The Optimization of a Design using VHDL Concepts

Iuliana CHIUCHISAN
Alin Dan POTORAC
"Stefan cel Mare" University of Suceava
str.Universitatii nr.13, RO-720229 Suceava
iuliap@eed.usv.ro
alinp@eed.usv.ro

Abstract — The primary purpose of this paper is to study the field of Hardware Description Languages such as VHDL. The study is significant for several reasons. First, the utilization of Hardware Description Languages in real life engineering applications will become more conventional. Second, the study is significant due to the major implication that programmable logic based microcontrollers can be upgraded as the requirements of a system increase as shown in the case of the counter. Third, it is demonstrated how the utilization of VHDL benefits not only engineering applications, but also plays an important role accelerating the design of digital systems. VHDL were employed to describe the models for a different sized counter. The internal view of the device specified the functionality of the counter using the concept of architecture, while the external view specified the interface of the device through which it communicated with the other models in its environment.

Index Terms—VHDL, Very high speed integrated circuit, Very-large-scale integration, counter, entity, architecture.

I. INTRODUCTION

The growing sophistication of applications continually pushes the design and manufacturing of integrated circuits to new levels of complexity. Due to major advances in the development of electronics and miniaturization, vendors are capable of building and designing products with increasingly greater functionality, higher performance, lower cost, lower power consumption, and smaller dimensions [2].

The electronics industry requires systems to be capable of in-site reprogramming, where the upgrading task depends more on software than on hardware. This situation has fostered the need for adoption of modern technologies in design and testing. There are now two industry standard hardware description languages, VHDL and Verilog. The complexity of ASIC and FPGA designs has meant an increase in the number of specialist design consultants with specific tools and with their own libraries of macro and mega cells written in either VHDL or Verilog [5].

Of the several existing methodologies, high-density Programmable Logic Devices (PLDs) as well as the Very High Speed Integrated Circuits (VHSIC) Hardware Description Language (VHDL) and Verilog Hardware Description Language are key elements in the evolution of electronic devices. It was demonstrated how the utilization of VHDL generates benefits not only for engineering applications, but also playing an important role accelerating the design of digital systems [2].

VHDL usage has risen rapidly since its inception and is used by literally tens of thousands of engineers around the globe to create sophisticated electronic products. VHDL is a powerful language with numerous language constructs that are capable of describing very complex behavior. Learning all the features of VHDL are not at all a simple task and the designer abilities and experiences still remain an important issue [1].

The article is illustrating the implementation and simulation of a VHDL logic design based on behavioral functional description. A reusable HDL code for a presetable up/down 4-bit counter is generated and described in order to demonstrate the principle. Simulation is used to validate the procedure.

II. SHORT OVERVIEW ON VHDL LANGUAGE

The VHSIC Hardware Description Language is an industry standard language used to describe hardware from the abstract to the concrete level. The concept of a Hardware Description Language was born from the necessity of bringing the worlds of hardware and software back together again. Vendors wanted the design descriptions to be computer readable and executable. This was followed by the arrival of Very High Speed Integrated Circuits (VHSIC) Hardware Description Language (VHDL) [2]. Its roots are in the ADA language, as will be seen by the overall structure of VHDL as well as other VHDL statements.

VHDL is a hardware description language employed to model a digital system or digital hardware device at many levels of abstraction, ranging from the algorithmic level down to the gate level. The complexity of the digital system being modeled could vary from that of a simple gate to a complete digital electronic system, or anything in between. The digital system can also be described hierarchically [2].

The VHDL language can also be described as a combination of languages as: sequential language, concurrent language, netlist language, waveform generation language and timing specifications.

Therefore, VHDL has constructs that enable the user to express the concurrent or sequential behavior of a digital system with or without timing characteristics. It also allows the modeling of systems as an interconnection of components. Test waveforms can also be generated using the same constructs. All the above constructs may also be combined to provide a comprehensive description of the system in a single model [3].

III. TRANSLATION OF A CIRCUIT INTO A VHDL CODE

VHDL describes the behavior of an electronic circuit or system, from which the physical circuit or system can then be implemented [4].

The basic building blocks of VHDL design are the *entity declaration* and the *architecture body*. A VHDL entity specifies the name of the entity, the ports of the entity, and entity-related information.

The next example is a design of a 4-bit loadable up/down counter. A graphical schematic for a 4-bit counter is depicted in Figure 1.

