Regular contribution

Verification of the Link layer protocol of the IEEE-1394 serial bus (FireWire): an experiment with E-LOTOS

Mihaela Sighireanu, Radu Mateescu

INRIA Rhône-Alpes and DYADE / VASY group, 655, avenue de l’Europe, F-38330 Montbonnot St. Martin, France; http://www.inrialpes.fr/vasy, E-mail: [Mihaela.Sighireanu, Radu.Mateescu]@inria.fr

Abstract. The IEEE-1394 Standard (“FireWire”) describes a high-speed serial bus for multimedia PCs intended to unify several serial buses such as VME, MultiBus II, and Future Bus. This paper deals with the formal description of the Link layer protocol of the IEEE-1394 Standard in the formal description techniques E-LOTOS and LOTOS, and its subsequent verification using model checking. The E-LOTOS descriptions are based on both the standard and the μCRL descriptions written by Luttik. The verifications are performed using the CADP (César/ALDEBARAN) toolbox. As a preliminary step, the E-LOTOS descriptions are translated, using the Traian compiler, in LOTOS, which is the input language for the CADP toolbox. Five correctness criteria stated in natural language by Luttik are formalized in the action-based temporal logic ACTL. These logic formulas assess safety and liveness properties of the Link layer protocol, such as deadlock freedom, appropriate sequencing of messages, etc. To contain the state space explosion, the verification is carried out only for a few scenarios involving a restricted number of nodes connected to the bus and a fixed number of transaction requests per node. Even under these restrictions, an undesirable deadlock situation in the protocol is detected. The error is caused by the ambiguous semantics of the state machines given in the standard, and it can be misleading for implementors of the IEEE-1394 protocol. A correction of this deadlock is proposed and validated.


1 Introduction

The design and development of complex, critical applications such as distributed systems and communication protocols are difficult tasks requiring a careful methodology in order to avoid errors as far as possible.

A real-life example of such application is the “FireWire” high performance serial bus defined in the IEEE-1394 Standard [17]. The bus involves \( n \) nodes (addressable entities that run their own part of the protocol) connected by a serial line. On each node the IEEE-1394 protocol consists of three stacked layers: the transaction layer, the link layer, and the physical layer. The protocol implemented by the link layer is designed to transmit data packets over an unreliable medium to a specific node or to all nodes (broadcast), and it is similar to an “acknowledged datagram” protocol. The transmission can be performed synchronously or asynchronously. The desired functioning of the link layer protocol can be characterized by means of several correctness properties [22]: the protocol has to be free of deadlocks; at a given time, the protocol should allow only a single link layer to send a packet over the bus; a non-broadcast packet should always be acknowledged; a packet requiring an immediate response must give priority over the bus to its destination node, etc.

One approach that proved its usefulness in the design of critical applications is the use of formal methods throughout the design process, by means of specialized tools. For this purpose, the application must be described using an appropriate high-level language such as μCRL\(^1\) [13], LOTOS\(^2\) [18], or E-LOTOS\(^3\) [28]. Such descriptions provide a formal, unambiguous basis upon which the verification of the desired correctness properties can be attempted.

A verification method that has been extensively studied over the last few years, and for which various algorithms and tools have been developed, is called model checking. In this approach, the correctness properties

---

\(^1\) micro Common Representation Language
\(^2\) Language Of Temporal Ordering Specification
\(^3\) Extended LOTOS
are verified on a model automatically generated from the high-level description of the application under design. Although restricted to finite-state systems, model checking provides a simple, efficient way to detect errors in the early steps of the design process.

