Fourth LACCEI International Latin American and Caribbean Conference for Engineering and Technology (LACCET'2006)
"Breaking Frontiers and Barriers in Engineering: Education, Research and Practice"
21-23 June 2006, Mayagüez, Puerto Rico.

Design Progression With Verilog 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
Electronic systems now perform a wide variety of tasks in daily life. Electronic systems in some cases have replaced older mechanisms that operated mechanically, hydraulically, or by other means; electronics are usually smaller, more flexible, and easier to use. In other cases electronic systems have created totally new applications. The progress of electronic systems depends mostly in the harmonization of software and hardware. The role of software designers and hardware designers is essentially the same: solve a problem. Their primary job is to develop an algorithm that solves a problem and translate that algorithm into hardware. Integrated Circuit (IC) technology allows us to build hardware systems with many more transistors, allowing much more computer power to be applied to solving a problem [1]. This ability makes possible the development of special-purpose systems that are more efficient than general-purpose computers for the task at hand. As a computer language, a Hardware Description Language (HDL) allows the use of many of the time saving software methodologies; but, as a hardware language, the HDL allows expression of concepts that previously could not be expressed by manual notation. The Verilog Hardware Description Language is a popular HDL; it has syntax similar to C and Pascal. It is now known as IEEE1364. The authors have studied the field of Hardware Description Languages and Verilog and deployed its capabilities. It was demonstrated how the utilization of Verilog benefits not only engineering applications, but also plays an important role accelerating the design of digital systems.

Keywords
Very-large-scale integration, Verilog, VHDL.

1. Introduction

In the first half of the 1990s the electronics industry experienced an explosion in the demand for electronic devices such as personal computers, cellular phones, and high-speed data communications devices. To accomplish this, vendors created highly integrated, complex systems with fewer Integrated Circuit devices and less Printed Circuit Board (PCB) area. The submicron semiconductor development, large-scale PCB manufacturing, and surface mount packaging technologies supported increased integration.

Given the existing Electronic Design Automation (EDA) tools and accelerated time to market schedules, the drawback for some vendors seemed to be the ability of engineers to handle the increasing complexity

of designs [2]. In most cases, the demand for faster, more reliable and smaller devices required a major increase in the complexity of electronic devices. Moreover, the necessity to upgrade systems with digital circuitry became a major issue. Most electronic devices were not upgradeable since the IC technology embedded in their logic architectures was not reprogrammable. The market required systems to be capable of in-site reprogramming, allowing the systems to adjust and grow as its necessities changed [3]. This situation promoted the need for widespread adoption of modern technologies in design and testing.

Designers utilizing HDLs are capable of programming devices by writing code in a mode similar to that of computer languages such as C and C++. This offers the designer the ability and flexibility to program microcontrollers by writing the code in an English-like environment, or as it is commonly known, translate words into algorithms. Verilog Hardware Description Language is suitable for the programming and testing of reprogrammable devices such as Complex Programmable Logic Devices (CPLDs) and Field Programmable Gate Arrays (FPGAs).

This article is organized as follows: Section 2 will present a brief overview of Verilog and its capabilities. Section 3 will discuss the implementation of several Verilog 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. Verilog Overview

Verilog HDL 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 to the switch-level. The complexity of the digital system being modeled could vary from that of a simple gate to a complete electronic digital system. The digital system can be described hierarchically and timing can be explicitly modeled within the same description [4].

The Verilog HDL language includes capabilities to describe the behavioral nature of a design, the dataflow nature of a design, a design's structural composition, delays and a waveform generation mechanism including aspects of response monitoring and verification, all modeled using one single language [5]. In addition, the language provides a programming language interface through which the internals of a design can be accessed during simulation including the control of a simulation run.

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 Verilog simulator. The language inherits many of its operator symbols and constructs from the C programming language.

The basic building units of description in Verilog are the *module*, *declaration*, and *statements*. A *Module* describes the functionality or structure of a design and also describes the ports through which it communicates externally with other modules. The structure of a design is described using switch level primitives, gate level primitives and user-defined primitives. A module in Verilog encapsulates the description of a design [5]. Verilog *Declarations* are used to define the various items, such as registers and parameters, used within a module. *Statements* are used to define the functionality or structure of the design. Declarations and statements can be interspersed within a module; however, a declaration must appear before its use. Listing 1 is an example of a module containing declarations and statements that models the half-adder circuit shown in Figure 1.

```
module HalfAdder (A, B, Sum, Carry);  //Name
                                      //of
                                      //module

input A, B;                  // Declaration line 1
```

```
output Sum, Carry;          // Declaration line 2

assign Sum = A ^ B;         // Statement line 1
assign Carry = A & B;       // Statement line 2

endmodule                   // End of module
```

Listing 1: Half-Adder Verilog Module

The name of the module is *HalfAdder*. It has four ports; two input ports *A* and *B*, and two output ports *Sum* and *Carry*. All ports are size 1-bit since no range has been specified in the declarations. The module contains two continuous assignment statements that describe the behavior of the half adder.

