Chapter 1 : Verilog HDL: A Guide to Digital Design and Synthesis, Second - calendrierdelascience.com

Part 1 Basic Verilog Topics Overview of Digital Design with Verilog HDL Evolution of CAD, emergence of HDLs, typical HDL-based design flow, why Verilog HDL?, trends in HDLs.

Names of signals, modules, ports, etc. However, it is a general term used to refer to a Verilog HDL user or a verification engineer. Following were the primary contributors to my creation: I would like to start by thanking all those people once again. For this second edition, I give special thanks to the following people who helped me with the review process and provided valuable feedback: Some of the material in this second edition of the book was inspired by conversations, email, and suggestions from colleagues in the industry. I have credited these sources where known, but if I have overlooked anyone, please accept my apologies. Trends in HDLs 17 1. The earliest digital circuits were designed with vacuum tubes and transistors. Integrated circuits were then invented where logic gates were placed on a single chip. As technologies became sophisticated, designers were able to place circuits with hundreds of gates on a chip. At this point, design processes started getting very complicated, and designers felt the need to automate these processes. Chip designers began to use circuit and logic simulation techniques to verify the functionality of building blocks of the order of about transistors. The circuits were still tested on the breadboard, and the layout was done on paper or by hand on a graphic computer terminal. Technically, the term Computer-Aided Design CAD tools refers to back-end tools that perform functions related to place and route, and layout of the chip. For the sake of simplicity, in this book, we will refer to all design tools as EDA tools. Because of the complexity of these circuits, it was not possible to verify these circuits on a breadboard. Computeraided techniques became critical for verification and design of VLSI digital circuits. Computer programs to do automatic placement and routing of circuit layouts also became popular. The designers were now building gate-level digital circuits manually on graphic terminals. They would build small building blocks and then derive higher-level blocks from them. This process would continue until they had built the top-level block. Logic simulators came into existence to verify the functionality of these circuits before they were fabricated on chip. As designs got larger and more complex, logic simulation assumed an important role in the design process. Designers could iron out functional bugs in the architecture before the chip was designed further. Similarly, in the digital design field, designers felt the need for a standard language to describe digital circuits. HDLs allowed the designers to model the concurrency of processes found in hardware elements. Both Verilog and VHDL simulators to simulate large digital circuits quickly gained acceptance from designers. Even though HDLs were popular for logic verification, designers had to manually translate the HDL-based design into a schematic circuit with interconnections between gates. The advent of logic synthesis in the late s changed the design methodology radically. Thus, the designer had to specify how the data flows between registers and how the design processes the data. The details of gates and their interconnections to implement the circuit were automatically extracted by logic synthesis tools from the RTL description. Thus, logic synthesis pushed the HDLs into the forefront of digital design. Designers no longer had to manually place gates to build digital circuits. They could describe complex circuits at an abstract level in terms of functionality and data flow by designing those circuits in HDLs. Logic synthesis tools would implement the specified functionality in terms of gates and gate interconnections. HDLs also began to be used for system-level design. A common approach is to design each IC chip, using an HDL, and then verify system functionality via simulation. In , the original standard IEEE was approved. Unshaded blocks show the level of design representation; shaded blocks show processes in the design flow. In any design, specifications are written first. Specifications describe abstractly the functionality, interface, and overall architecture of the digital circuit to be designed. At this point, the architects do not need to think about how they will implement this circuit. A behavioral description is then created to analyze the design in terms of functionality, performance, compliance to standards, and other high-level issues. Behavioral descriptions are often written with HDLs. These tools can be used instead of writing behavioral descriptions in

