UCI University of California, Irvine

EECS 31L: INTRODUCTION TO DIGITAL DESIGN LAB LECTURE 2

Salma Elmalaki
salma.elmalaki@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

LECTURE 2: LOGISTICS

- Lab 1 deadline is Sunday 1/17/2021 at 11:00 pm on canvas
  - Late submission between 11:00pm to 11:59pm
  - Server closes immediately at midnight

- Lab 2 will be assigned on Monday 1/18/2021
  - You will have two weeks to finish it

- Monday 1/18 (MLK) - no lab session
  - Makeup: Extended OH for Maryam
  - You can attend any lab session this week

UCI University of California, Irvine

LECTURE 2: OVERVIEW

- Deeper insight in Verilog structures with additional examples

- Verilog data types
- Verilog operators
- Some examples for concurrency

- Lab 2 description

LECTURE 2: OVERVIEW

- Deeper insight in Verilog structures with additional examples

- Verilog data types
- Verilog operators
- Some examples for concurrency

- Lab 2 description

VERILOG BASICS

- Each design in Verilog consists of one main part:

module

Circuit Ports (I/Os) (mandatory)

Generic constants (optional)

Other declaration (optional)

Structure 1

```
 module module_name (port_name1, port_name2, … );
        port_mode1  port_name1, port_name2;
        port_mode2  port_name3;
//body of module
endmodule
```

Structure 2

```
 module module_name (port_mode1 port_name1,
port_name2, port_mode2  port_name 3);

//body of module
endmodule
```

Note the semicolon at the end of the port list!

VERILOG BASICS

- Each design in Verilog consists of one main part:

module

The main code (functionality)

How to describe functionality?

Behavioral

What a module does

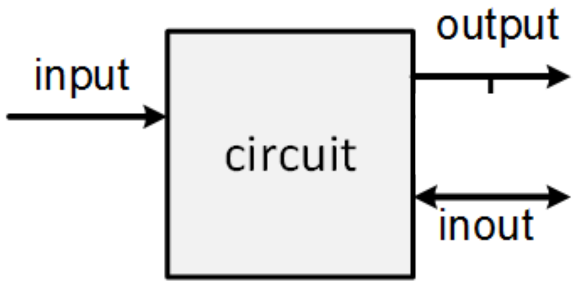
Structural

How a module is built

```
module identifier(port_list);
[ports_declaration]

//module_body

endmodule
```

VERILOG BASICS: PORT MODES

- input: can only be read
- output: can be on either side of assignment
- inout: can be read or written (is a bidirectional bus)



UCI University of California, Irvine

VERILOG BASICS: CONTINUOUS ASSIGNMENT STATEMENTS

assign statement

Describes only combinational logic

These statements are re-evaluated anytime any of the inputs on the right hand side changes

```
assign <out> = <in0> <operator> <in1> … ;
```

Example

```
assign Y = A + B;
```

VERILOG BASICS: CONTINUOUS ASSIGNMENT STATEMENTS

Find the bug in this 2 input AND gate!

```
module assign_example(A,B,Y);
    input A;
    input B;
    output Y

    Y = A & B;
endmodule
```

VERILOG BASICS: CONTINUOUS ASSIGNMENT STATEMENTS

Fixed version

```
module assign_example(A,B,Y);
    input A;
    input B;
    output Y;

    assign Y = A & B;
endmodule
```

UCI University of California, Irvine

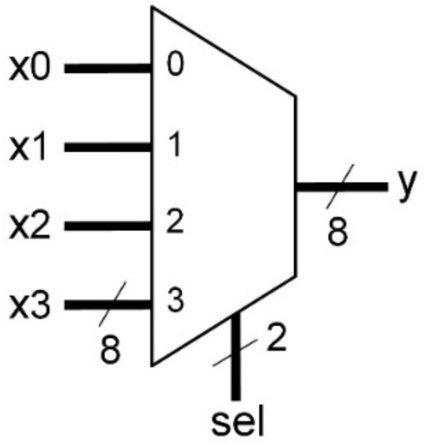
VERILOG CODE STRUCTURE EXAMPLE

4x1 multiplexer

```
module mux_4_1 (x0,x1,x2,x3,sel,y);
  input [7:0] x0,x1,x2,x3;
  input [1:0] sel;
  output [7:0] y;

  assign y =  (sel==2'b00) ? x0:
        (sel==2'b01) ? x1:
         (sel==2'b10) ? x2:
          x3;

endmodule
```

