HDL Idioms

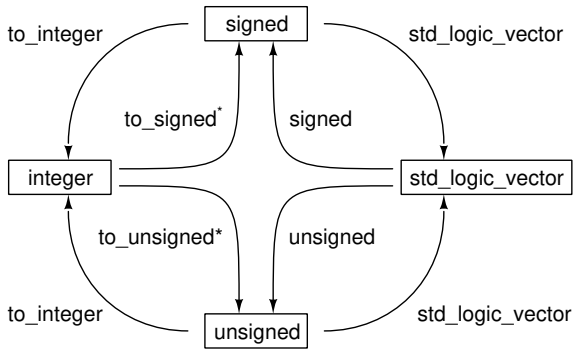
Additional VHDL

Reserved Words

VHDL has many reserved words, including some that you may inadvertently use as identifiers in your designs such as **`bus`**, **`next`**, **`open`**, **`register`**, or **`time`**. Using an editor with syntax highlighting is a convenient way to identify and avoid using reserved words.

Type Conversions

The diagram below shows the available type conversion functions. The functions marked with an asterisk require a second argument that specifies the number of bits in the result.



*specify number of bits as a second argument

For example, to convert an **`std_logic_vector`** signal **`x`** to an **`unsigned`** signal **`y`** you would use: **`y <= unsigned(x)`**. To convert an **`integer`**[1] to a 4-bit **`unsigned`** signal you would use: **`y <= to_unsigned(x,4)`**.

Modeling Memory with Arrays

Arrays in VHDL can be used to model look-up tables (memories). For example a lookup table with 16 elements, each of which is an 8-bit **`std_logic_vector`** could be declared and used as follows:

[1]Integers are 32 bits.

```
entity decoder is
  port ( led_out : out std_logic_vector (7 downto 0) ;
         digit : in unsigned (1 downto 0) ) ;
end decoder ;

architecture rtl of decoder is
  type byte_array is array (natural range <>) of
    std_logic_vector(7 downto 0) ;

  signal led_decoder : byte_array(0 to 3)
     := ( 8x"7E", 8x"30", 8x"6D", 8x"79" ) ;
begin
     led_out <= led_decoder(to_integer(digit)) ;
end rtl ;
```

This code declares and initializes a lookup-table with 16 8-bit values numbered 0 to 15. **`led_out`** is set to the value of the **`digit`**'th element in the **`led_decoder`** table.

This is a more concise way of defining lookup tables than a conditional or selected assignment.

Tri-State Outputs

Tri-state outputs (those with H, L and high-impedance output states) can be modeled using **`std_logic`**. When such outputs are assigned the value '**`Z`**' the output is placed in the high-impedance ("tri-stated") mode.

```
-- as part of a port map:
x : output std_logic ;
-- in an architecture:
x <= b when oe_n = '0' else 'Z' ;
```

Common HDL Idioms

Logic synthesizers such as Quartus convert HDL descriptions into circuits (actually, netlists). They do this by recognizing a relatively small number of idioms.

You must be able to visualize the hardware that would be generated by an HDL description in order create efficient designs. This lecture describes some common HDL constructs and their corresponding hardware implementations.

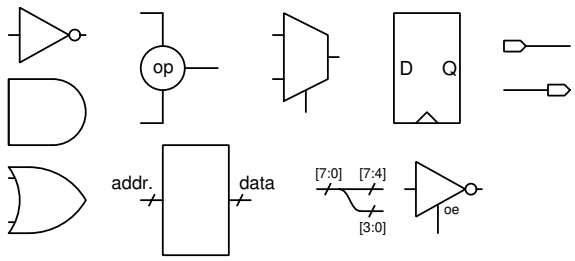
Combinational Logic

The following HDL constructs are typically synthesized as follows:

- logical operators are converted to the equivalent logic gates as you would expect.
- arithmetic and comparison operators are converted to blocks of combinational logic that implement the required operation.
- access to a value in an array is implemented as a read-only memory (ROM). Access to bits of **`unsigned`** or **`std_logic_vector`** signals is implemented simply as a connection to specific bits.
- conditional and selected assignments are converted into multiplexers; nested as necessary.
- an output to which '**Z**' can be assigned is converted to a tri-state output.