Verilog HDL. The designer has to describe the data flow that will implement the desired digital circuit. From this point onward, the design process is done with the assistance of EDA tools. Logic synthesis tools convert the RTL description to a gate-level netlist. A gate-level netlist is a description of the circuit in terms of gates and connections between them. Logic synthesis tools ensure that the gate-level netlist meets timing, area, and power specifications. The gate-level netlist is input to an Automatic Place and Route tool, which creates a layout. The layout is verified and then fabricated on a chip. Thus, most digital design activity is concentrated on manually optimizing the RTL description of the circuit. Designing at the RTL level has shrunk the design cycle times from years to a few months. It is also possible to do many design iterations in a short period of time. Behavioral synthesis tools have begun to emerge recently. These tools can create RTL descriptions from a behavioral or algorithmic description of the circuit. As these tools mature, digital circuit design will become similar to high-level computer programming. Designers will simply implement the algorithm in an HDL at a very abstract level. EDA tools will help the designer convert the behavioral description to a final IC chip. It is important to note that, although EDA tools are available to automate the processes and cut design cycle times, the designer is still the person who controls how the tool will perform. Garbage In Garbage Out" phenomenon. If used improperly, EDA tools will lead to inefficient designs. Thus, the designer still needs to understand the nuances of design methodologies, using EDA tools to obtain an optimized design. Designers can write their RTL description without choosing a specific fabrication technology. Logic synthesis tools can automatically convert the design to any fabrication technology. If a new technology emerges, designers do not need to redesign their circuit. They simply input the RTL description to the logic synthesis tool and create a new gate-level netlist, using the new fabrication technology. The logic synthesis tool will optimize the circuit in area and timing for the new technology. Since designers work at the RTL level, they can optimize and modify the RTL description until it meets the desired functionality. Most design bugs are eliminated at this point. This cuts down design cycle time significantly because the probability of hitting a functional bug at a later time in the gate-level netlist or physical layout is minimized. A textual description with comments is an easier way to develop and debug circuits. This also provides a concise representation of the design, compared to gate-level schematics. Gate-level schematics are almost incomprehensible for very complex designs. HDL-based design is here to stay. With rapidly increasing complexities of digital circuits and increasingly sophisticated EDA tools, HDLs are now the dominant method for large digital designs. No digital circuit designer can afford to ignore HDL-based design. These languages are better suited for functional verification. However, for logic design, HDLs continue as the preferred choice. It is similar in syntax to the C programming language. Thus, a designer can define a hardware model in terms of switches, gates, RTL, or behavioral code. Also, a designer needs to learn only one language for stimulus and hierarchical design. This makes it the language of choice for designers. Thus, designing a chip in Verilog HDL allows the widest choice of vendors. Designers have responded by designing at higher levels of abstraction. Designers have to think only in terms of functionality. EDA tools take care of the implementation details. With designer assistance, EDA tools have become sophisticated enough to achieve a close-to-optimum implementation. Behavioral synthesis allowed engineers to design directly in terms of algorithms and the behavior of the circuit, and then use EDA tools to do the translation and optimization in each phase of the design. However, behavioral synthesis did not gain widespread acceptance. Today, RTL design continues to be very popular. Verilog HDL is also being constantly enhanced to meet the needs of new verification methodologies. Formal verification and assertion checking techniques have emerged. Formal verification applies formal mathematical techniques to verify the correctness of Verilog HDL descriptions and to establish equivalency between RTL and gate-level netlists. However, the need to describe a design in Verilog HDL will not go away. Assertion checkers allow checking to be embedded in the RTL code. This is a convenient way to do checking in the most important parts of a design.

Chapter 2 : Verilog HDL: A Guide to Digital Design and Synthesis [With CDROM] by Samir Palnitkar

Comment: Spine creases, wear to binding and pages from reading. May contain limited notes, underlining or highlighting that does affect the text. Possible ex library copy, thatÃ¢â‚¬TMll have the markings and stickers associated from the library.