Size (bit-width)

sel_1	sel_0	Y
0	0	X_0
0	1	X_1
1	0	X_2
1	1	X_3



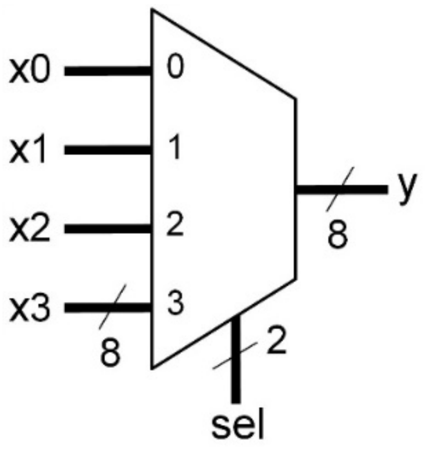
UCI University of California, Irvine

VERILOG CODE STRUCTURE EXAMPLE

4x1 multiplexer

```
module mux_4_1 (x0,x1,x2,x3,sel,y);
  input [7:0] x0,x1,x2,x3;
  input [1:0] sel;
  output [7:0] y;

  assign y =  (sel==2'b00) ? x0:
         (sel==2'b01) ? x1:
         (sel==2'b10) ? x2:
          x3;
/*Or it can be written as case statement in always block or if statement (revise lecture 1 behavioral structural)
 */
endmodule
```

sel_1	sel_0	Y
0	0	X_0
0	1	X_1
1	0	X_2
1	1	X_3



UCI University of California, Irvine

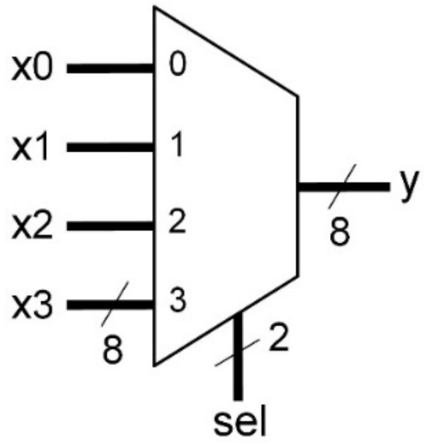
VERILOG CODE STRUCTURE EXAMPLE

```
module mux_4_1 (x0,x1,x2,x3,sel,y);
  input [7:0] x0,x1,x2,x3;
  input [1:0] sel;
  output [7:0] reg y;
always @(x0,x1,x2,x3,sel)
begin
        case (sel):
                2'b00:
                        y = x0;
                2'b01:
                        y = x1;
                2'b10:
                        y = x2;
                2'b11:
                        y = x3;
      endcase
end
endmodule
```

Sensitivity list

Rule
assign statements cannot be used inside *always* block

sel_1	sel_0	Y
0	0	X_0
0	1	X_1
1	0	X_2
1	1	X_3



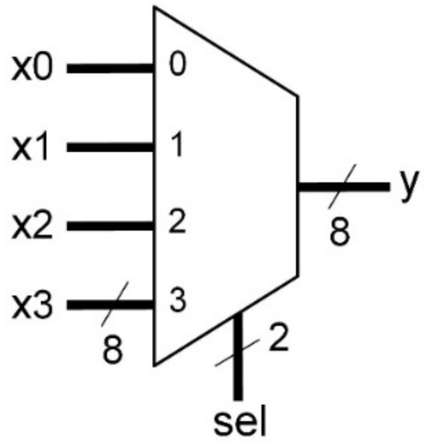
VERILOG CODE STRUCTURE EXAMPLE

```
module mux_4_1 (x0,x1,x2,x3,sel,y);
  input [7:0] x0,x1,x2,x3;
  input [1:0] sel;
  output [7:0] reg y;
always @(*)
begin
        case (sel):
                2'b00:
                        y = x0;
                2'b01:
                        y = x1;
                2'b10:
                        y = x2;
                2'b11:
                        y = x3;
        endcase
end
endmodule
```

