

Abstraction of Communication and Concurrency in VHDL

Peter Ashenden

University of Adelaide

Visiting Scholar at
University of Cincinnati

partially supported by Wright Laboratory
under USAF contract F33615-95-C-1638

Complexity Management

- For system-level design of behavior
 - abstraction of data
 - abstraction of communication & timing
 - abstraction of concurrency
- SUAVE:
 - SAVANT & University of Adelaide
VHDL Extensions
 - object-oriented data modeling
 - genericity
 - communication & concurrency

Design Objectives

- Abstract communication (*cf* signals)
- Dynamic process creation/termination
- Avoid bias toward hardware or software
- Integration with existing language and oo/genericity extensions
- superset of existing language

Communication

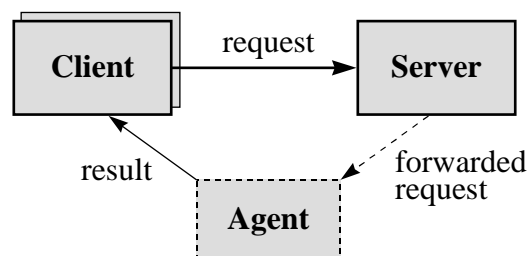
- Channel types
- Channel objects
- Interface channels
 - ports and parameters
- Dynamically created channels
 - access-to-channel types
- Message passing
 - send/receive
 - select (non-deterministic choice)

Concurrency

- Process declaration
 - generic and port clauses define interface
- Static instantiation
- Dynamic instantiation

Example

- Client-server system
 - number of clients not known *a priori*
 - multi-threaded server
 - creates new process to handle a request



System Overview

architecture system_level **of** client_server_system **is**

type result_value **is** ...;
type result_channel **is** **channel** result_value;
type result_ref **is** **access** result_channel;

type request_info **is** **record**
...; -- info for the transaction
result_please : result_ref;
end record request_info;

process client **is** ...

process server **is** ...

channel server_request : request_info;

begin

the_server : **process** server
 port map (request => server_request);

a_client : **process** client
 port map (request => server_request);

end architecture system_level;

Client Process Declaration

process client **is**
 port (**channel** request : **out** request_channel);
 variable result : result_ref := **new** result_channel;

begin

...
 send (..., result) **to** request;
 receive ... **from** result.all;

...
end process client;

Server Process Declaration

```
process server is
  port ( channel request : in request_channel );
  process agent is
    port ( channel request : in request_channel );
    variable info : request_info;
    begin
      receive info from request;
      ...; -- perform transaction
      send ... to info.result_please.all;
      terminate;
    end process agent;
    variable info : request_info;
    variable new_agent_request : request_ref;
  begin
    receive info from request;
    new_agent_request := new request_channel;
    process agent
      port map ( request => new_agent_request.all );
    send info to new_agent_request.all;
  end process server;
```

Summary

- Complexity management \Rightarrow abstraction
- SUAVE: abstraction for
 - data modeling
 - communication & timing
 - concurrency
- System-level modeling
 - pre hardware/software partitioning
- Further info
 - <http://www.ececs.uc.edu/~petera/suave.html>