Abstraction of Concurrency and Communication in VHDL

Third Workshop on System Level Design Languages
Barga, Italy, July 1997

Peter J. Ashenden
Dept. Computer Science, University of Adelaide

Philip A. Wilsey
Dept. ECECS, University of Cincinnati

Extended Abstract

One of the difficulties that has been identified with using VHDL for design at the system level is the concrete nature of communication between subsystems. Currently, each subsystem comprises one or more processes that communicate using signals. Communication events are scheduled at specific times, and a receiver must respond to an event when it occurs, or else miss the event. Hence, there is no notion of synchronization as is seen in conventional concurrent programming or system description languages. If more specific synchronization is required, it can be achieved either by implementing a two-way communication protocol over signals, or by instantiating communication intermediaries, such as explicit message queues. Either approach detracts from the abstraction of communication required at the system level of description.

Previous proposals have addressed extending VHDL to support system-level description. Published proposals include Vista OO-VHDL by Swamy *et al*, the LaMI proposal by Benzakki and Djaffri, and the Bournemouth/IBM proposal by Cabanis *et al*. All of these proposals are based on object-oriented extensions to VHDL, and use classes to describe design entities. A class interface includes operations that can be invoked by other objects in the system, and the class encapsulates state that can be accessed by the operations. We have identified deficiencies in these proposals in earlier papers. The main problem is that reliance on classes and their operations leads to a monitor-based solution to concurrency and communication. While monitors may be appropriate for some applications, they have significant problems than limit their use as a general modeling paradigm.

Alternative modeling paradigms are illustrated by several system-level description languages in wide use, including Statecharts, Estelle, SDL and CSP and its derivatives. These languages are based on concurrent processes communicating through message passing. Depending on the language, processes are statically or dynamically instantiated, and process behavior is specified in state-machine form or as a thread of sequential statements. The communication structure is statically specified, but message passing may be buffered or unbuffered. Event triggering in Statecharts can be viewed as message passing without data transfer. Additional forms of communication include use of shared variables (with appropriate concurrency control) and remote procedure call (such as the rendezvous facility in Ada).

Given this analysis, it appears that VHDL could be improved to support system level modeling by providing a more abstract mechanism for communication and synchronization between processes and by generalizing the process model to allow dynamic instantiation and termination of processes.

---

* This work was partially supported by Wright Laboratory under USAF contract F33615- 95- C- 1638.

Such extensions to VHDL would make it more suitable for describing systems before partitioning into hardware and software implementations, and would make it more feasible to use a single language uniformly throughout the design flow.

There are a number of language design issues to be addressed in considering extensions to VHDL for more abstract forms of concurrency and communication. Among them are:

- process abstraction and interface
- run-time process creation and termination semantics
- message passing mechanisms and semantics
- abstraction of communication mechanisms as part of a design entity interface
- integration with existing language
- integration with other proposed extensions (e.g., SUAVE)

The full presentation will survey these issues and point out some possible approaches.