Sensitivity list

<u>Rule</u>
assign statements cannot be used inside *always* block

sel_1	sel_0	Y
0	0	X_0
0	1	X_1
1	0	X_2
1	1	X_3



VERILOG DATA TYPES

-Net

-Register

-Vectors (Defined and accessed by brackets)

- multi-bit words of type reg or net (wire)

-Integer

- Defined by predefined word integer

–Real

- Defined by predefined word real

–Parameter

- Define global constants
- Defined by predefined word parameter

–Array (Defined and accessed by brackets)

- Arrays for integer, time, reg, and vectors of reg

VERILOG DATA TYPES

4-state types

Value	Definition
0	Logic 0 (false)
1	Logic 1 (true)
X	Unknown
Z	High impedance

- **Nets**
  - Defined by predefined word *wire,* with 4 known values
  - Can be driven in <u>concurrent statements</u>
  - They change continuously by circuit
  - Syntax: Example:

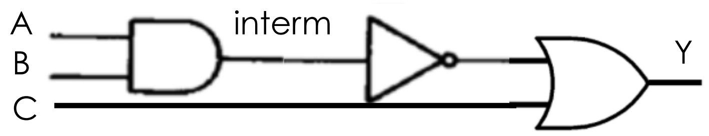
```
wire net_name;   //just name
wire S1= 1'b0;   //name and initial value
```

- **Registers**
  - Defined by predefined word *reg,* with 4 known values
  - Can be driven in <u>sequential statements</u>
  - In contrast to nets stores values until they are updated
  - Syntax: Example:

```
reg reg_name; //just name
reg S1= 1'b0; //name and initial values
```

VERILOG DATA TYPES

Example

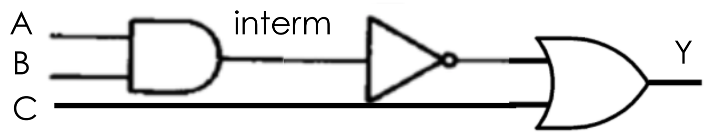


```
module always_ex(A,B,C,Y);
    input A;
    input B;
    input C;









endmodule
```

```
module assign_ex(A,B,C,Y);
    input A;
    input B;
    input C;









endmodule
```

VERILOG DATA TYPES

Example

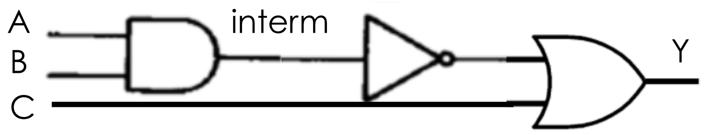


```
module always_ex(A,B,C,Y);
    input A;
    input B;
    input C;
    output reg Y;





endmodule
```

```
module assign_ex(A,B,C,Y);
    input A;
    input B;
    input C;
    output Y;





endmodule
```

VERILOG DATA TYPES

Example

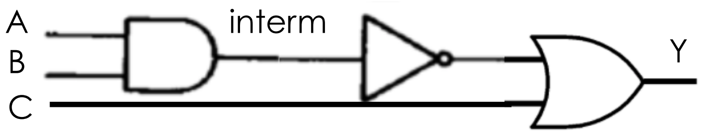


```
module always_ex(A,B,C,Y);
    input A;
    input B;
    input C;
    output reg Y;

    reg interm;




endmodule
```

```
module assign_ex(A,B,C,Y);
    input A;
    input B;
    input C;
    output Y;

    wire interm;




endmodule
```

VERILOG DATA TYPES

Example



```
module always_ex(A,B,C,Y);
    input A;
    input B;
    input C;
    output reg Y;

    reg interm;

    always @(*) begin
        interm = A & B;
        Y      = ~interm | C;
    end
endmodule
```

=

```
module assign_ex(A,B,C,Y);
    input A;
    input B;
    input C;
    output Y;

    wire interm;


    assign interm = A & B;
    assign Y      = ~interm | C;

endmodule
```

UCI University of California, Irvine

VERILOG DATA TYPES

•**Vector**

Multiple bits.
A reg or a net can be declared as a vector.
Vectors are declared by brackets [ ]
Examples:

```
reg [7:0] MB1; //8-bit reg vector with MSB=7 LSB=0
wire [0:7] MB2; //8-bit wire vector with MSB=0 LSB=7
reg [3:0] bitslice;
 // with initialization
wire [3:0] multiBitWord1 = 4'b1010;
reg [7:0] total = 8'd12;   //8 bits and decimal value 12
                          // 8'b0000_1100
```

