7

System-Level Hardware Design with μ-Charts

Jan Philipps, Peter Scholz
Technische Universität München, Institut für Informatik
D-80290 München, Germany
Tel.: ++49-89-289 {22398,28129}, Fax: ++49-89-289 28183
E-Mail: {philipps,scholzp}@informatik.tu-muenchen.de

Abstract

μ-Charts are a synchronous specification language for reactive systems with a compositional semantics. We show how a μ-Chart can be implemented in hardware, using a register and a combinational logic block that represents the transition relation of the system.

Keywords

System-Level Specification and Design, Formal Methods.

1 INTRODUCTION

The specification language μ-Charts is a dialect of Statecharts based on a modular syntax and a compositional semantics. Previous work has been focused on the semantical foundation of μ-Charts [5], on ways to extend them to a full-featured, flexible specification language, and on formal verification using model checkers [4].

In this contribution, we demonstrate how μ-Charts can be used as a hardware design language. Similar to the approach of Drusinsky [3] we aim at a single-block implementation, which uses a single combinational logic block and a state register. This eliminates the overhead of communicating finite state machines which results from approaches where each subchart is implemented as an independent state machine [2].

In contrast to Drusinsky's work, however, our approach is based on a compositional formal semantics. According to this semantics, each μ-Chart can be translated into a collection of transition predicates encoded with binary decision diagrams (BDDs). The BDDs can then be used both for verification and – if the specification is determinisitc – for a hardware implementation.

2 THE LANGUAGE

The syntactic representation of μ-Charts is modular. A specification is a tree of subcharts built from sequential automata, parallel composition, hierarchical decomposition, and an explicit feedback construction for broadcast communication. Since the semantics of μ-Charts is compositional, the initial configuration of a specification and its transition relation can be built hierarchically.

Sequential automata are denoted by $\mathsf{Seq}(N, \Sigma, \sigma_d, \delta)$, where N is a unique identifier, Σ denotes the nonempty, finite state set, and σ_d represents the default state. δ is a finite, total state transition function that takes a state and a finite set of signals and yields a — possibly empty — set of reactions. Each reaction consists of the subsequent state, paired with a finite set of output signals.

With $\mathsf{And}(S_1, S_2)$ we denote the parallel composition of two μ-Charts S_1 and S_2. Informally, the charts S_1 and S_2 operate independently; the output of the composition is the union of the outputs of S_1 and S_2. The following formulas, taken from [4], may give the reader an idea of how the semantics of parallel composition can be defined:

$$
\begin{aligned}
&\mathcal{I}_S((c_1, c_2)) \equiv \mathcal{I}_{S_1}(c_1) \wedge \mathcal{I}_{S_2}(c_2) \\
&\mathcal{T}_S((c_1, c_2), x, (c_1', c_2'), y, o) \equiv \\
&(\exists y_1, y_2 . \mathcal{T}_{S_1}(c_1, x, c_1', y_1) \wedge \mathcal{T}_{S_2}(c_2, x, c_2', y_2) \wedge y = y_1 \cup y_2) \vee \\
&((\nexists y_2, c . \mathcal{T}_{S_2}(c_2, x, c, y_2)) \wedge \mathcal{T}_{S_1}(c_1, x, c_1', y) \wedge c_2' = c_2) \vee \\
&((\nexists y_1, c . \mathcal{T}_{S_1}(c_1, x, c, y_1)) \wedge \mathcal{T}_{S_2}(c_2, x, c_2', y) \wedge c_1' = c_1)
\end{aligned}
$$

Here $\mathcal{I}_S(c)$ is true whenever c is an initial configuration of the μ-Chart S. The predicate $\mathcal{T}_S(c, x, c', y)$ is true whenever the μ-Chart S can transition from the configuration c to the configuration c' when input x is given; then, the output y is produced.

For finite signals sets, these Boolean predicates can easily be encoded as BDDs. The details of this translation, and the encoding of the configurations are presented in [4].

Neither parallel composition nor hierarchical decomposition introduce communication between subcharts. Broadcast communication is introduced explicitly with the feedback operator. This construction is denoted by $\mathsf{FB}(S, L)$, where L is the set of those signals that can be communicated within the chart S. Broadcast communication is achieved by feeding back signals in L instantaneously; these signals are added to the environment input and can cause a reaction in other subcharts. Due to space limitations, hierarchical decomposition is not considered here. As shown in [4], the formalism can easily be extended to incorporate a simple programming language on finite data states.

The predicates can be input to the μ-calculus verifier μcke [1]; using this tool, it is straightforward to verify properties of a specification, or to check whether it is deterministic, i.e. for each configuration and input, there is exactly one successor configuration and output.

3 HARDWARE GENERATION

In contrast to the tool Statemate, we aim at a *direct* implementation, and not at a compilation to VHDL code. Two previous approaches for direct hardware implementation of Statecharts were presented by Drusinsky in [2, 3]. The former implemented a Statechart as a network of communicating finite state machines; this scheme introduces considerable communication overhead. The latter realizes a Statechart as a single (possibly quite large) logic block. However, neither of these two approaches is based on a formal semantics. Thus, it is not possible to formally verify designs based on these approaches. Moreover, neither allows us to specify systems with data states.

In our implementation scheme, implementations are generated from the same transition relations that are used for formal verification of μ-Charts with symbolic model checkers. The logic block of the implementation is derived in the following way. The transition relation $\mathcal{T}$ is converted to a family of Boolean functions, one for each output signal, and one for each bit in the encoding of the configuration. The Boolean function for each bit is derived from $\mathcal{T}$ by existentially quantifying the other outputs. The conversion to Boolean functions is possible, since for hardware generation we restrict ourselves to deterministic μ-Charts.

Each Boolean function for an output signal contains an abstraction of the complete specification. This is the reason that our implementation scheme does not require explicit communication between the logic blocks. The abstraction contains those aspects of the complete system specification that are needed to calculate the signal's value — neither more nor less. This is the reason the individual Boolean functions can be represented with comparatively small BDDs, much smaller than the complete transition relation $\mathcal{T}$.

REFERENCES

[1] A. Biere. *Effiziente μ-Kalkül-Modellprüfung mit binären Entscheidungsdiagrammen.* PhD thesis, University of Karlsruhe, 1996. To appear. (in German).

[2] D. Drusinsky-Yoresh. Using Statecharts for Hardware Description and Synthesis. IEEE Transactions on Computer-Aided Design 8(7), pages 798 – 807, 1989.

[3] D. Drusinsky-Yoresh. A State Assignment for Single-Block Implementation of State Charts. IEEE Transactions on Computer-Aided Design 10(10), pages 1569 – 1576, 1991.

[4] J. Philipps and P. Scholz. Formal Verification of μ-Charts. 1996. To appear in: TACAS'97.

[5] J. Philipps and P. Scholz. Specifying Reactive Systems with μ-Charts. 1996. To appear in: TAPSOFT'97.