18

Control optimization and hardware translation of Esterel programs

François-Xavier Fornari
INRIA
INRIA, BP 93, 06902 Sophia-Antipolis, FRANCE. Telephone: (33) 93 95 74 93, Fax: (33) 93 95 74 88. e-mail: fornari@sophia.inria.fr

Abstract

This paper describes an original application of hardware optimization techniques for ESTEREL programs. These techniques are tuned to the target technology which may be software or hardware. The control portion of an ESTEREL program consists of a set of sorted boolean equations and a set of boolean memory cells. It computes the next state of the program and drives data handling actions. The presented techniques heavily reduce the size of the control while preserving the sequential constraints between the actions. For hardware implementations, we describe a translation into HARDWAREC. This final synthesis step is done using the Stanford OLYMPUS synthesis system.

Keywords

Synchronous languages, ESTEREL, digital circuits, sequential optimizations, high level synthesis.

1 INTRODUCTION

Designing and implementing controllers is known to be a hard task especially when concurrent components are used. One way to reduce the development time, to increase the reliability, and to be technology independent is to use a high-level programming language. In this paper we present work on the translation of **full** ESTEREL (Berry and Gonthier 1992) language into digital circuits, extending previous work by Berry implementing the PURE ESTEREL synchronization subset of ESTEREL (Berry 1992). The ESTEREL language was designed for modular description of control. It fully combines sequencing, parallelism, instantaneous broadcast, exceptions and watchdogs. It has a well-defined mathematical semantics that relies on the **perfect synchrony hypothesis**, which states that the response of a program to a stimulus from its environment is instantaneous. This model is known in the hardware domain as the zero-delay model.

ESTEREL was primarily designed for control intensive software applications. Several tools have been built for the ESTEREL language: a compiler that produces software implementations of encoded finite state machines, graphical simulators, a symbolic debugger, and automata verification tools. Since programs can be compiled into FSMs, the existing FSM techniques for sequential hardware synthesis can be used. Unfortunately, optimal state encoding is hard to achieve and current techniques do not scale with the size of programs.

G. Saucier et al. (eds.), *Logic and Architecture Synthesis*

```
loop
  [
    await A
  ||
    await B
  ];
  emit O
each R
```

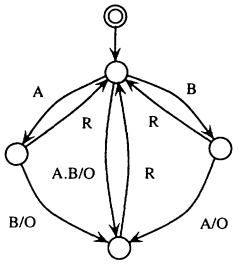

Figure 1 An ESTEREL program and its FSM representation

In (Berry 1992) Berry proposed a structural hardware translation that follows the structure of the behavioral description and translates the synchronization kernel of the program into a sequential circuit. The state encoding is done implicitly. This work was limited to the PURE ESTEREL language, which is the synchronization kernel of ESTEREL where data handling is forbidden. Pure programs are translated into hardware and the circuits obtained are heavily optimized (Touati and Berry 1993).

The problem we solve here is the translation of the **full** language into hardware including data handling. We rely on the existing ESTEREL v4 compiler that transforms an ESTEREL program into a control part and a separate set of data handling actions. The control part is expressed as a set of sorted boolean equations that drive the actions. Optimizing the control becomes not trivial since one must respect sequentiality constraints between actions. The optimization step is done with the SIS program (Sentovich *et al.*) and hierarchical optimization is available. Once the control is optimized the final translation to hardware is done using the Stanford OLYMPUS (De Micheli *et al.*) synthesis system to recombine the control and the data path.

Section 2 presents the language and gives a flavor of the structural translation to hardware. Section 3 presents the control optimizations: control part extraction, specific optimizations and control-data recombination. The final hardware translation is presented in Section 4. Section 5 summarizes and offers directions for future development.

2 THE ESTEREL LANGUAGE

2.1 The language

ESTEREL has three main features: it assumes a perfect synchrony hypothesis, it permits communications through instantaneous broadcast signals, and it has a deterministic parallelism. Computations are triggered by input events. Sequencing, concurrency, looping and presence of signals perform instantaneous control transmission. Instantaneous preemption are done by `watching` and `trap` statements. Non instantaneous statements are various forms of explicit delay. The language and its semantics are presented in (Berry and Gonthier 1992).

