

Design Progression With VHDL Helps Accelerate The Digital System Designs

Jaime Marcelo Montenegro, Eng. Ph.D. Candidate

VLSI Lab Manager, Florida International University, Miami, Florida, USA, montenegro@ieee.org

Dr. Subbarao Wunnava, Ph.D., P.E.

Professor of Electrical & Computer Engineering, Florida International University, Miami, Florida, USA,
subbarao@fiu.edu

Abstract

Integrated Circuit technology (IC) is the enabling technology for a whole host of innovative devices and systems that have changed the way we live. Integrated Circuits are much smaller and consume less power than the discrete components used to build electronic systems before the 1960s. Integrated circuits are also easier to design and manufacture and are more reliable than discrete systems. The growing sophistication of applications continually pushes the design and manufacturing of integrated circuits and electronic systems 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 [1]. However, the bottleneck for some vendors appears to be the ability of designers to target the necessary increase in the complexity of electronic devices. Furthermore, 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 widespread adoption of modern technologies in design and testing. 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) are key elements in the evolution of electronic devices. The authors have studied the field of programmable logic and deployed its capabilities. It was demonstrated how the utilization of VHDL benefits not only engineering applications, but also plays an important role accelerating the design of digital systems.

Keywords

Very-large-scale integration, VHDL, Verilog.

1. Introduction

Higher density Programmable Logic Devices, including Complex PLDs (CPLDs) and Field Programmable Gate Arrays (FPGAs), are microcontrollers that can be used to integrate large amounts of logic in a single IC. Semi custom and full-custom Application Specific Integrated Circuit (ASIC) devices are also used for integrating large amounts of digital logic, but CPLDs and FPGAs provide additional flexibility; they can be used with tighter schedules, for low volume products, and for first production runs even with high volume products [2]. Devices such as CPLDs and FPGAs were developed to follow the criteria that in order to upgrade or improve the performance and capabilities of an electronic system, only the devices themselves should be reprogrammed with new instructions and parameters to fulfill the new

demands of that system. With that in mind, the electronics market has a major tool for the development of the next generation of electronic devices. By using these microcontrollers, the issue of achieving adequate scalability, proper upgrade, and stability of a system became a reality. However to efficiently program these logic devices, there was a need for powerful computer languages. A Hardware Description Language such as VHDL is particularly well suited for designing with programmable logic devices.

This article is organized as follows: Section 2 will present a brief overview of VHDL and its capabilities. Section 3 will discuss the implementation of several VHDL logic modules on CPLDs, their simulations, and how they can be upgraded. Section 4 will present the conclusions based on the results obtained and further recommendations.

2. VHDL Overview

The goal of using the current generation of general-purpose computers to help design the next generation of special and general-purpose computers required bringing the worlds of hardware and software back together again [2]. The concept of the Hardware Description Language (HDL) was born from this union. Vendors were looking for a computer language to document electronic systems with the aim to support the state of the art in silicon-based technology. They 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) [3].

VHDL is a hardware description language that can be used to model a digital system at many levels of abstraction, ranging from the algorithmic level to the gate level [4]. 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. Timing can also be explicitly modeled in the same description.

The VHDL language can also be described as a combination of languages as shown in Figure 1:

$$\sum \left[\begin{array}{l} \text{SEQUENTIAL LANGUAGE} \\ \text{CONCURRENT LANGUAGE} \\ \text{NETLIST LANGUAGE} \\ \text{TIMING SPECIFICATIONS} \\ \text{WAVEFORM GENERATION LANGUAGE} \end{array} \right] = \text{VHDL}$$

Figure 1: VHDL as an Integration of Languages

Therefore, the language has constructs that enable the user to express the concurrent or sequential behavior of a digital system with or without timing [2]. 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.

