High Level Design using SyncCharts:

A Case Study

Charles André
I3S Laboratory - University of Nice Sophia-Antipolis / CNRS
2000 route des Lucioles, BP 121
06903 Sophia Antipolis cédex - France
andre@i3s.unice.fr

Abstract

This presentation shows how "SyncCharts" can be used in the design of a controller.
"SyncCharts" is a graphical synchronous formalism that inherits from Statecharts in its look and from Esterel in its foundations. SyncCharts supports hierarchical descriptions, concurrency and preemption. We use it to express the exepected behavior of control-dominated systems. Choosing the non trivial example of a binary encoder/decoder, we illustrate high-level description capabilities of this model. Then, thanks to the mathematical semantics of SyncCharts, we explain how to validate the design, using both interactive simulation and model checking. Finally, with the help of synthesis tools present in the Esterel software environment, we show how our approach makes the design easier, without loss of rigour or efficiency.

Keywords:

System specification and modeling, Validation, Synchronous programming.

1. Introduction

State-based models are often used to express expected behavior of reactive systems. Many people feel more comfortable with graphical representations. Explicit representation of states facilitates the understanding of behaviors; possible animation of the model makes it still easier. A danger of graphical representations may be a weak, or even worse, the absence of, semantics. Too many graphical models are semi-formal, indeed even informal. Ambiguity disqualifies such models in digital system design. Mealy machines are mathematically well-founded models but they are "flat" models, leading to huge and useless graphs for complex systems. We adopt a higher-level model, like Statecharts [1], able to deal with hierarchy, concurrency and pre-emption. The actual model we use is "SyncCharts" [2], clearly inspired by Statecharts. The two models differ in their underlying semantics. The SyncCharts[1] semantics is fully synchronous and perfectly fits Esterel's semantics [3]. The semantics of Statecharts, such as the one adopted in Statemate [4], is more complex (micro-step semantics). Moreover, SyncCharts offers richer constructions for preemption. Being akin to Esterel, SyncCharts may also include textual descriptions written in Esterel: the designer may choose textual or graphical descriptions for different parts of his/her design.

In fact, SyncCharts is now fully integrated in the "Esterel Studio" platform, marketed by Simulog [5]. As a consequence, SyncCharts has direct access the whole programming platform developed for Esterel: compilers, simulators(XES), model-checkers (XEVE [6]) and circuit optimizers that rely on SIS [7] and TiGeR [8] (an efficient BDD-based tool).

[1] "SyncCharts" is the name of the model, whereas a "syncChart" is an instance of the model.

In this presentation, we revisit the design of a binary stream encoder/decoder. Through this example we try to draw advantages of our approach, and contrast it with the classical approach proposed in Zahnd's book on Sequential Machines [9].

2. Example of an Encoder/Decoder

The Encoder/Decoder system, represented on Figure 1, is classical in the field of data transfer. It has been devised for electrical transmission by wire. Even if wireless communication has lessened the significance of this coding technique, it is still worth studying it because it raises several interesting algorithmic issues.

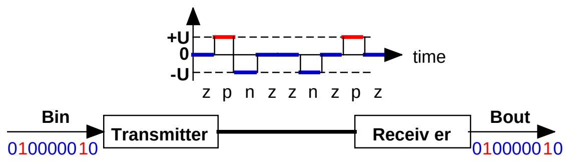


Figure 1. Encoder/decoder

Informal presentation

This encoding/decoding system is used to transmit binary streams. Bits are encoded into a three-level valued electrical voltage: positive ($p=+U$), negative ($n=-U$), or null ($z=0$), for a given constant positive voltage U.
Two requirements are imposed:

1. The mean voltage must be 0, and at each instant the accumulated voltage must stay between $-U$ and $+U$. This avoids electronic problems due to bias polarity.
2. The transmitted code shall never contains more than three consecutive null values. This prevents from clock de-synchronization and misinterpretation of a line break as a continuous stream of *z*'s.

Typically these requirements are imposed for physical/electrical reasons. The first requirement is easily captured by a simple encoding technique: *z* for 0, and either *n* or *p*, in alternation, for 1. The second requirement is trickier: sub-sequences of four consecutive 0's are also encoded with *n* or *p*. In order to set apart "true" 1's from "false" 1's (series of four 0's) "Violation" of polarity is used, instead of "Alternation".

Formal specification

This part is omitted in this short document. The important fact to notice is that the encoding may be either "standard", or "exceptional" according to the incoming bit flow. This frequent dynamic switching between encoding algorithms is typical of highly reactive system behavior.

3. A classical solution

In his book, Zahnd chose a Mealy machine as a model to represent the encoder example. The demonstration will show the great pedagogic interests of this model and also its drawbacks, making it not easy to modify and reuse.

