Verilog HDL

EE/CS 499-3

EE/CS 599-3

Brad Olsen

bolsen@amis.com

(208)233-4690 x7137

Section 1

Verilog Overview

Hardware Description Language

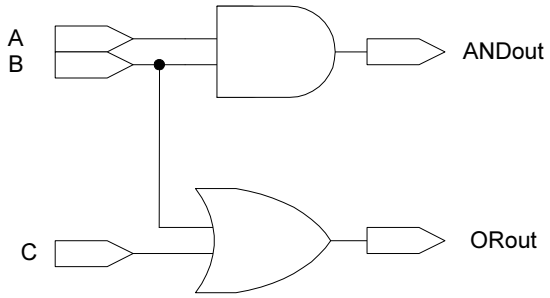
- Hardware Description Language (HDL)
  - High level programming language which describes circuit connectivity.
  - Describes both functionality and timing.
  - Describes the circuit at various levels of abstraction.
  - Has the ability to express concurrency
    - Typical programming languages are sequential.
    - Digital hardware has a number of actions that occur simultaneously.
    - HDL languages allow modeling of concurrent processes.

Concurrency

- Concurrency in Verilog

```
module andor (ANDout, ORout, A,B,C);
input A,B,C;
output OUT;
  assign ANDout = A & B;
  assign Orout = B | C;
endmodule
```

- Concurrency in C?



Conventional programming languages such a 'C' do not handle concurrency very well. Conventional programming languages usually execute the code line-by-line or subroutine by subroutine. The functionality of the OR gate is independent from the AND gate, but they are both active at the same time. Verilog processes each assign statement concurrently. There is no order or priority in which they're processed.

Why Use a HDL?

- Design at a higher level
  - Find problems earlier in design cycle
  - Easily explore design alternatives
- Your design is implementation independent (?)
  - Last minute changes are not catastrophic
  - Implementation decisions can be delayed
  - FPGA, CPLD, ASIC, etc.
- HDLs provide greater flexibility in reuse, choice of tools, and choice of vendors.
  - Cadence, Synopsys, ModelSim, Xilinx, Altera, etc.
- Provides faster design capture and easier to manage.
  - Easier to use than schematic capture.
  - You don't have to worry about Boolean equations, K maps, etc.

Pure HDL RTL code is portable across any design methodology and end process. Unfortunately, it is difficult to avoid implementation depended structures in your RTL code. Typically, precompiled macro cells, memory devices, etc. are vendor specific. We will discuss more about what makes code vendor specific throughout the course.

What is Verilog?

- Verilog is a HDL that facilitates the modeling of digital circuits.
- Allows the circuit to be described at different levels of abstraction.
- Allows the development of tests to verify the functionality of the circuit model.
- Verilog is a different language than VHDL, AHDL, etc.

Verilog History

- Verilog was written to be used by circuit designers.
- Created by Gateway Design Automation in the early ‘80s.
- Gateway was acquired by Cadence in 1990.
- Released as a public domain language in 1991

Verilog History (cont.)

- Open Verilog International (OVI) was formed in 1991 for the evolution, maintenance, and promotion of the language
- Ratified as IEEE standard 1364 in 1995.
- New IEEE standard 1364-2001.

Although there was a new Verilog standard released by IEEE in 2001. Most of the EDA vendors have yet to implement *all* of the new features. Therefore, this course will focus primarily on the pre-2001 Verilog syntax.

For more information on Verilog and the standards:

http://www.verilog.com

http://www.verilog-2001.com/

http://www.sutherland-hdl.com

Levels of Abstraction

- Verilog allows various levels of design abstraction.
  - Behavioral Modeling
  - Register Transfer Level (RTL) Modeling
  - Gate Level Modeling
  - Switch Level Modeling
- All levels of abstraction can be used in the same design.

-Behavioral modeling is used for high-level system design and modeling.