This paper aims to show the adequacy of the E-Lotos language and the Cadp toolbox for the verification of industrial applications. We illustrate this by analyzing the protocol of the asynchronous transmission mode of the link layer described in the IEEE-1394 Standard. The E-Lotos language is a new generation of LOTOS that extends the Fdt Isis [18]. The Cadp toolbox [7, 10, 11] is dedicated to the design and verification of communication protocols and distributed systems. Since Cadp supports LOTOS as input language, we use the Traian [32] compiler to translate E-Lotos descriptions into LOTOS ones. To express and verify the correctness properties of the protocol, we use the Actl [25] temporal logic, for which a model-checker is available within the Actl [23] tool of Cadp.

Using this methodology, we were able to identify and correct an undesirable deadlock occurring in the protocol. These results are encouraging, since they show the effectiveness of the approach in the framework of real-life industrial applications.

The paper is organized as follows. Section 2 introduces briefly the LOTOS and E-Lotos languages. Section 3 gives an informal presentation and an E-Lotos description of the IEEE-1394 three-layered architecture. Sections 4, 5, 6, and 7 contain informal presentations and E-Lotos descriptions of the data types, the Trans layer, the Link layer, and the Bus layer, respectively. Section 8 introduces the Cadp protocol engineering toolbox. Section 9 presents the generation of the LTS models corresponding to the E-Lotos descriptions. Sections 10 and 11 describe the correctness properties and their verification on the LTS models, respectively. Section 12 gives some concluding remarks. The complete E-Lotos descriptions of data types, Link layer, and Bus layer are given in Annexes A, B, and C, respectively.

2 The ISO language LOTOS and the E-Lotos language

LOTOS [18] is a standardized Formal Description Technique intended for the specification of communication protocols and distributed systems. Several tutorials for LOTOS are available, e.g. [2, 31].

The design of LOTOS was motivated by the need for a language with a high abstraction level and a strong mathematical basis, which could be used for the description and analysis of complex systems. As a design choice, LOTOS consists of two "orthogonal" sublanguages:

- The data part of LOTOS is dedicated to the description of data structures. It is based on the well-known theory of algebraic abstract data types [14], more specifically on the AcTOne specification language [6].

The control part of LOTOS is based on the process algebra approach for concurrency, and appears to combine the best features of CCS [24] and CSP [16].

LOTOS has been applied to describe complex systems formally, for example Osset [19, Annex H] and Ftam basic file protocol [21]. It has been mostly used to describe software systems, although there have been recent attempts to use it for asynchronous hardware description [4].

A number of tools have been developed for LOTOS, covering user needs in the areas of simulation, compilation, test generation, and formal verification.

Despite these positive features, a revision of the LOTOS standard has been undertaken within ISO since 1993, because feedback from users indicated that the usefulness of LOTOS is limited by certain characteristics related both to technical capabilities and user-friendliness of the language.

The ISO Committee Draft [28] appeared in May 1998 and proposes a revised version of LOTOS, named E-Lotos. Compared to LOTOS, the language defined in [28] introduces new features, from which we mention only those used in this case study:

- Modularity: an E-Lotos module is a collection of types, functions and/or process definitions, the visibility of which can be controlled by interface declaration; modules may be combined using importation and renaming.
- Data types: types are defined in a functional style; in addition, many useful types are predefined.
- Sequential composition operator: the action prefix, enabling, and ‘accept’ operators of LOTOS have been substituted with a new, simpler sequential composition operator.
- ‘If-then-else’ operator: to express conditional constructs, an explicit ‘if-then-else’ operator has been introduced instead of guarded commands combined with choice.
- Imperative features: to allow an imperative-like programming style, write-many variables as well as functions and processes with in/out parameters have been introduced. These features try to align E-Lotos notations with standard programming languages.
- Gate typing: gates must be explicitly typed [9].

This case study is based on a version of E-Lotos described in [29, 30], which is slightly different from the Committee Draft one. The two main differences are: (a) instead of record subtyping and anonymous records, we use named records and overloading of functions and constructors; (b) although [28] introduces quantitative time, the fragment of E-Lotos we consider here is untimed.

---

4 Formal Description Technique

5 Distributed Transaction Processing

6 File Transfer, Access, and Management