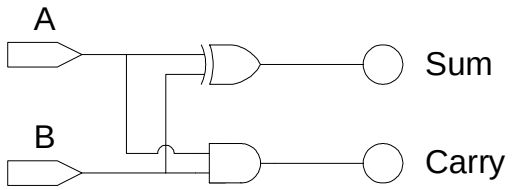

Figure 1: A Half-Adder Circuit Schematic

The description of a design within a module can be either one, or a combination of the following styles:

Dataflow style: The basic mechanism used to model a design in the dataflow style is the continuous assignment; where a value is assigned to a port. When the value of an operand used in the right hand side expression changes, the right hand side expression is evaluated, and the value is assigned to the left hand side variable.

Behavioral style: The behavioral descriptions provide a means to define the behavior of a circuit in abstract high -level algorithms.

Structural style: The structural descriptions define the structure of the circuit in terms of components and resemble a netlist that describes a schematic equivalent of the design. Structural Verilog descriptions contain hierarchy in which components are defined at different levels [5].

3. Verilog Logic Modules

This section will deal with the implementation of two logic devices employing Verilog. A 4-bit counter and a full adder will be described and simulated. Listing 2 shows the Verilog behavioral description of a module that will behave as a 4-bit counter design.

```
module Counter (trigger, reset, count);
        parameter   counter_size = 4;
        input       trigger;
        input       reset;
        inout       [counter_size:1] count;
        reg         [counter_size:0] tmp_count;

always @(posedge reset or posedge trigger)f
begin
if (reset == 1'b 1)
   tmp_count <= {(counter_size + 1){1'b 0}};
```

```
else
   tmp_count <= count + 1;
end
assign count = tmp_count;

endmodule
```

Listing 2: A 4-bit Counter Verilog Module

The module *Counter* has two control inputs (*trigger*, *reset*) and one input/output variable (*count*). The parameter *counter_size* is intended to define the size of the counter. For this example, *counter_size* is set to 4 in order to define a 4-bit counter. This is very useful because by changing only this parameter, a different size counter can be implemented. The functionality of the module is described in high-level algorithms by including *if* and *else* statements in the syntax of the code. Figure 2 illustrates the schematic of the 4-bit counter *Counter*.

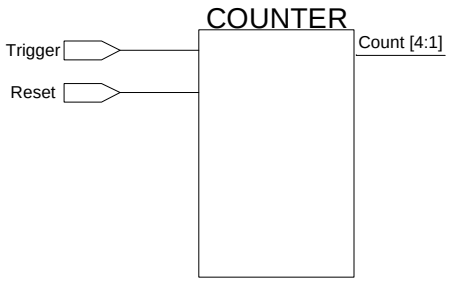

Figure 2: 4-bit Counter *Counter*

Figure 3 shows the simulation results obtained for the module *Counter*. The total running simulation time was 150µs, and the signal's duration value was 5µs. For demonstration purposes the input *trigger* always shifts values every 5µs. The *reset* signal remains at "0" except at the interval from 50µs to 55µs. This means that the output *count* will restart counting from "0" at 50µs.

