Verilog – Combinational Logic

Verilog for Synthesis
Verilog – logic and numbers

• Four-value logic system
  • 0 – logic zero, or false condition
  • 1 – logic 1, or true condition
  • x, X – unknown logic value
  • z, Z - high-impedance state

• Number formats
  • b, B binary
  • d, D decimal (default)
  • h, H hexadecimal
  • o, O octal
  • 16’H789A – 16-bit number in hex format
  • 1’b0 – 1-bit
Verilog types

- **Constants**
  - `parameter` DIME = 10;
  - `parameter` width = 32, nickel = 5;
  - `parameter` quarter = 8’d0010_0101;

- **Nets**
  - `wire` clock, reset_n;
  - `wire[7:0]` a_bus;

- **Registers**
  - `reg` clock, reset_n;
  - `reg[7:0]` a_bus;

- **Integer**
  - only for use as general purpose variables in loops
  - `integer` n;
Operators

• Bitwise
  - ~ negation
  - & and
  - | inclusive or
  - ^ exclusive or

  Verilog  VHDL
  - ~ a;  y = ~(a & b);
  - & a & b;  y = a AND b;
  - | a | b;  y = A OR b;
  - ^ a ^ b;  y = a XOR b;
  - y = & a_bus;

• Reduction (no direct equivalent in VHDL)
  - Accept single bus and return single bit result
    • & and
    • ~& nand
    • | or
    • ^ exclusive or

  y = & a_bus;
  y = & a_bus;
  y = | a_bus;
  y = a & b;  y = A AND b;
  y = ~(a & b);
  y = A NAND b;
  y = NOT a;
Operators (cont’d)

• Relational (return 1 for true, 0 for false)
  - `<` less than,     `<=`
  - `>` greater than   `>=`

• Equality
  - `==` logical equality
  - `!=` logical inequality

• Logical Comparison Operators
  - `!` logical negation
  - `&&` logical and
  - `||` logical or

• Arithmetic Operators
  - `+`
  - `-`
  - `*`
Operators (cont’d)

• Shift
  – \( \ll \) logical shift left, (\( \llll \) arithmetic)
  – \( \gg \) logical shift right (\( \gggg \) arithmetic)

• Conditional
  – Only in Verilog - selects one of pair expressions
    – \( ? : \)
      – Logical expression before \( ? \) is evaluated
      – If true, the expression before \( : \) is assigned to output
      – If false, expression after \( : \) is assigned to output
        • \( Y = (A > B) \ ? \ 1 \ : \ 0 \)
        • \( Y = (A == B) \ ? \ A + B \ : \ A - B \)
Simple Combinational Example

module simple(
    input [2:0] sw,
    input push,
    output [5:0] led
);

assign led[0] = 1'b1; // led 0 always on
assign led[1] = sw[0]; // turn led on when switch on
assign led[3] = push;
assign led[5:4] = 2'b10;
endmodule
View Technology Schematic
Decoder Tutorial Demo Example

sw0
sw1
sw2

→ led0
→ led1
→ led2
→ led3
→ led4
→ led5
→ led6
→ led7
Verilog Source Code

```verilog
module decoder(
    input [2:0] sw,
    output [7:0] led
);

assign led = (sw == 3’d000) ? 8’d00000001 :
    (sw == 3’d001) ? 8’d00000010 :
    (sw == 3’d010) ? 8’d00000100 :
    (sw == 3’d011) ? 8’d00001000 :
    (sw == 3’d100) ? 8’d00010000 :
    (sw == 3’d101) ? 8’d00100000 :
    (sw == 3’d110) ? 8’d01000000 :
    8’d10000000;
endmodule
```
Concurrent statements

- VHDL
  - Process
  - Signal assignments
- Verilog
  - `always` statement
  - Continuous assignment - `assign`
Verilog wire and register data objects

- **Wire** – net, connects two signals together
  - `wire clk, en;
  - `wire [15:0] a_bus;

- **Reg** – register, holds its value from one procedural assignment statement to the next
  - Does not imply a physical register – depends on use
  - `reg [7:0] b_bus;`
Index and Slice

- **VHDL**
  - Use `to` and `downto` to specify slice
  - Concatenation &
    - `c_bus(3 downto 0) <= b_bus(7 downto 4);`
    - `c_bus(5 downto 0) <= b_bus(7) & a_bus(6 downto 3) & '0';`

- **Verilog**
  - Use colon `:`
  - Concatenation `[,]`
    - `assign c_bus[3:0] = b_bus[7:4];`
    - `assign c_bus[5:0] = {b_bus[7], a_bus[6:3], 1'b0};`
Internal wires

- Declare internal wires:

```verilog
module buses(a_bus, b_bus, c_bus, d_bus);

input [7:0] a_bus;
input [7:0] b_bus;
output [7:0] c_bus;
output [7:0] d_bus;

wire [7:0] internal_bus;

assign c_bus[5:0] = {b_bus[7], a_bus[6:3], 1'b0};
assign internal_bus = {8'h5A & a_bus} | b_bus;
assign c_bus = ~internal_bus;

endmodule
```
Sequential Statements

- VHDL
  - reside in process statement

- Verilog
  - reside in an always statement
  - if statements (no endif)
  - case statements (endcase)
  - for, repeat while loop statements

- Note: use begin and end to block sequential statements
Decoder – always statement

- 2 to 4 decoder with enable
- Combinational logic using always statement with sensitivity list
  - similar to VHDL process – for cyclic behavior
  - (@) event control operator
  - begin .. end block statement
  - note reg for y
Decoder (cont’d)

