Digital System Design
with Verilog

Adapted from Z. Navabi
Portions Copyright Z. Navabi, 2006

Digital System Design
Automation with Verilog

Digital Design Flow
- Design entry
- Testbench in Verilog
- Design validation
- Compilation and synthesis
- Post-synthesis simulation
- Timing analysis
- Hardware generation

Verilog HDL
- Verilog evolution
- Verilog attributes
- The verilog language
Digital System Design Automation with Verilog

- As the size and complexity of digital systems increase, more computer-aided design (CAD) tools are introduced into the hardware design process.
- Early simulation and primitive hardware generation tools have given way to sophisticated design entry, verification, high-level synthesis, formal verification, and automatic hardware generation and device programming tools.
- Growth of design automation tools is largely due to hardware description languages (HDLs) and design methodologies that are based on these languages.
- Based on HDLs, new digital system CAD tools have been developed and are now widely used by hardware designers.
- One of the most widely used HDLs is the Verilog HDL.
- Because of its wide acceptance in digital design industry, Verilog has become a must-know for design engineers and students in computer, hardware-related fields.

Digital Design Flow

* FPGA Design Flow
Digital Design Flow

- Digital Design Flow begins with specification of the design at various levels of abstraction.

- **Design entry phase:** Specification of design as a mixture of behavioral Verilog code, instantiation of Verilog modules, and bus and wire assignments

```verilog
module testbench ();
generate data ;
process data ;
endmodule
```

```
module design (...);
assign ...
always ...
comp1( ...)
endmodule
```

```
always (posedge clk )
begin ...
end
```

```
if (...) bus = w;
else ...
```

```
assert U1 (...);
assert U2 (...);
...
```

```
Compu U1 (...);
Compu U2 (...);
...
```

Violations Report:
Time of Violation:
Monitor Coverage

Pass / Fail Report
Property Coverage
Counter Examples
Digital Design Flow

- **FPGA Design Flow (Continued)**

  * Presynthesis verification: Generating testbenches for verification of the design and later for verifying the synthesis output
Digital Design Flow

- **Synthesis process**: Translating the design into actual hardware of a target device (FPGA, ASIC or custom IC)
FPGA Design Flow (Continued)

- **Postsynthesis simulation**: Testing the behavioral model of the design and its hardware model by using presynthesis test data.
Digital Design Flow

- FPGA Design Flow (Continued)

Digital Design Flow ends with generating netlist for an application specific integrated circuits (ASIC), layout for a custom IC, or a program for a programmable logic devices (PLD)
Digital Design Flow

Design Entry
- Testbench in Verilog
- Compilation and Synthesis
- Timing Analysis
- Hardware Generation

Design Validation
- Design Entry
- Compilation and Synthesis
- Timing Analysis
- Hardware Generation

Postsynthesis Simulation
- Design Entry
- Compilation and Synthesis
- Timing Analysis
- Hardware Generation

Simulation
- Design Entry
- Compilation and Synthesis
- Timing Analysis
- Hardware Generation

Hardware Generation
- Design Entry
- Compilation and Synthesis
- Timing Analysis
- Hardware Generation

Design Entry
- Testbench in Verilog
- Compilation and Synthesis
- Timing Analysis
- Hardware Generation

Design Validation
- Design Entry
- Compilation and Synthesis
- Timing Analysis
- Hardware Generation

Postsynthesis Simulation
- Design Entry
- Compilation and Synthesis
- Timing Analysis
- Hardware Generation

Simulation
- Design Entry
- Compilation and Synthesis
- Timing Analysis
- Hardware Generation

Hardware Generation
- Design Entry
- Compilation and Synthesis
- Timing Analysis
- Hardware Generation
Design Entry

- The first step in the design of a digital system
- Describing the design in Verilog in a top-down hierarchical fashion
- Register Transfer Level (RTL): High-level Verilog designs usually described at this level
- Verilog constructs used in RT level design:
  - `procedural statements` for high-level behavioral description
  - `continuous assignments` for representing logic blocks, bus assignments, and bus and input/output interconnect specifications
  - `instantiation statements` for using lower-level components in an upper-level design

