

# Reuse Through Genericity in SUAVE

---

Peter J. Ashenden

*University of Adelaide*

Philip A. Wilsey, Dale E. Martin

*University of Cincinnati*

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

## SUAVE

---

SAVANT and  
University of  
Adelaide  
VHDL  
Extensions

# Outline

---

- Design Objectives
- Overview of Extensions
  - genericity

# Design Objectives

---

- 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
- Preserve correctness of existing models
- Design principles from VHDL-93
  - preserve “conceptual integrity”

## Existing Generics

---

- Generic clause on entity, component, block
  - specifies generic constants
- Used for
  - operational parameters (eg, timing)
  - bounds for array ports

```
entity mux is  
  generic ( Tpd : time;  
            width : positive;  
            trace : boolean := false);  
  port ( d0, d1 : in bit_vector(0 to width - 1);  
        . . . );  
end entity mux;
```

## Extended Genericity

---

- Object-orientation is not a panacea
  - OO extension meets many, but not all, objectives
  - Doesn't include type genericity needed for reuse
- Adopt generics from Ada-95
  - modified to integrate with VHDL generics
  - formal types, subprograms, packages
  - allow generic clause in subprograms and packages
- Instantiation done at elaboration-time

## Formal Types in Entities

```
entity generic_mux is
  generic ( type data_type is private );
  port ( control : in bit; in0, in1 : in data_type;
        data_out : out data_type );
end entity generic_mux;

architecture data_flow of generic_mux is
begin
  with control select
    data_out <= in0 when '0',
              in1 when '1';
end architecture data_flow;

int_mux : entity work.generic_mux(data_flow)
  generic map ( data_type => integer );
  port map ( . . . );
```

## Formal Types in Packages

```
package sets is
  generic ( type element_type is private );
  type set is access private;
  constant empty_set;
  procedure copy ( from : in set; to : out set );
  function "+" ( R : element_type ) return set;
  impure function "+" ( L : set; R : element_type ) return set;
  . . .
private
  type element_node;
  type element_ptr is access element_node;
  type element_node is record
    next_element : element_ptr;
    value : element_type;
  end record element_node;
  type set is new element_ptr;
end package sets;
```

## Formal Types in Packages (cont)

---

```
type test_vector is . . .  
package test_sets is  
  new sets  
    generic map ( element_type => test_vector );  
use test_sets.all;  
variable tests_to_perform : test_sets.set := empty_set;  
  . . .  
  
test_to_perform := test_to_perform + new_test;
```

## Formal Types in Subprograms

---

```
procedure swap  
  generic ( type data_type is private )  
  ( a, b : inout data_type ) is  
  variable temp : data_type;  
  
begin  
  temp := a; a := b; b := temp;  
end procedure swap;  
  
procedure swap_times is new swap  
  generic map ( data_type => time );  
  . . .  
  swap_times ( old_time, new_time );
```

## Example: Generic Shift Register

---

```
entity shift_register is
  generic ( type index is (<>);
            type element_type is private;
            type vector is array ( index_type range <> ) of element_type );
  port ( clk : in bit;
         data_in : in element_type; data_out : out vector );
end entity shift_register;
signal master_clk, carry_in : bit;
signal result : bit_vector(15 downto 8);
...
bit_vector_shifter : entity work.shift_register(behavioral)
  generic map ( index_type => natural,
                element_type => bit,
                vector => bit_vector )
  port map ( clk => master_clk, data_in => carry_in, data_out => result );
```

## Interaction: Derivation and Genericity

---

- Formal derived type
  - provides mechanism for “mix-in” inheritance
  - obviates need for multiple inheritance in many cases

## Derivation and Genericity Example

---

```
package indexed_addressing_mixin is
  generic ( type parent_instruction is
            abstract new instruction with private );
  type indexed_instruction is new parent_instruction with record
    index_base, index_offset : register_number;
  end record indexed_instruction;
  function effective_address ( instr : indexed_instruction ) return address;
end package indexed_addressing_mixin;

type load_instruction is abstract new instruction with record
  destination : register_number;
end record load_instruction;

package indexed_loads is
  new indexed_addressing_mixin
  generic map ( parent_instruction => load_instruction );
alias indexed_load_instruction is indexed_loads.indexed_instruction;
```

## Conclusion

---

- SUAVE improves VHDL's support for modeling
  - across the spectrum
    - system-level down to gate level
  - improves encapsulation, inheritance, genericity
  - integrates cleanly with existing language
- Full details in papers and TRs
  - <http://www.ececs.uc.edu/~petera/suave.html>
- Implementation in progress