Verilog for Control Logic

Paul Gratz

Outline

- Control Logic Modeling Methods
  - Non Procedural
    - Continuous assignments
  - Procedural
    - Always blocks
    - Conditional statements
    - Multi-way Branch (Case)- statements
    - Loops
  - State Machine example
- Synthesis to Structural Verilog

Non-Procedural Control logic

```
assign      head_plus = in_array_addr + 1;
assign      tail_plus = out_array_addr + 1;
assign      tail_plus2 = out_array_addr + 2;
assign      saq_wr_clk = ~clk & saq_we;
```

- Simple
- Explicit
- Difficult to express complicated expressions
- State must be saved for sequential logic

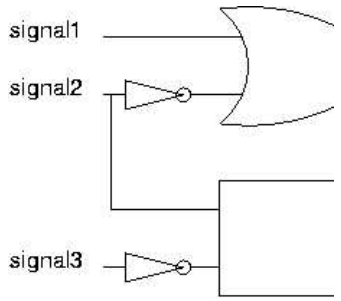
Procedural Control Logic

- Often easier to code.
- Can handle state without having to specify sequential cells.
- Is non-implementation specific.
- Potentially dangerous from the point of view of creating real hardware. Must pay attention to buildablity of the design.

Always blocks revisited

```
input  signal1, signal2, signal3;
output output1, output2;
reg        output1, output2;

always @ (signal1 or signal2 or signal3) begin
   output1 = signal1 | ~signal2;
   output2 = signal2 & ~signal3;
end
```



- Whenever any “signalX” changes the two outputs are recalculated.
- Easily done in an assignment statement as well.
- Must make sure all signals tested are listed in “@ ()” condition.
- This is non-sequential as shown

Conditional Statments

```
always @ (signal1 or signal2 or signal3) begin
     if (signal1 | ~signal2)
        output1 = 1'b1;
     else
        output1 = 1'b0;
     if (signal2 & ~signal3)
        output2 = 1'b1;
     else
        output2 = 1'b0;
  end
```

- Equivalent to the last slide but uses if statements.
- Must be sure to always include “else” for each conditional or will cause instantiation of latches (unless you want latches…).

