ELEX 2117 : Digital Techniques 2
2020 Fall Term

Introduction to Digital Design with Verilog HDL

This is an introduction to digital logic circuit design using the System Verilog Hardware Description Language. Details will be covered later.
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;*
- *use* `assign` *statements and* `always_comb` *procedural blocks to generate combinational logic;*
- *use* `if` *and* `case` *statements to model multiplexers and arbitrary combinational logic functions;*
- *write Verilog numeric literals in binary, decimal and hex bases.*

Introduction

Most of the functionality of modern electronics is defined by its software. But software can be too slow, or a processor too expensive or consumes too much power for certain applications. In these cases we may need to design custom digital hardware. This course explains how to design such circuits.

Today, all but the simplest designs are written using a Hardware Description Language (HDL) rather than by drawing schematics. In this course we will use System Verilog 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 ~~`example1.vhd`~~ `ex1.sv` contains the following ~~VHDL~~ 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.

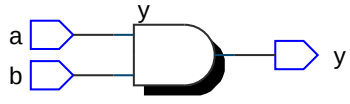
Capitalisation and indentation styles vary. In this course you will need to follow the coding style guide available on the course web site.

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

as the 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 ;`?

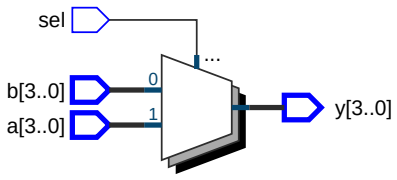
The output of a circuit such as this is a function only of the current combination of input values and is called a "combinational" logic circuit. "Sequential" logic circuits include memory components and so the current output can be a function of previous inputs as well as the current input.

Buses and Multiplexers

Verilog's `if` statement models a two-way multiplexer. The following example implements a multiplexer that selects from one of two 4-bit inputs:

```
module ex3 (input logic sel,
            input logic [3:0] a, b,
            output logic [3:0] y) ;

   always_comb begin
      if ( sel )
         y <= a ;
      else
         y <= b ;
   end

endmodule
```

which results in:



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`.

The `always_comb` statement, like the `assign` statement, is concurrent and operates at the same time as other statements in the module. However, the statements within `always_comb`, are "sequential" and are evaluated in order.

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]`?

Exercise 4: What changes might result in a 4-bit 4-to-1 multiplexer controlled by a 2-bit `sel` input?

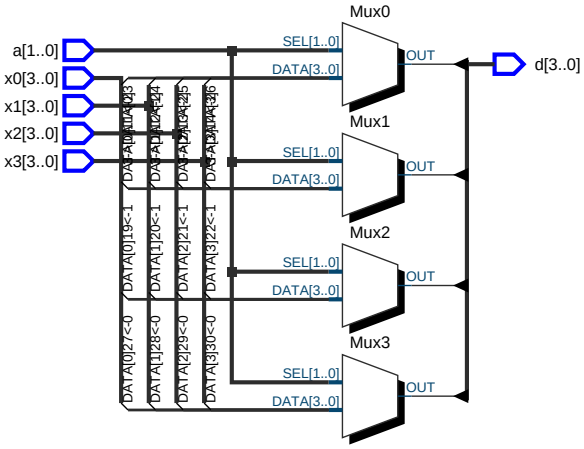
Case Statements and Numeric Constants

A Verilog `case` statement can model a multiplexer with more than two inputs.

In the following example, `a`, is a two-bit signal which selects one of four signals to be connected to the output `d`.

```
module ex34 (input logic [1:0] a,
             input logic [3:0] x0, x1, x2, x3,
             output logic [3:0] d) ;

   always_comb begin
      unique case (a)
        0: d = x0 ;
        1: d = x1 ;
        2: d = x2 ;
        default: d = x3 ;
      endcase
   end

endmodule
```

which synthesizes into:



`case` is a sequential statement that "executes" one statement as selected by the expression following `case`. If none of the values match then the value following `default` is executed. Always include a `default` case[1].

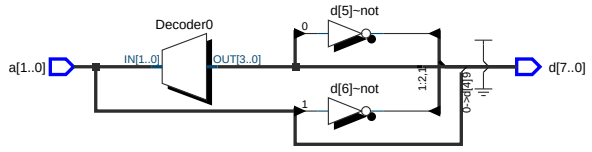
The selected values can be constants:

```
module ex4 (input logic [1:0] a,
            output logic [7:0] d) ;
```

[1] We will cover exceptions later.

```
   always_comb begin
      unique case (a)
        0: d = 8'hc0 ;
        1: d = 8'b1111_1001 ;
        2: d = 'ha4 ;
        default: d = 176 ;
      endcase
   end

endmodule
```

which synthesizes into:



and is way to create arbitrary logic functions.

Numeric constants ("literals") in Verilog are written as the number of bits (default 32), an optional quote (`'`) followed by the base (`b`=binary, `h`=hex, `d`=decimal), and the value. Underscore separators (_) are optional.

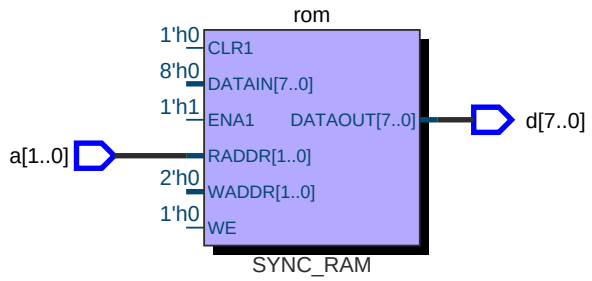
Exercise 5: What is the output in binary when the input is `a=2'b10` ?

Exercise 6: What are the values in decimal of the constants in the code above?

A concise way to define look-up tables is to use "unpacked arrays":

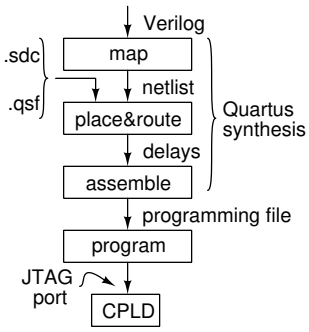
```
module ex35 (input logic [1:0] a,
             output logic [7:0] d) ;

   logic [7:0] rom [0:3] =
      '{ 8'hc0,  8'hf9, 8'ha4, 8'hb0 } ;

   assign d = rom[a] ;

endmodule
```

which synthesizes into a memory:



Implementation

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



After the design is mapped to gates and other logic elements it must be fit into a specific device. Additional information needed to "place and route" the design is supplied in two files. The `.qsf` (Quartus settings) file device contains, among other things, the device type (part number) and the pin assignments. For example:

```
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 and external device setup/hold times are defined in a `.sdc` (Synopsis Design Constraint) file. For example, the following statement requires that the design operate with a 50 MHz (20 ns period) clock signal in the design named `CLOCK_50`:

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

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