



COMP311-1 Logic Circuit Design

Chap. 3

Instructor: Taigon Song (송대건)

2019 Fall

In-class Assignment

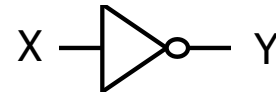
- Design an inverter and its testbench.
- Module
 - inverter
 - input: A
 - output: Z
- Testbench module
 - inverter_tb
 - one wire (output): output
 - one reg (input): clock
 - Clock changes in every #1 cycle
 - clock initializes as '0'
 - timescale: 1ns / 1ns

Hint: Bitwise operators

Symbol	Definition
~	Bitwise negation
&	Bitwise and
	Bitwise or
^	Bitwise xor
^~ or ~^	Bitwise xnor

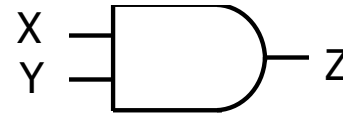
From Boolean expressions to logic gates

• NOT X' \bar{X} $\sim X$



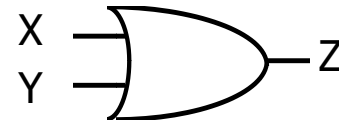
X	Y
0	1
1	0

• AND $X \cdot Y$ XY $X \wedge Y$



X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

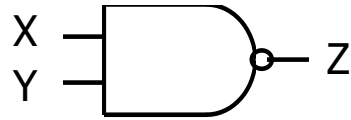
• OR $X + Y$ $X \vee Y$



X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

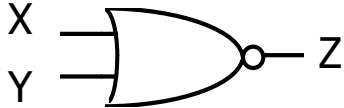
From Boolean expressions to logic gates (cont'd)

- NAND



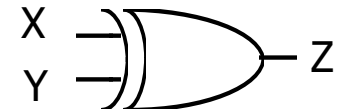
X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

- NOR



X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

- XOR
 $X \oplus Y$



X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

$X \text{ xor } Y = X Y' + X' Y$
X or Y but not both
("inequality", "difference")

- XNOR
 $X = Y$



X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

$X \text{ xnor } Y = X Y + X' Y'$
X and Y are the same
("equality", "coincidence")

In-class Assignment

- inv.v
- inv_tb.v

```
module inverter(a,z);  
    input a;  
    output z;  
  
    assign z = ~a;  
  
endmodule
```

inv.v

```
`timescale 1ns/1ns  
module inverter_tb();  
    wire z;  
    reg a;  
  
    inverter u1(a,z);  
  
    initial  
        a = 0;  
  
    always  
        #1 a = ~a;  
  
endmodule
```

inv_tb.v

Lexical Conventions

- Whitespace
 - Blank spaces (\b), tabs (\t), newlines(\n) comprise whitespace
 - Typically ignored by Verilog
 - Except when separates tokens
 - Not ignored in strings
- Comments
 - One line: //
 - Many-lines: /* */

```
a = b && c; // This is a one-line comment

/* This is a multiple line
   comment */

/* This is /* an illegal */ comment */

/* This is //a legal comment */
```

Lexical Conventions

- Operators

- Unary, binary, and ternary

```
a = ~ b; // ~ is a unary operator. b is the operand
```

```
a = b && c; // && is a binary operator. b and c are operands
```

```
a = b ? c : d; // ?: is a ternary operator. b, c and d are operands
```

- Number specification

- Sized numbers

```
4'b1111 // This is a 4-bit binary number
```

```
12'habc // This is a 12-bit hexadecimal number
```

```
16'd255 // This is a 16-bit decimal number.
```

- Unsized numbers

```
23456 // This is a 32-bit decimal number by default
```

```
'hc3 // This is a 32-bit hexadecimal number
```

```
'o21 // This is a 32-bit octal number
```

Lexical Conventions

- X or Z values

- Four bits in hexadecimal base
- Three bits for octal base
- One bit for in binary base

```
12'h13x // This is a 12-bit hex number; 4 least significant bits  
unknown
```

```
6'hx // This is a 6-bit hex number
```

```
32'bz // This is a 32-bit high impedance number
```

- Negative numbers

- Minus sign before the size for a constant number

6-bit

```
-6'd3 // 8-bit negative number stored as 2's complement of 3
```

```
-6'sd3 // Used for performing signed integer math
```

```
4'd-2 // Illegal specification
```


Lexical Conventions

- Underscore characters and question marks
 - “_” for readability
 - “?” for representing “z”

```
12'b1111_0000_1010 // Use of underline characters for readability
4'b10?? // Equivalent of a 4'b10zz
```

