

A Case Study in Computer-Aided Codesign of Embedded Controllers

L. Lavagno	M. Chiodo, P. Giusto	H. Hsieh, S. Yee	K. Suzuki
Politecnico di Torino	A. Jurecska	A. Sangiovanni-Vincentelli	Hitachi
Torino, Italy	Magneti Marelli	University of California	Tokyo, Japan
	Italy	Berkeley, CA	

Extended abstract

Abstract

With our codesign system, POLIS, we have specified and implemented a real-life design: a shock absorber controller. Through this experiment, we have shown the possibility of using such a system to design complex applications and to speed up the design cycle dramatically. All aspects of the design process are closely scrutinized including high level language translation and automatic hardware and software synthesis. We analyze different software implementation styles and draw some conclusions about our design process.

1 Introduction

The *POLIS* codesign environment is being jointly developed by the Department of Electrical Engineering and Computer Science of the University of California, Berkeley, Magneti Marelli, Torino Italy, and the Department of Electrical Engineering of the Politecnico di Torino, Italy. It is targeted towards real-time control systems that are composed of software on a micro-controller and semi-custom hardware components.

The environment is based on a uniform representation for hardware and software, Codesign Finite State Machines ([2]), that are Finite State Machines communicating by event broadcast, with an underlying unbounded reaction delay hypothesis. The system is described by the designer using a specification language for real-time reactive systems. At present we use ESTEREL [1] but our system is not intrinsically limited to it and is aimed to accept an array of appropriate languages in the future. After the design is described, it is partitioned by hand into cooperating hardware and software modules. The codesign system assists the designer in the partitioning task, by giving

estimates of the implementation costs as hardware or software. Hardware and software are then automatically generated for a given partition. Hardware is synthesized as a synchronous circuit, while software is synthesized as a collection of procedures, each implementing a CFSM. We are currently experimenting with various strategies for software implementation and optimization, based on a simplified control/data flow graph especially suited for real-time finite-state computations. In the future, close interaction with formal verification and validation tools will also allow to preserve correctness and ensure satisfaction of size and timing constraints.

In order to validate the overall design methodology, as well as the software tool supporting it, we carried out the design of an embedded controller subsystem of a vehicle. This example is an actual design being performed at Magneti Marelli. Its size was chosen in order to stress the capabilities of the codesign environment with a real-life case.

2 The Shock Absorber Controller

The subsystem being designed is a real-time controller which automatically modifies the response of the front- and rear-wheel shock absorbers according to sampled values from a set of sensors.

The system has the following input sensors:

- optical steering wheel sensor;
- vertical acceleration speed;
- longitudinal speed sensor;

The system outputs signals to four motors to set the response of the shock absorbers to three different levels: HARD, MEDIUM, and SOFT.

The function of the controller consists of the following tasks: input variable measurement, sensor fault detection, state variable computation, control strategy implementation, shock absorber motor control. The sensors send either voltage levels (vertical acceleration sensor) or pulses (steering angle and speed sensors) to the controller. The controller checks the incoming signals for open and short circuit, and for values outside the functional range. In the case of fault detection, it sends a DAMAGE signal to the user and sets the shock absorbers to HARD until RESET. The controller computes in real time the following state variables: horizontal vehicle speed and acceleration, steering angle and speed, vertical speed both at low and medium frequencies. For all of them the controller specification defines response curves to set the response of the shock absorbers. The motors are controlled in a feedback loop, according to the results of this response evaluation (prioritized from HARD to SOFT).

The current hand-designed board implementing the embedded controller contains several analog-to-digital data acquisition components, as well as power drivers for the motors and sensors for the feedback loop. The single Motorola 68HC11E9 micro-controller is used in expanded mode, using 32 Kbytes of external EPROM and 8Kbytes of external RAM, as well as its own internal RAM and EEPROM. There is also an on-board ASIC chip for controlling external memories, controlling the power MOS drivers, and diagnosing the motors. All the other tasks are implemented as SW.

