Module 2

Describing Verilog Applications

cadence

This module introduces the definition and use of a hardware description language.

Module Objective

In this module, you learn:

- The nature and use of a Hardware Description Language (HDL)

Topics

- What is an HDL?
- Levels of abstraction
- Benefits of using HDLs
- The Verilog HDL

cadence

Your objective is to be able to explain the nature and use of a Hardware Description Language. To do that, you need to know what a HDL is, what they are used for, and who uses one.

Terms and Definitions

Hardware Description Language (HDL)	A programming-like language that describes the functionality of digital electronic hardware circuits
Simulator	Software that imitates the functionality of electronic hardware as described by the HDL
Level of Abstraction	The level of detail provided by the utilized modeling style, such as behavioral and gate level
Application Specific Integrated Circuit (ASIC)	Integrated circuit developed for a specific application
ASIC Vendor	The company manufacturing the chips. This company is responsible for developing and providing the library cells
Bottom-Up Design Flow	A design methodology in which you build the low-level components first and then connect them to make large systems
Top-Down Design Flow	A design methodology in which you define the system at a very high level of abstraction and then iteratively refine it
Register Transfer level (RTL)	A level of abstraction that defines storage elements and their clocks
Tool Command Language (Tcl)	A scripting language used for issuing commands to interactive programs

cadence

- A Hardware Description Language (HDL) is a programming-like language that describes the functionality of digital electronic hardware circuits.
- A Simulator is software that imitates the functionality of electronic hardware as described by the HDL.
- A Level of Abstraction is the level of detail provided by the utilized modeling style, such as behavioral and gate level.
- An Application Specific Integrated Circuit (ASIC) is an integrated circuit developed for a specific application.
- An ASIC vendor is a company that manufactures ASICs. The company provides the cell libraries for its technology.
- A Bottom-Up Design Flow is a design methodology in which you build the low-level components first and then connect them to make larger systems.
- A Top-Down Design Flow is a design methodology in which you define the system at a very high level of abstraction and then iteratively refine it.
- The Register Transfer level (RTL) is a level of abstraction that defines storage elements and their clocks.
- The Tool Command Language (Tcl) is a scripting language used for issuing commands to interactive programs.

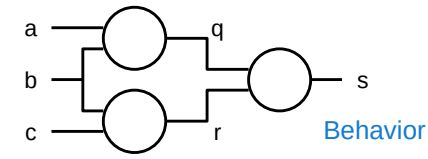
What Is a Hardware Description Language (HDL)?

An HDL is a programming language with special constructs for Modeling hardware concurrency and timing.

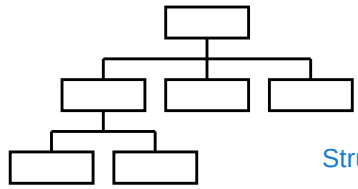
An HDL supports design at multiple levels of abstraction:

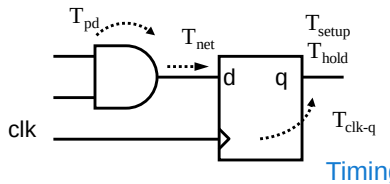
- Behavioral Modeling
  - Sequential behaviors
  - Parallel behaviors
- Structural Modeling
  - Hardware component hierarchy
  - Software subroutine hierarchy

An HDL typically supports simulation of estimated design timing.



Behavior


Structure



Timing

An HDL is similar to a procedural programming language, but also contains constructs to describe digital electronic systems. An HDL contains features and constructs to support description of:

- Behavior – Both serial and parallel. In serial behavior, you pass the output of one functional block to the input of another, which is similar to the behavior of a conventional software language. However in parallel behavior, you can pass a block output to the inputs of a number of blocks acting in parallel where many separate events happen at the same moment in time.
- Structure – Both physical, such as hierarchical block diagrams and component netlists, and software, such as subroutines. This allows you to describe large, complex systems and manage their complexity.
- Time – Programming languages have no concept of time. An HDL has to model propagation delays, clock periods, and timing checks.

