# Introduction to Digital Design with Verilog HDL

*This is a brief introduction to digital circuit design using the System Verilog Hardware Description Language (Verilog HDL). After this lecture you should be able to:*

- *define a module with single- and multi-bit* **`logic`** *inputs and outputs;*
- *write Verilog numeric literals in binary, decimal and hexadecimal bases;*
- *evaluate the value and length of expressions using* **`logic`** *signals, arrays, numeric literals and the operators described below;*
- *draw schematics for Verilog descriptions of multiplexers and registers; and*
- *use* **`assign`** *statements to design combinational logic and* **`always_ff`** *statements to design registers.*

## Introduction

Hardware Description Languages (HDLs) are used to design digital circuits. In this course we will use System Verilog, the modern version of the Verilog HDL, rather than the other popular HDL, VHDL.

Let's start with a simple example – a circuit called an **`ex1`** that has one output (**`y`**) that is the logical AND of two input signals (**`a`** and **`b`**). The file **`ex1.sv`** contains the following Verilog description:

```
// AND gate in Verilog

module ex1 ( input logic a, b,
             output logic y ) ;

   assign y = a & b ;

endmodule
```

Some observations on Verilog syntax:

- Everything following **`//`** on a line is a comment and is ignored.
- Module and signal names can contain letters, digits, underscores (**`_`**) and dollar signs (**`$`**). The first character of an identifier must be a letter or an underscore. They cannot be the same as certain reserved words (e.g. **`module`**).
- Verilog is case-sensitive: **`a`** and **`A`** would be different signals.
- Statements can be split across any number of lines. A semicolon ends each statement.

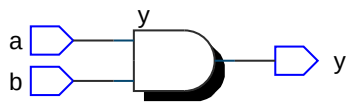
Capitalisation and indentation styles vary. In this course you will need to follow the coding style guide available on the course website.

The module definition begins by defining the input and output signals for the device being designed.

The body of the module contains one or more statements, each of which operates at the same time – *concurrently*. This is the key difference between HDLs and programming languages – HDLs allows us to define concurrent behaviour.

The single statement in this example is a signal assignment that assigns the value of an expression to the output signal **`y`**. Expressions involving **`logic`** signals can use the logical operators **`~`** (NOT), **`&`** (AND), **`^`** (exclusive-OR), and **`|`** (OR). Parentheses can be used to order the operations.

From this Verilog description a program called a logic synthesizer (e.g. Intel's Quartus) can generate a circuit that has the required functionality. In this case it's not too surprising that the result is the following circuit:




If you're familiar with the C programming language you'll note that Verilog uses similar syntax.

**Exercise 1:** What changes would result in a 3-input OR gate?

**Exercise 2:** What schematic would you expect if the statement was `assign y = ( a ^ b ) | c ;`?

## Syntax

### Reserved Words

System Verilog has about 250 reserved words that may not be used as module or signal names. Using one of these (e.g. **`time`**, **`wait`**, **`disable`**, **`reg`**, **`table`**,

`input`, ...) will result in a syntax error. An editor with syntax highlighting will help you identify and avoid using reserved words.

## logic values

Verilog's `logic` signals can have four values: `0` (false or low), `1` (true or high), `z` (high impedance) and `x` (unknown). `z` (high-impedance) is used for synthesis of tri-state outputs. `x` (unknown) is used for an unknown value.

## Arrays

An "array" is group of `logic` signals whose individual elements can be selected by a number. Arrays often represent numerical values in binary form.

The declaration `logic [3:0] a;` specifies an array named `a` with a 'width' of four bits. The bits are numbered from 3 to 0. `a[3]` is the leftmost (most significant) bit and `a[0]` is be the rightmost (least significant) bit.

**Exercise 3:** If the signal `i` is declared as `logic [2:0] i;`, what is the 'width' of `i`? If `i` has the value 6 (decimal), what is the value of `i[2]`? Of `i[0]`?

## Numeric Literals

Numeric literals, often called "constants", are written as a sequence of up to three parts: the number of bits; one of `'b` for binary, `'h` for hex, or `'d` for decimal; and the value. The default width is 32 and the default base is decimal. Underscores (`_`) may be used within the value to improve readability.

**Exercise 4:** What are the sizes and values, in decimal, of the following: `4'b1001`, `5'd3`, `6'h0_a`, `3`?

## Expressions

Logic circuits are defined using expressions as shown above. Operators operate on "operands": `logic` signals, arrays, and numeric literals. Operators with higher "precedence" are applied before those of lower precedence. If two operators are of equal precedence they are applied from right to left. Operator can change the number of bits in an expression (the "length") as they're applied. Values are padded with zeros on the left when necessary.

