# CSE 453

## *Verilog HDL*
## *Behavioral Modeling*

### Behavioral vs. Structural Modeling

- What is the difference?
- When is each used?
- Why do both exist?

### Assignment Operator

- Operator: ASSIGN
- Continuous Assignment
- May be omitted for implicit assignments
- Example
  - `assign x = a & b;`
  - Implicit Assignment
    - `x = a & b;`

### Bitwise Logical Operators

- AND
  - `&`
- OR
  - `|`
- Inversion
  - `~`
- Exclusive OR
  - `^`
- Exclusive NOR
  - `~^`

### Putting It All Together

- Simple Example

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

- Full Adder Example

```
module fulladder(a,b,cin,s,cout);
    output s, cout;
    input a,b,cin;

      s =  (~a & ~b & cin) | (~a & b & ~cin) |
       (a & ~b & ~cin) | (a & b & cin);

      cout = (a & b) | (a & cin) | (b & cin);

endmodule
```

## Combining Structural & Behavorial Verilog

- Example

```
`timescale 1ns / 1ps

module test(f,a,b,s);
   output f;          // When s=0, f=a or b
   input a,b,s;       // When s=1, f=a and b

   and g1 (w1,a,b);
   or g2 (w2,a,b);

   mux m1(f,w1,w2,s);

   endmodule


module mux(y, d1, d0, s);
    output y;
    input d1, d0, s;

   assign y = s ? d1 : d0;

endmodule
```

## Behavioral Modeling

- Rules for Synthesizing Sequential Systems
  - Sensitivity list
    - Determines which signals will cause block to be executed
  - Sensitivity list should ONLY include clock, reset, preset clock edges!
    - Why?
      - ✓ Those are the only times the system is allowed to change!
  - Specify reset & preset conditions in always block first
  - All registers assigned values in sequential block are clocked flip-flops and will be latched according to the triggering event.
    - Hence a purely combinational statement stored in a register will be latched only at the triggering event.

## Event Control

- Always Statement
  - "While True" Statement
  - Procedural statements in this block are repeatedly executed
  - Syntax

```
always
  begin
     statement(s);
  end
```

- Wait Statement
  - Block of statements is executed only after the simulation has waited for the specified event
  - Syntax

```
wait(event)
  begin
     statement(s);
  end
```

- Event-Based Timing Control
  - The event control statement "@" indicated the simulator should suspend control of the block until a change occurs on the named entities
  - Keywords
    - posedge
    - negedge
  - Multiple events can be specified by separation with the keyword or
  - Syntax

```
@(event(s))
```

  - Example

```
always @(posedge clk or negedge rst)
  begin
     statement(s);
  end
```

- An always block which runs when any of the inputs inside the always block changes
- Useful for implementing combinational logic
- Restrictions
  - Can only use blocking (=)
  - Cannot use nonblocking (<=)

## Modeling Clocked (Sequential) Circuits

- Blocking (=) vs. Non-blocking (<=) Assignments
  - Non-blocking assignments are used to synchronize assignments so they happen all at once
    - First, all values are evaluated concurrently
    - Finally, the assignments are made
  - In other words, the right hand side of all assignment statements are evaluated (before the triggering event), then all assignments are made once the triggering event occurs.
- Blocking (=) vs. Non-blocking (<=) Assignments
  - Example

```
State[1] = State[1] & EN
State[0] = ~State[1]
```

    - Initially, if

```
State[1] = 1
State[0] = 0
EN = 0
```

- Non-Blocking
  State[0] = 0
  State[1] = 0
- Blocking
  State[0] = 1
  State[1] = 0
- ∴ Synthesis may work properly, but simulation may not!

**Example #1**

- The following circuit will blink LED #0 when enabled of the BASYS2 FPGA Development Board by Digilent

  - Period = 2 seconds
  - Enable = Switch #0
    - When Switch #0 is off, LED #0 is off

```
module blinking_led(clk, led, enable);
   input clk;
   output led;
   input enable;
   reg led;
   integer count;

   always @(posedge clk)
   begin
      if (enable==0)
    begin
          count=0;
          led=0;
       end
      else
       begin
          if (count > 50000000)
             begin
                count=0;
                led=~led;
               end
          count = count+1;
       end
   end
endmodule
```

- Pin Assignments
  - Clk
    - B8
  - Enable (SW0)
    - P11
  - LED (LD0)
    - M5

**Example #2**

- Switch #1 turns LED #0 on or off of the BASYS2 FPGA Development Board by Digilent