An HDL typically supports multiple abstraction levels. You can describe hardware behaviorally, without and with sufficient detail for logic synthesis, and as a structured netlist of predefined components that can themselves be as simple as a transistor and as complex as another behavioral design.

An HDL Supports Multiple Levels of Abstraction

Behavioral Level

- Algorithms

Register Transfer Level (RTL)

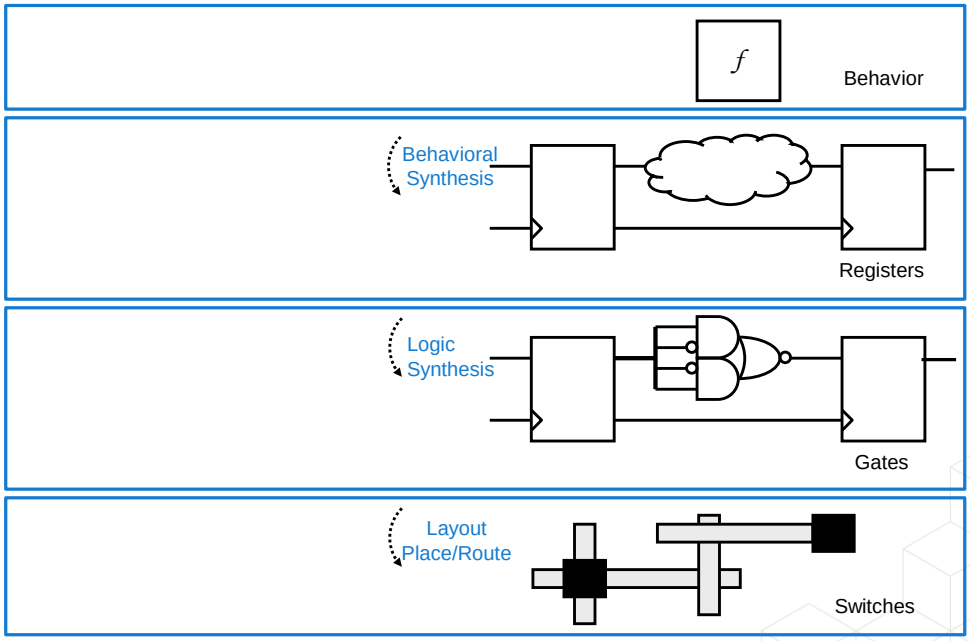
- Nets and registers

Gate Level

- Built-in and user-defined primitives

Switch Level

- Built-in switch primitives



You can describe a system as a group of hierarchical models of varying amount of detail.

An HDL supports multiple levels of such detail. The three main levels of abstraction are:

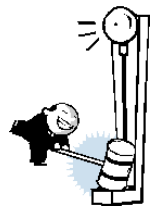
- The behavioral level:
  - You describe the system using mathematical equations; and
  - You can omit timing – the system may simulate in zero-time like a software program.
- The Register Transfer Level (RTL):
  - You partition the system into combinational and sequential logic, using constructs and coding styles supported by logic synthesis; and
  - You define timing in terms of cycles based on one or more defined clock(s).
- The structural level:
  - You instantiate and interconnect predefined components;
  - Which can include vendor-provided macrocells; and
  - Which can include logic primitives built into the language.

Abstraction Level Tradeoffs

Simulation effort is proportional to detail:

- Behavioral Level
  - Algorithms
- Register Transfer Level (RTL)
  - Nets and registers
- Gate Level
  - Built-in and user-defined primitives
- Switch Level
  - Built-in switch primitives

