Verilog: `wire` vs. `reg`

Chris Fletcher
UC Berkeley
Version 0.2008.9.25

October 2, 2008

1 Introduction

Sections 1.1 to 1.3 discuss the difference between `wire` and `reg` in Verilog, and when to use each of them.

1.1 `wire` Elements (Combinational logic)

`wire` elements are simple wires (or busses of arbitrary width) in Verilog designs. The following are syntax rules when using `wire`s:

1. `wire` elements are used to connect `input` and `output` **p**orts of a module **i**nstantiation together with some other element in your design.
2. `wire` elements are used as `input`s and `output`s within an actual module declaration.
3. `wire` elements must be driven by something, and cannot store a value without being driven.
4. `wire` elements cannot be used as the left-hand side of an `=` or `<=` sign in an `always@` block.
5. `wire` elements are the only legal type on the left-hand side of an `assign` statement.
6. `wire` elements are a stateless way of connecting two peices in a Verilog-based design.
7. `wire` elements can only be used to model combinational logic.

Program 1 shows various legal uses of the `wire` element.

Program 1 Legal uses of the `wire` element

```
1 wire        A, B, C, D, E; // simple 1-bit wide wires
2 wire [8:0] Wide;     // a 9-bit wide wire
3 reg I;
4
5 assign A = B & C;    // using a wire with an assign statement
6
7 always @(B or C) begin
8     I = B | C;       // using wires on the right-hand side of an always@
9                      // assignment
10 end
11
12 mymodule mymodule_instance(.In (D),
13                             .Out(E)); // using a wire as the output of a module
```

1.2 `reg` Elements (Combinational and Sequential logic)

`reg` are similar to wires, but can be used to store information ('state') like registers. The following are syntax rules when using `reg` elements.

1. `reg` elements **c**an be connected to the input **p**ort of a module **i**nstantiation.
2. `reg` elements **c**annot be connected to the output **p**ort of a module **i**nstantiation.
3. `reg` elements **c**an be used as `output`s within an actual module declaration.
4. `reg` elements **c**annot be used as `input`s within an actual module declaration.
5. `reg` is the only legal type on the left-hand side of an `always@` block `=` or `<=` sign.
6. `reg` is the only legal type on the left-hand side of an `initial` block `=` sign (used in Test Benches).
7. `reg` cannot be used on the left-hand side of an `assign` statement.
8. `reg` can be used to create registers when used in conjunction with `always@(posedge Clock)` blocks.
9. `reg` can, therefore, be used to create both combinational and sequential logic.

Program 2 shows various legal uses of the `reg` element.

Program 2 Legal uses of the `reg` element

```
wire      A, B;
reg       I, J, K;    // simple 1-bit wide reg elements
reg[8:0]  Wide;       // a 9-bit wide reg element

always @(A or B) begin
    I = A | B;        // using a reg as the left-hand side of an always@
                      // assignment
end

initial begin         // using a reg in an initial block
    J = 1'b1;
    #1
    J = 1'b0;
end

always @(posedge Clock) begin
    K <= I;           // using a reg to create a positive-edge-triggered register
end
```

1.3 When `wire` and `reg` Elements are Interchangable

`wire` and `reg` elements can be used interchangably in certain situations:

1. Both can appear on the right-hand side of `assign` statements and `always@` block `=` or `<=` signs.
2. Both can be connected to the input **p**orts of module instantiations.