Exercise 1

Using the following schematic symbols:



and these declarations:

```
type byte_array is array (natural range <>) of
  std_logic_vector(7 downto 0) ;
signal a, b, c, d, w, clk : std_logic ;
signal x, y, z, y_next, n : unsigned (7 downto 0) ;
signal r, s : unsigned (31 downto 0) ;
signal m : byte_array (0 to 3) ;
signal p : unsigned(1 downto 0) ;
```

convert each of the following VHDL expressions into a schematic:

```
c <= a nand b ;

y <= x+1 ;

a <= x(3) ;

x <= unsigned(m(to_integer(y(7 downto 6)))) ;

z <= x when a = '1' else  y;

d <= a when x = x"ff" else
     b when y = x"00" else
     c ;

y <= x"30" when x(2) = '1' else
     x"20" when x(1) = '1' else
     x"10" when x(0) = '1' else
     x"00" ;

x <= y+1 when y < z else y-1 ;

a <= b when c = '1' else 'Z' ;

n <= x"07" when a = '1' else
     x-1 when y = z and b = '1' else
     n ;

r <= s(30 downto 0) & a ;

d <= not n(7) ;
```

Sequential Logic

The following HDL constructs are synthesized as follows:

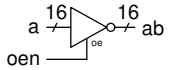
- conditional assignments without an unconditional **`else`** are converted to flip-flops or registers. The signal assigned to is the register output.
- combinational logic that computes the values assigned to register outputs are used to describe specialized sequential logic such as counters and shift registers. The register's current value is often used to compute the next value.

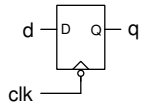
Exercise 2

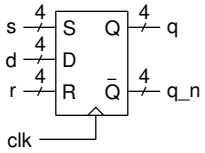
Using the schematic symbols and declarations above, convert each of the following VHDL expressions into a schematic:

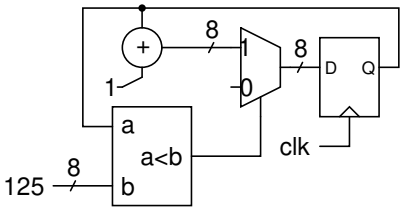
```
y <= x when rising_edge(clk) ;

y <= y(6 downto 0) & a when rising_edge(clk) ;

y_next <= x when a = '1' else y ;
y <= y_next when rising_edge(clk) ;


y_next <= x"00" when b = '1' else
       y+1 when a = '1' else
       y ;
```

```
y <= y_next when rising_edge(clk) ;


y_next <= x"00" when a = '1' or b = '1' else
       y+1 ;
y <= y_next when rising_edge(clk) ;


b <= x(7) when rising_edge(clk) ;


m(to_integer(p)) <= std_logic_vector(x) when rising_edge(clk) ;


x <= unsigned(m(to_integer(p))) ;


p <= p+1 when a = '1' and b = '1' ;
```





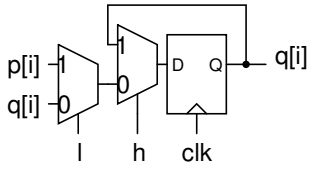


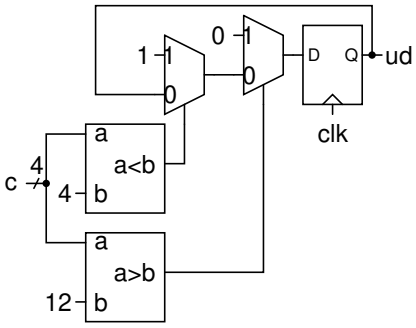


Schematics to HDL

It's also important to be able to write the HDL that will result in a specific hardware architecture.

Each circuit element is converted into the corresponding HDL construct and named signals are used to connect them according to the circuit topology.





Exercise 3

Write VHDL that would generate each of the following schematics. Include any required signal declarations.