Outline

• Netlists/Design flow
• What is a HDL?
• Verilog History
• Introduction to Verilog
• Verilog Basics

Netlist

• A key data structure (or representation) in the design process is the “netlist”:
  – Network List
• A netlist lists components and connects them with nodes:
  ex:
  
  n1 - g1 - n2       n6 - g3 - n7
  n3 - g2 - n4

  g1 "and" n1 n2 n5
  g2 "and" n3 n4 n6
  g3 "or" n5 n6 n7

  Alternative format:
  n1 g1.in1
  n2 g1.in2
  n3 g2.in1
  n4 g2.in2
  n5 g1.out g3.in1
  n6 g2.out g3.in2
  n7 g3.out
  g1 "and"
  g2 "and"
  g3 "or"

• Netlist is what is needed for simulation and implementation.
• Could be at the transistor level, gate level, ...
• Could be hierarchical or flat.
• How do we generate a netlist?

Design Flow
Design Flow

- Circuit is described and represented:
  - Graphically (Schematics)
  - Textually (HDL)
- Result of circuit specification (and compilation) is a netlist of:
  - generic primitives - logic gates, flip-flops, or
  - technology specific primitives - LUTs/CLBs, transistors, discrete gates, or
  - higher level library elements - adders, ALUs, register files, decoders, etc.

Design Entry
High-level Analysis
Technology Mapping
Low-level Analysis

High-level Analysis is used to verify:
- correct function
- rough:
  - timing
  - power
  - cost
- Common tools used are:
  - simulator - check functional correctness, and
  - static timing analyzer
    - estimates circuit delays based on timing model and delay parameters for library elements (or primitives).

Technology Mapping:
- Converts netlist to implementation technology dependent details
  - Expands library elements,
  - performs:
    - partitioning,
    - placement,
    - routing
- Low-level Analysis
  - Simulation and Analysis Tools perform low-level checks with:
    - accurate timing models,
    - wire delay
  - For FPGAs this step could also use the actual device.

Netlist: used between and internally for all steps.
Design Entry

- Schematic entry/editing used to be the standard method in industry
- Used in EECS150 2002
- Schematics are intuitive. They match our use of gate-level or block diagrams.
- Somewhat physical. They imply a physical implementation.
- Require a special tool (editor).
- Unless hierarchy is carefully designed, schematics can be confusing and difficult to follow.

- Hardware Description Languages (HDLs) are the new standard
  - except for PC board design, where schematics are still used.

HDLs

- Basic Idea:
  - Language constructs describe circuits with two basic forms:
  - Structural descriptions similar to hierarchical netlist.
  - Behavioral descriptions use higher-level constructs (similar to conventional programming).
- Originally designed to help in abstraction and simulation.
  - Now “logic synthesis” tools exist to automatically convert from behavioral descriptions to gate netlist.
  - Greatly improves designer productivity.
  - However, this may lead you to falsely believe that hardware design can be reduced to writing programs!

- “Structural” example:
  ```verilog
  Decoder(output x0,x1,x2,x3; inputs a,b)
  {wire abar, bbar;
   inv(bbar, b);
   inv(abar, a);
   and(x0, abar, bbar);
   and(x1, abar, b);
   and(x2, a, bbar);
   and(x3, a, b);
  }
  ```

- “Behavioral” example:
  ```verilog
  Decoder(output x0,x1,x2,x3; inputs a,b)
  {case [a b]
   00: [x0 x1 x2 x3] = 0x1;
   01: [x0 x1 x2 x3] = 0x2;
   10: [x0 x1 x2 x3] = 0x4;
   11: [x0 x1 x2 x3] = 0x8;
  endcase;
  }
  ```

Verilog

- Supports structural and behavioral descriptions
- Structural
  - Explicit structure of the circuit
  - How a module is composed as an interconnection of more primitive modules/components
  - E.g., each logic gate instantiated and connected to others
- Behavioral
  - Program describes input/output behavior of circuit
  - Many structural implementations could have same behavior
  - E.g., different implementations of one Boolean function

Design Methodology

- Structure and Function (Behavior) of a Design
- HDL Specification
- Simulation
- Synthesis
- Verification: Design Behave as Required?
  - Functional: I/O Behavior
  - Timing: Waveform Behavior
- Generation: Map Specification to Implementation
Verilog

- A brief history:
  - Invented as simulation language. Synthesis was an afterthought. Many of the basic techniques for synthesis were developed at Berkeley in the 80’s and applied commercially in the 90’s.
  - Around the same time as the origin of Verilog, the US Department of Defense developed VHDL. Because it was in the public domain it began to grow in popularity.
  - Afraid of losing market share, Cadence opened Verilog to the public in 1990.
  - An IEEE working group was established in 1993, and ratified IEEE Standard 1394 (Verilog) in 1995.
  - Verilog is the language of choice of Silicon Valley companies, initially because of high-quality tool support and its similarity to C-language syntax.
  - VHDL is still popular within the government, in Europe and Japan, and some Universities.
  - Latest HDL: C++ based. OSCI (Open System C Initiative).

Verilog Introduction

- the module describes a component in the circuit
- Two ways to describe:
  - Structural Verilog
    - List of components and how they are connected
    - Just like schematics, but using text
    - A net list
    - tedious to write, hard to decode
    - Essential without integrated design tools
  - Behavioral Verilog
    - Describe what a component does, not how it does it
    - Synthesized into a circuit that has this behavior
    - Result is only as good as the tools
- Build up a hierarchy of modules. Top-level module is your entire design (or the environment to test your design).

Verilog Module

- Corresponds to a circuit component
  - “Parameter list” is the list of external connections, aka “ports”
  - Ports are declared “input”, “output” or “inout”
    - inout ports used on bidirectional wires
  - Port declarations imply that the variables are wires

module addr_cell (A, B, Cin, S, Cout);
input A, B, Cin;
output S, Cout;
assign {Cout, S} = A + B + Cin;
endmodule

module xor_gate (out, a, b);
input a, b;
output out;
wire abar, bbar, t1, t2;
not invA (abar, a);
not invB (bbar, b);
and and1 (t1, a, bbar);
and and2 (t2, b, abar);
or  or1 (out, t1, t2);
endmodule

Structural Model - XOR

- Composition of primitive gates to form more complex module
- Note use of wire declaration!
  - By default, identifiers are wires

module xor_gate (out, a, b );
input a, b; output out;
wire abar, bbar, t1, t2; declarations

  not invA (abar, a);
  not invB (bbar, b);
  and and1 (t1, a, bbar);
  and and2 (t2, b, abar);
  or  or1 (out, t1, t2);
endmodule

Instance name
Interconnections (note output is first)
Announcements

Homework 1 due tomorrow
Homework 2 out soon

Reading:
- these notes
- verilog code you see in lab
- verilog book on reserve
- THE reference on the web page