Mazaaq Posted by On Gaylord Hotel Manager Posted by On I wanted to learn basic digital design paradigms and the necessary Verilog HDL constructs that would help me build small digital circuits, using Verilog and run simulations. At that time I was searching for a book that broadly discussed advanced Verilog-based digital design concepts and real digital design methodologies. Finally, when I had gained enough experience with digital design and verification of real IC chips, though manuals of Verilog-based products were available, from time to time, I felt the need for a Verilog HDL book that would act as a handy reference. This book emphasizes breadth rather than depth. The book imparts to the reader a working knowledge of a broad variety of Verilog-based topics, thus giving the reader a global understanding of Verilog HDL-based design. The book leaves the in-depth coverage of each topic to the Verilog HDL language reference manual and the reference manuals of the individual Verilog-based products. This book should be classified not only as a Verilog HDL book but, more generally, as a digital design book. It important to realize that Verilog HDL is only a tool used in digital design. It is the means to an end- the digital IC chip. Therefore, this book stresses the practical design perspective more than the mere language aspects of Verilog HDL. Who Should Use This Book The book is intended primarily for beginners and intermediate-level Verilog users. However, for advanced Verilog users, the broad coverage of topics makes it an excellent reference book to be used in conjunction with the manuals and training materials of Verilog-based products. The book presents a logical progression of Verilog HDL-based topics. It starts with the basics, such as HDL-based design methodologies, and then gradually builds on the basics to eventually reach advanced topics, such as PLI or logic synthesis. Thus, the book is useful to Verilog users with varying levels of expertise as explained below. Students in logic design courses at universities Part 1 of this book is ideal for a foundation semester course in Verilog HDL-based logic design. Students are exposed to hierarchical modeling concepts, basic Verilog constructs and modeling techniques, and the necessary knowledge to write small models and run simulations. Part 1 of this book is a perfect jump start for designers who want to orient their skills toward HDL-based design. Users with basic Verilog knowledge who need to understand advanced concepts Part 2 of this book discusses advanced concepts, such as UDPs, timing simulation, PLI, and logic synthesis, which are necessary for graduation from small Verilog models to larger designs. Verilog experts All Verilog topics are covered, from the basics modeling constructs to advanced topics like PLIs and logic synthesis. For Verilog experts, this book is a handy reference to be used along with the reference manuals. However, the concepts explained in the book are general enough to be applicable to the design of FPGAs, PALs, buses, boards, and systems. The same concepts apply to VLSI designs. How This Book Is Organized: This book is organized into three parts. Part 1, Basic Verilog Topics, covers all information that a new user needs to build small Verilog models and run simulations. Note that in Part 1, gate-level modeling is addressed before behavioral modeling. I have chosen to do so because I think that it is easier for a new user to see a correspondence between gate- level circuits and equivalent Verilog descriptions. Once gate-level modeling is understood, a new user can move to higher levels of abstraction, like data flow modeling and behavioral modeling, without losing sight of the fact that Verilog HDL is a language for digital design and is not a programming language. Thus, a new user starts off with the idea that Verilog is a language for digital design. New users who start with behavioral modeling often tend to write Verilog the way they write their C programs. They sometimes lose sight of the fact that they are trying to represent hardware circuits by using Verilog. Part 1 contains nine chapters. Part 2, Advanced Verilog Topics, contains the advanced concepts a Verilog user needs to know to graduate from small Verilog models to larger designs. Part 2 contains five chapters. Part 3, Appendices, contains information useful as a reference. Useful information, such as strength-level modeling, list of PLI routines, formal syntax definition, Verilog tidbits, and large Verilog examples is included. Part 3 contains six appendices. Conventions Used in This Book. The word designer is used frequently in the book to emphasize the digital design perspective. However, it is a general term used to refer to a Verilog HDL user For more free books download visit this blog daily.

Chapter 3 : Verilog HDL: A Guide to Digital Design and Synthesis - Samir Palnitkar - Google Books

VERILOG HDL, Second Editionby Samir PalnitkarWith a Foreword by Prabhu Goel Written forboth experienced and new users, this book gives you broad coverage of VerilogHDL. The book stresses the practical design and verification perspective ofVerilog rather than emphasizing only the language aspects.

