ELEX 2117 : Digital Techniques 2
2021 Fall Term

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 expressions using* **`logic`** *signals and operators;*
- *write Verilog numeric literals in binary, decimal and hexadecimal bases.*
- *use* **`assign`** *statements to generate combinational logic;*

Introduction

Most of the features of modern electronics are defined in software. But for certain tasks software can be too slow or a processor can be too expensive or can consume too much power. In these cases we need to design specialized digital circuits. This course teaches how to do this.

Today, digital circuit designers use Hardware Description Languages (HDLs) instead of drawing schematics. In this course we will use System Verilog, the modern version of the Verilog HDL, rather than the other popular HDL, VHDL.

Combinational Logic

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.

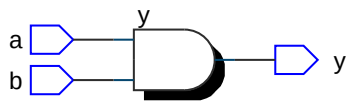
Capitalisation and indentation styles vary. In this course you will need to follow the coding style guide available on the course web site.

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 define the order of evaluation.

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 the same syntax for its bitwise operators.

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 ;`**?

Buses and Packed Arrays

A group of logic signals that is treated together is called a 'bus'. A bus is described in Verilog using a "packed array". A packed array can also be treated as a binary number.

The declaration `logic [3:0] a;` specifies a packed 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]`?

Verilog has many operators for working with packed arrays: arithmetic (`+, -, *, /, %`), comparison (`==, <, !=`, etc. equal, less-than and not-equal) and shift operators (`<<, >>`). These operators are similar to those in the C programming language. We will cover them in more detail later.

Verilog uses brackets (`[]`) in expressions for indexing and "slicing" – extracting parts of a packed array. Braces (`{}`) are used to combine signals into multi-bit values. For example, the expression `{a[0],a[1],a[2],a[3]}` would be a bit-reversed version of the packed array defined above. The order of the indices in a slice must be the same as in its declaration.

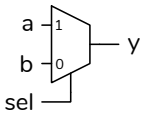
Exercise 4: Use slicing and concatenation to assign **y** the value of the byte **x** with its nybbles swapped.

C syntax is also used for comments – `//` for single-line and `/*` ... `*/` for multi-line.

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. 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 5: 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?

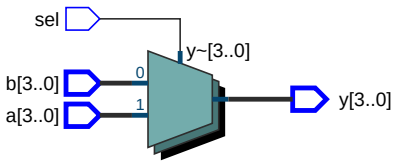
The following example implements a multiplexer that selects from one of two 4-bit inputs:

```
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:



Multiple 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 6: Draw the schematic corresponding to: `y = a ? ( b ? s1 : s2 ) : ( c ? s3 : s4 )`

Exercise 7: Write a Verilog description of a 4-bit 3-to-1 multiplexer controlled by a 2-bit `sel` input? Label the inputs **a** (for `sel=00`) through **c** (for `sel=10`).

Numeric Literals

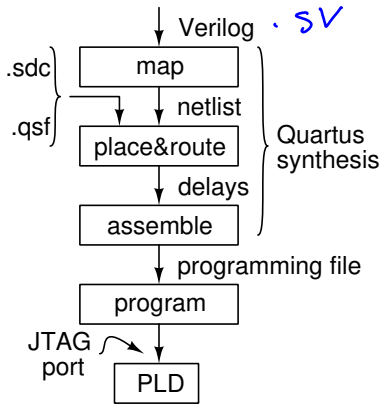
"Literals" (numeric constants) are written as a sequence of up to three parts: the number of bits, the base denoted with a quote[1] followed by a letter (`'b` for binary, `'h` for hex or `'d` for decimal), and the value. The default width is 32 and the default base is decimal. Underscores (_) may be used within the value to improve readability.

Exercise 8: What are the sizes and values, in decimal, of the following: 4'b1001, 5'd3, 6'h0_a, 3?

Implementation

How is Verilog used to implement logic circuits with a programmable logic device (PLD) such as a CPLD (Complex Programmable Logic Device) or FPGA (Field Programmable Gate Array)?

[1] Often pronounced "tick".



The design is first mapped by logic synthesis software (e.g. Quartus) to elementary logic functions such as gates and flip-flops. The result is a netlist – a list of logic functions and how they're connected.

A "Place and Route" (P&R) step then assigns each of the logic functions to one of the logic elements in the PLD. This requires additional information such as the part and the pin assignments. These are supplied in the **`.qsf`** (Quartus settings) file. For example, your **`.qsf`** file might contain the lines:

```
set_global_assignment -name DEVICE EPM240T100C5
set_location_assignment PIN_2 -to clk_in
   ...
set_location_assignment PIN_44 -to led[3]
```

Timing constraints such as clock frequencies are defined in a **`.sdc`** (Synopsis Design Constraint) file. For example, the following statement requires that the design operate correctly if the signal **`CLOCK_50`** has a 50 MHz (20 ns period) clock:

```
create_clock -period 20ns CLOCK_50
```

Finally, the placed and routed design is "assembled" to a file that can program the device, typically over a dedicated "JTAG" programming/diagnostic interface on the PLD.