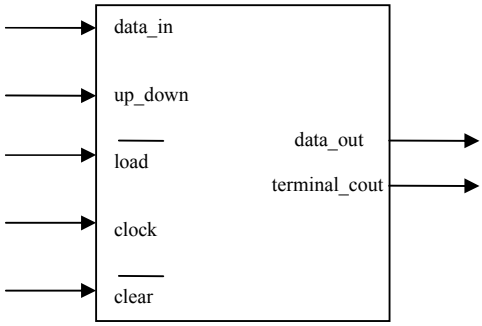

Figure 1. A 4-bit loadable up/down counter

The entity describes a component's connections to the rest of the design. It specifies the number of ports, the direction of the ports, and the type of the ports.

The entity *counter* contains a *clock* input port to clock the counter, a *load* input port that allows the counter to be synchronously loaded, a *clear* input port that synchronously clears the counter, a *up-down* input port that sets the counter to a up count or a down count, a *data_in* input port that allows values to be loaded into the counter's cells, a output port *terminal_count* that detects the end of the counter and a output port *data_out* that presents the current value of the counter to the outside world.

```
ENTITY counter_4bit IS
    PORT(
      data_in: IN std_logic_vector(3
      downto 0);
      clock: IN std_logic;
      load: IN std_logic;
      clear: IN std_logic;
      up_down: IN std_logic;
      terminal_count: OUT   std_logic;
      data_out: OUT std_logic_vector (3
      downto 0));
END counter_4bit;
```

VHDL allows the user to write the designs using various styles of architecture. The architecture can contain any combination of behavioral, structural or dataflow styles to define an entity's function. These styles allow programmers to describe a design at different levels of abstraction, from using algorithms to gate level primitives [2].

The architecture describes the underlying functionality of the entity and contains the statements that model the behavior of the entity. The architecture is always related to an entity and describes the behavior of that entity.

The architecture for the 4-bit loadable up/down counter device described earlier would look like this:

```
ARCHITECTURE counter_4bit_arh OF
counter_4bit IS
SIGNAL
count:std_logic_vector(3 downto 0)
      :="0000";
BEGIN
PROCESS (clock) BEGIN
    IF (clear = '0') THEN
       count <= "0000";
    ELSIF(load = '0') THEN
       count <= data_in;
    ELSE
    IF (clock'EVENT AND clock = '0')
AND(clock'LAST_VALUE = '1') THEN
    IF(up_down = '1') THEN
       count <= count + 1;
    END IF;
    IF(up_down = '0') THEN
       count <= count - 1;
    END IF;
END IF;
END IF;
    IF (count = "1111") THEN
       terminal_count <= '1';
    ELSE
       terminal_count <= '0';
    END IF;
data_out <= count;
END PROCESS;
END counter_4bit_arh;
```

The reason for the connection between the architecture and the entity is that an entity can have multiple architectures describing the behavior of the entity.

If the designer wants to use a different architecture that has another description he can use or reuse the VHDL configurations. Configurations are a primary design unit used to bind component instances to entities. For structural models, configurations can be thought of as the parts list for the model. For component instances, the configuration specifies for an entity which architecture to be used for a specific instance.

The configuration can also be used to provide a very fast substitution capability. Multiple architectures can exist for a single entity. One architecture might be a behavioral model for the entity, while another architecture might be a structural model for the entity. The architecture used in the description can be selected by specifying which architecture to be included into the configuration, by just recompiling it. After compilation, the simulated model uses the specified architecture.

The default configuration specifies the configuration name, the entity being configured and the architecture to be used for the entity.

