EPSL: **E**xecutable **P**rotocol **S**pecification **L**anguage

Edmund Clarke[†] Yuan Lu[‡] Helmut Veith[†] Dong Wang[‡]

[†] School of Computer Science
[‡] Dept of Elect. and Comput. Eng.
Carnegie Mellon University
{emc,yuanlu,veith,dongw}@cs.cmu.edu

Motivation. The verification of bus protocols, i.e., of communication protocols between hardware devices as in the case of the well-known PCI bus, is a central problem in hardware verification. Although bus protocol design and verification become increasingly important due to the integration of diverse components in IP Core-based designs, even standard bus protocols are usually specified in English which makes specifications often ambiguous, contradictory and certainly non-executable.

Traditional hardware verification languages are usually not well-suited for protocol specification because they are based on existing concrete designs (or abstractions thereof) instead of specifications, and their execution model therefore focuses on single-cycle transitions. With protocols, the specification is naturally represented by constraints on signals which may connect relatively distant time points. Another problem of transition-system based approaches is that naive composition of participants in the protocol may cover up important protocol inconsistencies due to synchronization faults or write conflicts among non-cooperative participants. On the other hand, it is important that the specification language is executable, i.e., that a machine model can be computed from the specification.

The EPSL logic. We propose the new logical language **EPSL** which facilitates specification, verification, and simulation of protocols. EPSL is based on a variant of linear temporal logic (LTL) where atomic propositions are constraints on signals in the protocol, e.g. $REQ \in \{low, high\}$. The core of EPSL are so-called *EPSL scripts,* i.e., finite collections of executable temporal formulas similar to Horn clauses. More precisely, a script is a finite collection of axioms $\varphi \Rightarrow \psi$ where φ is a temporal formula involving **P** (past-time) as unique temporal operator, and ψ is a positive temporal formula involving **X** as unique temporal operator. Atoms in φ are interpreted as tests about previous signals, while ψ asserts constraints in the future. Scripts may contain local variables which are not visible in the traces (i.e., models) of the script.

An *EPSL module* $\mathcal{E}$ is a finite collection $\mathcal{S}_1 \| \ldots \| \mathcal{S}_n$ of *EPSL scripts*. The intended semantics is that EPSL scripts describe sequential machines (similar to Mealy Machines), and that the EPSL module specifies the synchronous composition of the sequential machines. Different scripts of a module potentially conflict if they employ common output signals.

A finite collection $\mathcal{P}$ of EPSL modules with pairwise disjoint output signals is called an *EPSL protocol*. The semantics of EPSL protocol is the synchronous composition of its modules. EPSL modules cannot directly conflict with each other, but they may have receptiveness failures.

In real protocols, EPSL scripts describe distinct functionalities and features of hardware devices, while EPSL modules correspond to the devices themselves. Note that different scripts can constrain the same signal, which is important when translating natural language specifications.

Results and Experiments. We show that semantically EPSL expresses exactly the regular safety properties. This property is crucial for synthesizing executable models of the protocol, and distinguishes it from other executable logics as in [3]. Note that a fragment of LUSTRE has been shown to capture the same expressive power [7], but provides a fairly different framework. We present efficient algorithms to obtain executable models of the protocols. The algorithm is based on an LTL tableau construction [1]. The tableau construction is used to obtain tableaus for the script formulas. (Formally, scripts $\mathcal{S}$ are translated into quantified temporal logic (QTL) formulas $\exists l_1 \cdots l_n \mathbf{G}(\mathcal{S})$ where $l_1 \cdots l_n$ are the local variables.) A determinization algorithm transforms the tableau into a Mealy machine whose outputs are controlled by the assertion atoms (i.e., the atoms in ψ) of the script. A combinator machine then is used to obtain values for each output and internal signal as the intersection of their constraints in the respective script traces. Finally, the protocol can be translated into either Verilog or the SMV input language. In the full paper, the logic is described in detail, and the formal correctness and generality of our approach is demonstrated.

Since each script is converted into a Mealy machine separately, there is no state explosion during the translation phase. The execution of EPSL in Verilog and SMV makes it possible to use the power of well-known and highly developed verification paradigms; in particular, it is possible to verify important features about the EPSL protocol, such as synchronization contradictions between different scripts, receptiveness [6] of the EPSL modules, local redundancy of EPSL scripts, and completeness of the specification.

Another important debugging method is property checking, i.e., existing Verilog monitors and CTL formulas can be easily used to debug EPSL protocols. To improve coverage of the Verilog simulation, a dynamically biased random simulation test bench can also be written directly in EPSL scripts.

In a case study, we have used EPSL to specify the PCI bus protocol Rev 2.2 [4]. Many errors were identified, including some errors from the English specification. Verilog monitors and CTL formulas from [2] have been checked against the generated Verilog and SMV models.

References

[1] R. Gerth, D. Peled, M. Vardi, and P. Wolper. Simple On-the-fly Automatic Verification of Linear Temporal Logic. *PSTV'95*, 1995.

[2] K. Shimizu, D. Dill, and A. Hu. Monitor Based Formal Specification of PCI. Submitted to DAC 2000.

[3] D. Gabbay. The Declarative Past and Imperative Future: Executable Temporal Logic for Interactive Systems. *Temporal Logic in Specification*, 1987.

[4] PCI Special Interest Group. *PCI Local Bus Specification Rev 2.2,* Dec. 1998.

[5] K. L. McMillan. Symbolic Model Checking. *Kluwer Academic Publishers*, 1993.

[6] R. Alur and T.A. Henzinger. Reactive modules. *Proceedings of the 11th Annual Symposium on Logic in Computer Science*, 1996.

[7] N. Halbwachs, J.-C. Fernandez, and A. Bouajjanni. An executable temporal logic to express safety properties and its connection with the language Lustre. *Sixth International Symp. on Lucid and Intensional Programming*, ISLIP'93, Quebec, April, 1993.

[8] E. Clarke, O. Grumberg, and D. Peled. Model Checking. *MIT Publishers*, 1999.