Combinational Logic Design with VHDL

This lecture introduces the use of VHDL for combinational logic design.
After this lecture you should be able to convert an informal description of a combinational logic circuit into a VHDL entity and architecture.

VHDL is a Very-complex[1] Hardware Description Language that we will use to design logic circuits.

Example 1 - Signal Assignment

Let's start with a simple example – a type of circuit called an example1 that has one output signal (`c`) that is the AND of two input signals (`a` and `b`). The file `example1.vhd` contains the following VHDL description:

```
-- example 1: An AND gate

entity example1 is port (
   a, b: in bit ;
   c: out bit ) ;
end example1 ;

architecture rtl of example1 is
begin
      c <= a and b ;
end rtl ;
```

First some observations on VHDL syntax:

- VHDL is case-insensitive. There are many capitalization styles. I prefer all lower-case. You may use whichever style you wish as long as you are consistent.
- Everything following two dashes "`--`" on a line is a comment and is ignored.
- Statements can be split across any number of lines. A semicolon ends each statement. Indentation styles vary but an "end" should be indented the same as its corresponding "begin"
- Entity and signal names begin with a letter followed by letters, digits or underscore ("_") characters.

A VHDL description has two parts: an entity part and an architecture part. The entity part defines the input and output signals for the device or "entity" being designed while the architecture part describes the behaviour of the entity.

Each architecture is made up of one or more statements, all of which "execute[2]" at the same time (*concurrently*).

The single statement in this example is a signal assignment that assigns the value of an expression to the output signal `c`. Expressions involving signals of type bit can use the logical operators `and`, `nand`, `or`, `nor`, `xor`, `xnor`, and `not`. Parentheses can be used to force evaluation in a certain order.

From this VHDL description a program called a logic synthesizer can generate a circuit that has the required functionality. In this case it's not too surprising that the result is the following circuit:

Exercise: Write a VHDL description for the circuit that would generate the 'a' and 'b' outputs for the 7-segment LED driver shown previously.

Example 2 - Selected Assignment

The selected assignment statement mimics the operation of a multiplexer – the value assigned is selected from several other expressions according to a controlling expression. The following example describes a two-input multiplexer:

```
entity mux2 is
   port (
   a, b : in bit ;
   sel : in bit ;
   y : out bit ) ;
```

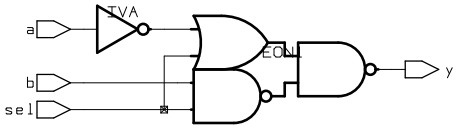
[1] Actually, the V stands for VHSIC. VHSIC stands for Very High Speed IC.

[2] The resulting hardware doesn't actually "execute" but this point of view is useful when using VHDL for simulation.

```
end mux2 ;

architecture rtl of mux2 is
begin
   with sel select y <=
      a when '0' ,
      b when others ;
end rtl ;
```

which synthesizes to:



Note the following:

- the keyword "others" indicates the default value to assign when none of the other values matches the selection expression
- commas separate the clauses

Example 3 - Bit Vectors

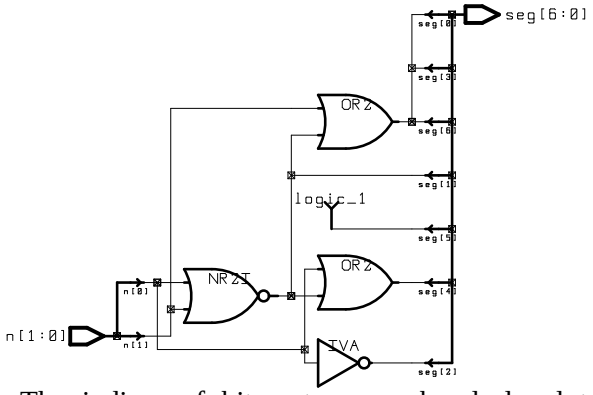
VHDL also allows signals of type `bit_vector` which are one-dimensional arrays of bits that model buses. Using bit_vectors and selected signal assignments we can easily convert a truth table into a VHDL description. The next example is a VHDL description of the 7-segment LED driver:

```
-- example 3: 7-segment LED driver for
--    2-bit input values

entity led7 is port (
   n: in bit_vector (1 downto 0) ;
   seg: out bit_vector (6 downto 0) ) ;
end led7 ;

architecture rtl of led7 is
begin
   with n select seg <=
      "1111111" when "00" ,
      "0110000" when "01" ,
      "1101101" when "10" ,
      "1111001" when others ;
end rtl ;
```

which synthesizes to:



The indices of bit_vectors can be declared to have increasing (`to`) or decreasing (`downto`) values. `bit_vector` constants are formed by enclosing an ordered sequence of bit values in double quotes.

Exercise: If `x` is declared as `bit_vector (0 to 3)` and in a process the assignment `x<="0011"` is made, what is the value of `x(3)`? What if `x` had been declared as `bit_vector (3 downto 0)`?

Substrings of vectors can be extracted by specifying a range in the index expression and vectors can be concatenated using the `&` operator. For example `x <= x(6 downto 0) & '0'` would shift the 8-bit value `x` left by 1 bit.

The logical operators (e.g. `and`) can be applied to bit_vectors and operate on a bit-by-bit basis.

Exercise: Write a VHDL description for a 2-to-4 decoder using a 2-bit input and a 4-bit output.