ECE 2300 Spring 2017
Verilog Tutorial Session

Ecenur Ustun (eu49)  Gaurab Bhattacharya (gb358)
Verilog

Verilog is a **Hardware Description Language (HDL)**. It is used to describe the structure and behavior of the hardware. The final product of a Verilog program is the generated hardware circuit.
How to describe functionality?

- **Behavioral**: *what a module does*
- **Structural**: *how a module is built*
HDL code

- Synthesizable modules
  - Describes the hardware

- Testbench
  - Intended only for simulation
  - Cannot be synthesized
Testbench

• An HDL module that is used to test another module, called the *unit under test* (*UUT*)
• Contains statements to apply inputs to the UUT
• The input and desired output patterns are called *test vectors*. 
HDL

Logic Simulation

1. Inputs are applied to the module
2. Outputs are checked to verify that the module operates correctly

Logic Synthesis

• The description of the module is translated into logic gates
module sillyfunction ( input x, y, z,  
                      output f );

assign f = ~x & ~y & ~z | ~x & y & ~z | ~x & y & z;

endmodule
Logic Simulation
Logic Synthesis

• To examine the circuit we use Tools -> Netlist Viewers -> RTL Viewer.

• We can also examine the resource usage on the Compilation Report -> Analysis and Synthesis -> Resource Usage Summary
Code snippets #1

VERILOG IS NOT A PROGRAMMING LANGUAGE !!!

module combinational1 ( input a, b, output d );

  wire c;
  assign d = a & c;
  assign c = ~b;
endmodule

module combinational2 ( input a, b, output d );

  wire c;
  assign c = ~b;
  assign d = a & c;
endmodule
Continuous Assignment Statements (*assign*)

- These statements are reevaluated anytime any of the inputs on the right hand side changes.

- Describes *only* combinational logic

- Evaluated *concurrently*
**always** Statements

- These statements are executed when an event in the sensitivity list takes place. Otherwise, signals keep their old value.
- Can be used to describe **sequential logic**, since the old state is kept as long as no new state is prescribed.

```verilog
module flipflop ( input clk, input [3:0] d, output reg [3:0] q);
  reg [3:0] n;
  always @(posedge clk)
  begin
    n <= d;
    q <= n;
  end
endmodule
```

Construct (necessary when multiple statements appear in the always statement)
Some rules

• Every signal is either a wire or a reg.
  • You can only assign to a wire outside always blocks.
  • You can only assign to a reg inside always blocks.
  • You can read from wire or reg anywhere.

• A wire is a physical connection between ports or a logical expression. (i.e. assign $X = Y \& Z$)

• A reg is not the same as a register, it is a signal which you intend to describe behaviorally. Quartus will give you a combinational circuit if that matches your described behavior.
In an *always* statement...

- **Blocking assignments** (=) are evaluated in the order in which they are given in the HDL code. Assignment to LHS is immediate.

- **Non-blocking assignments** (<=) are evaluated concurrently (i.e., all statements are evaluated before the signals on the left hand sides are updated). Assignment to LHS is delayed until end of always block.
Code snippets #2

Are the generated circuits same?

1. CMS
2. Go to the assignment
   Verilog Tutorial Session
3. Download project2
4. Unzip
5. Quartus:
   File → Open Project
Logic Simulation
Logic Synthesis

• To examine the circuit we use Tools -> Netlist Viewers -> RTL Viewer.

• We can also examine the resource usage on the Compilation Report -> Analysis and Synthesis -> Resource Usage Summary
e.g. Enabled Register with Synchronous Reset

module enreg (input clk, input reset, input en, input [3:0] d, output reg [3:0] q);

// synchronous reset
always @ (posedge clk) begin
  if (reset) q <= 4’b0;
  else if (en) q <= d;
end

endmodule
always Statements *cont’d*

- Can also be used to describe **combinational logic if** the sensitivity list includes *all of the inputs* and the body determines the output values for *every possible input combination*.

```verilog
always @ (DATA, RESET) begin
    if (RESET) 
        DONE = 1'b0;
    else if (DATA == 10'b0) 
        DONE = 1'b1;
    else 
        DONE = 1'b0;
end
```

Whenever DATA or RESET changes

- if RESET is now 1
  - set DONE to 0
- else if DATA is now 0
  - set DONE to 1;
- otherwise
  - set DONE to 0;
**always** Statements *cont’d*

- Generated circuit is combinational:

- We could specify the same circuit without an always block:

```verilog
circuit_template;
wire DONE;
assign DONE = RESET ? 1'b0 : (DATA == 10'b0);
```

- We can also use * in the sensitivity list to indicate “all signals”.

```verilog
always @ (*) begin
```
Quiz

Write down the Verilog code for the circuit given below.

