Simulation Modelling Practice and Theory 51 (2015) 69–86

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

Simulation of real-time systems with clock calculus

Kai Hu [a,*], Teng Zhang [b], Zhibin Yang [b,c,*], Wei-Tek Tsai [d]

[a] State Key Laboratory of Software Development Environment, Beihang University, Beijing, PR China
[b] School of Computer Science and Engineering, Beihang University, Beijing, PR China
[c] IRIT-CNRS, Université de Toulouse, Toulouse, France
[d] School of Computing, Informatics and Decision Systems Engineering, Arizona State University, Tempe, USA

ARTICLE INFO

Article history:
Received 12 July 2014
Received in revised form 20 October 2014
Accepted 23 October 2014
Available online 5 December 2014

Keywords:
SIGNAL
Clock calculus
Optimized clock tree
Code generation

ABSTRACT

Safety–critical real-time systems need to be modeled and simulated early in the development of lifecycle. SIGNAL is a data-flow synchronous language with clocks widely used in modeling of such systems. Due to the synchronous features of SIGNAL, clock calculus is essential in compilation and simulation. This paper proposes a new methodology for clock calculus that takes data dependencies into consideration. In this way, simulation code can be directly generated by using a depth-first traversal algorithm. In addition, a clock insertion method based on clock-implication checking is presented to obtain an optimized control structure.

1. Introduction

Safety–critical real-time systems such as automotive, avionics, and aerospace systems are usually reactive systems that need to interact with their environments. These systems often need to be modeled and simulated rigorously early in their development lifecycle.

Synchronous paradigm, based on *synchronous hypothesis* [1], is a formal method for modeling safety–critical systems. The synchronous hypothesis assumes that behaviors of the system will be represented as a sequence of discrete instants. At each instant, the system takes an input, performs computation, and produces an output with zero time. This abstraction simplifies timing analysis to allow the user to focus on the functional design without consideration of the computing platform. Other properties such as the temporal correctness can be considered when the platform has been chosen.

Synchronous languages such as Esterel [2], Lustre [3], QUARTZ [4] and SIGNAL [5], have been widely used. Esterel and QUARTZ are imperative languages while Lustre and SIGNAL are declarative. Apart from formal verification such as model checking and theorem proving, simulation is a major method for modeling of different kinds of systems [6].

An *abstract clock*, or a *clock*, is an important concept in synchronous languages. Each instant of a clock represents the status of computing including objects involved in computation. There are two kinds of clock models in synchronous languages:

* Corresponding authors. Tel.: +86 1082339460.
E-mail addresses: hukai@buaa.edu.cn (K. Hu), rahxephon89@163.com (T. Zhang), zhibin.yang@irit.fr (Z. Yang), wtsai50@gmail.com (W.-T. Tsai).

http://dx.doi.org/10.1016/j.simpat.2014.10.010
1569-190X/© 2014 Elsevier B.V. All rights reserved.

- Mono-clocked model: In this model, there is a global clock and other clocks are derived from the global clock. This is used in Esterel, Lustre and QUARTZ; and
- Multi-clocked model: there is no global clock, and clock relations are defined relative to each other. This is used in SIGNAL.

In mono-clocked models, all behavior activations are controlled by the global clock, and events triggered in components are subsets of events triggered by the global clock. All the clocks of individual components must be synchronized with the global clock.

Safety–critical real-time systems, however, may contain components with individual clocks in a distributed environment. These clocks are not usually synchronized globally. Instead, components are loosely coupled so that each component can execute on its own rate and the synchronization is only needed among interacting components.

Based on a typical multi-clocked model called Polychronous paradigm [7], SIGNAL is widely used in the design of GALS (Globally Asynchronous Locally Synchronous) system. Furthermore, an IDE tool for modeling, analysis and simulation based on SIGNAL called Polychrony[1] has been developed. However, to correctly simulate the SIGNAL specification, generation of correct and efficient code is still a research topic. The first difficulty is to extract control structure from synchronous equations based on the clock model. While a multi-clocked model is more realistic, its compilation is also complicated as relations among clocks are far more complex. As a result, *clock calculus*, a process to resolve the clocks and construct the control structure of the executive code, is a principal method in compilation. Note that although having the same terminology "calculus", clock calculus, a formal method dedicated for the compilation of Synchronous language, takes a very different approach from *timed* $\pi - calculus$ [8] which is mainly for the modeling of real-time system. Many studies have been carried out on clock calculus. [9,10] proposed the methodology of the clock calculus used in Polychrony. The idea is to extract the system of Boolean equations from the program and develop a hierarchical relation among clocks. If the program is endochronous [7], a single-rooted clock tree can be constructed and the sequential code can be generated based on the clock tree. Others [11–13] focus on the resolution of numerical expressions to improve the compilation precision.

Additionally, not only clock relations, data dependency information is also needed to generated simulation code. However, the current methods tend to separate clock calculus from the data-dependency analysis. Two different structures are needed: one for the clock hierarchy and the other for data dependency. To avoid the situation that hierarchical clock relations may not be consistent with data dependencies(eg. clock-to-data cycle), the two structures need to be combined [14]. In addition, when being executed, code with hierarchical and nested control structure will be more efficient than the corresponding flat code because in the nest structure, guard conditions enclosed will not be checked unless the outer ones evaluate to true. Therefore, it is desirable to design a clock calculus to generate efficient control structure.

To address these problems, this paper proposes a new clock calculus to analyze SIGNAL programs and generate simulation code. The calculus considers data dependency in constructing the clock tree. By using the DFS (Depth-First Search) traversal algorithm, simulation code meeting the data dependency constraints can be generated. Moreover, by using the idea of clock-implication checking, a clock can be inserted as deep as possible to generate a hierarchical and nested control structure for the simulation code.

The paper is organized as follows: Section 2 introduces SIGNAL and its code generation process; Section 3 provides a methodology on the insertion of the tree, and deal with the data dependency when constructing the clock tree; Section 4 presents a case study to illustrate the process of clock calculus and the simulation of SIGNAL program; Section 5 presents the related works; and Section 6 concludes this paper.

2. Introduction to SIGNAL

SIGNAL program (also called model in the remainder of the paper) is usually a set of equations on which relations among signals or their clocks are induced. Thanks to the multi-clock feature, every signal has its own clock to specify at which instant it can carry a value. This section introduces SIGNAL, including its syntax and semantics.

2.1. Signals and clocks

Based on the synchronous hypothesis, behaviors of the system are described as discrete sequences of instants and the system does the input-computation-output at each instant. The value sequence $(x_t)_{t \in N}$ manipulated in SIGNAL is defined as a *signal* where *t* is an index to represent the position of instant. At each instant, a signal can be present or absent (denoted by $\perp$). When present, a signal can hold values typed with integer, real, complex or boolean. Apart from usual types, there is a special type called *event*. When an event signal is present, its value is true. Otherwise, it is absent.

The *clock* is a set of instants where a *signal* is present. The clock of signal s is denoted as $\hat{s}$, which is an event typed signal. Moreover, clock relations among signals are defined in the program either explicitly or implicitly.

[1] http://www.irisa.fr/espresso/Polychrony.