Packages and Aliases

Prof. K. J. Hintz

Department of Electrical
and
Computer Engineering

George Mason University
Packages

- Method for Grouping Related Declarations Which Serve a Common Purpose
  - Set of subprograms to operate on particular data type
  - Set of declarations for particular model
  - Separate interface from implementation
  - Reusable
Packages

- Unclutter rest of model
- Allows declaration of “global” signals, *e.g.*, clocks.
  » Not a generally good since behavior can change through means other than signals declared in entity interface
Packages

■ Design Unit Similar to Entity Declarations and Architecture Bodies
  – Can be put in library and made accessible to other units
  – Access to items declared in the package is through using its *Selected Name*
    `library_name.package_name.item_name`
  – Aliases can be used to allow shorter names for accessing declared items
Packages

- Two Components to Packages
  - Package declaration
  - Package body
Package Declaration

- Subprograms Using Header, Implementation Is Hidden
  - “information hiding”

- Constants, Do Not Need to Be Initialized in Declaration
  - “information hiding”
Package Declaration

- Types, Must Be Completely Specified
  - Can have variable size arrays

- Signals Must Be Completely Specified
Package Declaration Syntax

\[
\text{package identifier is}
\{
\text{package_declarative_item}
\}
\text{end [ package ] [ identifier ] ;}
\]
package dp32_types is
  constant unit_delay : Time := 1 ns;
  type bool_to_bit_table is
    array ( boolean ) of bit;
  . . .
function bits_to_int
  ( bits : in bit_vector ) return integer ;
function bits_to_natural
  ( bits : in bit_vector ) return natural ;
procedure int_to_bits
  ( int : in integer ;
    bits : out bit_vector ) ;
end dp32_types ;

*Ashenden VHDL cookbook
Package Body

- Not Necessary If Package Declaration Does Not Declare Subprograms
- May Contain Additional Declarations Which Are Local to the Package Body
  – Cannot declare signals in body
Package Body

- Declared Subprograms Must Include the Full Declaration As Used in Package Declaration
  - Numeric literals can be written differently if same value
  - Simple name may be replaced by a selected name provided it refers to same item
package body identifier is
   { package_body_declarative_item } end [ package body ] [ identifier ] ;
package body dp32_types is

constant bool_to_bit :
  bool_to_bit_table := ( false => '0',
                        true => '1' );

function resolve_bit_32
  ( driver : in bit_32_array ) return bit_32 is

  constant float_value : bit_32 := x"0000_0000" ;
  variable result : bit_32 := float_value ;

*Ashenden VHDL cookbook
begin
  for i in driver'range loop
    result := result or driver ( i ) ;
  end loop ;
  return result ;
end resolve_bit_32 ;

*Ashenden VHDL cookbook
Library Clause

- Makes Items in a Library Available to a VHDL Model
- To Access Items in a Library Need to Use Their `selected_name`

```
library identifier { , . . . } ;
```
Use Clause

- Tedious to Always Use an Item’s Selected Name

- All Items Declared in a Package or Library Can Be Made “Visible” Through a Use Clause
Use Clause

- Can Be Used in Any Declarative Section

- Keyword “All” Imports All Identifiers
Use Clause Syntax

\[
\text{use } \text{selected_name} \{ , \ldots \} \\

\text{selected_name } \leq \text{name} . ( \text{identifier} \\
| \text{character_literal} \\
| \text{operator_symbol} \\
| \text{all} )
\]
Use Clause Example*

```
use work.dp32_types.all;
entity dp32 is
  generic ( Tpd : Time := unit_delay );
  port ( d_bus : inout bus_bit_32 bus;
         a_bus : out bit_32;
         read, write, fetch : out bit;
         ready, phi1, phi2, reset : in bit );
end dp32;
```

*Ashenden VHDL cookbook
Aliases

Alternative Identifier for an Item

Improves Readability

Allows One to Differentiate Among Identically Named Items in Different Packages

Can Refer to a Single Element or Part of a Composite Data Type, *e.g.*, 

```vhdl
alias interrupt_level is PSW(30 downto 26);
```
Aliases