The language not only defines the syntax but also defines very clear simulation semantics for each language construct. Therefore, models written in this language can be verified using a VHDL simulator [4]. The complete language has sufficient power to capture the descriptions of the most complex chips to a complete electronic system. VHDL is used to describe a model for a digital hardware device. This model specifies the external view of the device and one or more external views. The internal view of the device specifies the functionality or structure, while the external view specifies the interface of the device through which it communicates with the other models in its environment. When writing VHDL code, it is necessary to emphasize that the user is designing for hardware. The descriptions in VHDL code will be

synthesized into digital logic for a programmable logic device. The basic building blocks of VHDL design are the *entity declaration* and the *architecture body* [2]. The entity and architecture pairs can be used as complete design descriptions, or as components in a hierarchical design.

An entity declaration describes the design input/output (I/O) that may include parameters used to customize an entity. The entity declaration is analogous to a schematic symbol, which describes a component's connections to the rest of the design. The entity declaration specifies a name by which the entity can be referenced in a design's architecture. A graphical schematic for a 4-bit wide 4 to 1 logic multiplexer is depicted in Figure 2.

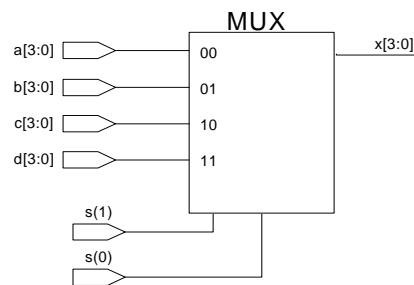


Figure 2: Block diagram of MUX

The multiplexer has a name (MUX), four 4-bit inputs (a, b, c, d), one 2-bit selection line (s), and one 4-bit output (x). The following listing describes the entity declaration in VHDL.

```
entity MUX is port (
  a, b, c, d:   in std_logic_vector(3 downto 0);
  s:           in std_logic_vector(1 downto 0);
  x:          out std_logic_vector(3 downto 0));
end MUX;
```

Listing 1: A 4 to 1 Multiplexer Entity

An architecture body describes the function and contents of a design entity. Every architecture body is associated with an entity declaration. If the entity declaration is viewed as a “black box,” for which the inputs and outputs are known but the details of what is inside the box are not, then the architecture body is the internal view of the black box [4]. VHDL allows the user to write the designs using various styles of architecture. An architecture can contain any combination of *behavioral*, *structural* or *dataflow* styles to define an entity's function. These styles allow you to describe a design at different levels of abstraction, from using algorithms to gate level primitives. Listing 2 contains the architecture body that defines the behavior of multiplexer MUX.

```
architecture archmux of mux is
begin
  with s select
    x <= a when “00”,
         b when “01”,
         c when “10”,
         d when others;
end archmux;
```

Listing 2: Architecture of a 4 to 1 Multiplexer

Based on the value of signal s , signal x is assigned one of four possible values (a , b , c , or d). This construct enables a concise description of the 4 to 1 multiplexer. Three values of s are explicitly enumerated (“00”, “01”, and “10”). The reserved word *others* is used to indicate the remaining possible values for s . That is, *others* is specified instead of “11”.

3. VHDL Logic Modules

This section will deal with the implementation of two logic devices employing VHDL. A 4 to 1 multiplexer, and an 8-bit counter will be described and simulated.

The entity declaration and architecture body described in Listings 1 and 2 respectively define the complete VHDL implementation of a 4 to 1 multiplexer. Listing 3 shows the entire VHDL code that describes the logic component.

```
library ieee;
use ieee.std_logic_1164.all;
entity mux is port (
a, b, c, d: in std_logic_vector(3 downto 0);
s:        in std_logic_vector(1 downto 0);
x:        out std_logic_vector(3 downto 0);
end mux;
architecture archmux of mux is
begin
with s select
    x <= a when "00",
        b when "01",
        c when "10",
        d when others;
end archmux;
```

Listing 3: A 4 to 1 VHDL Multiplexer

Figure 3 illustrates the simulation of the multiplexer (between 200ns to 500ns); where a , b , c , and d are the data inputs, s is the selection line, and x is the output.