## Operators

The following sections describe some useful Verilog operators in order of decreasing precedence.

**Slices** A range of bits in an array (a "slice") can be extracted using a range of "indices" in brackets (`[`*first*`:`*last*`]`) after the array name. The bit order cannot be reversed. The length is the number of bits in the slice.

**Negation** Logical negation (`!`) is zero (0) when applied to a non-zero operand and one (1) otherwise. The length is 1 bit.

Bitwise negation (`~`) inverts the value of each bit. The length is the width of the operand.

**Arithmetic** Multiplication (`*`) , division (`/`), addition (`+`) and subtraction (`-`) can be applied to arrays. The first two have higher precedence. The length is the largest of the two operand's lengths.

**Comparison** The comparison operators (`<`, `>`, `<=`, `>=`, `==`, `!=`) can be applied to arrays. The result is 1 if the comparison is true and 0 otherwise. The length is 1 bit.

**Bitwise Logical** Bitwise logical operators (`&`, `|`, `^`) are applied to the corresponding bits in two operands. The length is the largest of the two operand's lengths.

**Logical** The logical AND operator (`&&`) has value 1 if both operands are non-zero. The logical OR operator (`||`) has value 1 if either operand is non-zero. The length is 1 bit.

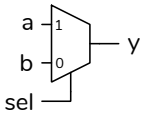
**Exercise 5:** An array declared as `logic [15:0] n;` and has the value `16'h1234`.

What are the values and lengths of the following expressions?

```
n[15:13]

!n

~n[3:0]

n + 1'b1

n[7:0] - n[3:0]

n >= 16'h1234

n ^ ~n

n && !n

 n * ( !n + 1'b1 )
```

**Conditional Operator** Verilog's conditional operator is a concise syntax for describing a two-way multiplexer. The operator consists of three parts: the condition, the true value and false value. The result of the operator is the true value if the condition is non-zero, or the false value otherwise. The length is the largest of the true and false value lengths. For example:

`assign y = sel ? a : b` ⇔



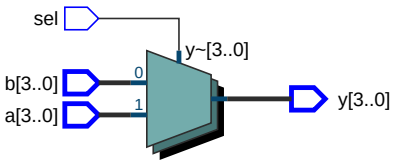

sets **y** to **a** when **sel** is non-zero and sets **y** to **b** whenever **sel** is zero.

**Exercise 6:** What is the value of the expression `3 ? 10 : 20`? Of the expression `x ? 1 : 0` if **x** has the value 0? If **x** has the value -1?

The following example implements a multiplexer that selects from one of two 4-bit inputs:

```
module ex36 (input logic sel,
             input logic [3:0] a, b,
             output logic [3:0] y) ;

   assign y = sel ? a : b ;

endmodule
```

which results in:




Conditional operators can be used to concisely describe trees of multiplexers. In the absence of parentheses a sequence of ternary operators is evaluated from left to right.

**Exercise 7:** Draw the schematics corresponding to:

```
y = a ? ( b ? s1 : s2 ) : ( c ? s3 : s4 );
```

```
y = a ? s1 : b ? s2 : c ? s3 : s4;
```

**Concatenation** The concatenation operator ({,}) combines expressions into a longer value. The length is the sum of the lengths.

**Exercise 8:** Use slicing and concatenation to swap the order of the bytes in **n**.

What is the value and length of `{n[7:0],n[15:8],4'b1111}`?

Use concatenation to shift **n** left by two bits.

Concatenations of variables can be used on the left hand side of an assignment.

**Assignment** Assignments (`=`, `<=`) connect the right-hand side (RHS) expression to the signal on the left-hand side (LHS). Use `=` for **`assign`** statements and `<=` for **`always_ff`** statements.

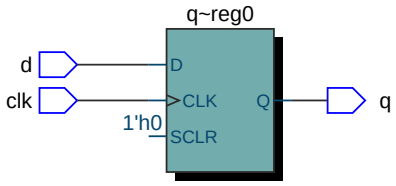
The length is the length of the LHS. The left-most bits are dropped if the RHS is wider than the LHS.

## Registers

The following Verilog:

```
module ex2 (input logic d, clk,
            output logic q) ;

   always_ff @(posedge clk)
     q <= d ;

endmodule
```

synthesizes a D-flip-flop that transfers the **d** input to the **q** output on the rising (positive) edge of **clk**:




**Exercise 9:** What would you change to make an 8-bit register? A 4-bit counter? A 3-bit shift register? Follow the course coding conventions.