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;*
- *use initialized unpacked arrays to implement arbitrary combinational logic functions*

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.

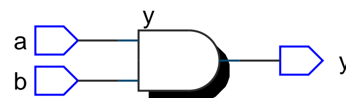
Capitalisation and indentation styles vary. In this course you will need to follow the coding style guide available on the course web site.

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 most of its operators including arithmetic (**`+, -, *, /, %`**), bitwise (**`&, |, ^, ~, <<, >>`**), comparison (**`>, >=, !=`**, etc.), logical (**`&&, ||, !`**),

array indexing (`[ ]`), and ternary conditional (`?:`). C syntax is also used for comments.

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 ;`?

Exercise 3: Write two `assign` statements that produce the same result. Does their order in the module matter?

The output of a circuit such as this is a function only of the current combination of input values and is called a “combinational” logic circuit. “Sequential” logic circuits include memory components and so the current output can be a function of previous inputs as well as the current input.

Buses and Multiplexers

Verilog’s conditional operator models a two-way multiplexer. 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:

The ternary (3-input) conditional operator uses the value before the `?` to select one of the two values separated by `:`. If the first value is non-zero then the second value is selected, otherwise the third value is selected.

Exercise 4: 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?

A group of logic signals that is treated together is called a ‘bus’. The declaration `logic [3:0]` specifies a bus with a ‘width’ of four bits. The bits in this bus will be numbered from 3 to 0. For example, `a[3]` would be the leftmost (most significant) bit of the 4-bit bus `a`.

Exercise 5: 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]`?

Exercise 6: Write the truth table and Verilog description of a 3-bit 4-to-1 multiplexer controlled by a 2-bit `sel` input? Label the inputs `a` (for `sel=00`) through `d` (for `sel=11`).

Literals and Unpacked Arrays

Numeric constants (“literals”) in Verilog are written as the number of bits (default 32), an optional base denoted with a quote[1] and letter (`'b`=binary, `'h`=hex, `'d`=decimal), and the value. The default size is 32 bits and the default base is decimal. Underscore separators (`_`) may be used to improve readability.

In the following example, `a`, is a two-bit signal which selects the element of the “unpacked array” `lut` to be output on `d`. In this example the array is initialized to four constant 4-bit values. The output `d` is selected (“indexed”) by the values, 0 through 3, of `a`. The first value in the array is `lut[0]` and has the value `4'b1000` while `lut[3]` is the final value and has the value `4'b0011`.

```
module ex37 (input logic [1:0] a,
             output logic [3:0] d) ;

   logic [3:0] lut [4] =
        '{ 4'b1000, 4'd1, 4'ha, 3 }  ;

   assign d = lut[a] ;

endmodule
```

which synthesizes into:

Exercise 7: What are the values, in decimal, of the constants in the code above?

Exercise 8: What is the output, in binary, when the input is `a=2'b10` ?

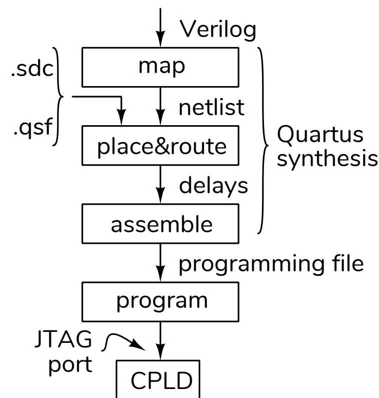
Unpacked arrays allow us to implement arbitrary combinational logic functions. The input is the array index and the output is the value of the array at the indexed location.

[1] Often pronounced “tick”.

Exercise 9: Write a Verilog module with a two-bit input `i` and a two-bit output `n` that uses an unpacked array to output the number of bits in the input that are "1". Start by writing the truth table.

Implementation

The process to implement a design using programmable logic device (PLD) such as a CPLD (Complex Programmable Logic Device) or FPGA (Field Programmable Gate Array) is shown below.



The design is first mapped to logic functions such as gates and flip-flops. The result is a netlist – a list of logic functions and how they're connected.

The "Place and Route" step then assigns each of the logic functions to one of the programmable logic elements in a specific device. This requires additional information such as the device type (part number) 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 PLD, typically over a dedicated "JTAG" programming/diagnostic interface port on the PLD.