Testbench in Verilog

Digital Design Flow

- Design Entry
- Design Validation
- Postsynthesis Simulation
- Hardware Generation

Testbench in Verilog
- Compilation and Synthesis
- Timing Analysis
Testbench in Verilog

- Simulation and Test of a designed system functionality before Hardware generation
- Detection of design errors and incompatibility of components used in the design
- By generation of a test data and observation of simulation results
- **Testbench**: A Verilog module
  - Use of high-level constructs of Verilog for:
    - Data Generation
    - Response Monitoring
    - Handshaking with the design
  - Inside the Testbench: Instantiation of the design module
  - Forms a simulation model together with the design, used by a Verilog simulation engine

Design Validation

- **Digital Design Flow**
  - Design Entry
  - Testbench in Verilog
  - Compilation and Synthesis
  - Timing Analysis
  - Postsynthesis Simulation
  - Hardware Generation
  - Design Validation
Design Validation

- An important task in any digital system design
- The process to check the design for any design flaws

- A design flaw due to:
  - Ambiguous Problem Specifications
  - Designer Errors
  - Incorrect Use of Parts in the Design

- Can be done by:
  - Simulation
  - Assertion Verification
  - Formal Verification
**Simulation**

- Simulation for design validation, done before a design is synthesized
- Also referred to as RT level, or Pre-synthesis Simulation
- Simulation at RTL level is accurate to the clock level
- The advantage: its speed compared with simulations at the gate or transistor levels
- The required test data: generated graphically using waveform editors, or through a testbench
- Outputs of simulators:
  - Waveforms (for visual inspection)
  - Text for large designs for machine processing
Two alternatives for defining test input data for a simulation engine

* Using a Testbench or a Waveform Editor for Simulation

**Verilog Code of a Counter Circuit**

```verilog
module Chap1CounterTester();
    reg Clk, Reset;
    wire [3:0] Count;
    initial begin
        Reset = 0; #5 Reset = 1; #115 Reset = 0;
        $stop;
    end
    always @Clk
        Chap1Counter #1 (Clk, Reset, Count);
endmodule
```

* Verilog Simulation with a Testbench

The simulation results in form of a waveform
Simulation

The testbench instantiates the design under test, and as part of the code of the testbench it applies test data to the instantiated circuit.

```verilog
module Chap1CounterTester();
    reg Clk=0, Reset=0;
    wire [3:0] Count
    initial begin
        Reset = 0; #5 Reset = 1; #115 Reset = 0;
        # 760 $stop;
        end
    always #26.5 Clk = ~ Clk;
    Chap1Counter U1 (Clk, Reset, Count);
endmodule
```

* Verilog Simulation with a Testbench (Continued)

The testbench instantiates the design under test, and as part of the code of the testbench it applies test data to the instantiated circuit.

```verilog
module Chap1Counter (Clk, Reset, Count);
    input Clk, Reset
    output [3:0] Count
    reg [3:0] Count
    always @(posedge Clk) begin
        if (Reset) Count = 0;
        else Count = Count + 1;
    end
endmodule
```

* Verilog Simulation with a Testbench (Continued)

---

Simulation

<table>
<thead>
<tr>
<th>Name</th>
<th>V...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clk</td>
<td>1</td>
</tr>
<tr>
<td>Reset</td>
<td>0</td>
</tr>
<tr>
<td>Count</td>
<td>0-8</td>
</tr>
</tbody>
</table>

Validates the functionality of the counter circuit being tested, Regardless of clock frequency
Simulation

- Obviously, an actual hardware component behaves differently.
- Based on the timing and delays of the parts used, there will be a nonzero delay between the active edge of the clock and the counter output.
- Furthermore, if the clock frequency applied to an actual part is too fast for propagation of values within the gates and transistors of a design, the output of the design becomes unpredictable.
- The simulation shown here is not provided with the details of the timing of the hardware being simulated.
- Therefore, potential timing problems of the hardware that are due to gate delays cannot be detected.
- This is typical of a presynthesis or high-level behavioral simulation.