- Less detail
  - Faster design entry
  - Faster simulation

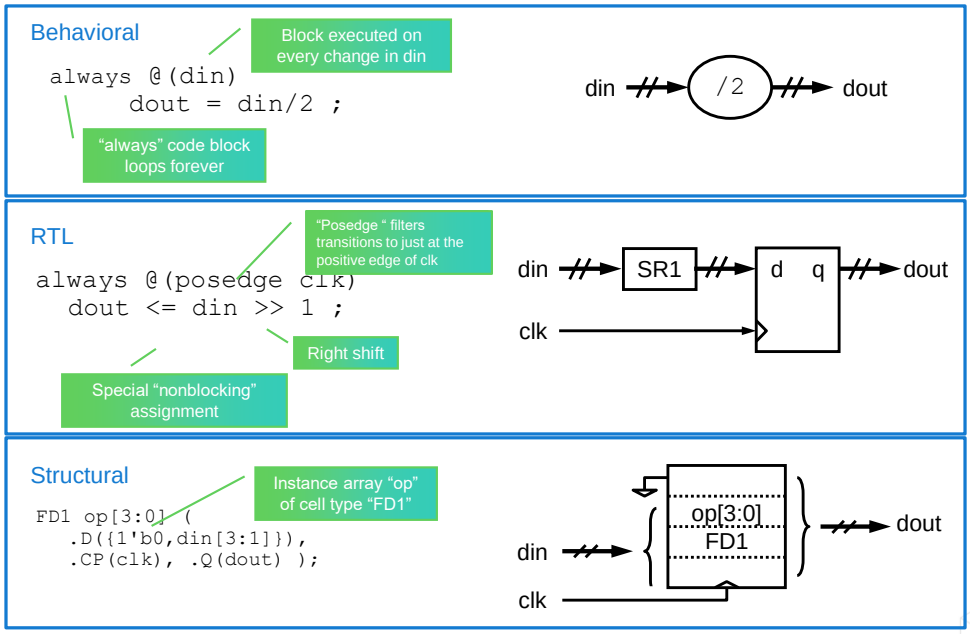


- More detail
  - Slower design entry
  - Slower simulation

Users of an HDL model at different levels of abstraction:

- Block architects model functionality at the behavioral level for maximum simulation speed and for the ease with which they can quickly modify the architecture as they explore alternatives;
- Block implementers refine the functional blocks to the RT level for a logic synthesis tool; and
- Library developers model cells at the gate level for simulation and the physical level for place and route tools.

Abstraction Level Example: Divide by 2



15

These code fragments illustrate the three main levels of abstraction:

- The behavioral level of abstraction describes the design behavior with no hint about how the operation is implemented;
- The RT level of abstraction describes the design behavior with sufficient detail that logic synthesis can infer an implementation involving an edge-triggered storage device; and
- The structural level of abstraction instantiates and connects a predefined storage device from a macro library, not caring how the component is itself implemented.

Why Use a Hardware Description Language?

The benefits of using an HDL include the following:

- You write HDL in simple ASCII text.
  - Capture the design quickly and easily manage modifications
- You can design at a higher abstraction level.
  - Easily explore design alternatives
  - Find problems earlier in the design cycle
- Use of a standard HDL enables design reuse.
  - Between design teams and partners, and through the design flow
  - No re-entry, reformatting, or translation
- The description is independent of the implementation.
  - You can delay the choice of implementation technology
  - You can more easily make architectural and functional changes
  - You can more easily adapt the design to future projects

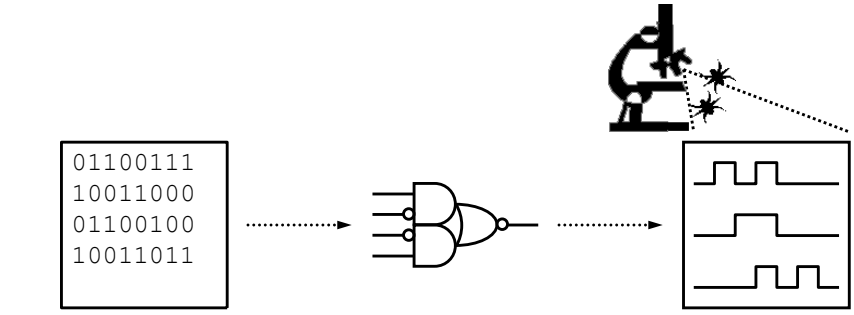
16

cadence

Benefits of using an HDL include the following:

- You can capture and modify the design more quickly in an HDL than by schematic capture.
- Earlier design capture at the higher abstraction level means earlier simulation, which facilitates design alternative exploration to produce a more optimal architecture and partitioning, and allows a more thorough verification.
- Use of a standard HDL facilitates reuse of designs from previous projects or from commercial Intellectual Property (IP) providers. You can move or switch between different tools and vendors without re-entry, reformat or translation, of the design description.
- Your RT-level implementation is almost 100% independent of the implementation technology, so you can defer selection of the target ASIC or FPGA technology until the design is mostly entered, and you can switch technology or vendor with a minimum of redesign.