- Combinational logic using *always* statement with sensitivity list
  - similar to VHDL *process* – for cyclic behavior
  - (@) event control operator
  - *begin* .. *end* block statement
    - Statements execute sequentially
      - *if* statement
      - *case* statement
        - Note: *case* expression can concatenate signals ({,})
  - Sensitivity list
    - (a or b or c)
    - Verilog 2001 allows comma-separated list (a, b, c)
Decoder – CASE statement

- CASE is better for this type of design - no priority
  - Exactly same logic produced

```verilog
module decoder_case(
    input [1:0] sel,
    input en,
    output reg [3:0] y // reg required for y
);

always @(sel, en)
    case (#en, #sel) // combine two signals - note order
        3'b100: y = 4'b1110;
        3'b101: y = 4'b1101;
        3'b110: y = 4'b1011;
        3'b111: y = 4'b0111;
        default: y = 4'b1111;
    endcase
endmodule
```
Decoder – 3 to 8 with CASE

module decoder(
    input [2:0] sw,
    output reg [7:0] led
);

always @(sw)
    case (sw)
        3'b000: led = 8'b00000001;
        3'b001: led = 8'b00000010;
        2: led = 8'b00000100;
        3: led = 8'b00001000;
        3'b100: led = 8'b00100000;
        3'b101: led = 8'b00100000;
        3'b110: led = 8'b01000000;
        3'b111: led = 8'b10000000;
    endcase
endmodule
MUX example

- Example multiplexer with conditional operator
- Selects different values for the target signal
  - priority associated with series of conditions
  - (similar to an IF statement)
Synthesis Results – Technology Schematic

\[ \text{O} = ((I0 \cdot I1 \cdot I3) + (!I0 \cdot I1 \cdot I4) + (!I0 \cdot !I1 \cdot I5) + (I0 \cdot !I1 \cdot I2)); \]
Mux – with CASE statement

- Include all inputs on sensitivity list

Elaborating module <mux_case>.

WARNING: HDLCompiler:91 - "C:\ece3829\mux_case\mux_case.v" Line 34: Signal <i> missing in the sensitivity list is added for synthesis purposes. HDL and post-synthesis simulations may differ as a result.
Mux – fixed sensitivity list

- Exact same logic produced as using conditional operator
Priority Encoder

- Priority Encoder using conditional operator
- Priority order determined by sequence
  - similar to if-else statement
Encoder – Technology Schematic

---

HDL Synthesis

Synthesizing Unit <encoder>.

Related source file is "C:\ece3829\encoder\encoder.v".

WARNING:Xst:647 - Input <i0> is never used. This port will be preserved and left unconnected if it belongs to a top-level block or it belongs to a sub-block and the hierarchy of this sub-block is preserved.

Summary:

    inferred 2 Multiplexer(s).

Unit <encoder> synthesized.

---

HDL Synthesis Report

Macro Statistics

# Multiplexers : 2
2-bit 2-to-1 multiplexer : 2
Add ‘gs’ output

```verilog
module encoder(
    input  i0,
    input  i1,
    input  i2,
    input  i3,
    output [1:0] a,
    output gs
);

assign a = (i3 == 1'b1) ? 2'b11 :
            (i2 == 1'b1 ? 2'b10 :
             (i1 == 1'b1) ? 2'b01 :
              2'b00;

assign gs = i0 | i1 | i2 | i3;
endmodule
```
Synthesize - Design Summary

Clock Information:
------------------
No clock signals found in this design

Asynchronous Control Signals Information:
----------------------------------------
No asynchronous control signals found in this design

Timing Summary:
------------------
Speed Grade: -3

Minimum period: No path found
Minimum input arrival time before clock: No path found
Maximum output required time after clock: No path found
Maximum combinational path delay: 5.456ns
Implement Design

Device Utilization Summary:

Slice Logic Utilization:

- Number of Slice Registers: 0 out of 18,224 (0%)
- Number of Slice LUTs: 2 out of 9,112 (1%)
  - Number used as logic: 2 out of 9,112 (1%)
    - Number using O6 output only: 1
    - Number using O5 output only: 0
    - Number using O5 and O6: 1
    - Number used as ROM: 0
- Number used as Memory: 0 out of 2,176 (0%)

Slice Logic Distribution:

- Number of occupied Slices: 2 out of 2,278 (1%)
- Number of MUXCYs used: 0 out of 4,556 (0%)
- Number of LUT Flip Flop pairs used: 2
  - Number with an unused Flip Flop: 2 out of 2 (100%)
  - Number with an unused LUT: 0 out of 2 (0%)
- Number of fully used LUT-FF pairs: 0 out of 2 (0%)
- Number of slice register sites lost to control set restrictions: 0 out of 18,224 (0%)
Creating adder – using LUTs

module adder(
    input [7:0] a,
    input [7:0] b,
    output [7:0] c,
    output [7:0] d
);

    assign c = a + b;

    assign d = { a[5:0], b[7:6] };
endmodule
Technology Schematic
Example of simple mistake

- No errors or warnings!
Top-Down Design Hierarchy

- Instantiate module (counter example with decoder)

```verilog
module decoder(
    input [3:0] count,
    output [6:0] seven_seg
);

// instantiate decoder module in counter
// using position of ports
decoder d1 (count_val, seven_seg_val);

// or using formal and actual names
decoder d1 (.count(count_val), .seven_seg(seven_seg_val));
```
Tri-state example

- Using conditional operator in continuous assignment