Multi-way Branching

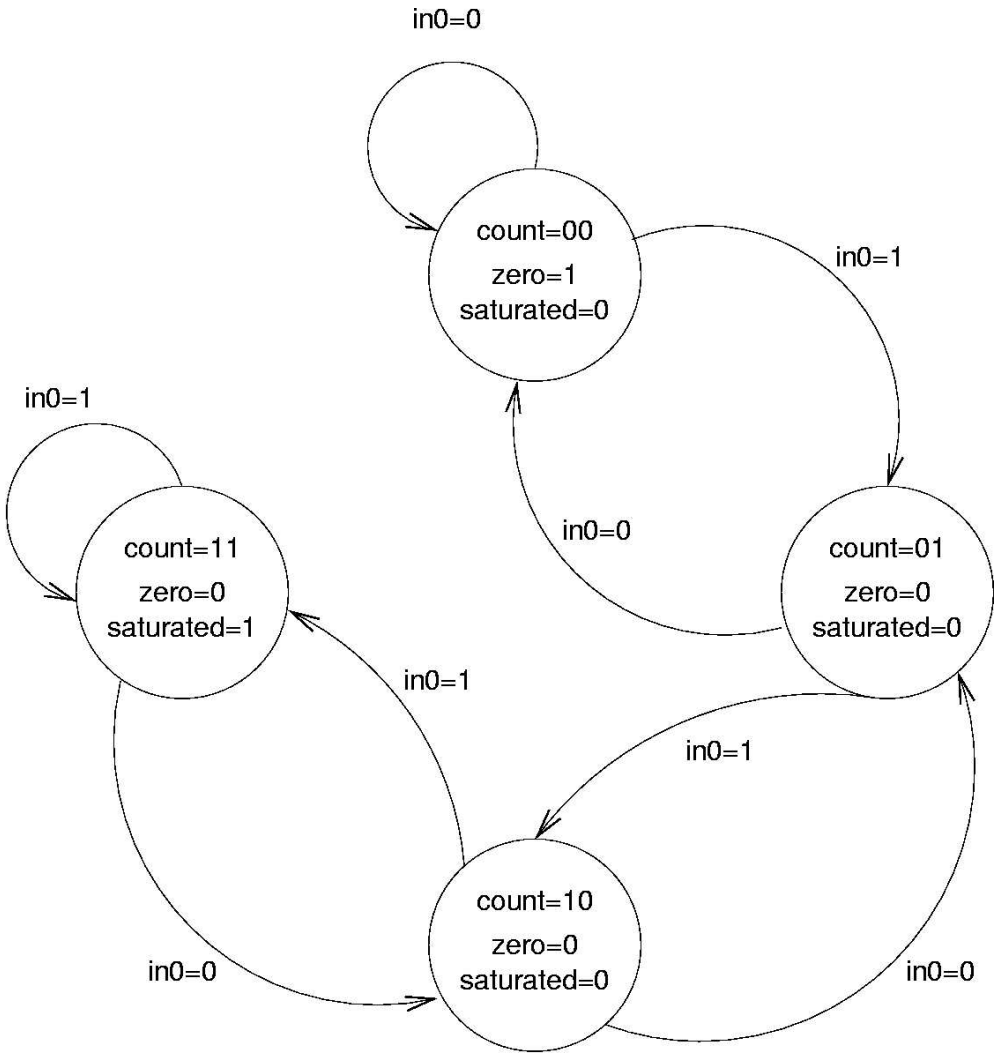
```
always @ ( signal1 or signal2) begin
     case ({signal1,signal2})        // Case for "or" part of logic block
         2'b00:
           output1 = 1'b1;
         2'b01:
           output1 = 1'b0;
         2'b10:
           output1 = 1'b1;
         2'b11:
           output1 = 1'b1;
         default:
           output1 = 1'b0;
     endcase // case({signal1,signal2})
end
```

- Always (always, always!) use a default statement unless you mean to instantiate a latch (usually you do not).
- Case statements are great for specifying state machines.

Loops

- Don’t use them outside of testbenches!
- The are not synthesizable (mostly).
- Have little relation to hardware (usually).
- Generally a bad idea unless you are sure you know what you are doing.

State Machine Example: Saturating Counter



State Machine Example: Continued

```
module always_test (/*AUTOARG*/
   // Outputs
   count, zero, saturated,
   // Inputs
   in0, clk, reset
   ) ;

   input  in0, clk, reset;
   output [1:0] count;
   output     zero, saturated;
   reg [1:0]  count, next_count;
   reg                      zero, saturated;

   always @ ( /*AUTOSENSE*/count or in0) begin
     case (count)
     2'b00: begin
        zero = 1'b1;
        saturated = 1'b0;
        if (in0)
          next_count = 2'b01;
        else
          next_count = 2'b00;
     end
     2'b01: begin
        zero = 1'b0;
        saturated = 1'b0;
        if (in0)
          next_count = 2'b10;
        else
          next_count = 2'b00;
     end
     2'b10: begin
        zero = 1'b0;
        saturated = 1'b0;
        if (in0)
          next_count = 2'b11;
        else
          next_count = 2'b01;
     end
     default: begin // also could have used "2'b11:" here
        zero = 1'b0;
        saturated = 1'b1;
        if (in0)
          next_count = 2'b11;
        else
          next_count = 2'b10;
     end
      endcase // case(count)
   end

   always @(posedge clk) begin
      if (reset)
     count = 2'b00;
      else
     count = next_count;
   end
endmodule // always_test
```

Synthesis

- Automated process.
- Used to make behavioral style verilog into structural instantiations of library cells.
- Give the tool design constraints and it will try to produce logic that fits them.
- Have to be careful to make sure the logic you intend is what you get.
- The only purpose of behavioral verilog is to be synthesized into real logic.