SUAVE: Extending VHDL for High-Level System Modeling

Peter Ashenden

University of Adelaide

Visiting Scholar at
University of Cincinnati

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

In a Nutshell ...

- VHDL is a standard hardware description language
- VHDL is good, but not good enough
- We're going to make it better!
  - at what it's already good for
  - and for modeling large complex systems
- SUAVE: SAVANT and University of Adelaide VHDL Extensions

Outline

- ☛What is VHDL?
- What is wrong with it?
- How other people want to fix it …
- How we are going to fix it …
  - If it ain't broke ...

Hardware Modeling

- Errors cost time and money
- Modeling: building a virtual system
  - Specification
  - Verification through simulation
  - Synthesis
- Model at different levels of abstraction
  - System level, register transfer level, gate level

Modeling Languages

- Verilog
  - originally proprietary simulator input language
  - influences (maybe) from C, Fortran
- VHDL
  - originated in VHSIC program
  - development sponsored by DoD, then IEEE
  - influences from Ada
    - similar requirements and development process

Entities

- Entity: basic hardware module
  - specifies interface of the module

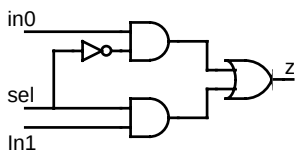
```
entity multiplexer is
    generic ( Tpd : delay_length );
    port (  sel : in bit;
            in0, in1 : in bit;
            z : out bit );
end entity multiplexer;
```

Architectures

- Architecture: implementation of an entity
  - can have alternative architectures for an entity
- Structural architecture
  - instances of entities interconnected with signals
- Behavioral architecture
  - process statements that implement an algorithm
  - processes respond to value changes in signals
  - processes schedule value changes on signals

Structural Example

```
architecture structural of multiplexer is
    signal sel_n, temp0, temp1 : bit;
begin
    sel_inv : entity gate_lib.inverter(basic)
                port map ( sel, sel_n );
    and_0 : entity gate_lib.and2(basic)
                port map ( in0, sel_n, temp0 );
    and_1 : entity gate_lib.and2(basic)
                port map ( in1, sel, temp1 );
    or_result : entity gate_lib.or2(basic)
                port map ( temp0, temp1, z );
end architecture structural;
```



Behavioral Example

```
architecture behavioral of multiplexer is
begin
    mux_behav : process is
    begin
        case sel is
            when '0' =>
                z <= in0 after Tpd;
            when '1' =>
                z <= in1 after Tpd;
        end case;
        wait on sel, in0, in1;
    end process mux_behav;
end architecture behavioral;
```


Behavioral Modeling

- sequential code in subprograms & processes
- concurrent processes communicating through signals
- logic modeling types
  - bit, multi-valued logic ('0', '1', 'Z', 'X', …)
- "abstract" user-defined types
  - integer, floating point, physical, enumeration
  - array, record, access (pointers), file

Packages

- Collection of declarations
  - eg, types, subprograms, etc.
  - package declaration: publicly visible
  - package body: separate & private
- Can be used to define an abstract data type

Package Example

```
package test_queues is
    use work.tests.all;
    constant max_size : positive := 100;
    type queue_array is array (0 to max_size – 1) of test;
    type queue is record
        head, tail : natural range 0 to max_size – 1;
        size : natural range 0 to max_size;
        the_buffer : queue_array;
    end record queue;
    function is_empty ( Q : queue ) return boolean;
    procedure add ( Q : in out queue;  item : in test );
    procedure remove ( Q : in out queue;  item : out test );
end package test_queues;
```

Package Example

```
package body test_queues is
    function is_empty ( Q : queue ) return boolean is
    begin
        return Q.size = 0;
    end function is_empty;
    procedure add ( Q : in out queue;  item : in test ) is . . .;
    procedure remove ( Q : in out queue;  item : out test ) is . . .;
end package body test_queues;
```


Outline

- What is VHDL?
- ☛What is wrong with it?
- How other people want to fix it …
- How we are going to fix it …
  - If it ain't broke …

High-Level Modeling Support

- Need ADTs to manage complexity
  - VHDL has poor encapsulation
- Need inheritance to classify abstractions and for re-use
  - VHDL has no form of inheritance
- Need late binding to support evolution
  - VHDL has only ad-hoc polymorphism
    - subprogram overloading

High-Level Modeling Support

- Need dynamic process creation to model dynamic reactive systems
  - VHDL has static process creation
- Need abstract communication
  - VHDL communication is via “hardware” signals

Outline

- What is VHDL?
- What is wrong with it?
- ☛How other people want to fix it …
- How we are going to fix it …
  - If it ain't broke …

Object-Oriented VHDL

- Add classes, like C++
  - provide encapsulation and inheritance
  - Willis *et al*; Radetzki *et al* (Objective VHDL)
  - replicates encapsulation mechanism (packages)
  - hiding state causes difficulties for signal objects
- Programming by extension, like Ada-95
  - Mills; Schumacher & Nebel
  - tagged records, derivation, class-wide types/ops