Referencing vectors

```
a = multiBitWord1[3];    // a = 1
bitslice1 = total[3:0];  // bitslice1 = 4'b1100
bitslice2 = total[7:4];  // bitslice2 = 4'b0000
```

VERILOG DATA TYPES

•Vector

Multiple bits.
A reg or a net can be declared as a vector.
Vectors are declared by brackets [ ]
Examples:

```
reg [7:0] MB1; //8-bit reg vector with MSB=7 LSB=0
wire [0:7] MB2; //8-bit wire vector with MSB=0 LSB=7
reg [3:0] bitslice;
```

•Array

Array as per definition in a collection of elements of same type.

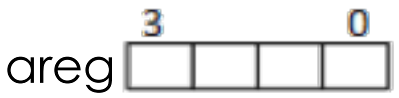
```
reg signed [3:0] anArray [0:4] ;
//anArray has 5 elements and each element is 4 signed
bits
```

Size of each element

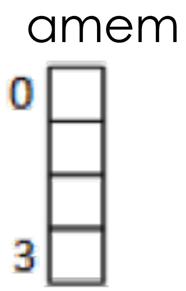
Size of the array

VERILOG DATA TYPES

```
//An 4-bit vector
wire [3:0] areg;
```

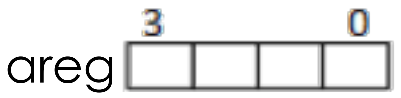


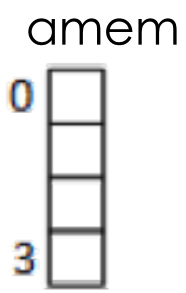
```
//A memory of 4 one-bit elements
reg amem [0:3];
```



VERILOG DATA TYPES

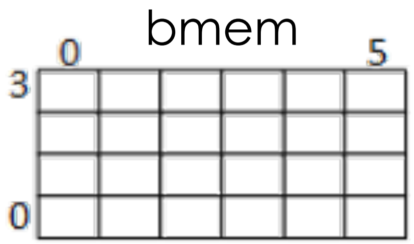
```
//An 4-bit vector
wire [3:0] areg;
```



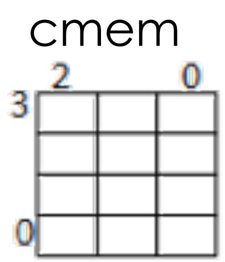


```
//A memory of 4 one-bit elements
reg amem [0:3];
```

```
//A memory of four 6-bit
reg [0:5] bmem [3:0];
```

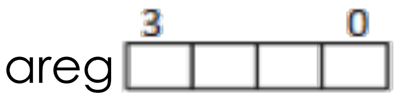


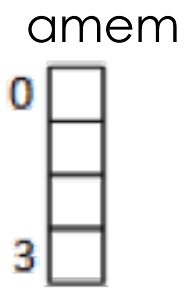
```
//A two dimensional memory of one-bit elements
reg cmem [3:0][2:0];
```

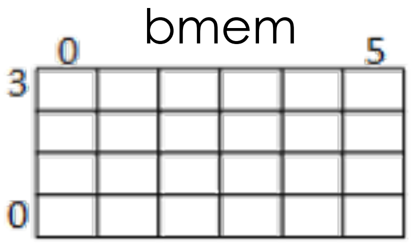


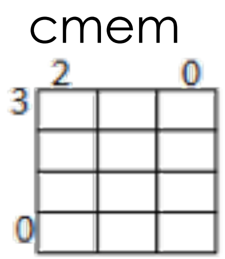
VERILOG DATA TYPES

```
//An 4-bit vector
wire [3:0] areg;

//A memory of 4 one-bit elements
reg amem [0:3];

//A memory of four 6-bit
reg [0:5] bmem [3:0];

//A two dimensional memory of one-bit elements
reg cmem [3:0][2:0];
```





