5. Model Organisation

The previous chapters have described the various facilities of VHDL somewhat in isolation. The purpose of this chapter is to show how they are all tied together to form a complete VHDL description of a digital system.

5.1. Design Units and Libraries

When you write VHDL descriptions, you write them in a *design file*, then invoke a compiler to analyse them and insert them into a *design library*. A number of VHDL constructs may be separately analysed for inclusion in a design library. These constructs are called *library units*. The *primary* library units are entity declarations, package declarations and configuration declarations (see Section 5.2). The *secondary* library units are architecture bodies and package bodies. These library units depend on the specification of their interface in a corresponding primary library unit, so the primary unit must be analysed before any corresponding secondary unit.

A design file may contain a number of library units. The structure of a design file can be specified by the syntax:

```
design_file ::= design_unit { design_unit }
design_unit ::= context_clause library_unit
context_clause ::= { context_item }
context_item ::= library_clause | use_clause
library_clause ::= library logical_name_list ;
logical_name_list ::= logical_name { , logical_name }
library_unit ::= primary_unit | secondary_unit
primary_unit ::=
    entity_declaration | configuration_declaration | package_declaration
secondary_unit ::= architecture_body | package_body
```

Libraries are referred to using identifiers called logical names. This name must be translated by the host operating system into an implementation dependent storage name. For example, design libraries may be implemented as database files, and the logical name might be used to determine the database file name. Library units in a given library can be referred to by prefixing their name with the library logical name. So for example, ttl_lib.ttl_10 would refer to the unit ttl_10 in library ttl_lib.

The context clause preceding each library unit specifies which other libraries it references and which packages it uses. The scope of the names made visible by the context clause extends until the end of the design unit.

There are two special libraries which are implicitly available to all design units, and so do not need to be named in a library clause. The first of these is called work, and refers to the working design library into which the

current design units will be placed by the analyser. Hence in a design unit, the previously analysed design units in a design file can be referred to using the library name work.

The second special libary is called std, and contains the packages standard and textio. Standard contains all of the predefined types and functions. All of the items in this package are implicitly visible, so no use clause is necessary to access them.

5.2. Configurations

In Sections 3.2.3 and 3.2.4 we showed how a structural description can declare a component specification and create instances of components. We mentioned that a component declared can be thought of as a template for a design entity. The binding of an entity to this template is achieved through a configuration declaration. This declaration can also be used to specify actual generic constants for components and blocks. So the configuration declaration plays a pivotal role in organising a design description in preparation for simulation or other processing.

The syntax of a configuration declaration is:

```
configuration_declaration ::=
    configuration identifier of entity_name is
        configuration_declarative_part
        block_configuration
    end [ configuration_simple_name ] ;

configuration_declarative_part ::= { configuration_declarative_item }

configuration_declarative_item ::= use_clause

block_configuration ::=
    for block_specification
        { use_clause }
        { configuration_item }
    end for ;

block_specification ::= architecture_name | block_statement_label

configuration_item ::= block_configuration | component_configuration

component_configuration ::=
    for component_specification
        [ use binding_indication ; ]
        [ block_configuration ]
    end for ;

component_specification ::= instantiation_list : component_name

instantiation_list ::=
    instantiation_label { , instantiation_label }
    | others
    | all

binding_indication ::=
    entity_aspect
    [ generic_map_aspect ]
    [ port_map_aspect ]

entity_aspect ::=
    entity entity_name [ ( architecture_identifier ) ]
    | configuration configuration_name
    | open

generic_map_aspect ::= generic map ( generic_association_list )
```

```
entity processor is
    generic (max_clock_speed : frequency := 30 MHz);
    port ( port list );
end processor;

architecture block_structure of processor is

    declarations

begin

    control_unit : block
        port ( port list );
        port map ( association list );
        declarations for control_unit
    begin
        statements for control_unit
    end block control_unit;

    data_path : block
        port ( port list );
        port map ( association list );
        declarations for data_path
    begin
        statements for data_path
    end block data_path;

end block_structure;
```

Figure 5-1. Example processor entity and architecture body.

port_map_aspect ::= **port map** (*port*_association_list)

The declarative part of the configuration declaration allows the configuration to use items from libraries and packages. The outermost block configuration in the configuration declaration defines the configuration for an architecture of the named entity. For example, in Chapter 3 we had an example of a processor entity and architecture, outlined again in Figure 5-1. The overall structure of a configuration declaration for this architecture might be:

```
configuration test_config of processor is

    use work.processor_types.all

    for block_structure
        configuration items
    end for;

end test_config;
```

In this example, the contents of a package called processor_types in the current working library are made visible, and the block configuration refers to the architecture block_structure of the entity processor.

Within the block configuration for the architecture, the submodules of the architecture may be configured. These submodules include blocks and component instances. A block is configured with a nested block configuration. For example, the blocks in the above architecture can be configured as shown in Figure 5-2.