Figure 1 shows a program that waits for the signals `A` and `B` in parallel. The program emits signal `O` once both signals are received and stops. This behavior is restarted at each occurrence of signal `R`.

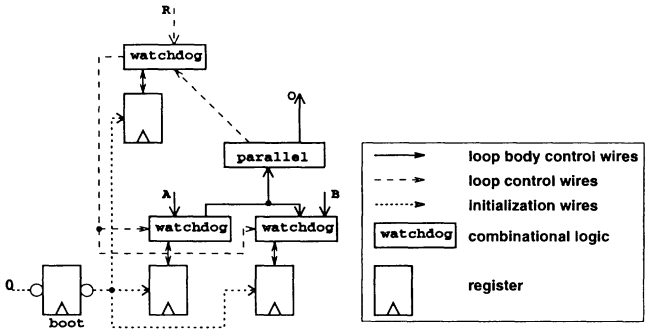

Figure 2 Structural translation of the `loop ... each` program

2.2 The structural translation

The state of an ESTEREL program is fully determined by its **distributed program counter**, i.e the points where the program is waiting for an input event. The structural translation proposed by Berry implements each of these halt points by a register. For instance, the program given in Figure 1 contains three halt points. Figure 2 shows its structural translation in a simplified way. A register is set to "1" when the corresponding halt point is **active**. The `watchdog` boxes distribute control according to the presence of the associated signal and the associated register value. The difficulty is to generate the correct combinational logic that implements instantaneous constructs (Berry 1991). Notice also a fourth register, named `boot`, that is used for the initialization step. Its initial value is "1", and then "0" thereafter.

An ESTEREL program is compiled into an internal description composed of two parts. The first part describes the control of the program as a set of sorted boolean equations and registers. It is generated using the structural translation. The second part is the data path and describes actions such as tests of signals, calls to procedures and test evaluations. Given the current inputs, the control computes the next state and enables the appropriate data computations.

3 CONTROL OPTIMIZATION

Since the control is produced by direct structural translation, it has poor speed performance and is usually too large. But it may be easily reduced and improved. Control optimization is done in three steps. The first step is the separation of the control part and the data manipulation part. During this operation, dependencies between actions must be found and recorded. The second step is the optimization process done with the SIS program. The last step is the merging of the optimized control part with the data manipulation part.

3.1 Extraction step

The aim of the extraction step is to translate the control part into the input format of a sequential logic optimizer. The control part is already separated from the data manipulation part in our

internal format (Section 2.2). As we said, it is expressed as a set of sorted boolean equations and registers, and drives the data manipulation part through input and output signals. Inputs correspond to the inputs of the program and outputs enable the execution of actions. Test instructions generate input/output pairs of signals. The control description is behavioral: equations are evaluated following the sorted order. In particular, sequentiality constraints between actions are implicitly coded by the equation order.

To optimize the control, we consider it as a sequential circuit. The main problem is that we do not want to include the sequentiality constraints into the description of this sequential circuit. There are two reasons. First, it is not possible to express directly these constraints using the optimizer format. Second, we could code them as micro-steps, but it would add more logic and registers and it would change the behavior of our control. Therefore, we keep track of the sequentiality constraints and we record them. This done by analysis of the control path and data dependencies. The control part is then translated into the SIS input format. It is described as a single global circuit, or as a hierarchy of sub-circuits. This decomposition will be detailed later.

Another circuit may be possibly produced at this stage. It is a combinational circuit that describes the execution environment of the program. One can specify explicit relations between inputs of an ESTEREL program in the source code. There relations give signals that are mutually exclusive, and signals that imply other signals. The combinational circuit represents these relations and is used by the optimizer to define sequential don't care conditions.

3.2 Optimization step

The ESTEREL compiler generates circuits with some particularities:

- redundant registers,
- interdependent registers,
- equivalent nets.