The two architectures of the entity *counter* specify two different-sized counters that can be used for the entity. The first architecture specifies an 8-bit up/down counter. The second architecture specifies a 16-bit up/down counter. The architectures specify a synchronous counter with synchronous *load* and *clear* inputs. All operations for the device occur with respect to the *clock*.

```
ENTITY counter IS
    PORT(
      data_in : IN INTEGER;
      clock : IN std_logic;
      clear : IN std_logic;
      load : IN std_logic;
      up_down : IN std_logic;
      terminal_count: OUT std_logic;
      data_out : OUT INTEGER);
END counter;

ARCHITECTURE counter_8bit_arh OF
counter IS
BEGIN
PROCESS(clock)
    VARIABLE count : INTEGER := 0;
    BEGIN
    IF (clear = '0') THEN
       count := 0;
    ELSIF (load = '0') THEN
       count := data_in;
    ELSE
    IF (clock'EVENT AND clock = '1')
AND(clock'LAST_VALUE = '0') THEN
    IF (up_down = '1') THEN
    IF (count = 255)THEN
       count:=0;
    ELSE
       count := count + 1;
    END IF;   END IF;
    IF(up_down = '0')THEN
    IF (count = 0)THEN
       count:=255;
    ELSE
       count := count - 1;
    END IF;   END IF;
    END IF;   END IF;
    IF (count = 255) THEN
       terminal_count <= '1';
    ELSE
       terminal_count <= '0';
    END IF;
data_out <= count;
END PROCESS;
END counter_8bit_arh;
ARCHITECTURE counter_64k_arh OF
counter IS
BEGIN
PROCESS(clock)
VARIABLE count : INTEGER := 0;
BEGIN
    IF (clear = '0') THEN
       count := 0;
    ELSIF (load = '0') THEN
       count := data_in;
    ELSE
    IF (clock'EVENT AND clock = '1')
AND(clock'LAST_VALUE = '0') THEN
    IF (up_down = '1') THEN
    IF (count = 65535)THEN
       count:=0;
    ELSE
       count := count + 1;
    END IF;   END IF;
    IF(up_down = '0')THEN
    IF (count = 0)THEN
       count:=65535;
    ELSE
       count := count - 1;
    END IF;   END IF;
    END IF;   END IF;
    IF (count = 65535) THEN
       terminal_count <= '1';
    ELSE
       terminal_count <= '0';
    END IF;
data_out <= count;
END PROCESS;
END counter_64k_arh;
```

Each of the two configurations above specifies a different architecture for the entity *counter*. Below is an example of two configurations illustrated as *small_counter* and *big_counter*:

```
CONFIGURATION small_counter OF counter
IS
      FOR counter_8bit_arh
      END FOR;
END small_counter;
CONFIGURATION big_counter OF counter IS
      FOR counter_64k_arh
      END FOR;
END big_counter;
```

This example shows how two different architectures for the same counter entity can be configured using two default configurations. The entity for the counter does not specify any bit word width for the data to be loaded into the counter or output data generated by the counter. The data type for the input and output data is an integer. With a data type of integer, multiple types of counters can be supported up to the integer representation limit of the computer hosting the VHDL simulator.

The next description example contains a package that defines an 8-bit binary word range that causes the synthesis tools to generate an 8-bit counter. Changing the size of the range causes the synthesis tools to generate

different-sized counters.

```
PACKAGE count_types IS
SUBTYPE bit8 is INTEGER RANGE 0 to 255;
END count_types;

ENTITY counter IS
    PORT(
      data_in: IN bit8;
      clock: IN std_logic;
      load: IN std_logic;
      clear: IN std_logic;
      up_down: IN std_logic;
      terminal_count: OUT   std_logic;
      data_out: OUT bit8);
END counter;
```

A. The test-bench

A simulator needs two inputs: the description of the design as basics and stimulus to drive the simulation. Sometimes designs are self-stimulating and do not need any external stimulus, but in most cases, VHDL designers use a VHDL test-bench of one kind or another to drive the design being tested. The top-level design description instantiates two components: the first being the design under test (DUT) and the second the stimulus driver. These components are connected with signals that represent the external environment of the DUT. The top level of the design does not contain any external ports, just internal signals that connect the two instantiated components. When the designer makes a small change to fix an error, the change can be tested to make sure that it did not affect other parts of the design [1].

The test-bench encapsulates the stimulus driver, known good results and DUT and includes internal signals to make the proper connections. The stimulus values drives inputs into the DUT. The DUT responds to the input signals and produces output results. Finally, a compare function within the test-bench compares the results from the DUT against those known good results and reports any discrepancies. That is the basic function of a test-bench, but there are a number of methods of writing a test-bench and each method has advantages and disadvantages [1].

The following are the most common test-bench types:

- Stimulus only — contains only the stimulus driver and DUT; does not contain any results verification.
- Full test-bench — Contains stimulus driver, known good results, and results comparison.
- Simulator specific — Test-bench is written in a simulator-specific format.
- Hybrid test-bench — Combines techniques from more than one test-bench style.
- Fast test-bench — Test-bench written to get ultimate speed from simulation.

The advantages and disadvantages of each kind of test-bench type are shown in Table 1.

Table 1.

	Speed	**Flexibility**	**Portability**
Stimulus only	Slow	High	High
Full testbench	Slow	High	High
Simulator specific	Medium	High	Low
Hybrid testbench	Medium	Medium	High
Fast testbench	Extremely fast	Low	High

B. VHDL Simulation

The VHDL description of the counter is simulated with a standard VHDL simulator to verify that the description is correct.