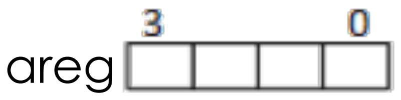


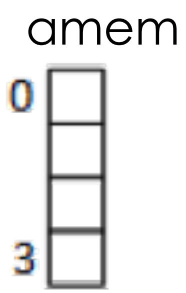


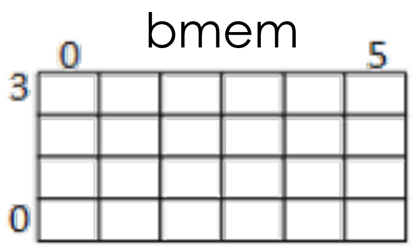
VERILOG DATA TYPES

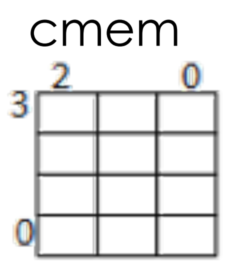
- **Parameter**

```
//An 4-bit vector
wire [3:0] areg;

//A memory of 4 one-bit elements
reg amem [0:3];

//A memory of four 6-bit
reg [0:5] bmem [3:0];

//A two dimensional memory of one-bit elements
reg cmem [3:0][2:0];
```









VERILOG DATA TYPES

- **Parameter**

Represents a global constant
Declared by a predefined word *parameter*
Example:

```
   parameter N = 3;

  //An 4-bit vector
  wire [N:0] areg;

  //A memory of 4 one-bit elements
  reg amem [0:N];

  //A memory of four 6-bit
  reg [0:5] bmem [N:0];

  //A two dimensional memory of one-bit elements
  reg cmem [N:0][2:0];
```

LECTURE 2: OVERVIEW

- Deeper insight in Verilog structures with additional examples

- Verilog data types
- Verilog operators
- Some examples for concurrency

- Lab 2 description

OPERATORS

- Logical operators
  - Such as AND, OR, XOR, …
- Relational
  - Compare relation between objects
  - Such as less than, equal, greater than or equal, …
- Arithmetic operators
  - Such as addition and subtraction
- Shift operators
  - Move bits of an object to a certain direction

BITWISE LOGICAL OPERATORS

- Single operand or multiple operands
- The operand(s) has(have) single or multiple bits
- Example

If A ="0101", B ="1100"

C = A ^ B   D = A & B

means C = "1001"

means D = "0100"

Operation	Verilog
AND	&
OR	|
NAND	~(&)
NOR	~(|)
XOR	^
XNOR	~(^)
NOT	~

BITWISE LOGICAL OPERATORS

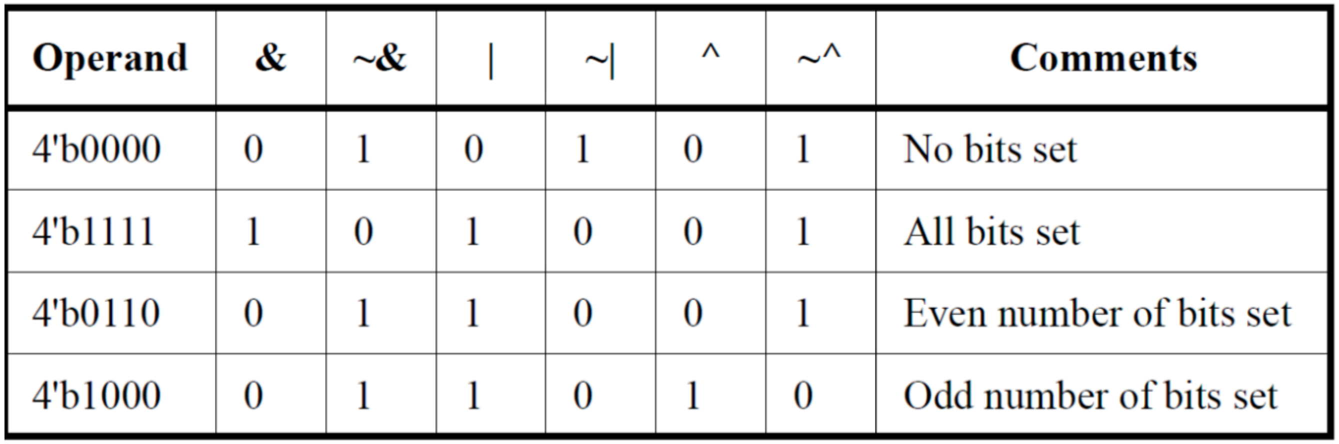
Reduction logical operators

–Need a single operand
–The result is a Boolean variable
–Example:
•If x = “1010”
•y = &x // y = (1 & 0 & 1 & 0)
•y = |x // y = (1 | 0 | 1 | 0)
•y = ~&x // y = (1 ~& 0 ~& 1 ~& 0)