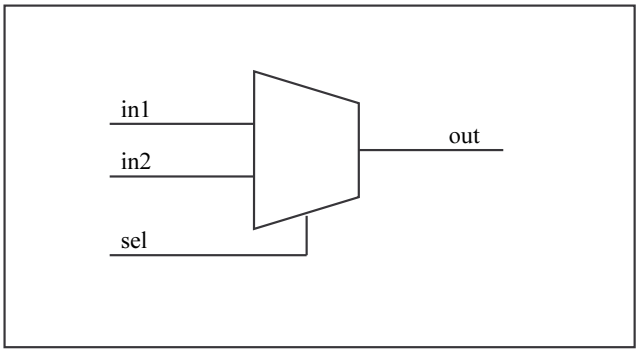
-RTL modeling is used for designs that will eventually be implemented at the chip level.

-Gate level modeling is used mostly for simulation cells, but can be used in RTL code for glue logic.

-Switch level modeling is available in Verilog, but is typically not used other than in some simulation cell development. Typically, switch level definitions for a design are non-Verilog and are handled by other EDA tools.

Levels of Abstraction (cont.)

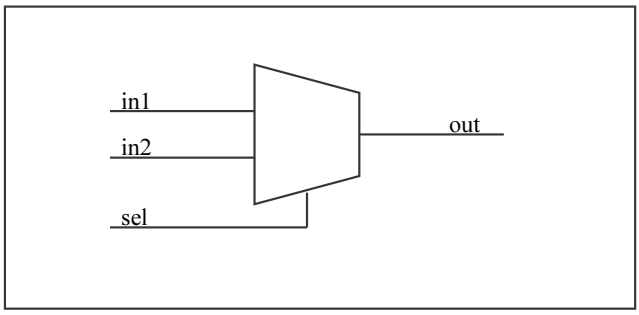
- Behavioral Modeling
  - Uses high level language constructs
  - Describes a design in terms of its algorithms.




```
module twoinmux (out, in1, in2, sel);

input in1, in2, sel;
output out;

assign #5 out = sel ? in2 : in1;

endmodule
```

Levels of Abstraction (cont.)

- RTL Modeling
  - Synthesizable
  - Describes a design in terms of its flow of data and control signals.
  - Assignments occur at signal edges



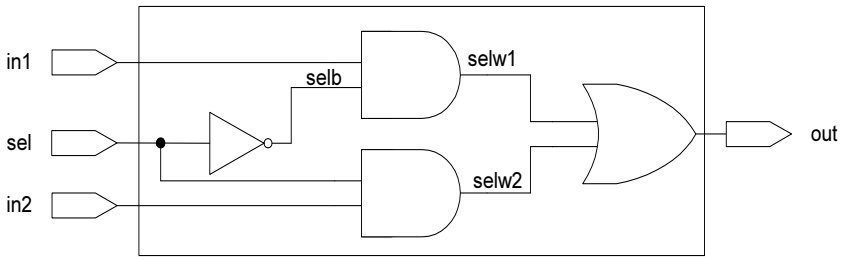

```
module twoinmux (out, in1, in2, sel);

input in1, in2, sel;
output out;
reg out;

always @(sel or in1 or in2)
  if (sel)
    out = in2;
  else
    out = in1;

endmodule
```


This RTL code is a subset of Verilog behavioral code. The behavioral example on the previous page is also RTL code.

The line between pure RTL code and general behavioral code can be fuzzy at times. Throughout this course, you will learn about RTL code, and what differentiates it from general behavioral code.

Levels of Abstraction (cont.)

- Gate Level (Structural) Modeling
  - Describes interconnection of gates.
  - Also called “netlist” format




```
module twoinmux (out, in1, in2, sel);

input in1, in2, sel;
output out;

not  u1 (selb, sel);
and u2 (selw1, in1, selb);
and u3 (selw2, in2, sel);
or   u4 (out, selw1, selw2);

endmodule
```


As you can see, the “netlist” format can become complex and confusing. When you are creating a large design, it is best to describe the design in RTL code and let the synthesis tool produce the “netlist.”