Legacy Schematic-Based Design



Low-level, Proprietary, Non-Portable Stimulus

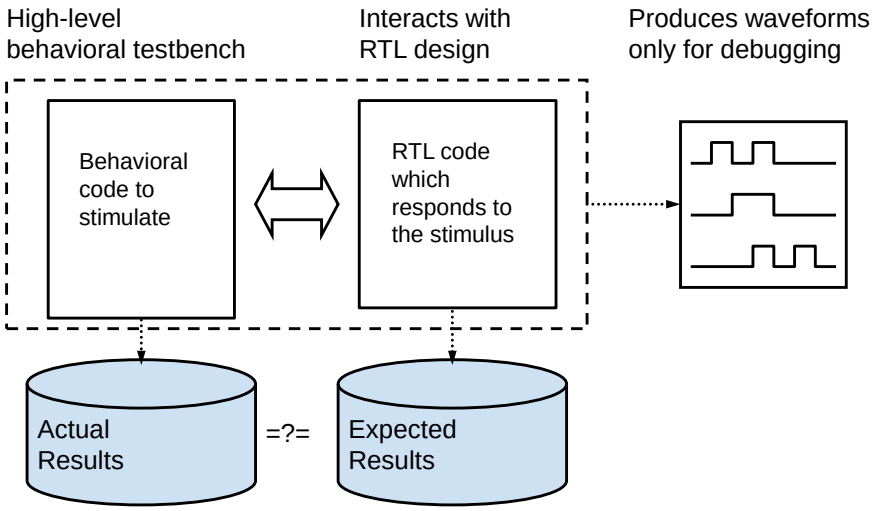
Apply to captured netlist

Observe waveform, manually verify function and timing

Let's compare to an HDL ...

Designers used to capture the design intent using a proprietary schematic capture tool, manually developed test stimulus in a proprietary tabular format, and simulated the design using a proprietary simulator. Those who had money to burn had graphical displays to examine the results, but most of us had plain old text terminals, so could examine the results only as the 0 and 1 characters. We did not use the Z and X characters because those values do not exist in hardware and we used simulation only to generate expected results for a device test machine.

HDL-Based Simulation



- Testbench can be self-checking
- Design and tests vendor independent

In more recent times, we capture the design intent and its test using the same standard HDL and simulate them together using a choice of tools running on a wide choice of third-party platforms and we have high-definition wide-screen displays that we do not have to stare at quite so much because our tests to a large extent pinpoint any problems with a high degree of accuracy.

Who Uses Hardware Description Languages?

The following personnel use an HDL:

- System architects doing high-level architectural exploration
- Verification personnel testing components and systems
- Hardware designers implementing RT-level code for synthesis
- Model developers describing system-level IP and ASIC/FPGA macrocells

cadence

- System architects create system-level HDL models for high level architectural exploration;
- Verification personnel create HDL testbenches for testing components and systems;
- Hardware designers implement the system-level models as Register Transfer-level HDL for synthesis; and
- Model developers describe system-level IP and ASIC or FPGA macrocells in an HDL.

Some HDL Adoption Issues

Issues in using HDLs include the following:

- Yet one more thing to learn!
  - A new programming language to understand
  - Simulation and synthesis EDA tools to learn
- Major change in design methodology
  - No standard “off the shelf” design methodology
  - Needs to be planned and implemented
- Primarily targets digital design
  - Analog extensions exist
- Design planning and partitioning required before coding
- Coding style influences project time-to-closure
- Other specific issues as we will see during the course

cadence

Issues in using an HDL include that:

- Adopting an HDL requires you to learn a new language, and if you previously used schematic capture, to change to a more software-like design methodology.
- The way in which you write your code can affect the simulation and synthesis efficiency and the area and performance of the product.

