Accepted for publication in *Proceedings of the 1999 International Hardware Description Languages Conference and Exhibit (HDLCON '99).*

Communication and Synchronization Using Bounded Channels in SUAVE*

Peter J. Ashenden and Robert Esser
Department of Computer Science
University of Adelaide, SA 5005
Australia
petera@cs.adelaide.edu.au
esser@cs.adelaide.edu.au

Philip A. Wilsey
Dept. ECECS, PO Box 210030
University of Cincinnati
Cincinnati, OH 45221-0030
USA
phil.wilsey@uc.edu

Abstract

This paper described improvements to the abstract interprocess communication features added to VHDL as part of the SUAVE language design. Channel type declarations are extended to allow specification of bounded message buffers. This allows the designer to choose between asynchronous or synchronous message passing semantics. The latter makes models more amenable to formal verification based on state-space exploration. The language is also extended to allow specification of timeout intervals in select statements. This change makes the language more widely applicable, including for description of telecommunication protocols.

1. Introduction

As the complexity of integrated hardware and software systems increases, system-level design languages are becoming increasingly important. Such languages rely on abstraction as the key to managing complexity. Designers focus first on the abstract properties of a system in various domains and devise a systems architecture that will satisfy the requirements placed on the system. The domains under consideration include behavior, structure, performance, physical arrangement and packaging, power consumption, thermal, cost, and so on. In each domain, abstraction is used to focus on the major aspects of the system and minor detail is ignored. Judicious choice of abstractions makes architectural design and analysis tractable, and aids subsequent partitioning and refinement of the system design.

Hardware description languages focus on describing systems in the behavioral and structural domains. However, due to their origin as languages for hardware design, they frequently do not include strong capabilities for abstracting over data and for describing complex interactions. For example, in Verilog [12, 18], data types are closely bound to their binary representation, and signalling between modules includes aspects of electrical implementation. VHDL [1, 11], on the other hand, allows more abstract expression of data, and its type system is similar to that of conventional programming languages. However, its signalling features are still closely bound to electrical implementation.

To remedy these deficiencies, we have developed extensions to VHDL to improve its support for system-level modeling [2, 4–7]. These extensions are based on the requirement in a system-level description language for abstraction of data, concurrency, communication and timing. The mechanisms added to the language to support abstraction in these areas include:

- object-oriented data types,
- type generics,
- process declarations, with static and dynamic instantiation, and
- message-passing communication on channels.

This paper focuses on the last of these mechanisms. The communications mechanisms originally proposed provided asynchronous message passing on channel with unbounded buffer capacity. While this is appropriate for some modeling applications, other applications find synchronous communication or communication with bounded buffering more appropriate. Furthermore, formal proof of properties and refinements of communicating system are more readily achieved when synchronous communication or bounded buffering is used. Another deficiency in the original proposal was the lack of a mechanism to specify timeouts in communication statements. Such mechanisms

---

* This work was partially supported by a grant from Motorola Australia Software Centre.

are widely used in description of communication protocols, and its omission from SUAVE reduces the language's applicability in that area.

In this paper, we generalize the communication mechanisms of SUAVE to allow specification of buffer capacity of channels and specification of timeouts in communication statements. Section 2 reviews the communication mechanisms in the previous proposal and discusses the problems that arise. Section 3 presents revised mechanisms for describing channels, including mechanisms for specifying buffering capacity, and discusses the implications for formal proof of model properties. Section 4 presents revised communication statements, including a mechanism for specifying timeouts, and compares the mechanism with features of SDL [13]. We summarize and draw conclusions in Section 5.

2. SUAVE's Existing Features for Abstract Communication

In a previous paper [3], we identified a number of issues influencing our design of communication mechanisms in SUAVE. Based on a consideration of these issues, we chose asynchronous message passing as the communication paradigm. Messages are transferred between processes over named, typed communication channels. Channels may be connected to multiple receiving processes, allowing a form of multicast communication.

2.1 Channel types and objects

A model that uses channels must first declare the necessary *channel types* using channel type declarations. The syntax rule is

channel_type_definition ::=
    channel of subtype_indication
    | **null channel**

One or more channels may be declared using a channel declaration. The syntax rule is:

channel_declaration ::=
    channel identifier_list : subtype_indication ;

Channel declarations may appear within entity declarations, architecture bodies, block statements, generate statements, and package declarations. The subtype indication in the channel declaration denotes a channel type.

A channel is analogous to a signal, except that information is transferred using the *send* and *receive* message passing operations (described below). There is no notion of resolution of multiple source values, nor of specific times at which values occur on channels. A channel object denotes a first-in/first-out buffer of *messages*. When the channel object is created, the buffer is initially empty.

SUAVE also allows *interface channels*, which may appear as formal ports of design entities, components or blocks, or as formal channel parameters of subprograms. The syntax rule is:

interface_channel_declaration ::=
    channel identifier_list : [mode] subtype_indication

The mode, if present, is one of **in** or **out**, and the subtype indication denotes a channel type. An **in** mode channel may be used to receive messages, and an **out** mode channel may be used to send messages.

SUAVE provides mechanisms based on access types for dynamically creating channels in order to communicate with dynamically created processes. An access type may be declared to have a channel type as its designated type. Such an access type is called an *access-to-channel* type. A channel may be dynamically allocated using an allocator with a subtype indication denoting a channel type. The access value returned by the allocator designates the newly allocated channel.

2.2 Communication Statements

SUAVE extends the set of sequential statements to include *send statements*, *receive statements* and *select statements*. A send statement adds a message to the buffer of a channel. The syntax rule is:

send_statement ::=
    [label :] **send** [expression] **to** *channel*_name ;

The expression is disallowed if the channel is of a null channel type. Otherwise, the expression is required and denotes the value to be sent as a message. If the channel is of a null channel type, a data-less message is sent. Execution of a send statement involves adding the message to the tail of the message buffer of the named channel. The process executing the send statement then continues executing. If multiple processes execute send statements to the same channel concurrently, the order in which the messages are added to the message buffer is not defined. (It is implementation dependent.)

A process accepts a message from a channel using a *receive statement*. The syntax rule is:

receive_statement ::=
    [label :] **receive** [target] **from** *channel*_name ;

The target is disallowed if the channel is of a null channel type, otherwise it is required. The target must denote a variable name or an aggregate of variable names. Execution of a receive statement involves examining the message buffer of the named channel. If the message buffer is empty, the process suspends until a message arrives. When there is a message available, it is removed from the buffer. If the channel is not of a null channel type, the value of the message is assigned to the target using the same rules as variable assignment.

If multiple processes can read a message channel, all processes receive each message sent to the channel. Furthermore, all processes receive the messages from the channel in the same order. An implementation may achieve this effect either by providing one message buffer for the channel, from which each process copies message values, or by replicating the message buffer at each process.

A process may choose between a number of channels for message r