This appendix serves as a quick reference guide to the structure of a first VHDL model. A template for a general VHDL model is presented. This template can help with the syntactical arrangement of programming constructs. It is useful when trying to remember where to place statements within a program relative to other program constructs. This template can serve as a handy reference for quickly constructing our first VHDL models. The goal here is to provide a template that contains the most basic and common (and therefore, for our purposes, important) language features and will enable the reader to rapidly proceed to the construction of useful VHDL models.

7.1 A Simulation Template

We can combine the procedures for constructing behavioral and structural simulation models that are described in the early chapters and identify a sequence of common operations. The first step is the construction of a schematic of the system being modeled.
Construct_Schematic

1. Represent each component (e.g., gate) of the system to be modeled as a delay element. The delay element simply captures all of the delays associated with the computation represented by the component and propagation of signals through the component. For each output signal of a component associate a specific value of delay through the component for that output signal.
2. Draw a schematic interconnecting all of the components. Uniquely label each component.
3. Identify the input signals of the system as input ports.
4. Identify the output signals of the system as output ports.
5. All remaining signals are internal signals and should be uniquely labeled.
6. Associate a type, such as bit, bit_vector, or std_logic_vector, with each input port, output port, and internal signal.
7. Ensure that each input port, output port, and internal signal is labeled with a unique name.

This schematic can now be translated into a VHDL model containing behavioral and structural models of the components that comprise the system. In fact, the architecture body shown in Figure G-1 can be structured as a series of program statements. Each statement can be one of the following:

1. A concurrent signal assignment statement
   - simple signal assignment
   - conditional signal assignment
   - selected signal assignment
2. A process
   The process may have a sensitivity list, and may comprise a large block of sequential code. Recall that a process execution takes no simulation time and may produce events on signals that are scheduled at some time in the future.
3. A component instantiation statement
   If components have been declared in addition to signals, these components may be instantiated and their input and output ports mapped to signals declared in the architecture. In this manner, these components can be "connected" to, or communicate with, other components, CSAs, or processes.
   This leads to the following procedure for constructing general models reflecting the behavior of the digital system.

Construct_Behavioral_Model

1. At this point I recommend using the IEEE 1164 value system. To do so, include the following two lines at the top of your model declaration.

   ```vhdl
   library IEEE;
   use IEEE.std_logic_1164.all;
   ```

   Single-bit signals can be declared to be of type std_logic while multibit quantities can be declared to be of type std_logic_vector.
2. Select a name for the entity (entity_name) for the system and write the entity description specifying each input or output signal port, its mode, and associated type.
3. Select a name for the architecture (arch_name) and write the architecture description. Within the architecture description, name and declare all of the internal signals used to connect the components. These signal names are shown on your schematic. The architecture declaration states the type of each signal and possibly an initial value.
4. For each delay element decide if the behavior of the block will be described by concurrent signal assignment statements, processes, or a component instantiation statement. Depending upon the type, perform the following:
   4.1 CSA: For each output signal of the component select a concurrent signal assignment statement that expresses the value of this signal as a function of the signals that are inputs to that component. Use the value of the propagation delay through the component provided for that output signal. The output signal and/or one or more input signals may be a port of the entity.
   4.2 Process: Alternatively, if the computation of the signal values at the outputs of the component are too complex to represent with concurrent signal assignment statements, describe the behavior of the component with a process. One or more processes can be used to compute the values of the output signals from that component. For each process perform the following:
      4.2.1 Label the process. If you are using a sensitivity list, identify the signals that will activate the process.
      4.2.2 Declare variables used within the process.
      4.2.3 Write the body of the process computing the values of output signals and the relative time at which these output signals assume these values. If a sensitivity list is not used, specify wait statements at appropriate points in the process to specify when the process should suspend and when it should resume execution. It is an error to have both a sensitivity list and a wait statement within the process.
      4.2.4 Complete the process with a set of signal assignment statements, assigning the computed values to the output signals. These output signals may be signals internal to the architecture or may be port signals found in the entity description.
   4.3 Component Instantiation: For those components for which entity–architecture pairs exist.
4.3.1 Construct component declarations for each unique component that will be used in the model. A component declaration can be easily constructed from the component’s entity description. For example, the port list is identical.

4.3.2 Within the declarative region of the architecture description (i.e., before the begin statement), list the component declarations.

4.3.3 Within the declarative region of the architecture description (i.e., before the begin statement), list the configuration specification if not using the default binding for the component entities.

4.3.4 Write the component instantiation statement. The label is derived from the schematic followed by the port map construct. The port map statement will have as many entries as there are ports on the component. If necessary, include a generic map statement.

5. If there are signals that are driven by more than one source, the type of this signal must be a resolved type. This type must have a resolution function declared for use with signals of this type. For our purposes use the IEEE 1164 types std_logic for single-bit signals and std_logic_vector for bytes, words, or multibit quantities. These are resolved types. Make sure you include the library clause and the use clause to include all of the definitions provided in the std_logic_1164 package.

6. If you are using any functions or type definitions provided by a third party make sure that you have declared the appropriate library using the library clause and declared the use of this package via the presence of a use clause in your model.

These steps will produce a fairly generic model. In particular, this approach implies that all design units (entity, architecture, and configuration information) is placed in one physical file. This is clearly not necessary. For example, we know from Chapter 8 that configurations are distinct design units that may be described separately. However, it is often easier to start in the fashion shown here. As our expertise grows, we will be able to avail ourselves of the advantages of dealing with design units separately and managing them effectively. Finally, note that in Figure G-1 the location of the packages is shown as library IEEE. Depending on the packages, this may not be the case, and when writing models we must have knowledge of the location of any vendor-supplied packages that are being used. We may also be creating our own libraries for retaining user packages.

7.2 A Synthesis Template

The construction of a synthesis template largely mirrors that of the simulation template with a few important exceptions.

1. Do not specify any delays within the design. They will be generally be ignored.

2. Follow restrictions required of synthesis compilers. For example, generally only one wait statement is permitted in a process. Check with hints listed in Appendix A.

FIGURE G-4 Anatomy of a VHDL model
3. Synthesis compilers may not support configurations.

4. Support for resolved types may differ across synthesis compilers. Use the IEEE 1164 value system and associated packages if at all possible.

With these exceptions the template shown in Figure G-1 can also be used as a starting point for synthesis templates.