  - Enable = Switch #0
    - If Switch #0 is on, Switch #1 is enabled

```
module circuit(switch, enable, led);
    input switch;
    input enable;
    output led;
    integer count;
    reg led;

always @(switch)
begin

   if (enable==1)
      begin
         led=switch;
      end
   else
      begin
         led=0;
      end
end
endmodule
```

- Pins
  - Enable (SW0)
    - P11
  - LED (LD0)
    - M5
  - Switch (SW1)
    - L3

## Conditional Statements

- If…Then…Else
  - Syntax

```
IF (Expression)
BEGIN
    Statements;
END
ELSE
BEGIN
    Statements;
END
```

  - Note
    - Else clause is optional
- The Conditional Operator (?:)
  - Alternate Syntax for IF…THEN…ELSE
    - ✓ *Condition* ? *Then Assignment* : *Else Assignment*;
    - Example
      - ✓ 2-to-1 Multiplexor

```
module mux(y, d1, d0, s);
    output y;
    input d1, d0, s;

    assign y = s ? d1 : d0;

endmodule
```

- Case
  - Syntax

```
case (expression)
   alternative 1 : statement(s) 1 ;
   alternative 2 : statement(s) 2 ;
        . . .
   alternative n : statement(s) n ;
   default : default statement(s) ;
endcase
```

  - Default
    - Implemented when none of the alternatives are true
  - Variations
    - Replace *case* with *casex* or *casez* for comparing don't cares or high impedance states

## Relational Operators

- Used in Conditional Statements
- Operators
  - Equality (==)
  - Inequality (!=)
  - Case Equality (===)
  - Case Inequality (!==)
  - Less Than (<)
  - Less Than or Equal To (<=)
  - Greater Than (>)
  - Greater Than or Equal To (>=)
- Unary
- Perform bitwise operations on a single operand
- Example
  - Consider the 4-bit vector, *a*, and scalar *f*
  - `f = &a;`
  - All four bits are AND'ed together to produce a single bit output
  - If *a*=1011, *f*=0
  - If *a*=1111, *f*=1

## Operator Summary

| ***Operation Type*** | ***Symbol*** | ***Operation*** |
|---|---|---|
| Arithmetic | * | Multiplication |
| | / | Division |
| | + | Addition |
| | - | Subtraction |
| | % | Modulus |
| Logical | ! | Negation |
| | && | AND |
| | \|\| | OR |
| Relational | > | Greater Than |
| | < | Less Than |
| | <= | Greater Than or Equal To |
| | >= | Less Than or Equal To |
| Equality | == | Equality |
| | != | Inequality |
| | === | Case Equality |
| | !== | Case Inequality |

Samir Palnitkar, *Verilog HDL A Guide to Digital Design and Synthesis*, Prentice Hall, Inc., 4th Edition, Table 6-1, pp. 92-93, 1996

| *Operation Type* | *Symbol* | *Operation* |
|---|---|---|
| Bitwise Operation Type | ~ | Bitwise Negation |
| | & | Bitwise AND |
| | \| | Bitwise OR |
| | ^ | Bitwise XOR |
| | ~^ *or* ^~ | Bitwise XNOR |
| Reduction | & | Reduction AND |
| | ~& | Reduction NAND |
| | \| | Reduction OR |
| | ~\| | Reduction NOR |
| | ^ | Reduction XOR |
| | ~^ *or* ^~ | Reduction XNOR |
| Shift | >> | Right Shift |
| | << | Left Shift |
| Concatenation | { } | Concatentation |
| | { { } } | Replication |
| Conditional | ?: | Conditional |

Samir Palnitkar, *Verilog HDL A Guide to Digital Design and Synthesis*, Prentice Hall, Inc., 4th Edition, Table 6-1, pp. 92-93, 1996

## Operator Precedence

| *Operators* | *Symbols* | *Precedence* |
|---|---|---|
| Unary | + - ! ~ | Highest |
| Multiply, Divide, Modulus | * / % | |
| Add, Subtract | + | |
| Shift | - | |
| Relational | % | |
| Equality | ! | |
| Reduction | &, ~& | |
| | ^, ~^ | |
| | \|, ~\| | |
| Logical | && | |
| | \|\| | |
| Conditional | ?: | Lowest |

Samir Palnitkar, *Verilog HDL A Guide to Digital Design and Synthesis*, Prentice Hall, Inc., 4th Edition, Table 6-4, pp. 101-102, 1996

## Numbers

- Syntax
  - ☞ Sized
    - ↳ *Size'Format Number*
    - ↳ *Size*
      - ✓ Number of digits
    - ↳ *Format*
      - ✓ h (Hexadecimal)
      - ✓ d (Decimal)
      - ✓ o (Octal)
      - ✓ b (Binary)
    - ↳ Number
      - ✓ Number specified
  - ☞ Unsized
    - ↳ *'Format Number*
- Examples