BITWISE LOGICAL OPERATORS

Reduction logical operators

–Need a single operand
–The result is a Boolean variable
–Example:
•If x = “1010”
•y = &x   // y = (1 & 0 & 1 & 0)
•y = |x   // y = (1 | 0 | 1 | 0)
•y = ~&x   // y = (1 ~& 0 ~& 1 ~& 0)

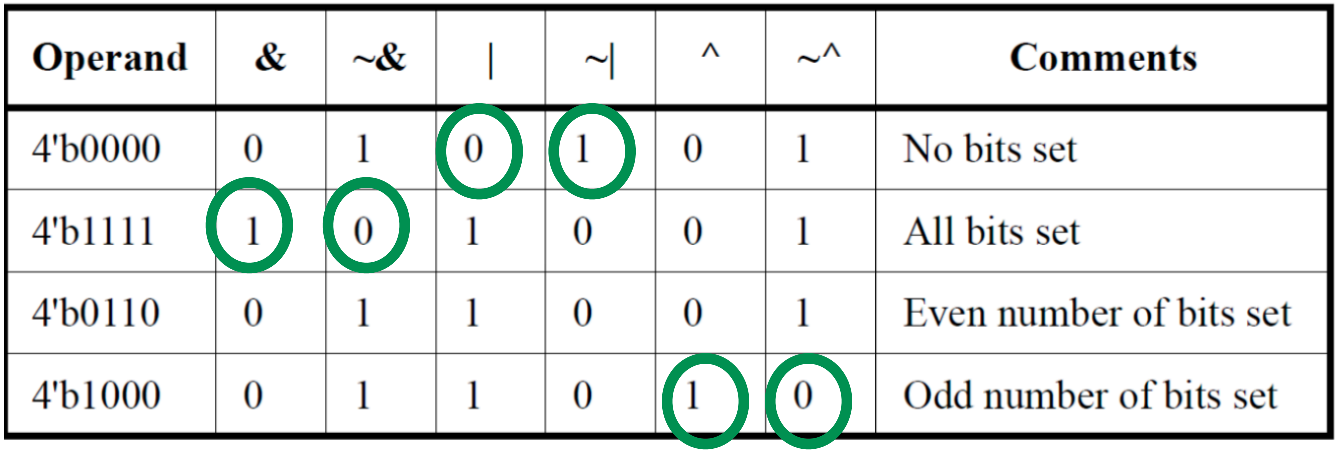


Operand	&	~&	|	~|	^	~^	Comments
4'b0000	0	1	0	1	0	1	No bits set
4'b1111	1	0	1	0	0	1	All bits set
4'b0110	0	1	1	0	0	1	Even number of bits set
4'b1000	0	1	1	0	1	0	Odd number of bits set

BITWISE LOGICAL OPERATORS

Reduction logical operators

–Need a single operand
–The result is a Boolean variable
–Example:
•If x = "1010"
•y = &x   // y = (1 & 0 & 1 & 0)
•y = |x   // y = (1 | 0 | 1 | 0)
•y = ~&x   // y = (1 ~& 0 ~& 1 ~& 0)



Operand	&	~&	|	~|	^	~^	Comments
4'b0000	0	1	0	1	0	1	No bits set
4'b1111	1	0	1	0	0	1	All bits set
4'b0110	0	1	1	0	0	1	Even number of bits set
4'b1000	0	1	1	0	1	0	Odd number of bits set

BOOLEAN LOGICAL OPERATORS

•Need at least two operands

•Return Boolean type value

–false (0)

–true(none-zero)

•More than one condition can be checked

Operation	Verilog
and	&&
or	||
not	!

Examples:

If (cond1 && cond2) ….