Includes bibliographical references p. Evolution of Computer Aided Digital Design. Popularity of Verilog HDL. Components of a Simulation. X or Z values. Underscore characters and question marks. System Tasks and Compiler Directives. Stopping and finishing in a simulation. Example of illegal port connection. Connecting Ports to External Signals. Connecting by ordered list. Connecting ports by name. Rise, Fall, and Turn-off Delays. Implicit Continuous Assignment Delay. Expressions, Operators, and Operands. Application of nonblocking assignments. Sequential and Parallel Blocks. Special Features of Blocks. Differences Between Tasks and Functions. Task Declaration and Invocation. Use of Input and Output Arguments. Function Declaration and Invocation. Force and release on registers. Force and release on nets. Conditional Compilation and Execution. Initializing Memory from File. Value Change Dump File. Types of Delay Models. Rise, fall, and turn-off delays. Min, max, and typical delays. Delay Specification on Switches. Parts of UDP Definition. Example of a Combinational UDP. Example of a Sequential UDP. Guidelines for UDP Design. Mechanics of Access Routines. Types of Access Routines. Examples of Access Routines. Mechanics of Utility Routines. Types of Utility Routines. Example of Utility Routines. What Is Logic Synthesis? Impact of Logic Synthesis. Interpretation of a Few Verilog Constructs. The case statement for loops. Technology Mapping and Optimization. Final, Optimized, Gate-Level Description. Verification of Gate-Level Netlist. Modeling Tips for Logic Synthesis. Use meaningful names for signals and variables. Avoid mixing positive and negative edge-triggered flip-flops. Use basic building blocks vs. Use continuous assign statements. Use if-else or case statements. Use parentheses to optimize logic structure. Be careful with multiple assignments to the same variable. Define if-else or case statements explicitly. Example of Sequential Circuit Synthesis. Strength Modeling and Advanced Net Definitions. List of PLI Routines. This complete Verilog HDL reference progresses from the basic Verilog concepts to the most advanced concepts in digital design. Nielsen Book Data Subjects.

Chapter 4 : Verilog HDL : a guide to digital design and synthesis in SearchWorks catalog

Verilog HDL: A Guide to Digital Design and Synthesis / Edition 1 Verilog HDL is a language for digital design, just as C is a language for programming. This complete Verilog HDL reference progresses from the basic Verilog concepts to the most advanced concepts in digital design.

Includes bibliographical references p. Each chapter concludes with a Summary and Exercises. Evolution of Computer-Aided Digital Design. Popularity of Verilog HDL. Components of a Simulation. System Tasks and Compiler Directives. Expressions, Operators, and Operands. Sequential and Parallel Blocks. Difference between Tasks and Functions. Conditional Compilation and Execution. Types of Delay Models. Guidelines for UDP Design. What Is Logic Synthesis? Impact of Logic Synthesis. Verification of the Gate-Level Netlist. Modeling Tips for Logic Synthesis. Example of Sequential Circuit Synthesis. Strength Modeling and Advanced Net Definitions. List of PLI Routines. System Tasks and Functions. Module and Generated Instantiation. UDP Declaration and Instantiation. This complete Verilog HDL reference progresses from the basic Verilog concepts to the most advanced concepts in digital design. Palnitkar covers the gamut of Verilog HDL fundamentals, such as gate, RTL, and behavioral modeling, all the way to advanced concepts, such as timing simulation, switch level modeling, PLI, and logic synthesis. Verilog HDL is a hardware description language with a user community of more than 50, active designers used to design and document electronic systems. This completely updated reference progresses from basic to advanced concepts in digital design, including timing simulation, switch level modeling, PLI, and logic synthesis. Nielsen Book Data Subjects.

Chapter 5 : Palnitkar, Verilog HDL: A Guide to Digital Design and Synthesis (Bk/CD-ROM) | Pearson

Book Description. Appropriate for all courses in digital IC or system design using the Verilog Hardware Description Language (HDL). Fully updated for the latest versions of Verilog HDL, this complete reference progresses logically from the most fundamental Verilog concepts to today's most advanced digital design techniques.