  4'h a729
  'd 62923
  8'b 1101zzzz
  16'h x

## Concatenation Operator

- {…,…}
- Allows several wires to be combined together into a single multi-bit wire

## Loops

- While
  - ☞ Syntax

```
while (condition)
  begin
    statement(s);
  end
```

- For
  - ☞ Syntax

```
for (initial condition; termination condition; control
   variable change)
  begin
    statement(s);
  end
```

- Repeat
  - ☞ Repeats a given number of times based on the number, variable, or signal value given

- Syntax

```
repeat (number, variable, or signal value)
  begin
    statement(s);
  end
```

- Forever
  - Statement executes forever
    - Can be disabled by the keyword disable
  - Syntax

```
forever statement;
```

  - Example

```
forever #10 clk=~clk;
```

**Example #3**

- A 0, 1, 2, or 3 is displayed on the seven-segment display, depending upon whether button #0, #1, #2, or #3 is pressed of the BASYS2 FPGA Development Board by Digilent

- Enable
  - Switch #0

```
module ckt(btn, clk, a, b, c, d, e, f, g, an, rst);
input [3:0] btn;
input clk, rst;
output a, b, c, d, e, f, g;
output [3:0] an;
reg a, b, c, d, e, f, g;
reg [2:0] cstate, nstate;
reg [3:0] an;

 always @(posedge clk or negedge rst)
begin
  if (~rst) cstate<=7;
  else cstate<=nstate;
  an=14;
end

always @(btn or cstate)
case (btn)
  4'b1000:  nstate=3;         // Button 3 pressed
  4'b0100:  nstate=2;         // Button 2 pressed
  4'b0010:  nstate=1;         // Button 1 pressed
  4'b0001:  nstate=0;         // Button 0 pressed
  4'b0000:  nstate=cstate;    // No button pressed
  default:  nstate=7;         // No button pressed yet or
                    // multiple buttons pressed
```

```
endcase

always @(posedge clk)
case (cstate)
  3:  begin  // Button 3 pressed
         a=0; b=0; c=0; d=0; e=1; f=1; g=0;
      end
  2:  begin   // Button 2 pressed
         a=0; b=0; c=1; d=0; e=0; f=1; g=0;
      end
  1:  begin   // Button 1 pressed
        a=1; b=0; c=0; d=1; e=1; f=1; g=1;
      end
  0:  begin   // Button 0 pressed
        a=0; b=0; c=0; d=0; e=0; f=0; g=1;
      end
  7:  begin   // No button pressed yet or multiple buttons pressed
        a=1; b=1; c=1; d=1; e=1; f=1; g=1;
      end
endcase

endmodule
```

- Pins
  - Switch #0
    - Pin P11
  - Buttons #0 - #3
    - Pins G12, C11, M4, A7
  - 7-Segment Display
    - Common Anodes (A-G)
      - ✓ L14, H12, N14, N11, P12, L13, M12
    - Nodes (AN0-AN3)
      - ✓ F12, J12, K14, M13
      - ✓ Note K14 & M13 are both listed as AN2 in the Reference Manual
        - K14 = AN2
        - M13 = AN3
  - Clock
    - Pin B8

## References


- Michael D. Ciletti, *Advanced Digital Design with the Verilog HDL*, Pearson Education, Inc. (Prentice Hall), 2003
- Donald E. Thomas and Philip R. Moorby, *The Verilog Hardware Description Language*, Kluwer Academic Publishers, 1998
- Samir Palnitkar, *Verilog HDL A Guide to Digital Design and Synthesis*, Prentice Hall, Inc., 4th Edition, 1996

- David R. Smith and Paul D. Franzon, *Verilog Styles of Digital Systems*, Prentice Hall, Inc., 2000
- *Digilent Basys 2 Board Reference Manual*, Digilent, Inc., May 25, 2009
- *Digilent BASYS 2 System Board Schematics*, Digilent, Inc., December 12, 2008
- Richard E. Haskell and Darrin M. Hanna, *Digital Design Using Digilent FPGA Boards – Verilog/Active-HDL Edition*, LBE Books, 2009
- Richard E. Haskell and Darrin M. Hanna, *Introduction to Digital Design Using Digilent FPGA Boards – Block Diagram/Verilog Examples*, LBE Books, 2nd Edition, 2012
- http://www.utdallas.edu/~kad056000/index_files/verilog/reduction.html
- http://inst.eecs.berkeley.edu/~cs150/fa08/Documents/Always.pdf