Verilog

Hung-Wei Tseng
Verilog

• Verilog is a hardware description language (HDL).
• In this class, we use Verilog to implement and verify your processor.
• C/Java like syntax
Data type in Verilog

• Bit vector is the only data type in Verilog
• A bit can be one of the following
  • 0: logic zero
  • 1: logic one
  • X: unknown logic value, don’t care
  • Z: high impedance, floating
• Bit vectors expressed in multiple ways
  • binary: 4'b11_10 ( _ is just for readability)
  • hex: 16'h034f
  • decimal: 32'd270
Operators

• Arithmetic: + - * / % ** (don’t use the last three)
• Logic: ! && ||
• Relational: > < >= <=
• Equality: == != === !===
• Bitwise: ~ & | ^ ^^~
• Reduction: & ~& | ~| ^ ^~
• Shift: >> << >>> <<<<
• Concatenation: { }
• Conditional: ? :
Wire to connect things together!

- **wire** is used to denote a hardware net
  - single wire
    ```verilog
    wire my_wire;
    ```
  - array of wires
    ```verilog
    wire[7:0] my_wire;
    ```
- **For procedural assignments, we will use** `reg`
  - again, can either have a single reg or an array
    ```verilog
    reg[7:0] result; // 8-bit reg
    ```
  - reg is not necessarily a hardware register
  - you may consider it as a variable in C
A Verilog module has a name and a port list
  - ports: must have a direction (input, output, inout) and a bitwidth

Think about an 1-bit adder
  - input: 1-bit * 3
  - output 1-bit * 1 and 1-bit * 1

module FA(
  input a,
  input b,
  input cin,
  output cout,
  output sum
);
assign sum = a^b^cin;
assign cout = (a&b) | (a&cin) | (b&cin);
endmodule

Adapted from Arvind & Asanovic’s MIT 6.375 lecture
Always block

- Executes when the condition in the sensitivity list occurs

```verilog
always@(posedge clk)
begin
  module FA( input a,
              input b,
              input cin,
              output cout,
              output sum );
  reg s, cout
  always@(a or b or cin)
  begin
    sum = a^b^cin;
    cout = (a&b) | (a&cin) | (b&cin);
  end
endmodule
```
Blocking and non-blocking

- Inside an always block, `=` is a blocking assignment
  - assignment happens immediately and affect the subsequent statements in the always block
- `<=` is a non-blocking assignment
  - All the assignments happens at the end of the block

Initially, \( a = 2, b = 3 \)

```verilog
reg a[3:0];
reg b[3:0];
reg c[3:0];
always @(posedge clock)
begin
  a <= b;
  c <= a;
and
Afterwards: \( a = 3 \) and \( c = 2 \)
```

Sequential logic

```verilog
reg a[3:0];
reg b[3:0];
reg c[3:0];
always @(*)
begin
  a = b;
  c = a;
and
Afterwards: \( a = 3 \) and \( c = 3 \)
```

Combinational logic
Initial block

- Executes only once in beginning of the code

```plaintext
initial
begin
  ...
  ...
end
```
Modules

- A verilog module can instantiate other modules

```verilog
module adder(
  input [3:0] A,
  input [3:0] B,
  output carry,
  output [3:0] sum);

wire c0, c1, c2
FA fa0(A[0],B[0],cin,c0,sum[0]); // implicit binding
FA fa1(.a(A[1]), .b(B[1]), .cin(c0), .sum(sum[1]), .cout(c1)); // explicit binding
FA fa2(A[2],B[2],c1,c2,sum[2]);
FA fa3(A[3],B[3],c2,cout,sum[3]);
endmodule
```

Adapted from Arvind & Asanovic’s MIT 6.375 lecture
Testing modules

```verilog
`timescale 1ns/1ns // Add this to the top of your file to set time scale
module testbench();
reg [3:0] A, B;
reg C0;
wire [3:0] S;
wire C4;
adder uut (.B(B), .A(A), .sum(S), .cout(C4)); // instantiate adder

initial
begin
A = 4'd0; B = 4'd0; C0 = 1'b0;
#50 A = 4'd3; B = 4'd4; // wait 50 ns before next assignment
#50 A = 4'b0001; B = 4'b0010; // don’t use #n outside of testbenches
end

endmodule
```
Resources

- Check out MIT’s 6.375 course webpage http://csg.csail.mit.edu/6.375/
  - thanks to Asanovic & Arvind for slides

- Tips for using Altera tools https://sites.google.com/a/eng.ucsd.edu/using-the-altera-tools/
  - thanks to Swanson and other CSE141L winter 2012 staffs
Q & A