More VHDL

This chapter covers a few additional VHDL features that are commonly used for logic design.
You should be able to: declare enumerated types and subtypes of array types in architectures and packages; declare and use entities with generics; instantiate tri-state outputs; and create RAM and ROM memories.

Type Declarations

It's often useful to make up new types for a project. We can do this in VHDL by using type declarations. The most common uses for defining new types are to create signals of a given width (i.e. a bus) and to declare types that can only have one of a set of possible values (called enumeration types).

Type declarations are often placed in packages to make them available to multiple design units. The following example shows a package called `dsp_types` that declares two new types:

```
package dsp_types is
   type mode is (slow, medium, fast) ;
   subtype word is std_logic_vector (15 downto 0) ;
end dsp_types ;
```

Note that we need to use a `subtype` declaration in the second example because the `std_logic_vector` type is already defined.

Exercise 1: Write a declaration for a signal that indicates whether the value in a register should be loaded, incremented, decremented, or held. Write the declaration for an 8-bit signal type called `byte`.

Generics

An entity can be declared with a bus or register size that is left undefined until the component is used ("instantiated") by adding a *generic* clause in its entity and component declarations. For example, a register with negated outputs could be declared in the file `nregister.vhd` as:

```
-- register with negated output

entity nregister is
   generic ( width : integer ) ;
   port ( d : in  bit_vector (width-1 downto 0) ;
          q : out bit_vector (width-1 downto 0) ;
          clk : in bit ) ;
end nregister ;

architecture rtl of nregister is
   signal tmp : bit_vector(width-1 downto 0) ;
begin
   process(clk)
   begin
      if clk'event and clk='1' then
         tmp <= d ;
      end if ;
   end process ;
   q <= not tmp ;
end ;
```

you might declare the `nregister` component in a package as:

```
package registers is
   component nregister
      generic ( width : integer ) ;
      port ( d : in  bit_vector (width-1 downto 0) ;
             q : out bit_vector (width-1 downto 0) ;
           clk : in bit ) ;
   end component ;
end registers ;
```

and then use it in another architecture as follows:

```
use work.registers.all ;
...
   r1: nregister
      generic map ( 8 )
      port map ( din, dout, clk ) ;
...
```

You should use generics if your component might have to be instantiated with various signal widths.

Attributes

Each signal has a number of properties associated with it which can be extracted and used in expressions by using VHDL's *attributes*. For example, the number of elements in an array `x` is given by `x'length`. Other useful attributes are `left`, `right`, `high`, `low` which extract the appropriate index limits and `range` which extracts the index range.

Exercise 2: Which attribute have we been using to make register outputs change only when the clock signal changes?

Tri-State Buses

A tri-state output can be set to high and low logic levels as well as to a third state: high-impedance ('Z'). This type of output is used where different devices' outputs are connected together and drive a common bus (hopefully at different times!). To specify that an output should be set to the high-impedance state, we use a signal of type `std_logic` and assign it a value of 'Z'.

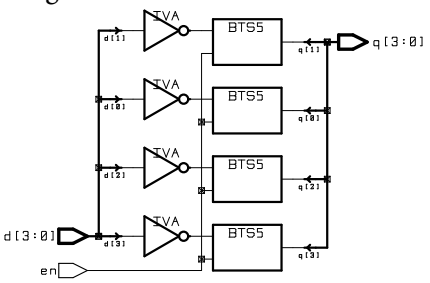
The following example shows an implementation of a 4-bit buffer with an enable output. When the enable is not asserted the output is in high-impedance mode :

```
library ieee ;
use ieee.std_logic_1164.all ;

entity tbuf is port (
   d : in std_logic_vector (3 downto 0) ;
   q : out std_logic_vector (3 downto 0) ;
   en : in std_logic
   ) ;
end tbuf ;

architecture rtl of tbuf is
begin
   q <=
      d when en = '1' else
      "ZZZZ" ;
end rtl ;
```

The resulting schematic for the `tbuf` is:



Tri-state outputs are used primarily to implement bidirectional bus signals. Bidirectional buses are declared of type `inout` rather than `in` or `out` and their values can be both 'read' and 'written' within the architecture (unlike signals of type `out`). When the bus is to act as an input, the bidirectional bus signals are driven to the high-impedance state and in this case it's the value of other signals that determine the signal's value.

The `std_logic` types use the following table[1] to determine the result when a signal is multiply-driven:

```
CONSTANT resolution_table : stdlogic_table := (
--      ---------------------------------------------------------
--      |  U    X    0    1    Z    W    L    H    -        |   |
--      ---------------------------------------------------------
        ( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- | U |
        ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | X |
        ( 'U', 'X', '0', 'X', '0', '0', '0', '0', 'X' ), -- | 0 |
        ( 'U', 'X', 'X', '1', '1', '1', '1', '1', 'X' ), -- | 1 |
        ( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X' ), -- | Z |
        ( 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X' ), -- | W |
        ( 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X' ), -- | L |
        ( 'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X' ), -- | H |
        ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' )  -- | - |
    );
```

For example, a signal driven by `1` and `Z` "resolves" to `1`.

Exercise 3: What is the result of driving an `std_logic` signal with both '1' and '0'?

The tri-state enable is usually controlled by an address decoder or other logic that "arbitrates" bus use.

Memory Models

VHDL also allows the use of arrays with signal indices to model random-access memory (RAM). The following example demonstrates the use of VHDL arrays as well as bi-directional buses. We must use the type-conversion function `to_integer` because the address input, `a`, is of type `unsigned` while the array index must be of type `integer`.

```
library ieee ;
use ieee.std_logic_1164.all ;
use ieee.numeric_std.all ;

entity ram is   port (
   -- bi-directional data signal
   d : inout std_logic_vector (7 downto 0) ;
   -- address input
   a : in unsigned (1 downto 0) ;
   -- output enable and write strobe (clock)
   oe, wr : in std_logic ) ;
end ram ;



architecture rtl of ram is
   subtype byte is std_logic_vector (7 downto 0) ;
   type byte_array is array (0 to 3) of byte ;
   signal ram : byte_array ;
begin
   -- output value is the indexed array element
   d <=
      ram(to_integer(a)) when oe = '1' else
      "ZZZZZZZZ" ;

    -- register the indexed array element
```

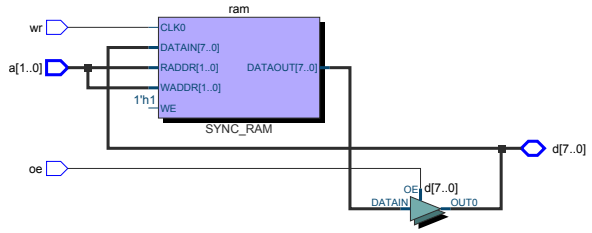
[1] From the IEEE 1164-1993 standard.

```
   process(wr)
   begin
      if wr'event and wr = '1' then
         ram(to_integer(a)) <= d ;
      end if ;
   end process ;
end rtl ;
```

Exercise 4: Modify the design above to create a 16-element, 4-bit wide RAM with separate input and output signals.

Exercise 5: How would you model a ROM?

The result of synthesizing this description using Quartus II is:



For many implementation technologies (FPGAs, gate arrays, or standard-cell ASICs) there are usually vendor-specific ways of implementing memory arrays that give better results. However, using a VHDL-only model as shown above is more portable and may be practical for small memories such as CPU "register files."

Exercise 6: Why is portability desirable?