We decide to use a fast test-bench that is optimized for speed and typically does not limit the speed of the simulation. The fast test-bench looks similar to the other test-bench styles consisting in a top-level entity that instantiates a DUT and a process that generates the stimulus. What's different is the fact that instead of reading the stimulus vectors from a file, the vectors are compiled into the test-bench model.

The advantages of the fast test-bench are related with the fact that it is executed extremely fast and doesn't suffer due to the operating system (file overheads are included when reading a file). A disadvantage is that the compiled model can get very large if the number of vectors is large, making compiling time longer and simulator memory usage excessive. Another disadvantage of the fast test-bench is that the model is not easily exchanged between simulations to be run. Changing the test-bench requires a recompilation step. Therefore, the fast test-bench is most useful for models that need fast vector inputs to be applied so that the vectors can be run in a small or medium-sized loop where those vectors are applied again and again.

C. Results of Simulation

The result for the entity *counter_4bit* is shown in Figure 2. For demonstration purposes the input *clock* always shifts values every 5 ns. The *load* signal is '0' between 0 time to 10 ns and allows the counter to be loaded with *data_in* value and to count up. The *clear* signal remains at '1' except at the interval from 40 ns to 45 ns. This means that the output *data_out* will restart counting down from '15' at 55 ns time.

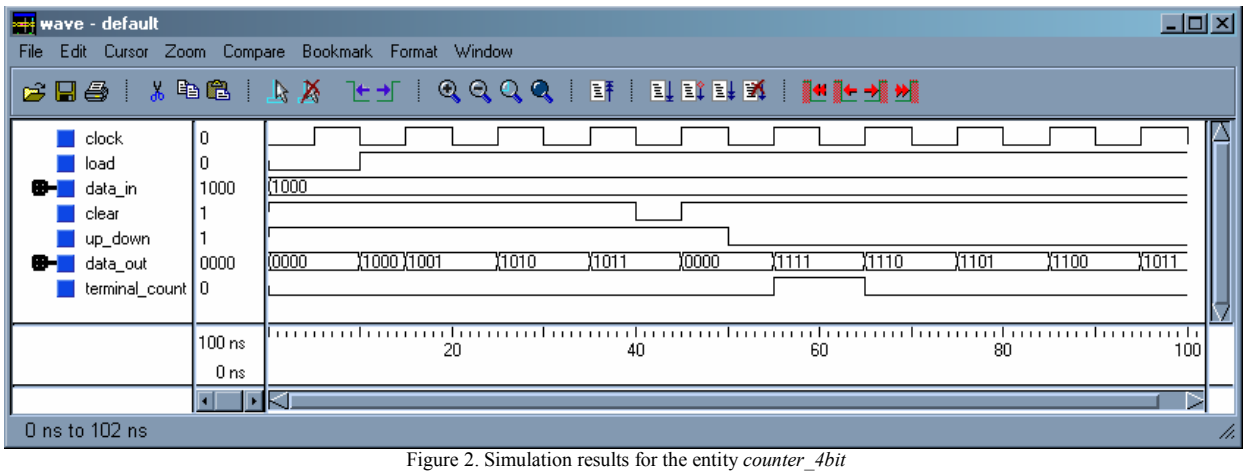

Figure 2. Simulation results for the entity *counter_4bit*

Figures 3 and 4 shows the simulation results obtained for the configurations *small_counter* and *big_counter* for the entity *counter*.

