not sufficiently clear) relations between classical, macro-step synchronous models and the more operational, micro-step models that cover the implementations of various synchronous languages and formalisms, especially when desynchronization is involved.

3. Visits

- September 2004 – visit at Newcastle University, UK. Invited by prof. Alex Yakovlev for work on the GALS implemetation of synchronous specifications.
- Visits et presentations in the projects TICK (AOSTE) and S4 of INRIA, France, in the SaxoRT group of CNET France Télécom Grenoble, France, and in the Vérimag laboratory in Grenoble.
- From 1999 on, I participated to the annual « Synchron » Workshops on synchronous languages.
- I participated to the 68NQRT seminar of IRISA, Rennes, France.