Where a submodule is an instance of a component, a component configuration is used to bind an entity to the component instance. To illustrate, suppose the data_path block in the above example contained an

```
configuration test_config of processor is
    use work.processor_types.all
    for block_structure
        for control_unit
            configuration items
        end for;
        for data_path
            configuration items
        end for;
    end for;
end test_config;
```

Figure 5-2. Configuration of processor example.

```
    data_path : block
        port ( port list );
        port map ( association list );
        component alu
            port (function : in alu_function;
                  op1, op2 : in bit_vector_32;
                  result : out bit_vector_32);
        end component;
        other declarations for data_path
    begin
        data_alu : alu
            port map (function => alu_fn, op1 => b1, op2 => b2, result => alu_r);
        other statements for data_path
    end block data_path;
```

Figure 5-3. Structure of processor data-path block.

instance of the component alu, declared as shown in Figure 5-3. Suppose also that a library project_cells contains an entity called alu_cell defined as:

```
entity alu_cell is
    generic (width : positive);
    port (function_code : in alu_function;
          operand1, operand2 : in bit_vector(width-1 downto 0);
          result : out bit_vector(width-1 downto 0);
          flags : out alu_flags);
end alu_cell;
```

with an architecture called behaviour. This entity matches the alu component template, since its operand and result ports can be constrained to match those of the component, and the flags port can be left unconnected. A block configuration for data_path could be specified as shown in Figure 5-4.

Alternatively, if the library also contained a configuration called alu_struct for an architecture structure of the entity alu_cell, then the block configuration could use this, as shown in Figure 5-5.

```
for data_path
    for data_alu : alu
        use entity project_cells.alu_cell(behaviour)
            generic map (width => 32)
            port map (function_code => function, operand1 => op1, operand2 => op2,
                      result => result, flags => open);
    end for;
    other configuration items
end for;
```

Figure 5-4. Block configuration using library entity.

```
for data_path
    for data_alu : alu
        use configuration project_cells.alu_struct
            generic map (width => 32)
            port map (function_code => function, operand1 => op1, operand2 => op2,
                      result => result, flags => open);
    end for;
    other configuration items
end for;
```

Figure 5-5. Block configuration using another configuration.

5.3. Complete Design Example

To illustrate the overall structure of a design description, a complete design file for the example in Section 1.4 is shown in Figure 5-6. The design file contains a number of design units which are analysed in order. The first design unit is the entity declaration of count2. Following it are two secondary units, architectures of the count2 entity. These must follow the entity declaration, as they are dependent on it. Next is another entity declaration, this being a test bench for the counter. It is followed by a secondary unit dependent on it, a structural description of the test bench. Following this is a configuration declaration for the test bench. It refers to the previously defined library units in the working library, so no library clause is needed. Notice that the count2 entity is referred to in the configuration as work.count2, using the library name. Lastly, there is a configuration declaration for the test bench using the structural architecture of count2. It uses two library units from a separate reference library, misc. Hence a library clause is included before the configuration declaration. The library units from this library are referred to in the configuration as misc.t_flipflop and misc.inverter.

This design description includes all of the design units in one file. It is equally possible to separate them into a number of files, with the opposite extreme being one design unit per file. If multiple files are used, you need to take care that you compile the files in the correct order, and re-compile dependent files if changes are made to one design unit. Source code control systems can be of use in automating this process.

```
-- primary unit: entity declaration of count2

entity count2 is
    generic (prop_delay : Time := 10 ns);
    port (clock : in bit;
          q1, q0 : out bit);
end count2;

-- secondary unit: a behavioural architecture body of count2

architecture behaviour of count2 is

begin

    count_up: process (clock)

        variable count_value : natural := 0;

    begin
        if clock = '1' then
            count_value := (count_value + 1) mod 4;
            q0 <= bit'val(count_value mod 2) after prop_delay;
            q1 <= bit'val(count_value / 2) after prop_delay;
        end if;
    end process count_up;

end behaviour;

-- secondary unit: a structural architecture body of count2

architecture structure of count2 is

component t_flipflop
        port (ck : in bit; q : out bit);
    end component;

    component inverter
        port (a : in bit; y : out bit);
    end component;

    signal ff0, ff1, inv_ff0 : bit;

begin

    bit_0 : t_flipflop port map (ck => clock, q => ff0);

    inv : inverter port map (a => ff0, y => inv_ff0);

    bit_1 : t_flipflop port map (ck => inv_ff0, q => ff1);

    q0 <= ff0;
    q1 <= ff1;

end structure;
```

Figure 5-6. Complete design file.

```
-- primary unit: entity declaration of test bench

entity test_count2 is
end test_count2;

-- secondary unit: structural architecture body of test bench

architecture structure of test_count2 is

    signal clock, q0, q1 : bit;

    component count2
        port (clock : in bit;
                q1, q0 : out bit);
    end component;

begin

    counter : count2
        port map (clock => clock, q0 => q0, q1 => q1);

    clock_driver : process
    begin
        clock <= '0', '1' after 50 ns;
        wait for 100 ns;
    end process clock_driver;

end structure;


-- primary unit: configuration using behavioural architecture

configuration test_count2_behaviour of test_count2 is

    for structure     -- of test_count2
        for counter : count2
            use entity work.count2(behaviour);
        end for;
    end for;

end test_count2_behaviour;


-- primary unit: configuration using structural architecture

library misc;

configuration test_count2_structure of test_count2 is

    for structure     -- of test_count2
        for counter : count2
            use entity work.count2(structure);
            for structure     -- of count_2
                for all : t_flipflop
                    use entity misc.t_flipflop(behaviour);
                end for;
                for all : inverter
                    use entity misc.inverter(behaviour);
                end for;
            end for;
        end for;
    end for;

end test_count2_structure;
```

Figure 5-6 (continued).