- Strings
 - Letters enclosed in double quotes (“ ”)
 - Recommend not to extend one line
 - Used mostly during testbench and debugging

```
"Hello Verilog World" // is a string
"a / b" // is a string
```

Lexical Conventions

- Identifiers and Keywords

- Keyword (키워드)
 - In lowercase
- Identifier (식별자)
 - Alphanumeric characters, underscore("_"), or the dollar sign (\$)
 - Case sensitive → However, recommend to use small case letters/numbers only
 - \$ as first character is reserved for system tasks

```
reg value; // reg is a keyword; value is an identifier
input clk; // input is a keyword, clk is an identifier
```

- Escaped Identifiers

- Begin with backslash(\) and end with whitespace (space, tab, newline)
- **Don't use them!**


```
\a+b-c
\**my_name**
```

Data Types

- Value Set
 - Value levels

Value Level	Condition in Hardware Circuits
0	Logic zero, false condition
1	Logic one, true condition
x	Unknown logic value
z	High impedance, floating state

- Strength levels
 - Stronger gets the value when in conflict

Strength Level	Type	Degree
supply	Driving	strongest
strong	Driving	
pull	Driving	
large	Storage	
weak	Driving	
medium	Storage	
small	Storage	weakest
highz	High Impedance	

Data Types

- Nets
 - Connection between hardware elements
 - Keyword **wire**
 - Default one-bit, unless explicitly declared as vectors
 - Default value: z
 - Net: not a keyword, but represents a class of data types (wire, wand, wor, tri, triand, trior, trireg...etc)

```
wire a; // Declare net a for the above circuit
wire b,c; // Declare two wires b,c for the above circuit
wire d = 1'b0; // Net d is fixed to logic value 0 at declaration.
```

Data Types

- Registers

- Represent data storage elements
- Retain value until another value is placed onto them
- Not physical FF (flip-flop)s
- No need for a driver
- Value can change anytime in simulation by assignment
- Keyword: **reg**
- Default data type: x

```
reg reset; // declare a variable reset that can hold its value
initial // this construct will be discussed later
begin
    reset = 1'b1; //initialize reset to 1 to reset the digital circuit.
    #100 reset = 1'b0; // after 100 time units reset is deasserted.
end
```

- Signed register

```
reg signed [63:0] m; // 64 bit signed value
integer i; // 32 bit signed value
```

Data Types

- Vectors
 - Multiple bit widths
 - If bit width not declared, default is scalar (1-bit)
 - Can be declared as [high# : low#] or [low# : high#]
 - Left is always the most significant bit of the vector

```
wire a; // scalar net variable, default
wire [7:0] bus; // 8-bit bus
wire [31:0] busA, busB, busC; // 3 buses of 32-bit width.
reg clock; // scalar register, default
reg [0:40] virtual_addr; // Vector register, virtual address 41 bits
wide
```

- Vector part select

```
busA[7] // bit # 7 of vector busA
bus[2:0] // Three least significant bits of vector bus,
// using bus[0:2] is illegal because the significant bit should
// always be on the left of a range specification
virtual_addr[0:1] // Two most significant bits of vector virtual_addr
```

Data Types

- Variable Vector Part Select
 - Another syntax of selecting vectors
 - Start could be a variable, but not the width

```
reg [255:0] data1; //Little endian notation
reg [0:255] data2; //Big endian notation
reg [7:0] byte;
```

```
//Using a variable part select, one can choose parts
byte = data1[31 -: 8]; //starting bit = 31, width = 8 => data[31:24]
byte = data1[24 +: 8]; //starting bit = 24, width = 8 => data[31:24]
byte = data2[31 -: 8]; //starting bit = 31, width = 8 => data[24:31]
byte = data2[24 +: 8]; //starting bit = 24, width = 8 => data[24:31]
```

```
//The starting bit can also be a variable. The width has
//to be constant. Therefore, one can use the variable part select
//in a loop to select all bytes of the vector.
for (j=0; j<=31; j=j+1)
    byte = data1[(j*8) +: 8]; //Sequence is [7:0], [15:8]... [255:248]
```

```
//Can initialize a part of the vector
data1[(byteNum*8) +: 8] = 8'b0; //If byteNum = 1, clear 8 bits [15:8]
```

Data Types

Integer, real, time, and register data types

- Integer
 - General purpose register data type
 - For purposes such as counting
 - Default width is host-machine specific, but typically 32-bits or more.
 - Registers store values as unsigned quantities
 - Integers store values as signed quantities
 - Mostly used for testbenches and module debugging

```
integer counter; // general purpose variable used as a counter.  
initial  
    counter = -1; // A negative one is stored in the counter
```


