Efficient automatic code generation for embedded systems

D. Pilaud

Verilog, Agence Rhône-Alpes, Centre Espace, 2 rue de Vignette-38610 Gières, France

Abstract

Developing a safety critical real-time application raises high challenge: "failure is not an option". The code has to be readable, reliable and efficient. For doing so, VERILOG has developed an environment based on formal approach. This is the only way to be in a position to prove that the code is doing what it is supposed to do, always.

Keywords: Safety critical systems; Data flow models; Lustre; SAO⁺/SAGA environment

1. Introduction

Today, in the domain of real-time critical systems, software engineers do not only program, but also specify, validate and implement, to attain the end goal of a proven fault-free system. Block diagram type languages are extensively used today for avionics systems, where the software component has a dominant role.

The designers' primary need therefore remains a formal structure adapted to their culture, enabling them to guarantee that their designs will function correctly, and to discover design errors as early in the process as possible. For other aspects, faced with the increasing complexity of software, the automation engineers share most concerns of the software designers:

- specify and simulate a system;
- automatically derive target code based on a high-level description;
- detect any inconsistencies as early as possible;
- deploy a development methodology based on their own experience and company culture;
- improve software reliability and security by simple means (readability of product code, reuse), or more sophisticated methods (formal verification, certification of tools used);
- be capable of automatically generating a complete set of documents based on what they develop;
- if necessary, demonstrate to the safety authorities that the software developed is fault-free: guaranteed long-term viability of the tools they use.

Synchronous data-flow languages provide a satisfactory solution for designers of critical real-time systems. The data-flow model is in fact directly based on a description in the form of the block diagrams preferred by automation engineers. Moreover, the synchronous model perfectly matches the execution mode of the systems described, in terms of input sampling, calculation, output transmission and return in an infinite loop.

In Section 2, some results concerning the data-flow approach in avionics are presented. Section 3 is devoted to the description of the basic concept of Lustre language. This language allows the data-flow diagram to be describe formally. In Section 4, the SAO⁺/SAGA tool is detailed and some results of the utilization of the automatic code generator are shown.

2. The synchronous approach as applied to avionics

The concrete benefits of this type of approach have already been shown through the development of the critical software for the Airbus A320 and A340 jetliners. Several articles [1–4] have already spotlighted the specific advantages:

- A very significant decrease in the number of coding errors (from several hundreds to just a few), due to the extensive use of automatic code generation based on detailed specifications. For the Airbus A340, automatically-generated code accounted for 70% of
the total. Progress in successive Airbus aircraft is summarized in Table 1.
- Total control over the software life-cycle. More specifically, the specification changes required during the development stage are under perfect control. Those involved pointed out the ability to very quickly and reliably deploy the modified software after a request for change.
- Sextant Avionique focused on the gain in productivity. Reaching about 20%, this gain is all the more significant since it is achieved despite the software doubling in size every five years—a factor that would normally facilitate a loss in productivity. If an effort of 1 is needed to write 100 000 lines of code, then we assume that without changing the method, the cost varies between 3 and 12 for 200 000 lines of code! The use of the synchronous approach and appropriate tools have made an important contribution to keeping costs under control despite the increase in software volume.

Results in the avionics sector have been confirmed by similar results obtained by Schneider Electric in the development of critical real-time software for a power application. Other advantages mentioned by users are ease of training and the rapid acquisition of a common culture by the entire project team.

3. Lustre: A synchronous data-flow language

One method for reliable design is to use high-level languages, which allow a natural expression of problems. Within the domain of reactive real-time systems, many people are used to designing their systems by means of networks of operators, transforming flows of data (gates, switches, analog devices, transfer functions, and dynamical equations) to capture the behaviour of the system.

The data-flow model is based on a block diagram description. A system is made up of a network of operators acting in parallel and in time with their input rate. Such formalism possesses several of the advantages required for a high level language:
- it is a functional model, and therefore has mathematical cleanliness. It clearly fits formal verification allowing a natural expression of properties,
- reusability is made easier by incremental construction of operators networks,
- a maximum use of parallelism is ensured (the only constraints are dependencies between data),
- the data-flow model has a natural graphical expression as well as textual representation using mathematical equations. Consequently, it fits the different cultural backgrounds of the end-users.