Assertion Verification

Design Validation

Simulation

Assertion Verification

Formal Verification
Assertion Verification

- **Assertion Monitors**: Used to continuously check for design properties during simulation.
- Instead of having to inspect simulation results manually or by developing sophisticated testbenches.
- **Design Properties**: Certain conditions have to be met for the design to function correctly.
- Assertion Monitors developed to assert that the Design Properties are not violated.
- Firing of an assertion verification: alerts the malfunctioning of design according to the designer’s expectation.
- **Open Verification Library (OVL)**: provides a set of assertion monitors for monitoring common design properties.

---

Formal Verification

- **Design Validation**
- **Simulation**
- **Assertion Verification**
- **Formal Verification**
Formal Verification

- **Formal verification**: The process of checking a design against certain properties
- Examining the design to make sure that the described properties by the designer to reflect correct behavior of the design hold under all conditions
- **Property's Counter Examples**: Input conditions making a property to fail
- Property coverage indicates how much of the complete design is exercised by the property

Compilation and Synthesis

Digital Design Flow

- Design Entry
- Design Validation
- Postsynthesis Simulation
- Hardware Generation
- Testbench in Verilog
- Compilation And Synthesis
- Timing Analysis
Compilation and Synthesis

- **Synthesis**: The process of automatic hardware generation from a design description that has an unambiguous hardware correspondence.
- A Verilog description for synthesis:
  - Cannot include signal and gate level timing specifications, file handling, and other language constructs that do not translate to sequential or combinational logic equations
  - Must follow certain styles of coding for combinational and sequential circuits
- Compilation process has three phases:
  - **Analysis Phase**
  - **Synthesis Phase**
  - **Routintg and Placement Phase**

**Compilation and Synthesis Process**

**Input**: Hardware description consisting of various levels of Verilog

**Output**: A detailed hardware for programming an FPGA or manufacturing an ASIC
Compilation and Synthesis

**Analysis Phase:** Translates various parts of the design to an intermediate format.

```
module design (...);
    assign ...
    always ...
    compi(...)
endmodule
```

* Compilation and Synthesis Process (Continued)

**Synthesis Phase:** Links all parts together and generates the corresponding logic.

```
always (posedge clk )
    begin ...
    end
if (...) bus = w;
else ...
```

* Compilation and Synthesis Process (Continued)
Compilation and Synthesis

Routing and Placement Phase: Places and routes components of the target hardware, and generates timing details.

* Compilation and Synthesis Process (Continued)
Before the complete design is turned into hardware
Analyzing the design and generating a uniform format for all parts of it
Also checks the syntax and semantics of the input Verilog code
Generic Hardware Generation

- **Generic Hardware Generation**: Turning the design into a generic hardware format such as a set of Boolean expressions or a netlist of basic gates
Logic Optimization

- Logic Optimization:
  - Reducing expressions with constant input
  - Removing redundant logic expressions
  - Two-level minimization
  - Multilevel minimization that include logic sharing
- Output:
  - Boolean expressions
  - Tabular logic representations
  - Primitive gate netlists
Binding

- **Binding**:  
  - Decide exactly what logic elements and cells are needed for the realization of the circuit using information from target hardware.  
  - Output is specific to the FPGA, ASIC, or custom IC being used.
Routing and Placement

- Decides on the placement of cells of the target hardware
- Determines wiring of inputs and outputs of the cells through wiring channels and switching areas of the target hardware
- The output is specific to the hardware being used and can be used for programming an FPGA or manufacturing an ASIC.
* An Example Synthesis Run (Continued)

Routing and Placement

module Chap1Counter (Clk, Reset, Count);
    input Clk, Reset;
    output [3:0] Count;
    reg [3:0] Count;
    always @(posedge Clk) begin
        if (Reset) Count = 0;
        else Count = Count + 1;
    end
endmodule

An example of a synthesis run: The counter circuit is being synthesized

Design to Synthesize

Target hardware specification
- List of primitive components
- Flip-flops
- Logic elements
- Timing specifications
- Pin-to-pin timing

Back to Design to Synthesize

Verilog Description of the Design

Synthesis Tool

* An Example Synthesis Run (Continued)
Routing and Placement

The output of synthesis tool

An Example Synthesis Run (Continued)