Data Types

Integer, real, time, and register data types

- Real
 - Real number constants and real register data types
 - Floating point numbers
 - Decimal notation (e.g., 3.14)
 - Scientific notation ($3.14e6 = 3.14 \times 10^6$)
 - Default value is 0
 - When a real value assigned to an integer, the real number is rounded off (반올림) to the nearest integer.

```
real delta; // Define a real variable called delta
initial
begin
    delta = 4e10; // delta is assigned in scientific notation
    delta = 2.13; // delta is assigned a value 2.13
end
integer i; // Define an integer i
initial
    i = delta; // i gets the value 2 (rounded value of 2.13)
```

Data Types

Integer, real, time, and register data types

- Time
 - A special time register data type is used in Verilog to store simulation time
 - The width for time register data types is implementation-specific but is at least 64 bits
 - The system function \$time is invoked to get the current simulation time
 - Simulation time measured in terms of simulation 'seconds'
 - Further discussed in Sec. 9.4.

```
time save_sim_time; // Define a time variable save_sim_time
initial
    save_sim_time = $time; // Save the current simulation time
```

Data Types

Integer, real, time, and register data types

- Arrays
 - Allowed in Verilog for reg, integer, time, real, realtime, and vector register data types.
 - A vector is a single element that is n-bits wide.
 - Arrays are multiple elements that are 1-bit or n-bits wide.

```
integer count[0:7]; // An array of 8 count variables

reg bool[31:0]; // Array of 32 one-bit boolean register variables

time chk_point[1:100]; // Array of 100 time checkpoint variables

reg [4:0] port_id[0:7]; // Array of 8 port_ids; each port_id is 5 bits
wide

integer matrix[4:0][0:255]; // Two dimensional array of integers

reg [63:0] array_4d [15:0][7:0][7:0][255:0]; //Four dimensional array

wire [7:0] w_array2 [5:0]; // Declare an array of 8 bit vector wire

wire w_array1[7:0][5:0]; // Declare an array of single bit wires
```

Data Types

Integer, real, time, and register data types

- Arrays
 - Some examples

```
count[5] = 0; // Reset 5th element of array of count variables
chk_point[100] = 0; // Reset 100th time check point value
port_id[3] = 0; // Reset 3rd element (a 5-bit value) of port_id array.
```

```
matrix[1][0] = 33559; // Set value of element indexed by [1][0] to
33559
```

```
array_4d[0][0][0][0][15:0] = 0; //Clear bits 15:0 of the register
//accessed by indices [0][0][0][0]
```

```
port_id = 0; // Illegal syntax - Attempt to write the entire array
```

```
matrix [1] = 0; // Illegal syntax - Attempt to write [1][0]..[1][255]
```

Data Types

Integer, real, time, and register data types

- Memories
 - Modeled as a one-dimensional array of registers.

```
reg memlbit[0:1023]; // Memory memlbit with 1K 1-bit words
reg [7:0] membyte[0:1023]; // Memory membyte with 1K 8-bit words (bytes)
membyte[511] // Fetches 1 byte word whose address is 511.
```

Data Types

Integer, real, time, and register data types

- Parameters
 - Similar as parameter definition in C/C++
 - Cannot be used as variables
 - Allows module instance customization
 - Discussed more on Chap. 9.

```
parameter port_id = 5; // Defines a constant port_id
parameter cache_line_width = 256; // Constant defines width of cache
line
parameter signed [15:0] WIDTH; // Fixed sign and range for parameter
// WIDTH
```

- Local parameters
 - Used with localparam keyword
 - E.g., state encoding for a state machine
 - State encoding cannot be changed.

```
localparam state1 = 4'b0001,
state2 = 4'b0010,
state3 = 4'b0100,
state4 = 4'b1000;
```

Data Types

Integer, real, time, and register data types

- Strings
 - Can be stored in reg.
 - Width of reg should be large enough to hold the string.
 - Each character in the string takes up 8 bits (1 byte).
 - Empty reg filled with 0 after string is inserted
 - If String width > reg width, → truncates the leftmost bits of the string

```
reg [8*18:1] string_value; // Declare a variable that is 18 bytes wide
initial
    string_value = "Hello Verilog World"; // String can be stored
                                         // in variable
```

- Special characters
 - Represented in escape characters (\)

Escaped Characters	Character Displayed
\n	newline
\t	tab
%%	%
\\	\
\"	"
\ooo	Character written in 1?3 octal digits

System Tasks and Compiler Directives

System tasks

- In \$<keyword>
 - Such as
 - Displaying on the screen
 - Monitoring values of nets
 - Stopping simulation
 - Finishing simulation

System Tasks and Compiler Directives

System tasks

- **\$display**

Usage: `$display(p1, p2, p3,....., pn);`

- Displaying information. Similar to `printf` in C
- Typically used inside “initial” statement.
- A `$display` without any arguments produces a new line.