3 Experiment Setup and Procedure

We first completed all of the initial specification task, translating an informal specification on paper into an interconnection of ESTEREL modules, one for each major sub-systems. These module are then validated individually through the use of the ESTEREL debugger ([3, 4]). The validation process is a tedious one as it is necessary to “formalize” the informal specification. Often time it is necessary to talk with the original designer to figure out what the words in the specification really mean. The interconnection of these ESTEREL modules is done with a simple hierarchical graphical user interface. At this graphical level the user may also specify the implementation attribute of each hierarchical component, thus partitioning the design into hardware and software components. After compiling the ESTEREL modules along with the hierarchical connectivity into SHIFT (Software/Hardware Intermediate Format, an intermediate representation format describing a network of CFSMs), a simulator

is used to validate the specification as a hierarchical network of interacting CFSMs. As previously stated, CFSMs are reactive FSMs which respond to external events with unbounded non-zero delay. This model allows to describe all implementation choices, since synchronous hardware has a minimum delay of 1 clock and software always reacts with some, possibly unbounded, delay.

The ESTEREL specification is currently composed of 64 modules, functionally organized into 8 hierarchical netlist files interconnecting the modules. Each SHIFT module (which corresponds to an ESTEREL module) takes on different design path from this point depending on whether it is labeled as hardware or software.

Hardware modules are bit-encoded and translated down to a logic implementation using the *SIS* logic synthesis system [5]. We are currently using an experimental setup in which Xilinx FPGAs are used to implement the hardware. The output of *SIS* hence is an XNF file that can be read by Xilinx physical design tools. All the necessary interfacing logic to the software domain are attached as part of the synthesis process.

Software modules go through several different synthesis and optimization steps, with various size/timing trade-offs. We use a standard control/data flow graph for software optimization, but we perform more powerful optimizations than are allowed to a general purpose compiler (which must solve a much more difficult problem). Beside automatically attaching the necessary hardware interfacing functions and signals, the synthesis process also produces a simple round-robin scheduler (we are planning to experiment with more sophisticated schedulers, such as for example rate-monotonic, later on). Micro-controller configuration routines are also included with the resulting C code, that can be compiled, loaded on the micro-controller, and run as a stand-alone program.

Validation of the final implementation comes in two forms: simulation and formal verification. At this stage we do not yet consider co-simulation. We believe that our framework could fit into existing co-simulation environments like Ptolemy [6]. Simulation of software and hardware are currently done independently, using a simple command interpreter appended to the software routine (like [4]) and commercial logic simulation tools. Formal verification can be done at the specification (implementation independent) level or with implementation attributes. An example of a property that the design should satisfy could be “Whenever an ERROR occurs (DAMAGE), the shock

absorber setting should be always HARD until a RESET event occurs”. It is expected that most existing verifications algorithm will have problems with the complexity of this design. We have an ongoing effort to find a suitable abstraction for the formal verification of real-life reactive system like our shock absorber.

4 Experimental Results

We have currently completed two versions of the shock absorber formal specification. One was written in ESTEREL and automatically translated into the SHIFT internal representation. The main problem encountered during this phase of the design was the interpretation of the original written informal specification. One person spent about a month interviewing the designers of the original system in order to formalize the specification. After completing the ESTEREL design, and manually inspecting the resulting SHIFT files, we decided to perform a second specification pass, in order to reduce expensive operations (e.g., divisions) as well as the amount of memory used (which initially was allocated exactly as in the original hand design). This second specification was done by one person in one week, directly in SHIFT¹.

We report mainly the results of a pure software implementation of the system, essentially because the number of arithmetic operations required would make an FPGA implementation impractical.

The computation performed during an execution cycle of a CFSM can be divide roughly in the following “phases” (which may partially or totally overlap): input event detection, arithmetic and boolean function computation, next state and output event emission. Our software synthesis system uses two major paradigms to implement this reactive behavior²:

- using the program counter to encode the “state” of the computation, e.g. to remember which events have been detected, which comparison operators have returned the value *true* and so on.
- using a set of variables (memory locations or registers) to represent the same information in a more “distributed” form.