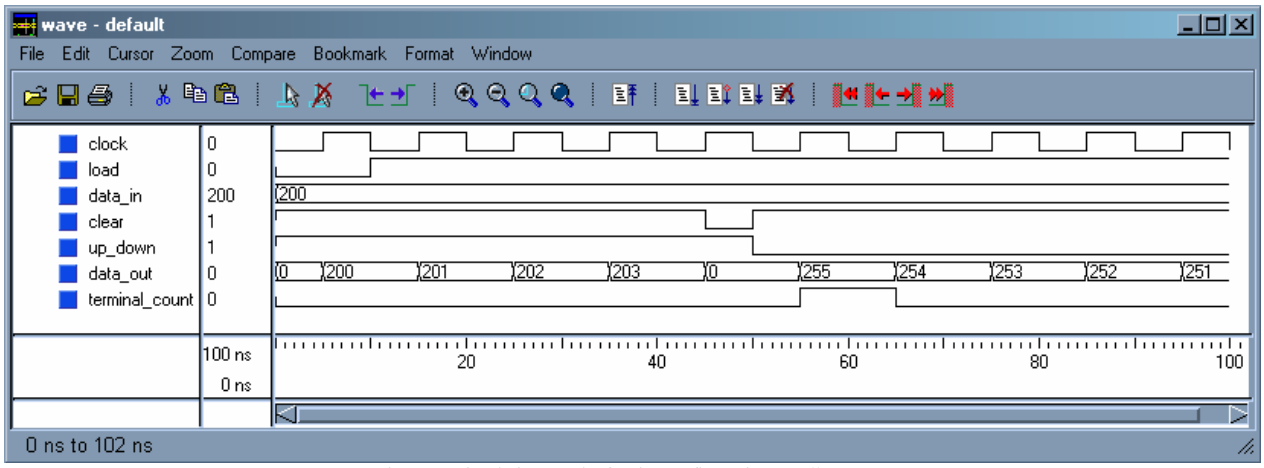

Figure 3. Simulation results for the configuration *small_counter*

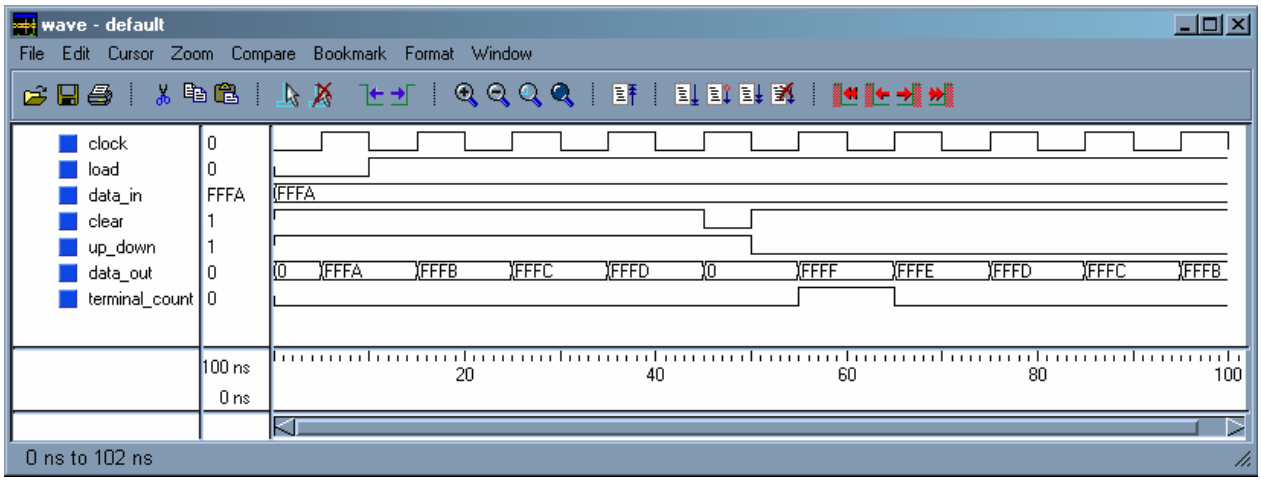

Figure 4. Simulation results for the configuration *big_counter*

I. CONCLUSION

A fundamental motivation to use VHDL is that VHDL is a standard, technology/vendor independent language, and is therefore portable and reusable. The two main immediate applications of VHDL are in the field of Programmable Logic Devices (including CPLDs – Complex Programmable Logic Devices and FPGAs – Field Programmable Gate Arrays) and in the field of ASICs (Application Specific Integrated Circuits). Once the VHDL code has been written, it can be used either to implement the circuit in a programmable device (from Altera, Xilinx, Atmel, etc.) or can be submitted to a foundry for fabrication of an ASIC chip. Currently, many complex commercial chips are designed using such an approach.

REFERENCES

[1] Perry, D. L. - "VHDL Programming by Example", 4th edition, Mc-Grow Hill, USA, 2002

[2] Wunnava, S. - "Tutorial on VHDL and Verilog applications", LACCEI, 2004

[3] Skahill, K. - "VHDL for Programmable Logic", 1st edition, Addison-Wesley, 1996

[4] Volnei, A. Pedroni - "Circuit Design with VHDL", MIT Press, 2004

[5] Patentariu, I., Potorac, A. D. - „Hardware Description Languages, A Comparative Approach", Advances in Electrical and Computer Engineering, Faculty of Electrical Engineering, Ştefan cel Mare University of Suceava, pp 303-308, 2003