The synchronous model [5] was introduced to supply primitives allowing the program to handle external data instantaneously. Each output in the program is attributed to a precise data in relation to the flow of entities in input. A discrete time-scale is introduced. The time granularity is considered to be adapted a priori to the time constraints imposed by the dynamics of the environment on which the system is to react. It is verified a posteriori.

Each instant of the time scale corresponds to a calculated cycle, i.e. to the arrival of new inputs. The synchronous hypothesis presumes that the means of calculation are powerful enough for the level of granularity of the discrete time-scale. Consequently, outputs are calculated and inputs are taken into account “at the same time” (with regard to discrete time-scale);

The synchronous data-flow approach consists of adding a time dimension to the data-flow model. A natural way of doing this is to associate time with the rate of data-flow. The entities manipulated can naturally be interpreted as functions of time. A basic entity (or a flow) is a couple made of:
- a suite of values of a given type,
- a clock representing a suite of graduations (on the discrete time scale).

A flow takes the \( t \)th value in its suite at the \( t \)th instant of its clock. In the example shown in Fig. 1:

\[
\text{count}(n) = \begin{cases} 
\text{if} \ reset(n) \text{ then init}(n) \\
\text{else} \ count(n - 1) + incr(n) 
\end{cases}
\]

where \( \text{count}(n) \) indicates the value of count at the \( n \)th instant.

Lustre [6] is a synchronous language based on the data-flow model. The synchronous aspect introduces constraints on the type of input/output relations that can be expressed: the output of a program at a given instant cannot depend on the future inputs (causality), and can depend on only a limited number of inputs. The operators allowed to express the operators networks are the classical arithmetic and logic operators. Moreover, each cycle can memorize the value of the previous input using a specific operator called pre. Another
specific operator can force a value of a data at the first cycle. Finally, operator networks can be computed at different rhythms manipulating the clocks of the data-flows.

4. SAO+/SAGA environment

Based on the experience acquired by Schneider Electric [7] and Aerospatiale in designing critical real-time systems, Verilog decided to use the Lustre language for its SAO+/SAGA (computer-aided specifications) environment. Working in close harmony with manufacturers and research teams, Verilog was able to construct this environment based on simple principles, with a constant focus on meeting user requirements, while at the same time taking advantage of advanced research into Lustre language applications.

SAO+/SAGA is an open environment comprising two main tools.

The first is a multiview graphics editor, for which parameters can be set according to the methodological rules established by the user. This editor can be connected to a documentor to produce structured documents in various formats (PostScript, Interleaf, SGML, etc).

The second is an automatic code generator, that produces optimized, readable ANSI C code. It can also be connected, via an intermediate format, to a Verimag formal property verifier.

The automatic code generator tool generates 100% of the C code that implements a description either directly in the Lustre text language, or using the SAO+/SAGA environment editor.

The code generated meets several requirements resulting from the needs expressed by the users:

- work in conjunction with Siemens Automotive gives some results concerning the expansion factor of the automatic code with optimized handwritten code obtained from a real application. Where the assembly code is generated automatically, it is less than 15% larger than when it is generated manually. For this experiment, we have used the Siemens 16 bit 80C166 micro controller and BSO/Tasking Cl66 compiler to obtain these results.

- The code generated is readable: its structure matches the hierarchy of the Lustre description and the variables generated are named so as to ensure traceability with the data circulating in the Lustre description. Code readability is very important for the user for two reasons. It strengthens his confidence in the code generated, and, concerning critical systems, there are safety authorities who want to make sure that the software developed meets certain reliability and readability requirements.

- The code generated can be simulated without being instrumented, thanks to the production of a main program. The obvious advantage here is that what is simulated is also what is installed on the target machine.

In other words, when reliability and security are paramount, the code generator is another key to project success.

5. Conclusion

Designers of real-time critical systems are therefore seeing the emergence of languages and tools which, at last, respond to their main concerns. This type of technology has already generated significant gains in productivity. Aerospatiale, for example, has achieved a productivity gain of 55% in Airbus A320 design tasks, using its in-house computer-aided specifications tool.

In critical areas such as these, safety authorities require that designers prove the software is "fault-free". To meet this requirement, Verilog is preparing the certification of its new software tools. These tools were developed following the strong quality process required for system critical software.

References