¹We decided not to use ESTEREL for this pass to obtain a finer control over the CFSM network; we are planning to perform the same optimizations at the ESTEREL level as well and compare the results of the automated translation with the hand-written SHIFT file.

²One third possibility would be to use a table-driven approach, where input events and values are concatenated to index a giant (possibly multi-level) table immediately determining the CFSM reaction. We have not yet implemented this approach.

The first paradigm lends itself to an *if-then-else* implementation, while the second paradigm can be implemented almost without branching, just using Boolean machine-level instructions.

For example, consider a specification that requires to emit event *d* if and only if either events *a* and *b* or event *c* have been detected. It can be implemented by the following decision graph:

```
begin:
  if a then goto test_b
  else      goto test_c
test_b:
  if b then goto emit_d
  else      goto test_c
test_c:
  if c then goto emit_d
  else goto end
emit_d:
  emit d
end:
```

Alternatively, we can encode it in the following sequence of assignments:

```
begin:
  t0 = a and b
  t1 = t0 or c
  if t1 then emit d
end:
```

Both paradigms have some advantages and some disadvantages, and in the future we are planning to integrate them into a single optimization subsystem. For now, we can show global trade-offs between implementations of a complete CFSM using either technique.

Another possibility of optimization within the first paradigm is the choice, whenever the need to test a multi-valued input variable arises, between testing single bits or performing a multi-way branch (like *switch* in C or *case* in Pascal). In this particular example multi-way branches are more expensive than bit tests, but this may not always be the case (e.g., there may be significant differences with the use of other micro-controllers).

The current implementation of the *if-then-else* paradigm relies heavily on the use of Binary Decision Diagrams ([7, 8]) to determine the best ordering for input event testing and output event emission.

Table 1 shows the ESTEREL source code size³ and the number of SHIFT code lines for the first and sec-

³Measured as number of lines, both including and excluding comments; the comments include the value of constants defining the curves, which are not part of the ESTEREL source *per se*.

Component	ESTEREL		SHIFT	
	With comments	Without	ver.1	ver.2
bat_diag	196	88	54	57
long_acc	3112	297	5896	559
long_speed	1021	518	661	346
mot_ctrl	824	178	3739	843
steer_ang	2064	590	2742	479
steer_speed	1665	428	2220	342
timer	152	96	128	77
ver_acc	2655	849	5218	591

Table 1: ESTEREL and SHIFT source code size

Component	Var. data	Const. data	Code
bat_diag	102	75	1604
long_acc	1545	2082	16088
long_speed	374	367	5395
mot_ctrl	621	664	11893
steer_ang	715	912	8874
steer_speed	590	772	6392
timer	109	70	1367
ver_acc	1233	1299	13952
total	3462	7125	57859

Table 2: Executable code and data size

ond version of the top-level components of the shock absorber control.

Table 2 shows the code and data memory (ROM + RAM) size in bytes of the system components. Each component was compiled, using the *INTROL-C* compiler, as a stand-alone program to be run on the microcontroller MC68HC11E9 (including port I/O drivers and a simple round-robin scheduler). The table reports the results for the implementation strategy described above, using if-then-else structures with bit tests and optimized event ordering. The total is somewhat less than the sum of the individual files because compiling everything together reduces the overhead associated with the scheduler and I/O handler. Note that the final result does *not* fit in the target microcontroller. The problem is due to the fact that table look-ups for the response curves are currently specified as a sequence of tests and assignments. We believe that we can substantially reduce the size of the code by using arrays of constants to perform the same task.

Table 3 shows the comparison (for some components) between the different paradigms to implement the software, as discussed above.