The earliest digital circuits were designed with vacuum tubes and transistors. Integrated circuits were then invented where logic gates were placed on a single chip. As technologies became sophisticated, designers were able to place circuits with hundreds of gates on a chip. At this point, design processes started getting very complicated, and designers felt the need to automate these processes. Chip designers began to use circuit and logic simulation techniques to verify the functionality of building blocks of the order of about transistors. The circuits were still tested on the breadboard, and the layout was done on paper or by hand on a graphic computer terminal. Technically, the term Computer-Aided Design CAD tools refers to back-end tools that perform functions related to place and route, and layout of the chip. For the sake of simplicity, in this book, we will refer to all design tools as EDA tools. Because of the complexity of these circuits, it was not possible to verify these circuits on a breadboard. Computer-aided techniques became critical for verification and design of VLSI digital circuits. Computer programs to do automatic placement and routing of circuit layouts also became popular. The designers were now building gate-level digital circuits manually on graphic terminals. They would build small building blocks and then derive higher-level blocks from them. This process would continue until they had built the top-level block. Logic simulators came into existence to verify the functionality of these circuits before they were fabricated on chip. As designs got larger and more complex, logic simulation assumed an important role in the design process. Designers could iron out functional bugs in the architecture before the chip was designed further. Similarly, in the digital design field, designers felt the need for a standard language to describe digital circuits. HDLs allowed the designers to model the concurrency of processes found in hardware elements. Both Verilog and VHDL simulators to simulate large digital circuits quickly gained acceptance from designers. Even though HDLs were popular for logic verification, designers had to manually translate the HDL-based design into a schematic circuit with interconnections between gates. The advent of logic synthesis in the late s changed the design methodology radically. Thus, the designer had to specify how the data flows between registers and how the design processes the data. The details of gates and their interconnections to implement the circuit were automatically extracted by logic synthesis tools from the RTL description. Thus, logic synthesis pushed the HDLs into the forefront of digital design. Designers no longer had to manually place gates to build digital circuits. They could describe complex circuits at an abstract level in terms of functionality and data flow by designing those circuits in HDLs. Logic synthesis tools would implement the specified functionality in terms of gates and gate interconnections. HDLs also began to be used for system-level design. A common approach is to design each IC chip, using an HDL, and then verify system functionality via simulation. In , the original standard IEEE was approved. Unshaded blocks show the level of design representation; shaded blocks show processes in the design flow. In any design, specifications are written first. Specifications describe abstractly the functionality, interface, and overall architecture of the digital circuit to be designed. At this point, the architects do not need to think about how they will implement this circuit. A behavioral description is then created to analyze the design in terms of functionality, performance, compliance to standards, and other high-level issues. Behavioral descriptions are often written with HDLs. Designers can write their RTL description without choosing a specific fabrication technology. Logic synthesis tools can automatically convert the design to any fabrication technology. If a new technology emerges, designers do not need to redesign their circuit. They simply input the RTL description to the logic synthesis tool and create a new gate-level netlist, using the new fabrication technology. The logic synthesis tool will optimize the circuit in area and timing for the new technology. By describing designs in HDLs, functional verification of the design can be done early in the design cycle. Since designers work at the RTL level, they can optimize and modify the RTL description until it meets the desired functionality. Most design bugs are eliminated at this point. This cuts down design cycle time significantly because the probability of hitting a functional bug at a later time in the gate-level netlist or physical layout is minimized. Designing with HDLs is analogous to computer programming. A textual description with comments is an easier way to develop and debug circuits. This also provides a concise representation of the design, compared to gate-level schematics. Gate-level schematics are almost incomprehensible for very complex designs. HDL-based design is here to stay. No digital circuit designer can afford to ignore HDL-based design. These languages are better suited for functional verification. However, for logic design, HDLs continue as the preferred choice. It is similar in syntax to the C