Lecture 8:
Combinational Verilog

CSE 370, Autumn 2007
Benjamin Ylvisaker

Where We Are

• Last lecture: Minimization with K-maps
• This lecture: Combinational Verilog
• Next lecture: ROMs, PLAs and PALs, oh my!
• Homework 3 ongoing
• Lab 2 done; lab 3 next week

Specifying Circuits

• Schematics
  • Structural description
  • Build more complex circuits using hierarchy
  • Large circuits are unreadable
• HDLs (Hardware description languages)
  • Not conventional programming languages
  • Very restricted parallel languages
  • Synthesize code to produce a circuit
Quick History Lesson

- Abel (~1983)
  - Developed by Data-I/O
  - Targeted to PLDs
- Verilog (~1987)
  - Developed by Gateway (now part of Cadence)
  - Syntax similar to C
  - Moved to public domain in 1990
- VHDL (~1987)
  - DoD sponsored
  - Syntax similar to Ada

Verilog and VHDL Dominant

- Both “IEEE standard” languages
- Most tools support both
- Verilog is “simpler”
  - Less, more concise syntax
- VHDL is more structured
  - More sophisticated type system
  - Better modularity features

Simulation and Synthesis

- Simulation
  - “Execute” a design with some test data
- Synthesis
  - Generate a physical implementation
Simulation and Synthesis (cont’d)

• Simulation
  • Model circuit behavior
  • Can include timing estimates
  • Allows for easier design exploration

• Synthesis
  • Converts HDL code to "netlists"
  • Can still simulate the generated netlists
  • Simulation and synthesis in the CSE curriculum
    • 370: Learn simulation
    • 467: Learn something about synthesis

Simulation

• You provide an environment
  • Use non-circuit constructs (Active-HDL waveforms, random number generators, etc)
  • Can write arbitrary Verilog code

![Simulation Diagram]

Specifying Circuits in Verilog

• There are three major styles
  • Instances ‘n wires
  • Continuous assignments
  • "always" blocks

“Structural”

wire E;
and g1(E,A,B);
not g2(Y,C);
or g3(X,E,Y);

“Behavioral”

wire E;
reg E, X, Y;
assign E = A & B;
assign Y = ~ C;
assign X = E | Y;
always @ (A or B or C)
begin
  E = A & B;
  Y = ~ C;
  X = E | Y;
end
Data Types

- Values on a wire
  - 0, 1, x (unknown or conflict), z (unconnected)
- Vectors
    - Interpreted as an unsigned binary number
    - Indices must be constants
  - Concatenation
    - $B = 4[A[0:2]]$
    - Style: good to use unnecessary size specs sometimes
      - $A[7:0] = b[7:0] + c[7:0]$

Data Types That Do Not Exist

- structures (records)
- Pointers
- Objects
- Recursive types

(Remember, Verilog is not C or Java or Lisp or ...)

Numbers

- Format: <sign><size><base format><number>
  - 14
    - Decimal
    - ~4'h11
      - 4-bit 2's complement of 0011
    - 12'b000_0100_0110
      - 12 bit binary number ( '_s ignored)
    - 12'h4Ab
      - 12 bit hexadecimal number
## Operators

<table>
<thead>
<tr>
<th>Noting Operator</th>
<th>Name</th>
<th>Functional Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>not</td>
<td>logical operator</td>
</tr>
<tr>
<td>&amp;</td>
<td>and</td>
<td>boolean operator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or</td>
</tr>
<tr>
<td>^</td>
<td>xor</td>
<td>boolean operator</td>
</tr>
<tr>
<td>&lt;=</td>
<td>&lt;=</td>
<td>relational operator</td>
</tr>
<tr>
<td>&lt;</td>
<td>&lt;</td>
<td>relational operator</td>
</tr>
<tr>
<td>&gt;=</td>
<td>&gt;=</td>
<td>relational operator</td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt;</td>
<td>relational operator</td>
</tr>
<tr>
<td>==</td>
<td>==</td>
<td>relational operator</td>
</tr>
<tr>
<td>!=</td>
<td>!=</td>
<td>relational operator</td>
</tr>
</tbody>
</table>

Similar to C operators

## Two Abstraction Mechanisms

- **Modules**
  - More structural
  - Heavily used in 370 and "real" Verilog code

- **Functions**
  - More behavioral
  - Used to some extent in "real" Verilog, but not much in 370