The Verilog HDL

The Verilog HDL is used primarily for design, verification, and implementation of digital logic.

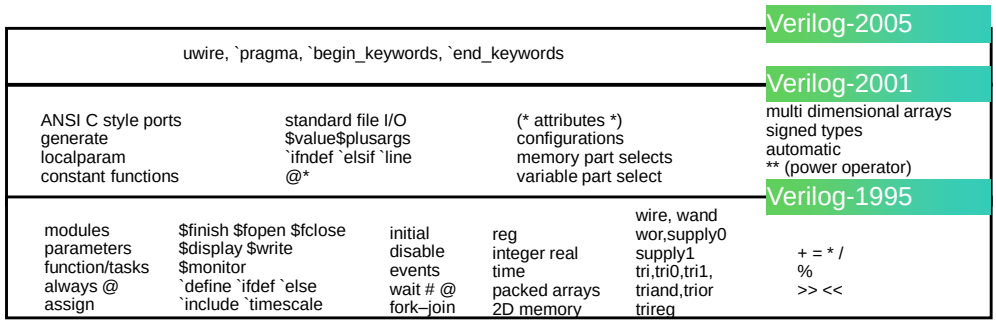
- Developed in 1983 by Phil Moorby and Prabhu Goel at Automated Integrated Design Systems
  - Currently IEEE Std. 1364-2005
- Syntax: Similar to C programming language
- Paradigm: Procedural assignment of values to variables
- Types: Loosely typed Language
- Comprehensive language with no facility for user extensions

cadence

- Verilog was initially developed in 1983 by Phil Moorby and Prabhu Goel at Automated Integrated Design Systems. The Verilog syntax is similar to C, and like C, Verilog is loosely typed and case-sensitive.
- For variable types, Verilog includes only reg, integer, time, and real.
- Verilog has several net types. The net type determines how to resolve the value of a net having multiple drivers.
- Verilog has several transistor and gate-level primitives, a construct for user-defined primitives, and a Programmable Logic Array (PLA) modeling mechanism.
- For nets and primitives, you can optionally specify a propagation delay.
- For continuous assignments and primitives, you can optionally specify a drive strength.
- For modules, you can optional specify path delays, pulse filtering, and timing checks.

Verilog Language Versions

The IEEE Std 1364-2005 had relatively few major extensions over Verilog-2001.



The IEEE Std 1364 standardized in 1995 what was previously the proprietary Verilog hardware description language.

The 2001 update to the Verilog standard added several convenience features, targeted at hardware design and verification. The IEEE Std 1364-2005 had relatively few major extensions over Verilog-2001, such as the uwire type, encryption, a `pragma directive, and a `begin_keywords compatibility directive.

Module Review

What is an HDL?

How is the HDL description independent of the product implementation and why is this an advantage?

What level of abstraction is used in:

- Testbenches?
- Synthesizable designs?
- Netlists?

cadence

This page does not contain notes.

Module Review Solutions

1. What is an HDL?
   - An HDL is a programming language with special constructs for modeling hardware, such as abstraction, concurrent behavior, and timing.
2. How is the HDL description independent of the product implementation and why is this an advantage?
   - The HDL is standard and the synthesizable subset is standard.
     You specify the target technology upon synthesis.
3. What level of abstraction is used in:
   a. Testbenches?
   b. Synthesizable designs?
   c. Netlists?
      - Testbenches – Behavioral
      - Synthesizable designs – RTL
      - Netlists – Structural

cadence

This page does not contain notes.

Lab

Lab 2-1 Exploring the VeriRISC CPU Design

- To become familiar with the lab model, which is a RISC CPU design.

cadence

The objective of this lab is to become familiar with the lab model, which is a RISC CPU Design.

For this part, you examine the lab model description in the lab instructions.