- Operations on Aliases Operate on Actual Items Except for the Following Attributes
  - `x'simple_name`
  - `x'path_name`
  - `x'instance_name`
Aliases

- Cannot Declare Aliases for
  - Labels
  - Loop parameters
  - Generate parameters (replicates items)
Data Alias Syntax

```
alias identifier
[ : subtype_indication ] is name ;
```
Data Alias

- Subtype_indication Allows for the Type to Be Changed
  - If scalar original
    » Direction cannot change
    » Bounds cannot change
    » Unconstrained type allowed
Data Alias

- Subtype_indication Allows for the Type to Be Changed
  - If array or array slice
    » Direction can differ
    » Bounds can differ
    » Base type must remain unchanged
Non-Data Alias Syntax

```
alias ( identifier
    | character_literal
    | operator_symbol )

is name [ signature ] ;
```
Non-Data Alias

- Alias for Enumeration Type Does Not Require Definition of Aliases for Enumeration Literals of Original

- Alias for Physical Type Retains Units Without Redefinition
Non-Data Alias Syntax

- Optional Signature
  - Only for subprograms and enumeration literals
  - Overloading of identifiers may require means of differentiating among alternatives
    » return type does this
  - Outer [ ] are required
signature <=
[ [ type_mark { , . . . } ] [ return type_mark ] ]
- e.g.,

alias high is std.standard.'1' [ return bit ]
Resolved Signals

- VHDL Requires a Function to Specify the Values Which Result From Tying Multiple Outputs Together
- Resolved Signal Includes Resolution Function
  - Inclusion of function indicates it is a resolved signal
Resolved Signals

- Resolution Function Must Be Written for an Indeterminate Number of Signals Since It Is Not Known When Declared How Many Signals Will Be Connected to It.
- The Value of a Signal at a Transaction Is Determined by the Resolution Function Operating on the Multiply Connected Signals.
Resolved Signal Syntax

\[
\text{subtype} \_\text{indication} \leq \\
[ \text{resolution}_{\text{function}\_\text{name}} ] \\
\text{type}\_\text{mark} [ \text{range} \\
( \text{range}_{\text{attribute}\_\text{name}} \\
| \text{simple}\_\text{expression} ( \text{to} | \text{downto} ) \\
\text{simple}\_\text{expression} ) \\
| ( \text{discrete}\_\text{range} \{ , \ldots \} ) ] ;
\]
Resolved Signal Example*

package MVL4 is

type MVL4_ulogic is ( 'X', '0', '1', 'Z' );

type MVL4_ulogic_vector is array
    (natural range <> ) of MVL4_ulogic ;

function resolve_MVL4
    (contribution : MVL4_ulogic_vector )

return MVL4_ulogic ;

*Ashenden
Resolved Signal Example*

```vhd
subtype MVL4_logic is
    resolve_MVL4 MVL4_ulogic;
end package MVL4;
```

*Ashenden*
Resolved Signal Example*

package body MVL4 is
    type table is array
        ( MVL4_ulogic ,
        MVL4_ulogic )
    of MVL4_ulogic ;
Resolved Signal Example*

```verbatim
class constant resolution_table : table :=
  -- 'X' '0' '1' 'Z'
  -- -------------------------
  ( 'X' 'X' 'X' 'X' ),       -- 'X'
  ( 'X' '0' 'X' '0' ),       -- '0'
  ( 'X' 'X' '1' '1' ),       -- '1'
  ( 'X' '0' '1' 'Z' ) ;      -- 'Z'
```

*Copyright © 1997/8/9, KJH, 545_10, 5/8/2001*
Resolved Signal Example*

```plaintext
function resolve_MVL4
  ( contribution : MVL4_ulogic_vector )
return MVL4_ulogic is
variable result : MVL4_ulogic := 'Z';
```
Resolved Signal Example*

begin
    for index in contribution’range loop
        result := resolution_table
            ( result, contribution ( index ) );
    end loop ;
    return result ;
end function resolve_MVL4 ;
end package body MVL4 ;
End of Lecture