programming language. Verilog HDL allows different levels of abstraction to be mixed in the same model. Thus, a designer can define a hardware model in terms of switches, gates, RTL, or behavioral code. Also, a designer needs to learn only one language for stimulus and hierarchical design. Most popular logic synthesis tools support Verilog HDL. This makes it the language of choice for designers. All fabrication vendors provide Verilog HDL libraries for postlogic synthesis simulation. Thus, designing a chip in Verilog HDL allows the widest choice of vendors. The Programming Language Interface PLI is a powerful feature that allows the user to write custom C code to interact with the internal data structures of Verilog. Designers have responded by designing at higher levels of abstraction. Designers have to think only in terms of functionality. EDA tools take care of the implementation details. With designer assistance, EDA tools have become sophisticated enough to achieve a close-to-optimum implementation. Behavioral synthesis allowed engineers to design directly in terms of algorithms and the behavior of the circuit, and then use EDA tools to do the translation and optimization in each phase of the design. However, behavioral synthesis did not gain widespread acceptance. Today, RTL design continues to be very popular. Verilog HDL is also being constantly enhanced to meet the needs of new verification methodologies. Formal verification and assertion checking techniques have emerged. Formal verification applies formal mathematical techniques to verify the correctness of Verilog HDL descriptions and to establish equivalency between RTL and gate-level netlists. However, the need to describe a design in Verilog HDL will not go away. Assertion checkers allow checking to be embedded in the RTL code. This is a convenient way to do checking in the most important parts of a design. New verification languages have also gained rapid acceptance. These languages also provide support for automatic stimulus creation, checking, and coverage. However, these languages do not replace Verilog HDL. They simply boost the productivity of the verification process. Verilog HDL is still needed to describe the design. For very high-speed and timing-critical circuits like microprocessors, the gate-level netlist provided by logic synthesis tools is not optimal. In such cases, designers often mix gate-level description directly into the RTL description to achieve optimum results. This practice is opposite to the high-level design paradigm, yet it is frequently used for high-speed designs because designers need to squeeze the last bit of timing out of circuits, and EDA tools sometimes prove to be insufficient to achieve the desired results. Another technique that is used for system-level design is a mixed bottom-up methodology where the designers use either existing Verilog HDL modules, basic building blocks, or vendor-supplied core blocks to quickly bring up their system simulation. This is done to reduce development costs and compress design schedules. Hierarchical Modeling Concepts Before we discuss the details of the Verilog language, we must first understand basic hierarchical modeling concepts in digital design. The designer must use a "good" design methodology to do efficient Verilog HDL-based design. In this chapter, we discuss typical design methodologies and illustrate how these concepts are translated to Verilog. A digital simulation is made up of various components. We talk about the components and their interconnections. Define a stimulus block and a design block. Explain two methods of applying stimulus. In a top-down design methodology, we define the top-level block and identify the sub-blocks necessary to build the top-level block. We further subdivide the sub-blocks until we come to leaf cells, which are the cells that cannot further be divided. Figure shows the top-down design process. Top-down Design Methodology In a bottom-up design methodology, we first identify the building blocks that are available to us. We build bigger cells, using these building blocks. These cells are then used for higher-level blocks until we build the top-level block in the design.

Chapter 6 : Palnitkar, Verilog HDL, 2nd Edition | Pearson

For Verilog HDL digital IC and system design professionals. Verilog HDL is a language for digital design, just as C is a language for programming. This complete Verilog HDL reference progresses from the basic Verilog concepts to the most advanced concepts in digital design.

Chapter 7 : VerilogÂ® HDL: A Guide to Digital Design and Synthesis, Second Edition [Book]

Part 1 Basic Verilog Topics [j LJ LJ [j [j [j [j [J [J Overview of Digital Design with Verilog HDL Evolution of CAD, emergence of HDLs, typical HDL-based design flow, why.

Chapter 8 : Verilog HDL - A Guide To Digital Design And Synthesis - PDF Free Download

digital design and is the basis for synthesis, verification, and place and route technologies. Samir's book is an excellent guide to the user of the Verilog language.

Chapter 9 : HDL- Verilog Tutorial

Verilog HDL: A Guide to Digital Design and Synthesis, Second Edition By Samir Palnitkar Publisher: Prentice Hall PTR Pub Date: February 21, ISBN: Pages: Written for both experienced and new users, this book gives you broad coverage of Verilog HDL.