SUAVE: Extending VHDL to Improve Modeling Support

Peter J. Ashenden
University of Adelaide

Philip A. Wilsey
University of Cincinnati

SUAVE

SAVANT and
University of
Adelaide
VHDL
Extensions

Outline

- Design Objectives
- Overview of Extensions
  - encapsulation
  - type derivation & inheritance
  - genericity
- What we have not done (yet)
  - entity/architecture inheritance
  - concurrency and communication

Design Objectives

- Support high-level modeling
  - improve encapsulation and information hiding
  - provide for hierarchies of abstraction
- Support re-use and incremental development
  - polymorphism, dynamic binding, type genericity
- Preserve capability for synthesis & other analysis
- Support hw/sw codesign
  - improved integration with programming languages

Design Objectives (cont)

- Refinement through elaboration of components
  - avoid repartitioning
- Preserve correctness of existing models
- Design principles from VHDL-93
  - preserve “conceptual integrity”

Overview of Extensions

- Borrow heavily from Ada-95
  - VHDL already has much in common with Ada
  - borrow encapsulation, information hiding, inheritance, genericity features
- Class-based *cf* programming by extension
  - class-based
    - replicates package features
    - choose one or the other, but not both!
  - programming by extension
    - integrates better with signal semantics

Encapsulation

- Strengthen existing package feature
  - used to define secure ADTs
- Package can have *visible part* and *private part*
- *Private type*
  - declare *partial view* in visible part
    - includes some contractual details
  - declare *full view* in private part
- Allow packages in any declarative region
  - local ADTs

Encapsulation Example

```
package complex_numbers is
    type complex is private;
    constant i : complex;
    function re ( C : complex ) return real;
    function im ( C : complex ) return real;
    function "abs" ( C : complex ) return real;
    function arg ( C : complex ) return real;
    function "+" ( L, R : complex ) return complex;
    function "–" ( L, R : complex ) return complex;
    . . .
private
    type complex is
        record
            re, im : real;
        end record complex;
end package complex_numbers;
```

Encapsulation Example (cont)

```
use complex_numbers.all;
signal a, b, sum : complex;
signal enable : bit;
. . .
sum <= a + b after 10 ns when enable = '1' else
          0.0 + 0.0*i after 10 ns;
```


Encapsulation: Contracts

- *Limited* private type
  - assignment not allowed, no predefined equality
  - use when deep copy/compare need