Verilog Green Card

Greg Gibeling
UC Berkeley
gdgib@eecs.berkeley.edu

October 29, 2007

1 Purpose

This document is intended as a printable quick reference for the Verilog HDL, and as a partial list of the constructs allowed in CS61C. You may wish to print it out to have handy, but if you do so, keep an eye out for updates, as mistakes are found.

2 Signal Declarations

There are two types of signals: **wire** and **reg**. **reg** signals are generated by **initial** or **always** blocks. **wire** signals are generated by **assign** statements, hierarchical module instantiations or primitives.

There are four uses for signals: internal, **input**, **output** and **inout**. Internal signals do not cross in or out of the module they are declared in, though they may be connected to ports on lower level modules. The remaining three uses of signals are for ports, and their name gives their direction. Internal, **output** and **inout** signals may of type **wire** or **reg**.

Decl.	Width	Desc.	Generated From
wire WireName;	1bit	Simple Wire	**assign**, hierarchy, primitives
wire [3:0] WireName;	4bit	Bus or BitVector	**assign**, hierarchy, primitives
reg WireName;	1bit	'State' Wire	**always**, **initial**
reg [8:1] WireName;	8bit	'State' Bus or BitVector	**always**, **initial**
input InputName;	1bit	Simple Input Wire	hierarchy (outside)
output [3:0] OutputName;	4bit	Output Bus or BitVector	**assign**, hierarchy, primitives
output reg [7:0] OutputName;	8bit	Output 'State' Bus or BitVector	**always**, **initial**

3 Instantiation & Modules

Below is an example module declaration. The order of the port declarations is unimportant. Because Verilog is not an imperative (command based) language, the order of the statements within a module is irrelevant, just as the order in which you draw the boxes in a schematic is irrelevant.

```
module ModuleName(Port0, Port1);
        input                   Port1;
        output                  Port0;
        // The actual logic (module level statements) goes here
endmodule // ModuleName
```

Below is an example showing two identical instantiations of the above module. By convention, the second one, which uses "named connections", is far proffered over the first, which relies on order of the connections.

```
        ModuleName FirstInstanceName(LocalWire0, LocalWire1);
        ModuleName SecondInstanceName(.Port1(LocalWire1), .Port0(LocalWire0));
```

4 TestBenches

TestBenches can be written using the full expressive power of the Verilog language. This includes **always** @ (*), **initial** and for example **$display**(). These constructs are allowed in testbenches because, while they do not map to circuits, a testbench is not intended to describe a circuit.

The below table lists the constructs which you may wish to use in a testbench. You may of course use any construct listed in section 5 in addition to those below.

Syntax	Use	Example	Description
always @ (*) **begin** ... **end**	module level		Combinational, procedural logic
always @ (**posedge** Clock)	module level		Register, sequential circuit
initial begin ... **end**	module level		Initial conditions and tests
$display(x, y, z)	statement	**$display**("Hello")!	Output text
$stop	statement		Stop the simulation
+	expression	**assign** #1 x = y + z;	Addition
−	expression	**assign** #1 x = y − z;	Subtraction
*	expression	**assign** #1 x = y * z;	Unsigned multiplication
/	expression	**assign** #1 x = y / z;	Unsigned division
%	expression	**assign** #1 x = y % z;	Unsigned modulo or remainder
<< and >>	expression	**assign** #1 x = y << z;	Unsigned shift

While there are tools which can turn + and − as well as the various kinds of **always** blocks into circuits, these tools are quite advanced. The other constructs in this table cannot be turned into circuits. Finally, rather than use the << or >> operators, you should use bit-selections and concatenation shown below.

5 Circuits

Module which are intended to describe a circuit may only use a subset of the Verilog language. In particular the constructs in the below table may be used in circuits. The table lists the vague syntax, where the construct may appear, an example and a description. x, y and z are signals, c is a constant.

Syntax	Use	Example	Description
assign	module level	**assign** #1 x = y \& z;	Express bitwise logic as equations
{x, y}	expression	**assign** #1 {x, y} = 0;	Bit-vector concatenation
c{x, y}	expression	16{1'b1}	Bit-vector replication, c is constant
x[c]	expression	**assign** #1 x[4] = y[2];	Bit-vector selection, result is a signal
x[c0:c1]	expression	x[4:3]	Bit-vector range selection
x ? y : z	expression	x[0] ? y : z	Multiplexor, select must be 1-bit
~x	expression	~x	Inverter, NOT gate, bitwise NOT
x | y	expression	x | y | z	Multi-input OR gate, bitwise OR
x & y	expression	x & y & z	Multi-input AND gate, bitwise AND
x ^ y	expression	x ^ y ^ z	Multi-input XOR gate, bitwise XOR
not name(...)	module level	**not** name(out, in)	Inverter, NOT gate
or name(...)	module level	**or** name(out, in0, in1, in2)	Multi-input OR gate
and name(...)	module level	**and** name(out, in0, in1, in2)	Multi-input AND gate
xor name(...)	module level	**xor** name(out, in0, in1, in2)	Multi-input XOR gate

6 Common Pitfalls

This section lists several common pitfalls which trap the novice Verilog user. Most important is that unlike Java or even C, Verilog does not check the types or widths of signals very strenuously. In all three below cases, the compiler will not warn you of your mistake.

Undeclared Wires: In Verilog any signal which is undeclared is automatically a 1-bit wire. Assigning to undeclared signals from an **always** or **initial** block will result in compiler errors. Forgetting to declare a bus or bit-vector will result in only the 0th being connected properly (see below).

Width Mistmatch (Small Wire): Connecting a small wire (e.g. 1-bit) to a large port (e.g. 4-bit), will cause a "port width mismatch" warning. Some of the signals intended for the bus will be lost, and only the lowest few bits will appear at the other end.

Width Mistmatch (Large Wire): Connecting a large wire (e.g. 4-bit) to a small port (e.g. 1-bit), will cause a "port width mismatch" warning. Many waveforms will appear in blue, denoting an undriven signal, because there are bits of the bus which have no source.