* A list of gates and flip-flops available in the target hardware and their interconnections

Postsynthesis Simulation

Digital Design Flow

- Design Entry
- Design Validation
- Postsynthesis Simulation
- Testbench in Verilog
- Compilation and Synthesis
- Timing Analysis
- Hardware Generation

CSE 467
Verilog Digital System Design
53

CSE 467
Verilog Digital System Design
54
Post-synthesis Simulation

- After the Synthesis Phase a complete netlist of target hardware components and their timings is generated.
- The generated netlist includes:
  - The details of gates used for the implementation of the design
  - Wiring delays and load effects on gates used in the postsynthesis design
- The netlist output is made available in various netlist formats including Verilog
- A Postsynthesis simulation checks:
  - Timing issues
  - Determination of a proper clock frequency
  - Determination of race, and hazard considerations
- The behavior of a design as intended by the designer and its behavior after postsynthesis simulation may be different due to delays of wires and gates.

Timing Analysis

Digital Design Flow

- Design Entry
- Design Validation
- Postsynthesis Simulation
- Testbench in Verilog
- Compilation and Synthesis
- Timing Analysis
- Hardware Generation
Timing Analysis

- A part of the compilation process, or in some tools after the compilation process
- Timing Analysis Phase generates:
  - Worst-case delays
  - Clocking speed
  - Delays from one gate to another
  - Required setup and hold times
- Results of timing analysis appear in Tables and/or Graphs
- The results is used by designers to decide on speed of their circuits.

Hardware Generation

Digital Design Flow

- Design Entry
- Design Validation
- Postsynthesis Simulation
- Testbench in Verilog
- Compilation and Synthesis
- Timing Analysis
- Hardware Generation
Hardware Generation

- Last stage in an automated Verilog-based design
- Generates a netlist for ASIC manufacturing, a program for programming FPGAs, or layout of custom IC cells

Verilog HDL

- Verilog Evolution
- Verilog Attributes
- The Verilog Language
Verilog HDL

- A language that can be understood by:
  - System Designers
  - RT Level Designers
  - Test Engineers
  - Synthesis Tools
  - Machines
- Has become an IEEE standard

Verilog Evolution

Verilog HDL

Verilog Evolution

Verilog Attributes

The Verilog Language
Verilog Evolution

- Designed in early 1984 by Gateway Design Automation
- Originally used as a simulation and verification tool
- After the initial acceptance of this language by electronic industry, a fault simulator, a timing analyzer, and later in 1987, a synthesis tool was developed based on this language.
- Since acquiring Gateway Design Automation and its Verilog-based tools by Cadence Design System, Cadence has been a strong force behind popularizing the Verilog hardware description language.
- In 1987 VHDL became an IEEE standard hardware description language.
- VHDL was adapted by the U.S. government for related projects and contracts.
- In an effort for popularizing Verilog, in 1990, OVI (Open Verilog International) was formed and Verilog was placed in public domain.

Verilog Evolution

- Additional Features of Verilog-2001:
  - New features for external file access for read and write
  - Library management
  - Constructs for design Configuration
  - Higher abstraction level constructs
  - Constructs for specification of iterative structures
Verilog Attributes

Verilog is a hardware description language for describing hardware from transistor level to behavioral.

- Supports timing constructs for switch level timing simulation and at the same time, has features for describing hardware at the abstract algorithmic level.
- A Verilog description may consist of a mix of modules at various abstraction levels with different degrees of detail.
Switch Level

- Features of the language for switch level modeling and simulation:
  - Primitive unidirectional and bidirectional switches with parameters for delay and charge storage
  - Circuit delays may be modeled as propagation delay, rise and fall delay, and line delays.
  - The charge storage feature for describing dynamic complimentary metal oxide semiconductor (CMOS) and metal oxide semiconductor (MOS) circuits.