UCI University of California, Irvine

RELATIONAL OPERATORS

The result of these operators are Boolean

- (in)equality
  - It compares 2 values(1,0)
    - The result is unknown (x) if operand are other than 1 and 0
  - Bit by bit comparison
  - If result of condition expression is x
    - Both true and false parts are executed
- (in)equality inclusive
  - It compares all 4 values(1,0,x,z)

Operation	Verilog
Equality	==
Inequality	!=
Less than	<
less than or equal	>=
greater than	>
greater than or equal	>=
equality inclusive	= = =
inequality inclusive	! = =

ARITHMETIC OPERATORS

C = A operation B

Description	Operator	A and B types	Y type
add, sub, multiplication, division,	+ , - , *, /	A numeric B numeric	numeric
Modulus	mod	A numeric (not real) B numeric (not real)	numeric (not real)
Exponent	**	A numeric B numeric	numeric
concatenation	{ , } [1]	A numeric or array B numeric or array	Same as A
Repetition	{N{A}} [1]	A numeric or array	Same as A

```
reg a = 1'b1; reg b = 2'b00
y = { 4{a}, 2{b} } // y = 8'b11110000
```

SHIFT OPERATORS

C = A shift_operation **1**

	Verilog	**C (before)**	**C (after)**
Shift logic left –fills with 0	A << 1	1010	0100
Shift logic right -fills with 0	A >> 1	1010	0101
Shift arithmetic left - fills with LSB	A <<< 1	1001	0011
Shift arithmetic right - fills with MSB	A >>> 1	1010	1101

Red font shows the location of same bit before and after shift
Green font shows the location of new values which is added to the empty location after shifting

Break 10 minutes

LECTURE 2: OVERVIEW

- Deeper insight in Verilog structures with additional examples

- Verilog data types
- Verilog operators
- Some examples for concurrency

- Lab 2 description

CONCURRENCY

```
module gates(input [3:0] a, b, output [3:0] y1, y2, y3,
y4,y5);

/* five different two-input logic gates acting on 4-bit
buses */

   assign y1=a&b;           //AND
   assign y2=a|b;           //OR
   assign y3=a^b;           //XOR
   assign y4 = ~(a & b);    // NAND
   assign y5 = ~(a | b);    // NOR
endmodule
```

Concurrent statements

CONCURRENCY

```
module and8( input [7:0] a, output y, z);
    assign y = &a;
    assign z = a[7] & a[6] & a[5] & a[4] & a[3] & a[2] & a[1] & a[0];
endmodule
```

What can you say about the value of y and z?

Both y and z outputs hold the same result

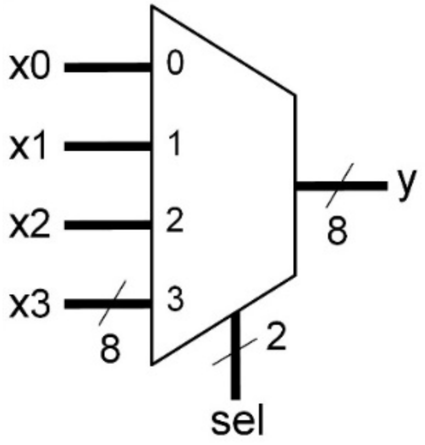
CONCURRENCY

Suppose that mux2to1 module is already defined

```
module mux2to1(d0, d1,s, y);
input [3:0] d0, d1;
input  s;
output [3:0] y;
// module body
endmodule
```

sel_1	sel_0	Y
0	0	X_0
0	1	X_1
1	0	X_2
1	1	X_3

```
module mux4to1(x0, x1, x2, x3, s, y);
input [3:0] x0, x1, x2, x3;
input [1:0] s;
output [3:0] y;
wire [3:0] low, high;
mux2to1 lowmux (x0, x1, s[0], low);
mux2to1 highmux (x2, x3, s[0], high);
mux2to1 finalmux(low, high, s[1], y);
endmodule
```



LECTURE 2: OVERVIEW

- Deeper insight in Verilog structures with additional examples

- Verilog data types
- Verilog operators
- Some examples for concurrency

- Lab 2 description

LECTURE 2: LAB 2

- Design an ALU (Arithmetic Logic Unit) module in Verilog
  - Modularity is essential to the success of large design
    - Remember 4to1 mux from 2to1 mux
  - Use what you learned here about mux to choose between different operations
  - Think about concurrency to make it efficient
- Make sure to attend the lab sessions for more explanation on the lab manual