Two nets are said equivalent if they express the same function with respect to reachable states and external don't care conditions. The following example illustrates the generation of redundant registers:

```
do
    every LowerLeft do emit StopwatchStartStop end every
||
    every Upperleft do emit StopwatchLap end every
watching LowerRight % exit stopwatch mode
```

This example is extracted from the wristwatch example (Berry 1991). The loop instruction `every S do` *stat* `end every` terminates and starts *stat* afresh whenever `S` occurs. The structural translation generates one register per `every` instruction. The two registers produced are always equal since the two instructions are always active together. Thus, they are redundant and one can suppress one of them. Furthermore, there is a third register which is the boot register. The example actually has only two states: the inactive state when the program is not yet started and one active state. The optimized register can be derived from the boot register by a simple combinational logic circuit, so finally only one register is needed.

Table 1 Hardware optimization results

Circuit	Standard			Optimized			num. of
	lit(fac)	regs	levels	lit(fac)	regs	levels	states
abc	227	13	9	157	6	3	9
arbiter12	959	50	52	224	12	3	14
controller	578	67	10	51	34	2	372
lights	153	8	7	86	8	2	9
game	184	7	17	185	5	2	7
tcint	1032	82	62	491	62	3	287
vme_control	116	14	5	26	7	1	44
watch	775	35	25	312	10	3	42
Gain	2.71	1.97	7.13				

It is not necessary to rewrite programs to generate less registers because the optimizations used automatically remove extra registers. They are based on the algorithms described in (Touati and Berry 1993). These algorithms exploit the state reachability of a circuit and are able to drastically reduce the number of registers while maintaining proper behavior.

Table 1 shows the results obtained with our standard benchmarks. *lit(fac)* is the number of literals in the factored form. *Gain* is the geometric mean of *Standard* over *optimized*. These examples show that our optimizations are well-suited to ESTEREL circuits, as shown by the *Gain* row.

These circuits have been improved using SIS. Practically, specialized scripts that group SIS commands have been developed to do specific optimizations, such as redundancy removal or depth reduction. These scripts have been then combined to form a general script that gives those results.

It should be noted that the control of an ESTEREL program is technology-independent at this level of abstraction. Therefore, we have developed other scripts to optimize the control for software implementation. The control equations are directly written in C logical expressions. The goal is to reduce the size of the object code and to improve the execution speed. This is done by simplifying the logic and factorizing the expressions as much as possible. Our experiments have shown that optimized code is 40% smaller and 50% faster than unoptimized code.

The decomposition into sub-circuits (Section 3.1) allows the optimization of larger circuits using **hierarchical optimization** (Fornari 1995). The decomposition is done during the extraction step. It exploits the module call tree of an ESTEREL program. A sub-circuit contains the part of the global control corresponding to a module call. The decomposition is used to do specific optimizations on particular sub-circuits. It may also help for circuits that are too large for optimizations based on state reachability. Strong optimizations are done for the leaves of the sub-circuit hierarchy, while lighter optimizations are applied to the whole hierarchy.

3.3 Insertion step

The insertion step recombines the optimized control and the data manipulation. It is the opposite problem of the extraction step. Here we start from a structural description and we end with a behavioral one. This conversion is done by a topological sorting. The difficulty comes from the fact that the optimizer has no knowledge about the sequentiality constraints. We have shown that the new structure of the circuit after optimization is always compatible with the sequentiality constraints (Fornari 1995). Thus, topological sorting is always possible. The final conclusion is that we can always optimize ESTEREL program control using the extraction-insertion technique.

4 HARDWARE GENERATION

ESTEREL programs are of two kinds: programs that only use the synchronization kernel (also called PURE ESTEREL) and programs that also use the data flow instructions. PURE ESTEREL programs can be directly translated into gates. This is possible because the control is the circuit, since it only uses pure signals and do not manipulate data. Full ESTEREL programs require the implementation of the implicit sequentiality between actions.

4.1 Pure Esterel Case

As PURE ESTEREL programs are directly translated into gates, optimizations are performed on the final circuits. We avoid the extraction-insertion steps, but we still use the same scripts developed for control optimization in case of hardware implementation. Experimental results (Touati and Berry) show that the quality of the circuits is close to hand-crafted circuits.