Gate Level

- Verilog Attributes
  - Switch Level
  - Pin-To-Pin Delay
  - Behavioral Level
- Gate Level
  - Bussing Specifications
  - System Utilities
- PLI
Gate Level

- Gate level primitives with predefined parameters provide a convenient platform for:
  - netlist representation
  - gate level simulation.
- For more detailed and special purpose gate simulations:
  - Gate components defined at the behavioral level.
- Verilog provides utilities for defining primitives with special functionalities:
  - A simple 4-value logic system used for signal values
  - 16 levels of strength in addition to the four values for more accurate logic modeling

Pin-To-Pin Delay

Verilog Attributes

Switch Level  Gate Level

Pin-To-Pin Delay  Bussing
Behavioral Specifications
Level

System Utilities

PLI
**Pin-To-Pin Delay**

- Verilog provides a utility for timing specification of components at the input/output level:
  - Can be used for back annotation of timing information in original predesigned descriptions
  - Enables modelers to fine-tune timing behavior of their models based on physical implementations

---

**Bussing Specifications**

- **Verilog Attributes**
  - Switch Level
  - Pin-To-Pin Delay
  - Behavioral Level
  - System Utilities
  - PLI
  - Gate Level
  - **Bussing Specifications**
Bussing Specifications

- Verilog provides:
  - Bus and register modeling utilities
  - For various bus structures, predefined wire and bus resolution functions using the 4-value logic value system.

- Combination of bus logic and resolution-functions enable modeling of most physical bus types.

- For register modeling, high-level clock representation and timing-control constructs can be used for representation of registers with various clocking and resetting schemes.

Behavioral Level

- Verilog Attributes
  - Switch Level
  - Pin-To-Pin Delay
  - Behavioral Level
  - Gate Level
  - Bussing Specifications
  - System Utilities
  - PLI
Behavioral Level

- Procedural blocks in Verilog enable algorithmic representations of hardware structures.
- Constructs similar to those in software programming languages are provided for describing hardware at this level.

System Utilities

- Verilog Attributes
  - Switch Level
  - Pin-To-Pin Delay
  - Behavioral Level
- Gate Level
  - Bussing Specifications
- PLI
  - System Utilities
System Utilities

- System tasks in Verilog provide designers with tools for:
  - Testbench generation
  - File access for read and write
  - Data handling
  - Data generation
  - Special hardware modeling.

- System utilities for reading memory and programmable logic array (PLA) images provide convenient ways of modeling these components.
- Verilog display and I/O tasks can be used to handle all inputs and outputs for data application and simulation.
- Verilog allows random access to files for read and write operations.

Program Language Interface

Verilog Attributes

- Switch Level
- Pin-To-Pin Delay
- Behavioral Level

Gate Level
- Bussing Specifications

System Utilities

PLI
Program Language Interface

- The Programming Language Interface (PLI) for Verilog is a mechanism to interface Verilog programs with programs written in the C language. It also provides mechanisms to access internal databases of the simulator from C programs.

- PLI is used for implementing system calls which would be hard to do otherwise (or impossible) using Verilog syntax. Or, in other words, you can take advantage of both paradigms - the parallel and hardware related features of Verilog and the sequential flow of C - using the PLI.

- Some of the most common applications of PLI are delay back annotation, writing delay calculators and developing user interface.

The Verilog Language

- Verilog HDL
  - Verilog Evolution
  - Verilog Attributes
  - The Verilog Language
The Verilog Language

- The Verilog HDL satisfies all requirements for design and synthesis of digital systems:
  - Supports hierarchical description of hardware from system to gate or even switch level.
  - Has strong support at all levels for timing specification and violation detection.
  - Timing and concurrency required for hardware modeling are specially emphasized in it.
  - A hardware component is described by the `module_declaration` language construct in it.

The Verilog Language (Continued):

- Description of a module specifies a component's input and output list as well as internal component busses and registers within a `module`, concurrent assignments, component instantiations, and procedural blocks can be used to describe a hardware component.
- Several modules can hierarchically be instantiated to form other hardware structures.
- Many Verilog tools and environments exist that provide simulation, fault simulation, formal verification, and synthesis.
- Simulation environments provide graphical front-end programs and waveform editing and display tools.
- Synthesis tools are based on a subset of Verilog.