## Basic Building Blocks: Modules

- INSTANTIATED, NOT CALLED
- ILLEGAL TO NEST MODULE DEFS
- INSTANCES "EXECUTE" IN PARALLEL
- WIRES ARE USED FOR CONNECTIONS
- AND, OR, NOT BUILT-IN PRIMITIVE MODULES
- LIST OUTPUT FIRST
- ARBITRARY NUMBER OF INPUTS NEXT
- NAMES ARE CASE SENSITIVE
- CANNOT BEGIN WITH NUMBER
- // FOR COMMENTS

---

```verilog
// first simple example
module smpl(X,Y,A,B,C);
  input A,B,C;
  output X,Y;
  wire E;
  and g1(E,B,B);
  not g2(Y,C);
  or  g3(X,E,Y);
endmodule
```
Module Ports

- Modules interact with the rest of a design through ports
  - input
  - output
  - inout

Same example with continuous assignments:

```verbatim
// first simple example
module smpl(X,Y,A,B,C);
    input A,B,C;
    output X,Y;
    assign X = (A&B)|~C;
    assign Y = ~C;
endmodule
```

Bigger Structural Example

```verbatim
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,abar,b);
    and       and2 (t2,bbar,a);
    or        or1 (out,t1,t2);
endmodule
```

Behavioral Full Adder

```verbatim
module full_addr (Sum,Cout,A,B,Cin);
    input     A, B, Cin;
    output    Sum, Cout;
    assign   {Cout, Sum} = A + B + Cin;
endmodule
```

{Cout, Sum} is a concatenation
Behavioral 4-bit Adder

- Module add4 (SUM, OVER, A, B);
  - input [3:0] A;
  - input [3:0] B;
  - output [3:0] SUM;
  - output OVER;
  - endmodule

Continuous Assignment

- Continuously evaluated
- Think of them as collections of logic gates
- Evaluated in parallel

assign A = X | (Y & ~Z);
assign B[3:0] = 4'b01XX;
assign C[15:0] = 4'h00ff;

Hierarchy Example: Comparator

- Module Compare1 (Equal, Alarger, Blarger, A, B);
  - input A, B;
  - output Equal, Alarger, Blarger;
  - assign Equal = (A & B) | (~A & ~B);
  - assign Alarger = (A & ~B);
  - assign Blarger = (~A & B);
  - endmodule
4-bit Comparator

// Make a 4-bit comparator from 4 1-bit comparators
module Compare4(Equal, Alarger, Blarger, A4, B4);
input [3:0] A4, B4;
output Equal, Alarger, Blarger;
wire e0, e1, e2, e3, Al0, Al1, Al2, Al3, Bl0, Bl1, Bl2, Bl3;
Compare1 cp0(e0, Al0, Bl0, A4[0], B4[0]);
Compare1 cp1(e1, Al1, Bl1, A4[1], B4[1]);
Compare1 cp2(e2, Al2, Bl2, A4[2], B4[2]);
Compare1 cp3(e3, Al3, Bl3, A4[3], B4[3]);
assign Equal = (e0 & e1 & e2 & e3);
assign Alarger = (Al3 | (Al2 & e3) | (Al1 & e3 & e2) | (Al0 & e3 & e2 & e1));
assign Blarger = (~Alarger & ~Equal);
endmodule

Sequential assigns don't make any sense

• assign B = X | (Y & ~Z);
• assign B = W | A;
• assign A = Y & Z;
• You can't reassign a variable with continuous assignments

Always Blocks

• reg A, B, C;
always @ (W or X or Y or Z)
begin
A = X | (Y & ~Z);
B = W | A;
A = Y & Z;
if (A & B)
begain
B = Z;
C = W | Y;
end
end
Functions

- Functions can be used for combinational logic that you want to reuse

```verilog
module and_gate (out, in1, in2);
    input         in1, in2;
    output        out;
    assign out = myfunction(in1, in2);
endmodule
function myfunction;
    input in1, in2;
    begin
        myfunction = in1 & in2;
    end
endfunction
```

Verilog Tips

- Do not write C code
  - Think hardware, not algorithms
    - Verilog is inherently parallel
    - Compilers don’t map algorithms to circuits well
- Do describe hardware circuits
  - First draw a dataflow diagram
  - Then start coding
- References
  - Tutorial and reference manual are found in ActiveHDL help
  - And in today’s reading assignment
  - copies for borrowing in hardware lab

Thank You for Your Attention
Thank You for Your Attention

- Read lab 2
- Continue homework 2
- Continue reading the book