Format	Display
%d or %D	Display variable in decimal
%b or %B	Display variable in binary
%s or %S	Display string
%h or %H	Display variable in hex

%c or %C	Display ASCII character
%m or %M	Display hierarchical name (no argument required)
%v or %V	Display strength
%o or %O	Display variable in octal
%t or %T	Display in current time format
%e or %E	Display real number in scientific format (e.g., 3e10)
%f or %F	Display real number in decimal format (e.g., 2.13)
%g or %G	Display real number in scientific or decimal, whichever is shorter

System Tasks and Compiler Directives

System tasks

- **\$display**

Usage: \$display(p1, p2, p3,....., pn);

```
//Display the string in quotes
$display("Hello Verilog World");
-- Hello Verilog World

//Display value of current simulation time 230
$display($time);
-- 230

//Display value of 41-bit virtual address 1fe0000001c at time 200
reg [0:40] virtual_addr;
$display("At time %d virtual address is %h", $time, virtual_addr);
-- At time 200 virtual address is 1fe0000001c

//Display value of port_id 5 in binary
reg [4:0] port_id;
$display("ID of the port is %b", port_id);
-- ID of the port is 00101

//Display x characters
//Display value of 4-bit bus 10xx (signal contention) in binary
reg [3:0] bus;
$display("Bus value is %b", bus);
-- Bus value is 10xx

//Display the hierarchical name of instance p1 instantiated under
//the highest-level module called top. No argument is required. This
//is a useful feature)
$display("This string is displayed from %m level of hierarchy");
-- This string is displayed from top.p1 level of hierarchy
```

```
//Display special characters, newline and %
$display("This is a \n multiline string with a %% sign");
-- This is a
-- multiline string with a % sign

//Display other special characters
```

System Tasks and Compiler Directives

System tasks

- **\$monitor**

Usage: `$monitor(p1,p2,p3,.....,pn);`

- Monitor a signal when any one variable or signal changes inside the statement
- Typically used inside “initial” statement
- Needs to be invoked only once
- Only one \$monitor can be active
 - Later one is always active
- On and off by \$monitoron and \$monitoroff

```
//Monitor time and value of the signals clock and reset
//Clock toggles every 5 time units and reset goes down at 10 time units
initial
begin
    $monitor($time,
              " Value of signals clock = %b reset = %b", clock,reset);
end

-- 0 Value of signals clock = 0 reset = 1
-- 5 Value of signals clock = 1 reset = 1
-- 10 Value of signals clock = 0 reset = 0
```

System Tasks and Compiler Directives

System tasks

- \$stop
 - Stops current simulation
- \$finish
 - Terminates the current simulation
 - Exits the current program

```
// Stop at time 100 in the simulation and examine the results
// Finish the simulation at time 1000.
initial // to be explained later. time = 0
begin
  clock = 0;
  reset = 1;
  #100 $stop; // This will suspend the simulation at time = 100
  #900 $finish; // This will terminate the simulation at time = 1000
end
```

56

System Tasks and Compiler Directives

Compiler directives

- 'define
 - Defines text macros (global variables)
 - Similar to #define in C/C++
 - Try not to use it, and use parameters

```
//define a text macro that defines default word size
//Used as 'WORD_SIZE in the code
'define WORD_SIZE 32

//define an alias. A $stop will be substituted wherever 'S appears
'define S $stop;

//define a frequently used text string
'define WORD_REG reg [31:0]
// you can then define a 32-bit register as 'WORD_REG reg32;
```

- 'include
 - Including source files

```
// Include the file header.v, which contains declarations in the
// main verilog file design.v.
#include header.v
...
...
<Verilog code in file design.v>
...
...
```

Homework Assignment

- Design a NOR gate and its testbench.
 - Module
 - NORgate
 - input: A, B
 - output: Z
 - Testbench module
 - NORgate_tb
 - one wire : z_1234 (last four digits should be your student id #)
 - two reg (input): clk1, clk2
 - Design clock cycles as you desire
 - clk1 and clk2 initializes as '0'
- Due: 9/16 8:59am
 - Explain in simple words on how you designed your module and testbench.
 - Screen capture your waveform results.
 - Homework report should not exceed 2 pages. **No cover page required.** Report required to be submitted in English.
 - 2 pages, PDF online, and paper offline. Double-sided report allowed (양면인쇄 가능)

Homework Assignment

A figure of results