Objective VHDL Classes

```
type complex is class
    class attribute re, im : real;
    function arg return real;
    for variable
        procedure clear_to_zero;
    end for;
    for signal
        procedure clear_to_zero;
    end signal;
end class complex;
variable c : complex;
. . .
c.clear_to_zero;
```


Objective VHDL Classes

```
type complex is class body
    function arg return real is . . .;
    for variable
        procedure clear_to_zero is
        begin
            re := 0.0;  im := 0.0;
        end procedure clear_to_zero;
    end for;
    for signal
        procedure clear_to_zero is
        begin
            re <= 0.0;  im <= 0.0;
        end procedure clear_to_zero;
    end signal;
end class complex;
```

Entity Classes & Inheritance

```
entity gated_mux is new multiplexer with
    port ( enable : in bit );  -- other ports inherited
end entity gated_mux;
```


Entities and Communication

```
entity memory is
    operation read ( address : in natural ) return word;
    operation write ( address : in natural;
                      data : in word );
end entity memory;
```

Outline

- What is VHDL?
- What is wrong with it?
- How other people want to fix it …
- ☛How we are going to fix it …
  - If it ain't broke …

Language Design Principles

- Simplicity of mechanism
- Orthogonality of mechanism
  - with clearly defined interactions
- Integration with existing
  - semantic mechanisms
  - syntax
  - language philosophies

Issues: Concurrency & OO

- Orthogonal aspects
- Could add process types
  - allows dynamic instantiation & termination
- Alternative communication models
  - monitors
  - message passing on static channels (CSP)
  - RPC, rendezvous (Ada)
- Comm's interface part of entity interface

Issues: Entities & Inheritance

- Don't touch it!
- Too hard to extend encapsulated behavior
  - hacking at the innards
- Composition hierarchy is usually ok
  - *cf.* "is-a" hierarchy

Issues: Data Modeling

- Go all out here!
- Borrow from Ada-95
  - tagged types, derivation
- Strengthen encapsulation
  - private types in packages
- Genericity
- Stylistic integration

Tagged Types

- Record type that can be extended

```
type instruction is tagged record
    opcode : opcode_type;
end record instruction;

type ALU_instruction is new instruction with
    src1, src2, dest : reg_number;
end record ALU_instruction;

type mem_instruction is abstract new instruction with
    base_reg : reg_number;
    offset : integer;
end record mem_instruction;
```

Primitive Operations

- Define subprograms that operate on a type
- Derived type inherits operations
  - can override and augment

```
procedure check_op ( instr : instruction;  . . . );
procedure perform_op ( instr : instruction );

procedure perform_op ( instr : ALU_instruction );
```


Class-Wide Types

- instruction'Class denotes hierarchy of types derived from instruction
- Polymorphism, dynamic dispatching

```
signal current_instr : instruction'Class;
. . .
perform_op ( current_instr );
```

Encapsulation: Private Parts

Encapsulation: Private Types

```
package test_queues is
    use work.tests.all;
    type queue is private;
    function is_empty ( Q : queue ) return boolean;
    . . .
private
    constant max_size : positive := 100;
    type queue_array is array (0 to max_size – 1) of test;
    type queue is record
        head, tail : natural range 0 to max_size – 1;
        size : natural range 0 to max_size;
        the_buffer : queue_array;
    end record queue;
end package test_queues;
```

Generic ADTs

```
package queues is
    generic ( max_size : positive;  type element is private );
    type queue is private;
    function is_empty ( Q : queue ) return boolean;
    . . .
private
    type queue_array is array (0 to max_size – 1) of element;
    type queue is record
        . . .
    end record queue;
end package queues;
. . .
package test_queues is new queues
    generic map ( max_size => 100,  element => work.tests.test );
```


Generic Entities

```
entity multiplexer is
    generic ( Tpd : delay_length;
              type data is private );
    port (  sel : in bit;
            in0, in1 : in data;
            z : out data );
end entity multiplexer;
```

Generic Entities

```
entity multiplexer is
    generic ( Tpd : delay_length;
              type data is private;
              type selector is (<>);
              choose_0 : selector );
    port (  sel : in selector;
            in0, in1 : in data;
            z : out data );
end entity multiplexer;
```


Generic Entities

```
architecture behavioral of multiplexer is
begin
    mux_behav : process is
    begin
        case sel is
            when choose_0 =>
                z <= in0 after Tpd;
            when others =>
                z <= in1 after Tpd;
        end case;
        wait on sel, in0, in1;
    end process mux_behav;
end architecture behavioral;
```

Conclusions

- VHDL needs improvement for high-level modeling
  - enhance abstraction, encapsulation
  - add inheritance, genericity
- Improvement across modeling domain
  - maintain support for synthesis
- Coherent integration with existing language
- OO is part of the solution, not a panacea