Table 4 shows the code size and estimated running time for different decision-based implementation techniques ("optimized" refers to the optimization of the

Component	Decision tree		Variable assignment	
	data	code	data	code
long_speed	320	2330	1782	1847
long_acc	118	660	2032	788
steer_ang	212	1983	2296	2247
steer_speed	88	400	820	497
timer	64	292	164	388

Table 3: Implementation paradigms

Component	2-way test			
	unoptimized		optimized	
	size	cycles	size	cycles
bat_diag	576	555	564	524
long_acc	563	1301	558	1293
long_speed	6022	3700	2330	3521
steer_ang	7054	7370	5299	7328
steer_speed	4458	6265	3716	6271
timer	352	728	292	728
ver_acc	10894	13657	9274	13737

Component	multi-way test	
	size	cycles
bat_diag	610	512
long_acc	571	1313
long_speed	3591	3920
steer_ang	14643	7558
steer_speed	8241	6422
timer	342	670
ver_acc	20416	13922

Table 4: Size and running time for test-based implementation

event testing order). The running time (in clock cycles on an MC68HC11E) is *estimated* from the control/data flow graph using a formula (with parameters obtained from benchmark programs) that estimates the delay of each node in the graph. The precision of the estimator with respect to true cycle counting, is currently on the order of $\pm 20\%$. It has the advantage of being much faster and directly coupled with the software synthesis data structure (i.e., it does not require to produce and compile C code).

The total run time for software synthesis and optimization of the shock absorber control for the bit test-based implementation is about 20 minutes of CPU time on a DECsystem 5000/260 with 128MB of memory. Most of the time is currently spent during the *encoding* of the bits. We believe that encoding does not affect too heavily the final result, and hence that the synthesis time can be decreased dramatically. As

a reference, the INTRON-C compilation time for the resulting program is about 5 minutes of CPU time on the same machine.

5 Conclusion and Future Work

The experiment that we have performed has shown the ability of the system to handle real-time designs, as well as exposing some limits of its current implementation. Namely we would like to improve the handling of arithmetic operators, which could result in significant cost saving especially for *hardware* implementations (right now the scheduling done by the ESTEREL translator is targeted towards a software implementation, where possible sharing among expensive operators does not matter, everything being sequential).

We are currently also setting up a hardware/software test-bed built around the Aptix field programmable interconnect board, used to flexibly interconnect a set of Xilinx FPGA chips (for the hardware partitions), with the target micro-controller (currently an in-circuit-emulator of the 68HC11). The shock absorber design, along with others, will be implemented and experimented with various implementation choices and trade-offs.

References

- [1] Frederic Boussinot and Robert de Simone. "The ESTEREL language," *Proceedings of the IEEE*, 79(9):1293-1304, September 1991.
- [2] Massimiliano Chiodo, Paolo Giusto, Harry Hsieh, Attila Jurecska, Luciano Lavagno, and Alberto Sangiovanni-Vincentelli. "A formal specification model for hardware/software codesign," *Technical Report UCB/ERL M93/48*, U.C. Berkeley, June 1993.
- [3] CISI Ingenierie, Agence Provence Est, Les Cardoulines B1 06560 Valbonne, France. *Esterel V-3, Language Reference Manual*, 1988.
- [4] CISI Ingenierie, Agence Provence Est, Les Cardoulines B1 06560, Valbonne, France. *Esterel V-3, Debug Format Manual*, 1988.
- [5] E.M. Sentovich, K.J. Singh, C. Moon, H. Savoj, R.K. Brayton, A.L. Sangiovanni-Vincentelli. "Sequential Circuit Design Using Synthesis and Optimization," *Proceedings of ICCD*, October 1992.
- [6] S. Lee, J. M. Rabaey. "A Hardware-Software Co-Simulation Environment," *Second International Workshop on Hardware-Software Co-Design*, October 1993.
- [7] C. Y. Lee. "Representation of switching functions by binary decision programs," *Bell System Technical Journal*, v. 38, 985-999, 1959.
- [8] R. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, C-35, 677-691, August 1986.