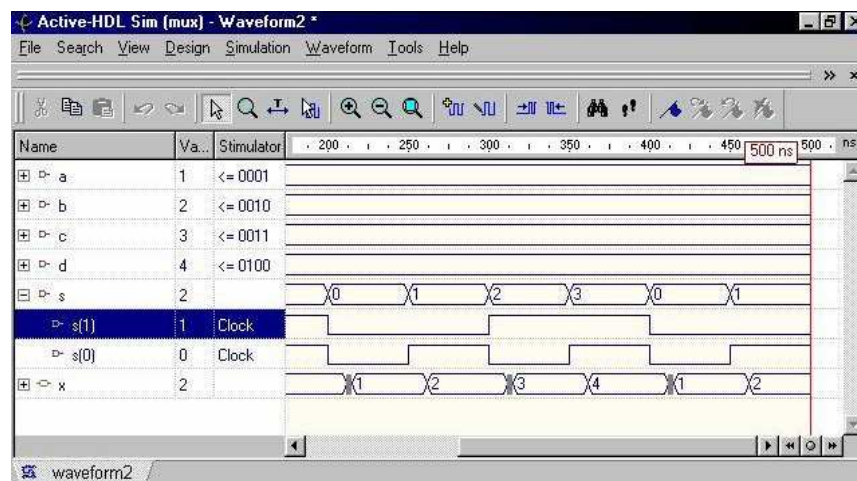


Figure 3: Simulation results of Multiplexer MUX

When the selection lines $s(1)$ and $s(0)$ become “0” (between 200ns to 250ns), the output x becomes “1”. The only data input that had this value was a . Therefore, by selecting “0” for both selection lines, the multiplexer asserted the first data input a as it was expected. One advantage of employing VHDL when designing a component is that the component itself can be modified and upgraded depending on the necessity of the system. For example, the same multiplexer description can be modified to be an 8 to 1 multiplexer by just changing the code as shown in Listing 4.

```

library ieee;
use ieee.std_logic_1164.all;
entity mux8 is port (
a, b, c, d: in std_logic_vector(3 downto 0);
e, f, g, h: in std_logic_vector(3 downto 0);
s:         in std_logic_vector(2 downto 0);
x:         out std_logic_vector(3 downto 0);
end mux8;
architecture archmux8 of mux8 is
begin
with s select
    x <= a when "000",
        b when "001",
        c when "010",
        d when "011",
        e when "100",
        f when "101",
        g when "110",
        h when others;
end archmux8;

```

Listing 4: An 8 to 1 VHDL Multiplexer

Most device architectures have blocks of combinational logic connected to the inputs of flip-flops as the basic building blocks for a CPLD macrocell or an FPGA logic cell. Most sequential logic designs are sensitive to changes in a *clock* signal or a *reset* signal. VHDL activates a process only when one of these signals presents a transition from one logic state to the other [2]. Listing 5 depicts the VHDL high level behavioral description of an 8-bit counter.

```

library ieee;
use ieee.std_logic_1164.all;
use work.numeric_std.all;
entity cnt8 is port(
    reset: in std_logic;
    clk:   in std_logic;
    outclk: buffer unsigned(7 downto 0));
end cnt8;
architecture cnt8 of cnt8 is
begin
count: process (clk, reset)
begin
    if reset = '1' then
        outclk <= (others =>'0');
    elsif (clk'event and clk='1') then
        outclk <= outclk + 1;
    end if;
end process count;
end cnt8;

```

Listing 5: An 8-bit Counter

The counter *cnt8* has two 1-bit inputs *clk* and *reset*; and one 8-bit output *outclk*, as shown in Figure 4.

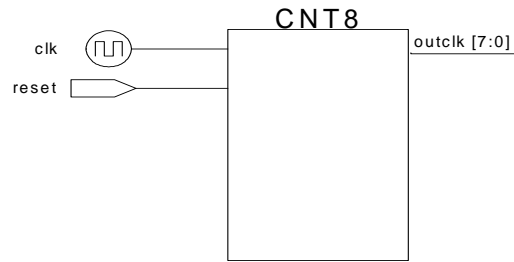


Figure 4: An 8-bit Counter

Figure 5 shows the simulation results obtained for the counter *cnt8*. The total running simulation time was 750ns and the period of the input signal *clk* was 100ns.