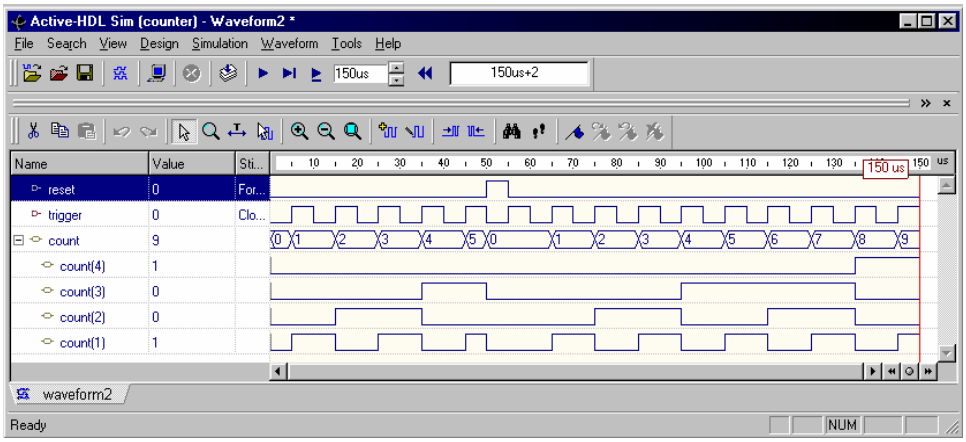

Figure 3: Simulation results of the 4-bit Counter

Listing 3 shows the description of a module that will behave as a full adder design.

```
module  FA_seq  (A,  B,  Cin,  Sum,  Cout);
          input  A,  B,  Cin;
```

```
        output Sum, Cout;
        reg Sum, Cout;
        reg T1, T2, T3;
always
        @ (A or B or Cin) begin
                Sum = (A ^ B) ^ Cin;
                T1 = A & Cin;
                T2 = B & Cin;
                T3 = A & B;
                Cout =(T1 | T2) | T3;
        end
endmodule
```

Listing 3: A 1-bit Full Adder Verilog Module

The module *FA_Seq* has three inputs and two outputs. The variables *Sum*, *Cout*, *T1*, *T2* and *T3* are of type *reg* because they are assigned values within the always statement. The always statement has a sequential block associated with an event control. This means that every time there is a change in the inputs *A*, *B* or *Cin*, the sequential block is executed. Statements within a sequential block execute sequentially, and the execution suspends after the last statement in the sequential block has executed. Once the block has executed, the always statement waits for a change in the inputs *A*, *B* or *Cin*. Figure 4 depicts the schematic and the logic circuitry of the full adder *FA_Seq*.

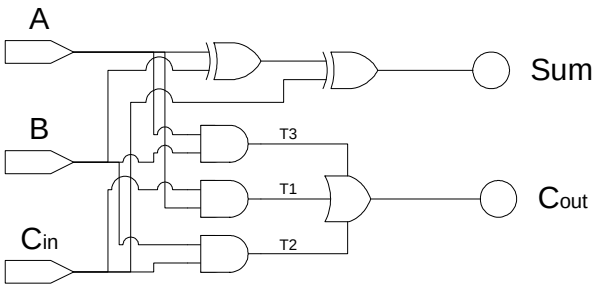

Figure 4: Full Adder *FA_Seq*

Figure 5 illustrates the successful simulation of the full adder, where *A*, *B*, and *Cin* are the inputs, and *Sum* and *Cout* are the outputs. The total running simulation time was 70µs and the signal's duration value was 10µs. That means that each signal remained at a logic state (0 or 1) for 10µs before any change occurs.

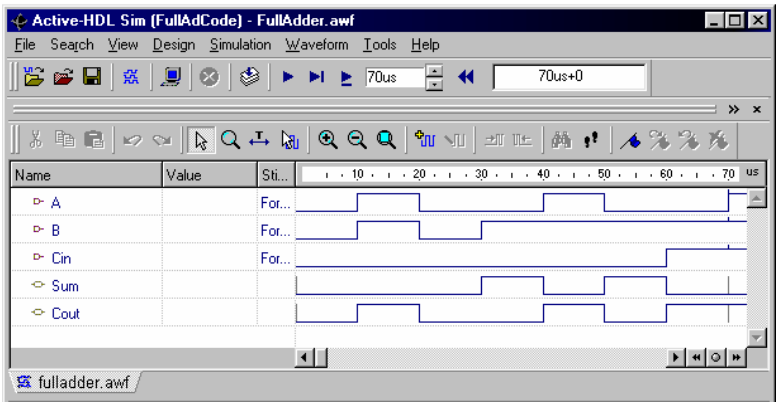

Figure 5: Full Adder Verilog simulations

4. Conclusions

The primary purpose of this research was to study the field of Hardware Description Languages and Verilog. The study was significant for several reasons. First, the utilization of Hardware Description Languages in real life Engineering applications will become more conventional. Second, the study was 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. This will help the ISR (In System Reprogrammable) CPLD and FPGA based hardware development, for industrial applications. Third, it was demonstrated how the utilization of Verilog benefits not only engineering applications, but also plays an important role accelerating the design of digital systems.

As Verilog and VHDL continue to gain momentum as the Hardware Description Languages of choice for programmable logic and are a requirement for the development of the next generation of design engineers, it is necessary that Engineering students, as well as design engineers, could benefit from these HDLs.

References

Wolf, W. (2002). "*Modern VLSI Design Systems on Chip Design*", 3rd edition, Prentice Hall, USA.

Skahill, K. (1996). "*VHDL for Programmable Logic*", Addison-Wesley, USA.

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.

Gordon Arnold, M. (1999) "*Verilog DIGITAL Computer Design, Algorithms to Hardware*", Prentice Hall, USA.

Bhasker, J. (1999) "*A Verilog HDL Primer*", Second Edition, Star Galaxy Publishing, USA.

Authorization and Disclaimer