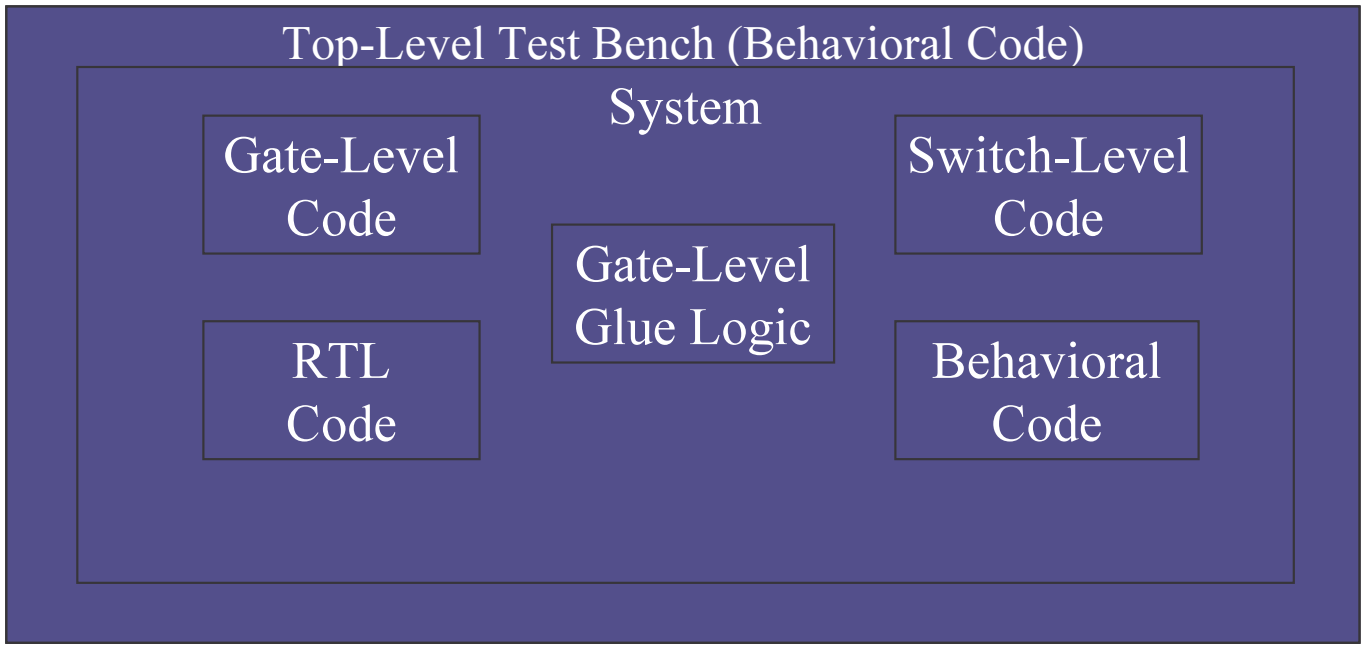
Levels of Abstraction (cont.)

- Switch Level Modeling
  - Describes logic behavior of transistor circuits.

```
module twoinmux (out, in1, in2, sel);

input in1, in2, sel;
output out;
wire c, selb;

supply1 pwr;
supply0 gnd;
pmos (c, pwr, sel);
pmos (selb, c, sel);
nmos (selb, gnd, sel);
nmos (selb, gnd, sel);
cmos (out, in1, selb, sel);
cmos (out, in2, sel, selb);

endmodule
```

One Language

- Verilog can be used at all levels of abstraction in one design.



Keep in mind that only the RTL code is synthesizable. This is an example of how an idea or concept can be modeled and simulated in Verilog

Review

- What is one major difference between a HDL and a typical programming language?
- What are the advantages of using a HDL?
- Is a HDL design implementation independent?
- What are the four different levels of abstraction?