!! Reset is synchronous !!
Inferred Latches

What if you forgot to specify the **else** branch on an if statement?

\[
\begin{array}{|l|l|}
\hline
\textbf{always @ (COUNT, RESET)} & \textbf{Whenever COUNT or RESET changes} \\
\texttt{begin} & \texttt{if RESET is now 1} \\
\texttt{if (RESET)} & \texttt{set DONE to 0} \\
& \texttt{else if COUNT is now 0} \\
\texttt{DONE = 1'b0;} & \texttt{set DONE to 1;} \\
\texttt{else if (COUNT == 10'b0)} & \texttt{otherwise} \\
\texttt{DONE = 1'b1;} & \texttt{DONE holds previous value;} \\
\texttt{end} & \texttt{This is implied!} \\
\hline
\end{array}
\]
Inferred Latched cont’d

HDL synthesizes the following sequential circuit instead of a combinational circuit:
Inferred Latches *cont’d*

```verilog
always@(*) begin
  case(X) begin
    2'b00:
      Y <= 2'b01;
    2'b01:
      Y <= 2'b10;
    2'b10:
      Y <= 2'b11;
    default:
      Y <= 2'bx;
  end
end
```

You could also get an inferred latch if you don’t include every possible case in a case statement. To avoid this you usually want a **default** case with don’t care outputs.
Verilog Gotcha’s

• Widths of numerical constants

```verilog
assign x1 = 1;    // no width is assumed 32 bit
assign x2 = 4’b1; // 4-bit 0001
assign x3 = 4’d5; // 4-bit decimal number 5
assign x4 = 4’hA; // 4-bit hex number A
```

• Adders automatically wrap

```verilog
wire [2:0] X;
wire [2:0] Y;

assign X = 3’b111;
assign Y = X + 3’b1;  // Y will be 3’b0
```
Common Verilog Compiler Messages and Pitfalls

• **Assigning to wire in always block**: “Error (10137): Verilog HDL Procedural Assignment error at universalShift.v(22): object "myErrorWire" on left-hand side of assignment must have a variable data type”

• **Assigning to reg outside always block**: One of several errors: “Error (10170): Verilog HDL syntax error at universalShift.v(17) near text "; expecting ",", or ")" or “Error (10219): Verilog HDL Continuous Assignment error at universalShift.v(17): object "myErrorReg" on left-hand side of assignment must have a net type”

• **Improper usage of always @(*) vs always @(posedge CLK)**: Not a compiler error or warning, but can result in incorrect behavior if the wrong type of always block is used”

• **Bit truncation warning** “Warning (10230): Verilog HDL assignment warning at universalShift.v(17): truncated value with size 10 to match size of target (8)”

• **Inferred latches** “Warning (10240): Verilog HDL Always Construct warning at universalShift.v(17): inferring latch(es) for variable "accidentalLatch", which holds its previous value in one or more paths through the always construct”
Common Verilog Compiler Messages and Pitfalls cont’d

• **Trying to assign to a wire in 2 places**: “Error (10028): Can't resolve multiple constant drivers for net "DoubleDrivenWire" at universalShift.v(17)”

• **Driving Input Ports** “Error (10231): Verilog HDL error at universalShift.v(15): value cannot be assigned to input "drivenInput" followed by “Error (10031): Net "drivenInput" at universalShift.v(15) is already driven by input port "drivenInput", and cannot be driven by another signal”

• **Declaring Inputs as Reg**: “Error (10278): Verilog HDL Port Declaration error at universalShift.v(15): input port "IN" cannot be declared with type "<a variable data type, e.g. reg>"

• **Dangling ports**: Compiler may not throw a warning or error, but occurs when you do not connect a wire to every port on a module.

• **Improper Port Declaration (declared in Port list but not listed as input or output)**: “Error (10161): Verilog HDL error at universalShift.v(5): object "OUT" is not declared” - declared as input or output but not in port list “Error (10206): Verilog HDL Module Declaration error at universalShift.v(14): top module port "OUT" is not found in the port list”
Thanks!