4. SyncCharts-based design

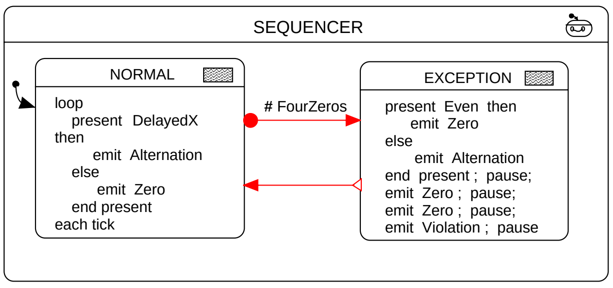
With SyncCharts, the application is seen as a collection of interacting agents. These agents are tightly coupled by instantaneous broadcasting of information and instantaneous reactions (synchronous hypotheses). The behavior of an agent is specified with a macro-state. A macro-state is translated into a module in Esterel. Figure 2 is the description of the behavior of the agent in charge of the effective encoding. Note the mixture of graphical and textual notation.

Compilation

The outlines of the compilation chain are:

- SyncCharts compiler: From a syncChart to a semantically equivalent Esterel program;
- Esterel compiler: From an Esterel program to output code:
  - C programs for simulation with XES
  - Blif description for optimization (with SIS) and verification (XEVE).

  Blif (Berkeley Logic Interchange Format) is a textual representation of a circuit. It is an input format to SIS.



```
loop
   present  DelayedX
then
      emit  Alternation
   else
      emit  Zero
   end present
each tick
```

```
present  Even  then
   emit  Zero
else
   emit  Alternation
end  present ;  pause;
emit  Zero ;  pause;
emit  Zero ;  pause;
emit  Violation ;  pause
```

Figure 2. SyncChart of the sequencer

5. Validation and Performance

To validate the design, we proceed in two steps: simulation and formal verification.

Test of scenarios

XES is an interactive simulator, which is part of the Esterel distribution. Given an Esterel program, XES automatically builds simulation panels that show the status of input signals (set by the user) and output signals (set by the program under test). Execution is traced in the source program, so that the user can visualize concurrent evolutions and pre-emptions. This possibility is now extended to SyncCharts. "Esterel Studio" can do the animation of the syncChart of the controller.

Safety properties

Even if the design passes successfully all the tests, it is not sure that all the cases have been covered. In order to establish a safety property, we have to check this property in all reachable states of the controller. The size of the actual reachability set can make the analysis untractable. Fortunately, there exist symbolic computations of the reachability set that allow for state abstraction without loss of exhaustivity. XEVE, a symbolic model-checker available in the Esterel platform, is able to compute (symbolically) the state space of a given program. XEVE can formally establish whether or not a safety property is satisfied. Safety properties can be expressed by temporal logic formulas. We prefer to use the *same formalism* to express both behaviors and properties: a property is given as an Esterel module or a syncChart.

The principle of the proof is to associate an **observer** with the property and compose this observer in parallel with the controller to check. An observer is a reactive agent that "observes" input and output signals of the program and emits a "violation signal" as soon as the property is not satisfied. XEVE symbolically executes the program composed with the observer. If the violation signal is never emitted, then the property is satisfied, otherwise XEVE returns a sequence leading to the counter-example. This is a very effective way to find out deeply hidden errors. Of course, several properties can be checked at once if you use several observers. Note that the safety property observers are used during the verification phase (i.e., the symbolic execution of the controller augmented with the observers). There are needless at run-tine for a guaranteed property.

According to the property to be checked, the observer can be written in Esterel or in SyncCharts. We will illustrate both methods in order to prove that the two requirements above mentioned are satisfied. We will also establish that the pair "Encoder Decoder" works correctly, i.e., outgoing stream and incoming stream are the same (up to a delay).

Performance

The translation from SyncCharts to Esterel is structural: The translation is fully automatic but not always clever. In many cases an expert in Esterel language, can find more efficient translations. However, this is not a problem because there exist tools able to optimize the generated code, at the circuit level. Efficiency of the generation will be compared to "hand-coded" implementations.

References

[1] D. Harel. "Statecharts: a Visual Formalism for Complex Systems", *Science of Computer programming*, 1987, vol 8, pp 231-274.

[2] C. André. "Representation and Analysis of Reactive Behaviors: A Synchronous Approach" IEEE-SMC Computational Engineering in Systems Applications (CESA), Lille (F), July 1996, pp 19—29.

[3] G. Berry, G. Gonthier. "The Esterel Synchronous Programming Language: Design, Semantics, Implementation" *Science of Computer Programming,* 1992, vol. 19, n°2, pp 87-152. (current version of Esterel v5_21: http://www.inria.fr/meije/verification/esterel)

[4] I-Logix, "Statemate MAGNUM", http://www.ilogix.com

[5] Simulog. "Esterel Studio", http://www.simulog.fr

[6] A.Bouali. "Xeve: an Esterel Verification Environement", *Int'l Conference on Computer-Aided Verification (CAV'98),* june/july 1998, Vancouver, BC Canada. Also available as a technical report INRIA RT-214, 1997.

[7] E.M Sentovitch, K.J Singh, et al. "SIS: a System for sequential circuit synthesis". Technical report, *UCB/ERL M92/41,* U.C. Berkeley, May 1992.

[8] O. Coudert, J.C Madre, H. Touati. "TiGeR version 1.0, User Guide", Digital Paris Research Lab. Dec 93, commercialized by Xorix.

[9] J. Zahnd. "Machines séquentielles". *Presses Polytechniques Romandes*. 1987.