PURE ESTEREL allows the use of static counters to count the occurrences of a signal or the number of iterations in a bounded loop. Programs with such counters are also translated into circuits. The final circuit is then composed of the control part and several fast counters. Finally, all circuits generated from PURE ESTEREL programs execute an ESTEREL transition within one clock cycle.

4.2 Full Esterel Case

We consider our description of a circuit as a behavioral description. So, we need a synthesis system that it is able to handle this level of abstraction. For our experiments we used the OLYMPUS synthesis system (De Micheli *et al.*) developed by De Micheli, Ku, Mailhot and Truong at Stanford University. It is a set of tools for ASIC synthesis from the high-level hardware description language HARDWAREC. We use the fact that the language allows the user to express several forms of parallelism and that the system generates an optimal control.

Thus, we translate an ESTEREL circuit into a HARDWAREC program. The user must provide the HARDWAREC implementation of the functions and procedures that were declared external in the ESTEREL program. The circuit is then synthesized in a technology-independent fashion: operations on data take an unknown number of clock cycles to execute. To ensure the correspondence between an ESTEREL transition and an execution of the circuit, we define a synchronization signal to start a transition. The circuit waits until this signal is set to "1" before it starts proceeding any other input event. The synchronization signal need not to be maintained during the computation. A special output signal synthesized by the OLYMPUS system defines the

end of the transition. Using these techniques, we have implemented a hardware version of the Reflex Game (Boussinot 1991) for a Xilinx XC3090 circuit. The design was down-loaded onto the PeRLe-1 board (Bertin *et al.* 1989).

PURE ESTEREL programs may be translated into circuits in the same fashion. There is no need for synchronization signals in this case. We still guarantee that an ESTEREL transition is performed within one clock cycle.

5 CONCLUSION

ESTEREL programs can be translated into hardware, using structural translation following by specific optimizations. These optimization techniques have been successfully extended to software compilation of ESTEREL programs. The ability to produce either software or hardware allows verification of a design before it is fabricated. One can use the full ESTEREL environment to do simulation, analysis and optimizations.

We now produce output in BLIF (Berkeley Logical Interchange Format), Lustre (Halbwachs *et al.* 1991) and VHDL formats. We are currently working on the improvement of the optimizations in terms of efficiency and speed.

REFERENCES

Berry, G. (1991) Programming a digital watch in Esterel v3_2. Rapport de recherche 08/91, Centre de Mathématiques Appliquées, Ecole des Mines de Paris, Sophia-Antipolis.

Berry, G. (1992) A hardware implementation of pure Esterel. *Sadhana, Academy Proceedings in Engineering Sciences, Indian Academy of Sciences*, 17(1):95–130. Rapport Centre de Mathématiques Appliquées de l'Ecole des Mines de Paris, numéro 06/91.

Berry, G. and Gonthier, G. (1992) The Esterel synchronous programming language: Design, semantics, implementation. *Science Of Computer Programming*, 19(2):87–152.

Bertin, P. Roncin, D. and Vuillemin J. 1989 Introduction to programmable active Memories. In E. Swartzlander Jr, J. McCanny, J. McWhirter, editor, *Systolic Array Processors*, 300–9. Prenctice Hall.

Boussinot (1991) Programming a reflex game in Esterel v3_2. Rapport de recherche 07/91, Centre de Mathématiques Appliquées, Ecole des Mines de Paris, Sophia-Antipolis.

Fornari, F-X. (1995) *Optimisation du contrôle et implantation en circuits de programmes* ESTEREL. Thèse d'informatique, Ecole des Mines de Paris.

Halbwachs, N., Caspi, P. and Pilaud, D. (1991) The synchronous dataflow program ming language lustre. *Another Look at Real Time Programming, Proceedings of the IEEE, Special Issue*.

De Micheli, Ku, Mailhot, and Truong The Olympus Synthesis System for Digital Design.

Sentovich, E. M. *et al.* (1992) SIS: a system for sequential circuit synthesis. memorandum UCB/ERL M92/41, University of California, Berkeley.

Touati, H. and Berry, G. (1993) Optimized controller synthesis using esterel. In *Proc. International Workshop on Logic Synthesis IWLS'93*, Lake Tahoe.