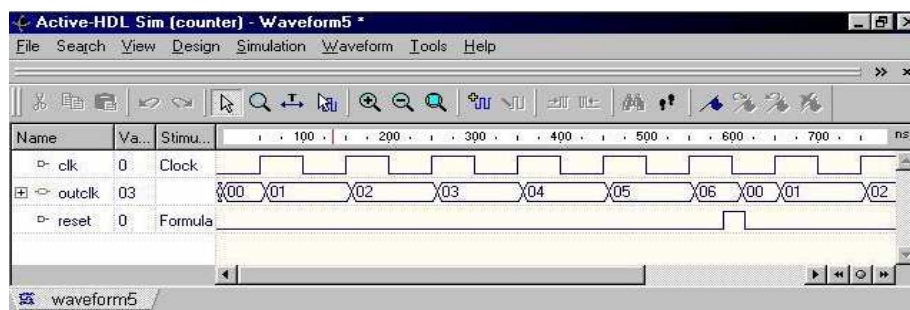


Figure 5: Simulation results of Counter *cnt8*

The *reset* signal remains at “0” throughout the whole simulation except at the interval between from 590ns to 615ns. This means that the output *outclk* will restart counting from “0” after 615ns.

Once again the VHDL counter can be modified depending on the necessity of the system. In this case, the same counter *cnt8* is modified to be a 4 bit counter by changing the code as shown in Listing 6.

```

library ieee;
use ieee.std_logic_1164.all;
use work.numeric_std.all;
entity cnt4 is port(
    reset: in std_logic;
    clk: in std_logic;
    outclk: buffer unsigned(3 downto 0));
end cnt4;
architecture cnt4 of cnt4 is
begin
count: process (clk, reset)
begin
    if reset = '1' then
        outclk <= (others =>'0');
    elsif (clk'event and clk='1') then
        outclk <= outclk + 1;
    end if;
end process count;
end cnt4;

```

Listing 6: A 4-bit Counter

4. Conclusions

The primary purpose of this research was to study the field of programmable logic. A Complex Programmable Logic Device (CPLD) was programmed by means of a Hardware Description Language. With the growing sophistication of applications continually pushing the design and manufacturing of integrated circuits, the CPLD proved to be easy to design and upgrade. The reprogramming feature of the CPLD was successfully tested in this study. A major requirement for today's electronic systems is the upgrading capability. The electronics industry requires systems to be capable of in-site reprogramming, where the upgrading task depends more on software than on hardware. In order to achieve the proper functionality of the system, the CPLD was reprogrammed several times depending on the system's specifications. In terms of time and cost, this feature saves a lot of troubleshooting time, and also reduces the cost of replacing electronic components every time a change is necessary.

A Hardware Description Language (HDL) such as VHDL was employed to program the CPLD. The HDL purpose was to document the electronic system with the aim to support the microcontroller technology. This language not only defines the syntax for the system but also defines very clear simulations for each language construct. VHDL was used to describe the model for a digital hardware device. This model specified the external view of the device and one or more external views. The internal view of the device specified the functionality or structure, while the external view specified the interface of the device through which it communicated with the other models in its environment.

References

- [1] Wolf, W. (2002). *“Modern VLSI Design Systems on Chip Design”*, 3rd edition, Prentice Hall, USA.
- [2] Skahill, K. (1996). *“VHDL for Programmable Logic”*, Addison-Wesley, USA.
- [3] Montenegro, J. (2002) *“Very High Speed Integrated Circuits (VHDL) and Verilog Based Microcontroller Implementation With In System Reprogrammable (ISR) Hardware Modules,”* M.S. Thesis, Florida International University, Florida, USA.
- [4] Bhasker, J. (1999) *“A Verilog HDL Primer”*, Second Edition, Star Galaxy Publishing, USA.
- [5] More, M., and Vidal, J. (1998) *“Experiences on VHDL based methodologies on industrial ASIC design”*, Proceedings of the International Semiconductor Conference, CAS v, IEEE, Piscataway, NJ, USA. Pp. 167-170.

Authorization and Disclaimer

Authors authorize LACCEI to publish the papers in the conference proceedings. Neither LACCEI nor the editors are responsible either for the content or for the implications of what is expressed in the paper.